Music Recommender System
Utilizing Users' Listening History and Social Network Information

CS229 Final Report

Tianye Lu
tianye@stanford.edu

Jing Xiong
Jxiong1@stanford.edu

Xiaoye Liu
xiaoye@stanford.edu

Abstract

A music recommender system based on users' listening history and social network was implemented in our project. We used collaborative filtering with both user-based and item-based strategies. For user-based collaborative filtering, we measured users' similarity with both the binary information and actual play count in their listening history. Our methods significantly increased the accuracy of recommendation. Furthermore, we modified the user-based collaborative filtering algorithm and came up with a method that combined the users' listening history and social relationships for music recommendation.

Introduction

Rapid developing of mobile devices with internet capability makes it possible for us to access different music resources freely. However, with millions of songs to choose from, people sometimes feel overwhelmed. Music service providers also need a more efficient way to manage songs and improve users' satisfaction.

Music recommender system is an information filtering system that builds on users' listening history to recommend users with new pieces of music that meet users' interest. Music service providers can utilize an effective music recommender system to maintain user group and attract new users. Users can also benefit from the recommender system with no more pain to make decisions on what to listen. Through studying users' listening history, we will be able to extract useful information and predict users' preference. Music service providers with social network information can also utilize friendship between users in recommending songs, because users might be interested in what their friends have listened to, and people with similar tastes tend to develop friendship.

In this report, the first section introduces the problem and the motivation for solving this problem. Background information including data sources is given in the second section. The "Methodology" section and "Results" section present the approaches and results of training the large scale data of users' listening history we used in our project. The "Discussion" section discusses the limitation of our project and further development possibilities; the last part concludes the project and summarizes the results.

Background

The Million Song Data Challenge was an open source data processing challenge concluded this August on Kaggle. The contest is to predict half of the listening history of 110,000 users by training the other half of the listening history and the full listening history of the other one million users. The data we worked on were the Million Song Dataset used in this contest and were downloaded from the Kaggle website. It is a freely available collection of audio features and metadata for a million of contemporary popular music tracks; each training data triplet contains user, song, and play count information [6].

Methodology

Simple Recommending Method -- Popularity

We created a baseline for later algorithm comparison through recommending songs with highest popularity without considering the listening history of users from the test set. For each song, we looked into the training set and calculated the number of users who had listened to the song. A higher number indicated a higher popularity. Sorting these numbers with a descending order, we obtained a list of songs with descending popularity; and the test of users were recommended with the most popular songs.

Collaborative Filtering

Collaborative filtering is a widely used method for recommendation systems [4]. The basic memory-based collaborative filtering algorithm is quite intuitive, relatively easy to implement and often has a good performance. The idea of collaborative filtering is that users who consumed similar items tend to have similar interests and will probably continue to consume similar items – in our case, listen to the same songs. Alternatively, items that are often consumed by the same user tend to appear in the future shopping list as well.

From these two perspectives, we can have both user-based and item-based strategies for collaborative filtering. For either strategy, we modeled the similarity between two users and two songs, and then assigned a score to every song that the user hasn't listened to accordingly.

Let u, v denote two users and i, j denote two items. $I(u)$ is the set of songs that u had listened to; $U(i)$ is the set of users who have listened to i. w_{uv} and w_{ij} are measures of similarity. We used a modification of cosine similarity, introducing a parameter α,

$$w_{uv} = \frac{|I(u) \cap I(v)|}{|I(u)|^{\alpha}|I(v)|^{1-\alpha}} \quad (1)$$

$$w_{ij} = \frac{|U(i) \cap V(j)|}{|U(i)|^{\alpha}|V(j)|^{1-\alpha}} \text{ [2].} \quad (2)$$

Then for a new user u and song i, the user-based scoring function is

$$h_{ui}^{U} = \sum_{v \in U(i)} f(w_{uv}) \text{ [2].} \quad (3)$$

The item-based scoring function is

$$h_{ui}^{I} = \sum_{j \in I(u)} f(w_{ij}) \text{ [2].} \quad (4)$$

Function f gives locality of different similarities. Specifically, we used exponential family of function

$$f(w) = w^{q} \text{[2].} \quad (5)$$

When q is large, users or songs with higher similarity contribute more to the final score, vice versa. In our implementation, we adjusted the two parameters α and q to see their impact on the results. Figure 1 gave interpretation of similarity in user-based and item-based strategies.

