Building Music Recommender Systems with Graph-based Method

Chang Li
CSE Department
A53100229
chl585@ucsd.edu

Jiaxiang Chen
CSE Department
A53100265
jic286@ucsd.edu

Yangyue Wan
CSE Department
A53106526
yaw109@ucsd.edu

***Abstract*—Today, most people enjoy music online and the music providers use personalized music recommender systems to recommend songs to users. In our project, we first collected 10 million users' "like" records from xiami.com, one of the biggest online music provider in China with our own crawler. After analyzing, we found the dataset has two important features: a majority of "like" records are for a little part of songs and songs are grouped into communities when considering the "like" they received from common users. Based on this observations, we tried to use three different ways: latent-factor model, item-rank and graph-based method to predict users' attitudes towards unseen songs and evaluated their performance on our own test set. The result shows that our methods have the best performance.**

Keywords*—*online music, recommender systems, graph-based.

I. Introduction

Nowadays, enjoying online music service has gradually become an important part of peoples lives. Facing the huge number of available songs (more than 20 million on Pandora), people usually rely on the music providers music recommender systems to find new songs. In such case, how to provide a highly personalized music recommender systems for each user and recommend exactly the kind of songs they like become a crucial problem.

As a specified recommender system, the music recommender system can benefit from many classic general recommending algorithms, like k-means clustering[1], k-nearest neighbor collaborative filtering[2] and SVD[3]. Though the general recommender systems has been researched for a long time and has many matured theories, there aren't many works for the music recommender systems. In[4], George Tzanetakis and Perry Cook have proposed a k-means clustering algorithm based on songs' audio features and in[5], Aggarwal and Wolf implemented a algorithm for graph theoretic approach. The methods in these works are still to naive and their research are all based on a very small dataset, which is definitely different from the real situation.

In this assignment, we collected around 10 million like records from one of the biggest online music provider(xiami.com) by our own crawler and tried three different methods to build the recommender system: latent-factor model, item-rank and graph-based methods. Our own dataset shows some important features which are different from the book review records in assignment one and the result shows that the graph-based model has a better recommender accuracy.

The remaining parts of this report are organized as this: Section 2 detailed describes how we collect these data and analyze some important features of this data set; Section 3 and 4 introduced how we applied the latent-factor model and item-Rank into this problem and Section 5 explained our graph-based method. How we constructed the test dataset and the performance evaluation are explained in the Section 6 and the last section is our conclusion.

II. Dataset analysis

In this section, we first describe how we collected our training data from the web with our own crawler, then we will use graphs to illustrate two important properties of this dataset: 80-20 rule and communities. At last, there will be a thorough analysis about how the feature of the dataset will affect the performance of our music recommender systems.

A. The data collection process

There are several biggest music providers in the world. In the north of America, most people use iTunes, Pandora or Spotify. If we could collect their data about people's like or dislike, it would be wonderful. However, after survey, we found these providers' user data were either invisible or hard to grab by crawler. At last, we decided to collect data from xiami.com, which has more than 80 million users and is one of the biggest online music provider in China. Each user in this website has an unique user ID and so does each song and each artist.

We implemented a simple Java web crawler to scan users personal music library (contains the songs they marked as liked and is open to everyone) and download these information. Each record in our dataset has three attributes: user ID, song ID and artist ID. We totally scanned around 1 million users. For many users are not active, we only download data from those who have liked more than 20 songs and finally collected 10 million pieces of like records from around 100000 users (to spped up the processing, we finally only use 6 million records of all). Unlike Pandora, xiami.com doesn't have the "dislike" buttion for user and so we cannot know which songs the user doesn't like.

B. The 80-20 Rule in this dataset

The 80-20 rule, also known as the Pareto Principle, states that, for many events, roughly 80% of the effects come from 20% of the causes. This rule is usually used to describe some economic phenomenons, but is also correct when we study the distribution of the amount of ”like” that each song received.

