Content-based Band Recommender System

Hu Junbo 14250926
Sun Jingxuan 14252392
Li Shiying 14251205
Ng Kwan Ho 14222264

1. Introduction

1.1 Background information

With the development of Internet industries, applications have appeared madly. Competitive pressure on similar products has increased dramatically.

In current environment of network society, user-individual functions based on artificial intelligence technology have gradually become a selling point for various application products to occupy the market and increase user stickiness. Just as Zhang Yiming, the founder of Jinri Toutia said before, tolerating users like mothers, providing them with what they want and gradually making users addicted to the application is the core secret that how this application became the largest media channel in China today.

1.2 User requirements

There are two things we need to pay attention to in terms of user needs. The first feature is user laziness. Compared to hands-on search, users want automatically provided information which is based on their preferences. This is the reason why people in mainland China seldom use Baidu to search information especially news. Nowadays, there are many news applications providing particular news by knowing users' habits. Secondly, the users' expectations for the product has changed. They are more and more concerned about the social functionality of the products.

1.3 Product vision

As the hot situation of musical video community applications, such as DouYin and Musical.ly, we can positively expect that these kinds applications will take the position of some traditional music applications. And after analyzing user requirements and background information, the first thing we need to do is to establish a music or musicians recommending system to let users more and more dependent on our applications. And this band recommending system is based on users' music lists. After scratching the music data, our system will analyze the data by two independent methods. We use the audio quantitative

analysis as the first method. This method can evaluate every song in 20 different dimensions. After the evaluation, we will get a data set for each song. Then we can easily find similar songs based on these data sets. Our second method is based on lyrics of songs. After preprocessing words of songs, we calculate the TF-IDF for a particular user music list. Then we just compare the number with other TF-IDF numbers of other music lists and recommend the most similar ones to user.

With these two core algorithms, our system can automatically provide similar banks which have similar music style of the music in the user list.

2. Data collection

2.1 Oerview

We choose a set of 100 bands and pool their lyrics as well as songs as ground truth. In the process of choosing suitable bands, several points are considered.

1. Sufficiency of data. Band history should be long enough to provide more than 30 songs.
2. Integrity of data. For the convenience of final evaluation, the characteristics of the music should not observe significant change (eg. change of music gnere).
3. Discriminating ability. For the convenience of final evaluation, the soundscape of bands from different style lists or genres should be able to be distinguished clearly by human.
4. Data source. Lyrics and songs should be found easily without copyright violation. Balancing the criteria above, 100 bands are selected and data is collected with the following techniques.

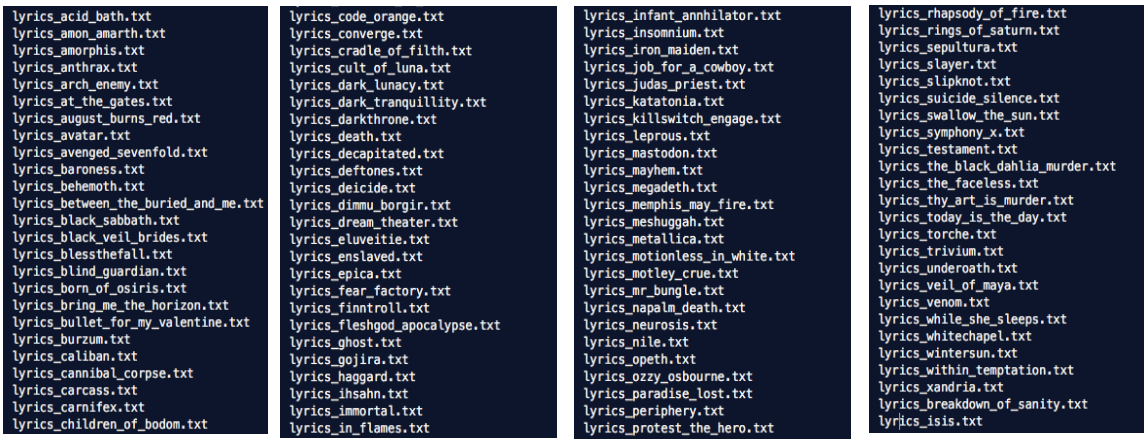

Figure 1. List of bands

2.2 Lyrics collection

As for text feature extraction, band song lyrics are collected from an online music database, darklyrics.com.[1] The essential part of web scraping is to analyze the structure of the html file. As our final goal is to recommend bands, lyrics of one band are treated as a unit. The architecture of darklyrics is:

1. For each search of band, there is one band's discography page, which contains the list of all albums from certain band;
2. For each album of the band, the album page lists all song URLs from that album on the top, below which are all song lyrics.

It is very convenient for our pre-processing for the TFIDF calculation. For more choices of working-around in text feature extraction step, it is designed to scrape all lyrics of one band into one file. Please refer to Section 3 for further lyrics processing.

Figure 2. Architecture of Darklyrics

Scripts are written to scrape the content from URLs with the Python library BeautifulSoup.[2]

1. Input to the script could be artist name or directly their URL in darklyrics. If the input is artist name, the full URL to their profile page would be automatically interpreted. The structure of the artist profile page URL is: www.darklyrics.com/the_first_letter_of_artist_name/full_artsit_name.html.
2. By searching for the pattern #1, which refers to the first song of certain album, in the profile page html, the list of albums is then gotten and the pattern containing album name is interpreted into album URL pattern.

3. On entering album page with album URL pattern, it is observed that the tag containing lyrics is in <h2>. Then lyrics are scraped and some irrelevant information such as copyright claim is deleted.
4. The whole cycle for one band is finished after all albums are scraped and all lyrics are output to a single .txt file.

```
<div class="album">
<h2>EP: <strong>"Lifesblood"</strong> (2001)</h2>
<a href="../lyrics/mastodon/lifesblood.html#1">Shadows That Move</a><br />
<a href="../lyrics/mastodon/lifesblood.html#2">Welcoming War</a><br />
<a href="../lyrics/mastodon/lifesblood.html#3">We Built This Come Death</a><br />
<a href="../lyrics/mastodon/lifesblood.html#4">Hail To Fire</a><br />
<a href="../lyrics/mastodon/lifesblood.html#5">Battle At Sea</a><br />
```

Figure 3. Step 2 of lyrics collection

```
<div class="albumlyrics">
<h2>EP: "Lifesblood" (2001)</h2>
<a href="#1">1. Shadows That Move</a><br />
<a href="#2">2. Welcoming War</a><br />
<a href="#3">3. We Built This Come Death</a><br />
<a href="#4">4. Hail To Fire</a><br />
<a href="#5">5. Battle At Sea</a><br />

<br /></div>
<script language="javascript" type="text/javascript" src="../../rect.js"></script>
<div class="lyrics">
<h3><a name="1">1. Shadows That Move</a></h3><br />
the us rhythm blows a diseased river flow<br />
yeah it's going to blow a change within you<br />
the us rhythm blows<br />
with change we'll explode<br />
and live<br />
<br /><br />
<h3><a name="2">2. Welcoming War</a></h3><br />
lost in a world<br />
a guided eye to strengthen weakness<br />
throat in noose pain<br />
feel it constricting a lost love<br />
welcome to war<br />
and it grows<br />
ocean lure<br />
fire of mankind<br />
universe<br />
throw it away<br />
destroyer of worlds found pain<br />
welcome to war <br />
and it grows<br />
<br /><br />
```

Figure 4. Step 3 of lyrics collection

2.3 Audio clip collection

Clips are partially downloaded or bought from music streaming services such as QQ music to avoid copyright infringement. For those songs that cannot be found in similar services, we work-around by using YouTube to mp3 convertors found online.[3-5] We try our best to find playlists from which the video clips are in Creative Commons rather than under Standard YouTube License. After some grueling search and change in the choice of bands, we manage to assemble enough audio data for 100 bands, each having more than 30 songs.