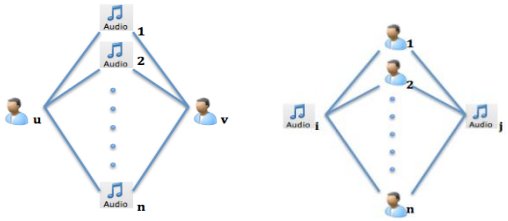

Figure 1. Similarity in user-based and item-based strategies.

Measure similarity with play counts

In the above approaches, we did not use play counts in processing our data. $I(u)$ was a vector in $\{0,1\}^{n}$, where n is the number of songs. With this notation, we can interpret the cosine similarity as

$$w_{uv} = \frac{I(u) \cdot I(v)}{(I(u) \cdot I(u))^{\alpha}(I(v) \cdot I(v))^{1-\alpha}}. \quad (6)$$

With this modification, we easily modified the user-based similarity function in the previous section, which was originally binary in nature, to include play counts information. Here, $I(u) \epsilon N^{n}$, where $I(u)[i]$ is the number of times that user u listened to song i.

Recommendation gets social

In this section, we suppose there was a social network with friend relationships built between the users. Our goal is to provide users with a comprehensive recommendation list that takes into account both the listening history and social network relationships of the user.

Figure 2 presents an idea of what an app will be like when adding this social component. We aim to provide users with a single list of songs as shown in the left part of Figure 2. The list consists of both songs recommended according to the user's listening history and songs that the user's friend listened to. In the "Filter Options", users will have an option of configuring their own parameters by dragging a scroll bar to decide how much social network component they would like to add to the recommendation list.

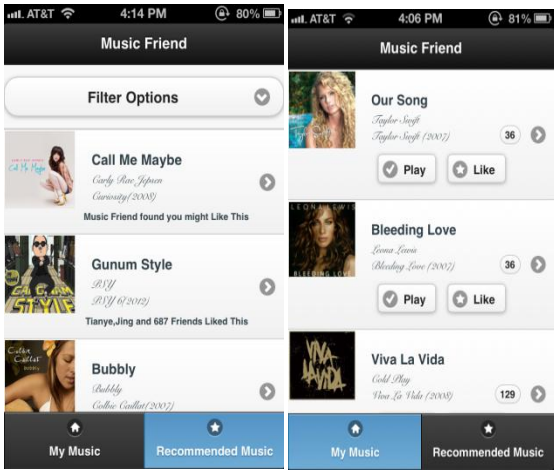

Figure 2. Demo of music recommender software with social network filtering option.

We can achieve this by modifying the user-based collaborative filtering method. When u and v are friends, we rescale w_{uv} to give more weight to their similarity. Specifically, the rescale function we used is

$$w'_{uv} = \begin{cases} k \cdot w_{uv} & if\ w_{uv} > 0 \\ \mathrm{k} & \text{otherwise} \end{cases}, \qquad (7)$$

where

$$\mathrm{k} = \mathrm{N} \cdot \log\frac{1+\beta}{1-\beta}\ . \qquad (8)$$

k is the rescale factor, which increases two friends' similarity by k times. This allows us to give more weight to all pairs of friends. Moreover, a good property of the rescaling scheme is that the relative similarity among friends remains unchanged after this modification. This property ensures that in our recommendation, the user's taste profile (modeled by similarity in listening history) and social network are both taken into account. One special case is that when two friends never listened to the same songs, we directly assign their similarity to be k.

The choice of k affects the percentage and ranking of the songs friends have listened to in the final recommendation list. We want to give users more control over k, but it is tedious to directly modify k. Therefore, we introduced a parameter $\beta\epsilon[0,1)$ and provided a nonlinear mapping from β to k. N is a constant which can be set before-hand in the system. We chose the function to be (8) so that k grows relatively steady with respect to β, and quickly grows to infinity when β approaches 1. When k passes a certain threshold, increasing k will not affect the result. Therefore, this mapping ensures that a user can control k within a reasonable range by changing β within the range of $[0,1)$. Figure 3 shows k as a function of β with N=10.

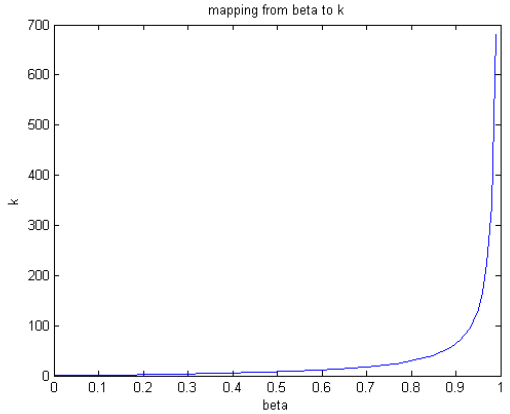

Figure 3. k as a function of β.