Specifically, We studied 6 million records in total and 780 thousand songs are included. In average, each song is expected to receive around 8 likes. However, after analysis we found that a few of them received a huge number of likes the majority of receive very few likes. For example, half of all songs (382 thousand songs) only received one like! There are 612 thousand songs (78% of all) received less than 5 likes and received 1 million likes (17% of all) in total. The other 22% songs just received all other 83% likes. This phenomenon can be expressed in figure 1: only 20% songs received the 80% likes.

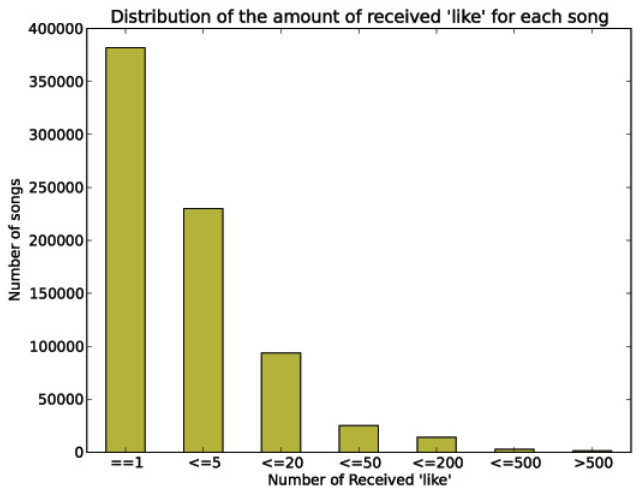

Fig. 1: Distribution of the amount of received ’like’

C. The communities in the dataset

When we talk about music, we all know that there are many different music styles just like country, rock, blues, jazz, etc. Generally speaking, different people may have different tastes and may like one or several kinds of music of all. So, if we represent each song as one node in a network and add one edge between these nodes if two songs are liked by the same person, its straightforward to think that there will be some communities in this network and each community may represent a certain kind of song taste or style. If people like one song in this community, its high probably that he will like the other songs in this community. To verify this idea, we constructed a network with a few songs in the dataset and the weight of each link means the amount of common users between the two songs. Finally we get the following figures:

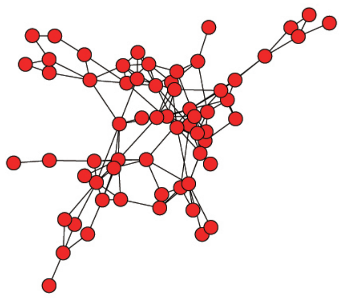

Fig. 2: The communities in very popular songs

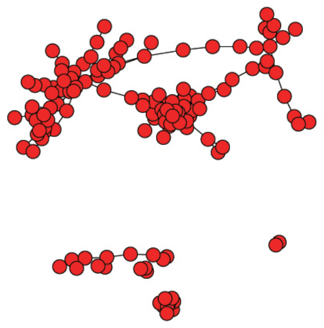

Fig. 3: The communities in normal popular songs

Figure 3 represents the network of a few songs randomly selected from the song set and they all received more than 500 likes. Actually, due to the large number of likes, nearly every two songs in the set has common links. To make the figure clear and easy to understand, we only keep these edges with weight more than 100 (both liked by more than 100 users). We can seen that there does exists several communities, but not very clear. In figure 4, we we chose the songs with likes number between 50 and 52. This time we get a much clear representation of networks, which confirms our expectation of communities in song sets. The reason that these less popular songs have a more clear structure of communities may be explained like this: for those really popular songs, they may gather some common features that people all like thus there doesn’t exist obvious groups. But for those less popular songs, they may only be liked by some people with certain taste, therefore it is easier to find the communities.

D. How would these features affect our recommender system

This music recommender system can be solved as a classic Collaborative Filtering (CF) problem. In general, there are two kinds of ways to do this: Memory-based and Model based.

The memory-based ways predict users’ attitudes towards new items according to other similar users’ attitudes. The advantages of memory-based is that it does not need to train data and therefore is simple to realize, and it has a strong scalability. The core issues of memory-based method is the amount of data and the definition of similarity. Obviously, the more records we have, the more similar users or songs we could find, then we can get a better prediction. As for the similarity, we can use cosin similarity, Jaccard similarity or path-based similarity in graph-based methods.