3. Text Feature Extraction

3.1 Overview

In this part, we converted lyrics of bands to vector format using TF-IDF. The basic scenario contains two steps: preprocessing, and vectorization. Prior to feed the text into TF-IDF library provided by sckit-learn, we performed some basic text preprocessing to clean the lyrics we crawled from the internet which includes basically tokenization, lemmatization and redundant message cleansing, all of which are implemented through either NLTK or Regular Expression library.[6-8] In the subsequent paragraphs, details of each steps in preprocessing will be explained.

3.2 Preprocessing

Tokenization is a process of demarcating a text string into a set of words and characters. Since lyrics crawled from the Internet contains a lot of weird symbols and since lyrics, especially metal song lyrics, uses a lot of causal words, using normal tokenizer may under some circumstances generate mistakes. Therefore, here we choose TweetTokenizer which is good at tokenize causal texts.

The lemmatization part is responsible for returning the words into its common based form. It is used in the project to convert same word in different conjugations/persons to the same format, reducing the corpus size, making the whole input data for TF-IDF less noisy. Unlike stemming which converts words into a stem form by removing their derivational affixes, lemmatization just return the word into its dictionary form. Thus, compared to stemming, lemmatization avoids the mistake that two different words are converted into the same stem after processing, and largely retains the properties of different words. Table 1 shows the performance of different stemming and lemmatization methods.

Also regular expression is used to filter out unnecessary information in the lyrics including urls, symbols, non-English characters and numbers. All the characters are converted to their lowercase. Besides, stopwords are filtered before feeding into TF-IDF vectorizer.

Word	**PorterStemmer**	**LancasterStemmer**	**SnowballStemmer**	**Wordnet_lemmatizer**
really	realli	real	realli	really
computer	comput	comput	comput	computer

computing	comput	comput	comput	computing
student	student	stud	student	student
playing	play	play	play	playing
playful	play	play	play	playful

Table 1: Stemming and Lemmatization

3.3 Vectorization

The next step is to vectorize the lyrics. As mentioned previously, this step is simply implemented through sckit-learn library. The max_df parameter equals to 0.5, meaning that the function will ignore terms with document frequency higher than 0.5. The min_df parameter equals to 2, which means that a term will be ignored if it appears less than 2 documents. The TF-IDF is trained in 2 ways. The first is to view all the song lyrics of one band as a document, and directly train the bands TF-IDF vector. The second is to view one song as one document and train the song vector first. Band vectors are calculated by getting the mean of all the vectors of its songs. Compared to the second one, the first method is definitely robust and easy to implement, as the song lyrics are crawled by band and songs of one band are simply stored in one document. However, if we calculate the TF-IDF in this way, it would be hard for us to get the user vector in the later phase. Thus we finally choose the second method.

Nevertheless, one huge drawback of TF-IDF vector for songs is that it could not give accurate document representations when the length and vocabulary size in different documents vary a lot. In the lyrics data, the length of lyrics varies a lot, which means that long lyrics would have more unique words whereas short ones only have a very limited vocabulary size. This will further lead to the result that most positions in a short TF-IDF vector is 0. When recommending songs to users, if a user song list happens to be short songs, the system will just recommend him/her bands whose songs are mostly short, regardless what genre they are. Thus, a more robust method comes out to alleviate this problem.

We then extract top 50 terms from each band using term frequency in TF-IDF model and get synonyms of these words using NLTK library. We then intersect the word lists of different bands to get the final corpus. Word with document frequency higher than 50 is ignored as the purpose is to get the more unique words for a document. Using this method, we can just evaluate the similarity of two bands by comparing how similar their word lists are.