Evaluation

Most recommendation systems with explicit feedback from users such as rankings or ratings would choose root mean-squared error (RMSE) as the metric for evaluating recommender quality. Such kind of feedback is a bounded numeric score which directly reflects the affinity of a user for an item. However, what we are working with now is implicit feedback, which is generally unbounded. Moreover, its indication on the user's affinity for an item is not as obvious as explicit feedback. Therefore, it's hard to interpret the result if we predict a play count for every song and evaluate by RMSE. [3]

Instead, we used an evaluation metric called mean average precision (mAP). Specifically, we recommended the top τ predicted songs for each user. We only made binary prediction for the user, i.e. whether the user listened to the song. When we made a correct prediction on a song, it contributed to the average precision according to its rank in our recommendation list. Top recommendations were given more weight.

To formalize the definition, let $\mathrm{M} \in \{0,1\}^{\mathrm{m}\times\mathrm{n}}$ denote the user-song feedback matrix, where M(u, i)=1 if user u listened to song i. Let r denote a song that we recommend to a user. r(j)=i means that song i is ranked at position j in our recommendation list. For $\mathrm{k} \le \tau$, precision-at-k ($\mathrm{P_k}$) is defined as

$$\mathrm{P_k}(\mathrm{u},\mathrm{r}) = \frac{1}{\mathrm{k}}\sum_{\mathrm{j}=1}^{\mathrm{k}} \mathrm{M}(\mathrm{u},\mathrm{r}(\mathrm{j}))\ . \qquad (9)$$

For each user the average precision is defined as

$$\mathrm{AP}(\mathrm{u},\mathrm{r}) = \frac{1}{\mathrm{n_u}}\sum_{\mathrm{k}=1}^{\tau} \mathrm{P_k}(\mathrm{u},\mathrm{r}) \cdot \mathrm{M}(\mathrm{u},\mathrm{r}(\mathrm{k}))\ . \qquad (10)$$

$\mathrm{n_u}$ is whichever smaller between τ and the actual number of songs the user listened to. The average mean precision is the average of the average precision of all users,

$$\mathrm{mAP} = \frac{1}{\mathrm{m}}\sum_{\mathrm{u}=1}^{\mathrm{m}} \mathrm{AP}(\mathrm{u},\mathrm{r_u})\ , \qquad (11)$$

where $\mathrm{r_u}$ is the ranked recommendation list for user u.

For recommendation with social network, our goal has changed, and we can't use the same method to evaluate the recommendation. In fact, it is quite ambiguous to set a standard against how to evaluate our result. Instead, we got a concrete result

of our algorithm to demonstrate how social network elements affected our recommendation list.

Results

Collaborative Filtering

One big challenge in our implementation of collaborative filtering is how to handle such a big dataset with millions of users and songs. One way to speed up is to build a sparse matrix from the user-song-play triplets and directly operate on the matrices, instead of looping over millions of users and songs. However, due to limitation of computational power, we did not run the algorithm on the full data from the Million Song Dataset Challenge, which is one million users for training and 110K for testing. Instead, we used the information of 100K users for training and tested on 10K users. For each user, we made top 500 recommendations and calculated the mAP. We implemented the two different strategies of collaborative filtering with different settings of parameters.

We used python to implement the algorithm and the scipy package for constructing and handling sparse matrix [5]. The following tables summarize our results. Figure 4 and Figure 5 show how mAP is affected with varying α.

Table 1. Baseline mAP

Baseline - Recommendation by popularity	0.02256

Table 2. mAP of 500 Recommended Songs for Item-based Collaborate Filtering Algorithm.

q \ α	0	0.5
1	0.12303	0.12817
2	0.12889	0.10912
3	0.12472	0.09968
4	0.12177	0.09482
5	0.11998	0.09193

Table 3. mAP of 500 Recommended Songs for User-based Collaborate Filtering Algorithm.

q \ α	0	0.5
1	0.08655	0.08587
2	0.10171	0.10382
3	0.10741	0.11408
4	0.10436	0.11778
5	0.10112	0.11738

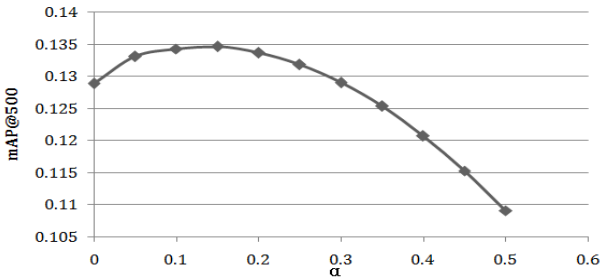

Figure 4. Item-based mAP with varying α.