The model-based methods try to build a model to directly reflect the relationship between the songs features and peoples attitudes. Latent factor model is one representation.for this methods. The advantages for model-based method is that it can have a better performance when the data is really sparse and its prediction process is really fast, for what you need to do is just calculate the output of a certain function. The disadvantages of model based methods are that it is really expensive to train and dont have a good scalability.

As for our problem, it has following important features: (i) majority records are related to a small percent of songs: according to the observed 80/20 rules, most likes are for the top 20% popular songs. Because peoples time is limited and will only listen to a quite small number of songs every day, these 20% songs have already consisted a big enough data set to recommend songs. So, to reduce the space and computing complexity, we removed around 50% less popular songs from the dataset; (ii) our data is not complete: because of the design of xiami.com, we don't know which songs users don't like. This means, if people listened to 10000 songs and likes only 100 songs of them, we can only see that 100 songs. What's more, the rating for the user is 0-1 based rather than a 1-5 start based, which further blur the users' attitudes. We think this incompleteness will bring a big accuracy issue in model based methods; (iii) these are communities in the songs: based on our study, we found that many songs are grouped into communities and if people like one songs in the communities, they are very likely also like the other songs in the communities. In this case, the memory based algorithm would have good performance, for it directly utilize these relationships to make predictions.

Based on these ideas, we tried three different methods to build the recommender system: latent factor model, item rank and path number model. The details are described in the following section.

III. Latent factor model

In this section, we use the latent-factor model to train the data, just like what we did in Assignment 1. We fit a predictor of this form

$$\text{like}(\text{user}, \text{song}) \approx \alpha + \beta_{\text{user}} + \beta_{\text{song}}$$

The result will be rounded off to either 0 or 1, which represents the user likes or doesn't like the song. We use the data collected from over 1 million users to train the predictor, and for the songs that never appeared in the train data, we just think the *like* value is 0 no matter for which user. As you can see, this predictor is relatively quite simple and naive. And by using this naive predictor, we get the following prediction result

true positive	false positive
96.3%	81%
true negative	false negative
19%	3.7%

From the result, we can easily tell that the performance of the predictor is not very good, it basically thinks that most of the songs will be liked by the user. Although it does a good job on predicting the *true like* songs, it marks too many songs as *like*. Therefore this predictor actually will recommend many songs that the user doesn't like.

IV. Item Rank

In this section, we implement an algorithm called Item-Rank[6] to give a rank for each song with respect to every single user. The higher the rank, the more possible that the user will like this song.

This algorithm is similar to PageRank algorithm. First we will define a n×n matrix called $\mathcal{C}$, where n is the amount of total songs. For song$_i$ and song$_j$, $\mathcal{N}_{ij}$ is the number of users who love song$_i$ and also song$_j$. And

$$\mathcal{C}_{ij} = \frac{\mathcal{N}_{ij}}{\sum_i^n \mathcal{N}_{ij}}$$

When computing the rank, we use a ItemRank value $\boldsymbol{SR_{u_i}}$ for every user, this is a vector contains the attitude of a user towards all the songs. The equation is expressed as

$$\boldsymbol{SR_{u_i}} = \alpha \cdot \mathcal{C} \cdot \boldsymbol{SR_{u_i}} + (1 - \alpha) \cdot \boldsymbol{d_{u_i}}$$

Where $\boldsymbol{d_{u_i}}$ is a vector built by the *like* value, which means for every song, if the user likes this song, then the pair equals 1 in the vector, and otherwise equals 1. And α is 0.85, this is a common choice. By using this equation, we can compute all the *like* value for every user with respect to every song by iteration, and it only needs about 20 iterations to converge.

The algorithm is designed to find both propagation and attenuation between songs and users. In other words, if a liked song from a user is connected with another song, and $\mathcal{C}_i j$ is large, this means the user would possibly like the other song too.Actually, the effect is quite satisfactory when using this algorithm on a small set of data. However when trying to use this algorithm to the train data with over 1 million users and 10 million songs, the computer cannot handle such a computation, just because the matrix needed for computing is too large. So this model is not suitable for this problem too.