3.4 Limitations and Future Work

There exists certain limitations and future works in this part. As is shown above, TF-IDF fails to capture the characteristics of each band. Also, it ignores the sequence of words and only concerns about word frequencies within the document and in the whole corpus. Moreover, when performing the evaluation, we find that even the robust method we propose later performs better than the TF-IDF. Here we draw a conclusion that TF-IDF is really bad for a set of documents whose size varies a lot. And rather than the robust way, another way to solve this problem is to use Word2Vec model. It is a 2-layer neural network which could give each word its vector representation based on its semantic meaning.[9] However, one problem is that a Word2Vec model usually requires a large number of training data but the lyrics data size is limited. An alternative solution is to use the pre-trained Word2Vec model provided by google and map the words in lyrics to its corresponding vectors. This step is not performed in our project since I am tried to perform it again and again and again…

4. Audio feature extraction

A Python library called pyAudioAnalysis [6] is majorly used in this step. No musicology knowledge is compulsively needed to carry out the extraction.

The functionality of the library is to sample the whole audio signal on one song into frames of a very short period, for example 0.05s per frame, and calculate the features from each frame. These features are called short-term features (ST) which represent local information. As the duration of each song is different, one mid-term feature (MT) per song is calculated as a global representation, by computing the mean and variation from all STs [6].

But for our goal, we need to combine the song features into a band feature, because we want to recommend a band instead of a song. Thus, a band vector is created by concatenating each song vector of the band, so it will be a 2D array in the end. As the number of songs a band has is different, principal component analysis (PCA) is used to reduce the band vectors to a unified dimensionality. Also, it can be interpreted that we choose the most musically representative songs of the band.

During implementation, there are in total 34 audio features extracted for each ST. Thus, the MT of a song dimension is 68, because for each of the 34 values, mean and variation are computed and the dimension doubles. For one band, we choose the number of components of PCA to be 10, so the final band vector dimension is (10, 68). The output file contains an array of (100, 10, 68) dimension.

For reference, the description of the 34 audio features could be found at.[9] I have done a bit understanding by myself. For example, feature number 22-33 chroma vectors refer to

the 12 different pitches in an octave scale. The vectors represent how many times in a certain frame each pitch appears. Similarly, other features might focus on difference, similarity or accumulation on volume, frequency, timbre and so on. For deeper understanding, you can also refer to.[10]

5. Experimental Evaluation

5.1 Overview

This section represents the experimental evaluation of our band recommendation system. We investigate the suggestions quality recommended by the system via a user playlist simulation framework.

Hardware environment. Our system and web crawling scripts are implemented in python. All the experiments are performed on a machine with a 2.70GHz Intel Core i7-7500u CPU and 16GB RAM.

Default settings. The default keyword set size is set to be 50. For band recommendation, 10 most similar bands are returned by our system each time. For parameters in the proposed ranking function, beta is set to be 0.7. For each parameter setting, in total 100 simulated playlists are used to evaluate our system.

Simulating the user interaction with our band recommendation system is relatively a hard task. We first highlight two main difficulties of evaluating our system.

Issue-1: How to accurately categorize songs according to their style? In order to evaluate the effectiveness of our proposed system, we need to first categorize all candidate band recommendations, then check if the style of the recommended band matches the style of the user's playlist or not. Unfortunately, songs under XX category is relatively harder to categorize when compared to other styles.

Issue-2: How to effectively get the user playlist from the internet? The number of available playlists is limited due to privacy issues. Besides, as we have mentioned, our system crawled XX bands from the internet to perform audio feature extraction and keyword extraction, which means that our system is only capable of handling users' playlists with only XX band songs at this stage. Based on these two limitations, it is hard for us to find suitable user playlist to perform the simulation.

Regarding the aforementioned issues, our proposed solution is to use the existing set of bands and songs that we crawled to generate a set of simulated user playlist. We first adopt the categorization of all the XX bands according to an XX music expert Miss Kelly Suen. Then, we randomly choose one band style to be the style that the virtual user loves. After that,

we randomly pick five bands from that style and select six songs from each band to form a simulated playlist with 30 songs. For each simulated user playlist, we use the recommendation system to generate a set of band recommendations. After getting the recommendations, we compare the band style and the simulated user playlist style. If it is with the same style, it indicates that our system successfully returns one useful suggestion. In the followings, we discuss the performance of our band recommendation system under the simulation framework.