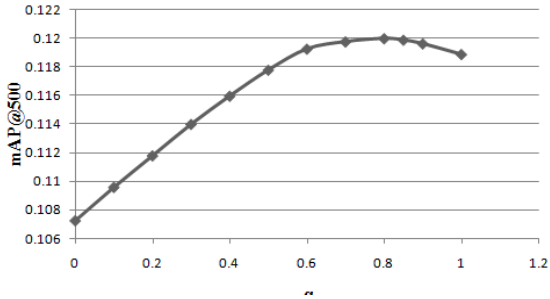

Figure 5. User-based mAP with varying α.

Adding Play Counts

By measuring similarity with play counts, we obtained results presented in the following table.

Table 4. mAP of 500 with play counts for user-based approach

q \ α	0	0.5
1	0.087798	0.086420
2	0.083760	0.088297

Collaborative Filtering in social network

To demonstrate the result when social network structure is included, we built a virtual friendship in our data and listed the top 10 recommendation for one particular user given by the system with different choice of β. When $\beta = 0$, the result is the original recommendation from the user-based collaborative filtering approach. As β increases, the songs the user's friends listened to ranked higher; more and more of these songs appeared in our recommendation list. When $\beta = 0.9$, all songs recommended were listened to by the user's friends. Table 5 illustrates how β changed our recommendation list to a certain user with a virtual friendship. Songs listened to by the user's friend are bolded.

Table 5. Recommendation list generated with varying β; songs listened to by friends are bolded.

beta	top ten songs recommended to user									
0	219472	250363	151010	**66627**	224602	367867	**27709**	**350475**	106132	**124578**
0.1	219472	250363	**66627**	151010	224602	367867	**27709**	**350475**	**124578**	**43217**
0.2	219472	**66627**	250363	**27709**	**350475**	151010	224602	367867	**124578**	**43217**
0.5	**66627**	**27709**	**350475**	219472	**124578**	**43217**	250363	151010	224602	367867
0.7	**66627**	**27709**	**350475**	**124578**	**43217**	219472	**210549**	250363	151010	224602
0.9	**66627**	**27709**	**350475**	**124578**	**43217**	**210549**	**92976**	**237366**	**57369**	**210782**

Discussion

The results show that both the basic user-based and item-based approaches of collaborative filtering significantly increased the recommendation accuracy. The settings of α and q, especially q, played an important role in the results. It shows that the locality of the scoring function, i.e. how much we emphasize users or items that are relatively more similar, is critical in the performance of the algorithm.

Besides, how we model similarity also affected the relative weight among different similarities. By changing parameter α, we actually changed how we model similarity. The user-based approach with play count similarity does not give a good result because the similarity model is highly biased to the few songs that are played multiple times.

Conclusion

In our study, our user-based and item-based music recommending algorithms significantly improved the recommending accuracies as compared to the baseline approach using solely popularity of songs. Both of the algorithms can be applied in music related services to provide users with more pre-selected options according to their listening history. We also combined the element of social networking into the user-based algorithm so that it can be easily applied to social-network based music recommending mobile apps or websites. Those features will help make contributions improving user experience and saving users time.

When computing similarity with play counts, we did not obtain as good result as we expected due to calculation noise generated by a few very popular songs. We also approached the content-based strategy through performing clustering for music pieces a user had listened to based on pair-wise Euclidean distance similarity computation. However, we did not finish this section due to technical limitation on large scale data computation and time constraints. Further work can be done in implementing content-based filtering.

Our work has implemented the main aspects of music recommendation system. We improved our project by adding innovative social component based on the idea that users might be interested in friends' listening habit and people with similar tastes tend to build friendship. Our work can be potentially utilized by social music service provider to increase user' satisfaction and boost their user group by attracting friends of existing users.

References

[1] T. Bertin et al., "The Million Song Dataset," Pro c. of the 12th International Society for Music Inform ation Retrieval Conference, 2011.

[2] F. Aioilli, "Preliminary Study on a Recommende r System for the Million Songs Dataset Challenge," ECAI12 Workshop on Preference Learning: Proble ms and Applications in AI, 2012.

[3] B. McFee et al., "The million song dataset challe nge," Proc. of the 4th International Workshop on Ad vances in Music Information Research, 2012.

[4]Su, X et al., "A survey of collaborative filtering te chniques," Advances in Artificial Intelligence, 2009.

[5] Sparse matrices http://docs.scipy.org/doc/scipy/reference/sparse.html

[6] Mahiux Ellis 2-11 http://labrosa.ee.columbia.edu/millionsong/tasteprofile