V. Graph-based Algorithm

In this section, we tried to used a graph model to represent our data and build recommender systems based on that model. In our graph, every song and every uses is a node, and edges between nodes means a song is liked by a user, so for a user node, the degree of it is the number of songs he liked, and for a song node, its degree means this song is liked by how many users. It is obvious that our graph is a bipartite graph.

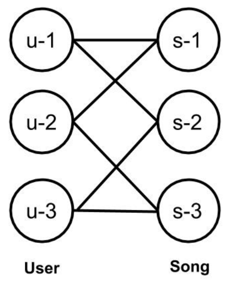

Fig. 4: Graph model for users and songs

A. Path number based model

The idea behind this method is quite simple, which state that the possibility of whether a user would like a song is closely related to the distance and the number of path between the user and the song in the graph. The shorter path and the more paths we can find, the more likely that the user will like that song. The assumption behind this method is similar to the k-nearest-neighbors methods' idea, which believe that user may like these things which are liked by similar users. But with the graph model, we can utilize more informations in the dataset, for those not directly connected nodes can also make contribution.

When we try to implement this model, we have met two important problems. The first one is that due to the huge data size, the graph would be really really big and memory consuming. Therefore, no matter the construction process or any path searching would very slow. To deal with this, we further shrinked our dataset by remove users and songs only related small size of records. After test, this only has a very small effect on the prediction result. The second one is that we found the relationship between users and songs are similar to the relationship between people in a social network, for example it satisfies the small-world phenomenon: we can cover nearly all songs from any user within 5 steps in the graph. This makes the path length difference much less useful, for most songs have the same shortest path length. In this case, we decided to only consider the songs we can reach in 3 steps and use the number of path as the major parameter for prediction or recommendation. For those songs we can not reach in 3 steps, we just think the connection between the user and the song are too weak and impossible to lead to a "like'.

In this method, if we want to generate a list of songs that the user may like, namely make a recommendation, we can first get the number of paths for all songs we can reach in three steps and sort these songs according to the number of path in descending order. Then we can just recommend the first several songs to users. If we want to predict whether a user would like a song, then this becomes a logistic regression problem. Using validation set, we can find a best value k, such that for every pairs with more than k paths, we predict that the user will like this song. If not, we just do the opposed prediction.

B. Points to improve the prediction performance

1) Use a more complex graph model: In our current graph model, we only use the information of users and songs, but ignore the artists. So, a good idea is that we can add artist into the graph and get a graph like below:

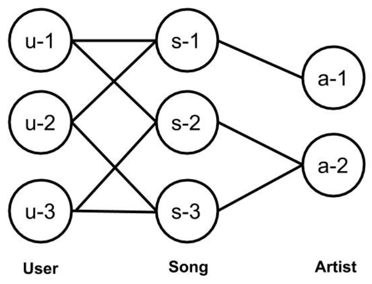

Fig. 5: Graph model with user, song and artist

What's more, we can introduce different weight for different edges. A straightforward way is that we give small weight for those very popular songs and bigger weight for those less popular ones. Because according to our analysis in section 2, we found that because very popular songs are liked by everyone, so they are not grouped into communities and may not yield so much information for the users taste.

2) User Path-number/Song number to do prediction: One obvious defect of the naive algorithm is that different user has a different number of already liked songs, which means their out degree in the graph may vary a lot. So, using a unified boundary for all users seem to be unreasonable. To improve this, we can divide the path number by the number songs that the user have already liked and get the path/user rate and this can more accurately reflect the relative connections between users and songs.

3) Add more features for the logistic regression: The path number of the path/user rate are only one valid features. In our dataset, we can utilized much more features actually, like the number of total likes one songs received, the total number of songs that the person liked. We can get a feature vector in this way and use gradient descent to get the best parameters for prediction.