5.2 Effectiveness qualities via simulation

Table 1 to Table 3 (shown in Appendix) presents the data for the effectiveness results. For quality metrics, we adopted 1 Hit, i.e. the successful time that our band recommendation system at least returns one recommendation with correct style. #useful shows the average number of correct recommendations at each iteration.

Varying □. In our system, □ is used to control the similarity preference between textual similarity and audio similarity. From Table 1, one can observe that the effectiveness of our system increases as □ increase, and is stable when □ > 0.6.

Varying *k*. Table 2 shows the effectiveness of our band recommendation system under different *k* values. In here, we did not consider $k > 10$, since *k* is usually bounded by the graphic user interface of the system, and users may not spend much time to examine each band recommendation.

Varying simulated playlist size. Keen readers may notice that the simulated playlist size will directly affect the effectiveness of the band recommendation system. That is, larger playlist can help us to better understand the style and the taste of a particular user. However, long playlist will increase the response time for the recommendation system. Therefore, we aim to find out how the system performs under varying playlist size to strike a balance between effectiveness and efficiency. Table 3 shows the effectiveness of our proposed system under varying simulated playlist size. It shows that the 1 Hit (%) reaches maximum when playlist size equals to 30, and remains stable when playlist size > 30. As such, for real users, we may choose a subset of 30 songs from the whole user playlist to generate recommendation.

5.3 Efficiency qualities via simulation

Next, Figure x and Figure x shows the efficiency measurements of our system under various settings. For quality metric, we measures the average response time, that is, the time it takes for our band recommendation system to return a set of band recommendations.

Appendix

Table 1 Quality metrics verse β

β	1 Hit (%)	#useful
0.0	45	1.03
0.2	52	1.04
0.4	59	1.04
0.6	70	1.06
0.8	71	1.05
1.0	69	1.04

Table 2 Quality metrics verse k

k	1 Hit (%)	#useful
2	21	1.02
4	37	1.03
6	51	1.03
8	64	1.04
10	72	1.06

Table 3 Quality metrics verse the # of songs in simulated playlist

# of songs	1 Hit (%)	#useful
10	59	1.03
20	66	1.04
30	71	1.05
40	69	1.04
50	69	1.04

Reference:

1. Welcome to the DARK LYRICS ! (n.d.). Retrieved from http://www.darklyrics.com/
2. Beautiful Soup Documentation¶. (n.d.). Retrieved from https://www.crummy.com/software/BeautifulSoup/bs4/doc/
3. QQ Music. (n.d.). Retrieved from https://y.qq.com/
4. YouTube (n.d.). Retrieved from https://www.youtube.com/
5. Guljaš, J., & Petković, M. M. (n.d.). YouTube to MP3 Converter - The Best Online. Retrieved from https://www.ytbmp3.com/
6. "Natural Language Toolkit¶," *Natural Language Toolkit — NLTK 3.2.5 documentation*. [Online]. Available: http://www.nltk.org/. [Accessed: 13-Oct-2017].
7. Regular expression operations¶. (n.d.). Retrieved from https://docs.python.org/2/library/re.html
8. Sklearn.feature_extraction.text.TfidfVectorizer¶. (n.d.). Retrieved from http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
9. Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (pp. 3111-3119).
10. T. (n.d.). Tyiannak/pyAudioAnalysis. Retrieved from https://github.com/tyiannak/pyAudioAnalysis/wiki/3.-Feature-Extraction
11. Computer research in music and acoustics (CCRMA). (n.d.). Retrieved from https://music.stanford.edu/programcenter-affiliation/computer-research-music-and-acoustics-ccrma