C. Another solution based on cosine similarity

This solution is based on the idea that people intend to like the music that is liked by those whose preference is similar to him/her. The first step is same as before, we find all users with at least one same liked song. Then for these users, we calculate their cosine similarity with the original user, and sort users in descend order of cosine similarity.

After analyzing the trend of cosine similarity, we found that cosine similarity descend quickly after the first a few users, so it can be considered that among all users who share same liked songs, only a few have a high similarity with the original user, since the graph grow very quickly when we consider all users with same liked songs, the less similar users preference is surely less similar to the original user, so we should emphasis on these users with high similarity.

We use the value of the cosine similarity as a kind of weight for a user, then accumulate songs' prediction value, for example, for a user with a cosine similarity of 0.8, we add all songs' prediction value in his liked list by the value of his cosine similarity. Apply this process to all selected users, and we will get a list of songs with accumulated weighted value, and then sort the result in descent order, output the prediction result.

The key to the accuracy of this prediction model is how many or in other words, what value of the ratio of the users are we going to select, there is a trade off on the ratio, so further examination on it is also required.

VI. Performance Evaluation

In this section, we evaluate the performance of these three methods on users recent playing history. We first introduced how we built the test set and then show the performance and give a detailed analysis.

A. Construct the test set

There are generally two ways to evaluate the the performance of recommender systems: offline and online. If we can directly access the user, we can use an online method, namely, give different user different recommendation and observe their feedback. But for us, we can only use the offline method. But even an offline test set is hard to get in our project. Because we don't know what songs users are explicitly dislike, we can only regard the songs that users did listen to but didnt mark as "like" as "dislike". But this is obviously not perfect: First: the songs that users listened are usually recommended by the music provider's system, so these songs are all already has a high probability that the user will like. This will make the predict much difficult. Second, it is almost sure that people like these songs that they marked as "like", but its not sure that they don't like the others. Whether people would marked one song as "like" may depend on many factors, like the time, the user's mood at that time, if the user are doing something else.

The test set contains the recent playing history collected from 248 active users and has around 60 thousand records. For each record, if it is has already been marked as "like", then we mark it as "like" and delete the "like" record from the train data set; if not, we directly marked it as "not like".

B. Performance Evaluation for Prediction Task

We decide to search a limited part of our ranked result, and then for every song, if it is not in test data, that means it is a song not yet be recommended by xiami.com, therefore is irrelevant in our test data, and we can simply ignore these songs. Then we measure the ratio of songs marked as like and disliked in test data, as removing irrelevant songs won't change the ratio of these two. Also the total amount of songs to be recommended is also relevant in this task since a fewer number of songs usually means a better result but losing many songs we want to predict.

For the prediction task, we first split the test set above evenly into training set and test set. We use the train set to get the best parameter for our prediction model. As the fore the measurement, we use the classic precision-recall and F1 score to measure the performance.

$$precision = \frac{tp}{tp + fp}$$

$$recall = \frac{tp}{tp + fn}$$

$$F1\ score = 2\frac{precision * recall}{precision + recall}$$

We tested the following four methods' performance: (1)simply based on the number of likes that song received; (2)based on the number of paths between the song and the users; (3)based on the rate of path-number and song liked; (4)logistic regression based on the following features:

$$[1, number_{paths}, like\,received_{song}, like\,received_{song's\ artist}]$$

and all of them has been normalized with this function, where k is the mean value for each feature:

$$f(num) = 1 - e^{\frac{-num}{k}}$$

The result is listed in the following table:

Method	Precision	Recall	F1 score
Song's "Like"	0.1983	0.8875	0.3242
Path Number	0.2188	0.7875	0.3423
Path Number Rate	0.2066	0.9375	0.3386
Logistic Regression	0.2377	0.8081	0.3674

From the table we can see, using the number of path and doing logistic regression with more features can improve the performance. But Use the rate seems to decrease the result somehow. This may because the relationship between the path numbers and the song already liked are a more complex than we expected and the simple division can not reflect it. But no matter in which method, the F1 score are all very small. As stated above, the test set we collected from the web may not be a good test set for 0-1 prediction. If people click "like" for one song, he certainly likes that song, but if he don't not click it, it may just because he forgot it. What's more, in most cases, these songs people listened online are already selected by the music provider, so the difference between them are already very small.

C. Resutlt for cosine similarity model

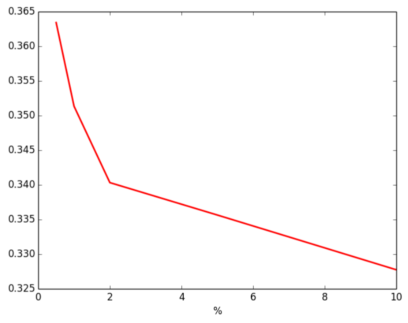

Fig. 6: cosine model with user, song

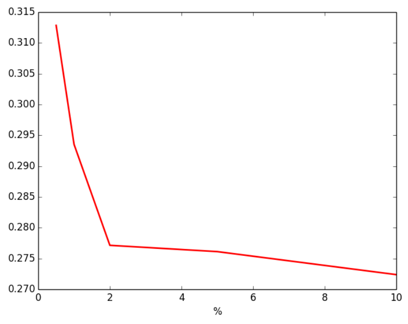

Fig. 7: cosine model with user, song and artist

For graph above X axis stands for the percent of the users we choose, and Y axis stands for the accuracy(ration of songs we found that are in test data and with tag 'like'=1). We decide to search a limited part of our ranked result, and then for every song, if it is not in test data, that means it is a song not yet be recommended by Xiami, therefore is irrelevant in our test data, and we can simply ignore these songs. Then we measure the ratio of songs marked as like and disliked in test data, as removing irrelevant songs wont change the ratio of these two. Also the total amount of songs to be recommended is also relevant in this task since a fewer number of songs usually means a bette result but losing many songs we want to predict.

From data above we can see that the ratio of like/dislike goes down as we choose to pick more users as similar user to vote for songs, but the total songs being picked goes up. And when we added artists information to the graph, and consider artists as users, then the overall accuracy will slightly changes while in both cases the trend of accuracy remains the same.

For the test data itself, the ratio is roughly 0.3, consider the truth that most users uses the recommendation system of Xiami, this can be partly considered as the accuracy of a baseline. But a major reason for the truth that this accuracy is lower than ours is that we only recommended a part of songs that are recommended by Xiami, so the accuracy is based on cost of losing completeness.

VII. Conclusion

We did four things in our project. Firstly, because we are really interested in the music recommender system but there is no available data set, we wrote crawler and select the data we want use and then build our own music recommendation dataset. Secondly, we did a deep analysis on this data set. We found that most records are about only a few songs so we use this knowledge to reduce the dataset size and facilitate the processing; we found the songs are grouped into communities, so we think recommender systems based on neighbors and similarity would be good. Thirdly, we built the recommender systems and test it. We tried different methods to do that and do some optimizations on that. Finally, we construct the test set and evaluated the our systems' performance. But because of the limitation of the data we collected, the performance isn't so good, we did found that our methods made improvement.

References

[1] K. Alsabti, S. Ranka, and V. Singh, "An Efficient k-means Clustering Algorithm," Proc. First Workshop High Performance Data Mining, Mar. 1998.

[2] J. S. Breese, D. Heckerman, and C. Kadie. Empirical analysis of Predictive Algorithms for Collaborative Filtering. In Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence, 1998. 43-52.

[3] Berry, M. W., Dumais, S. T., and O'Brian, G. W. (1995). Using Linear Algebra for Intelligent Information Retrieval. SIAM Review, 37(4).

[4] Musical Genre Classification of Audio Signals, George Tzanetakis and Perry Cook, IEEE Transactions on Speech and Audio Processing.

[5] AGGARWAL, C. C., WOLF, J. L., WU, K.-L., AND YU, P. S. 1999. Horting hatches an egg: A new graph theoretic approach to collaborative filtering.

[6] M. Brand. A random walks perspective on maximizing satisfaction and profit. In 2005 SIAM International Conference on Data Mining, 2005.