CS174: MACHINE LEARNING

PROJECT MILESTONE WRITE-UP

Personalized Song Recommender System

Michael Lau, Binjie Li, Beitong Zhang

February 17, 2015

1 INTRODUCTION

To begin building our music recommendation algorithm, we have implemented a naive collaborative filtering neighborhood model. Doing so has allowed us to keep the complexity of the project low while we figured out the best way to process and evaluate our data. Implementing the neighborhood model has also provided us with a decent baseline with which we can compare our future results.

The dataset we've chosen is rich in certain kinds of information, but it does not fit the standard model. Coming up with adaptations to the standard algorithms has been a nontrivial obstacle for every step of the process so far, but the solutions we've come up with should facilitate the second half of our project, matrix factorization with SVD.

2 DATASET AND DATA PREPROCESSING

We are using a real-world dataset [1] [3] containing the listening histories for 1000 different users from *Last.fm*. The size of the raw dataset is nearly **3GB** which is a really big dataset and surely quite difficult for us to analyze. Therefore, we first did analysis on the records to reduce the size of the dataset while keeping its underlying feature distribution as much as possible.

Fig. 2.1 shows the history size distribution in our original dataset and we can tell that the size ranges greatly among different users. For those users with small history sizes, we believe that they are not good samples for our analysis based on two reasons. First, if we only know a little about a user's history, then it would be difficult to find similar users and infer his taste in

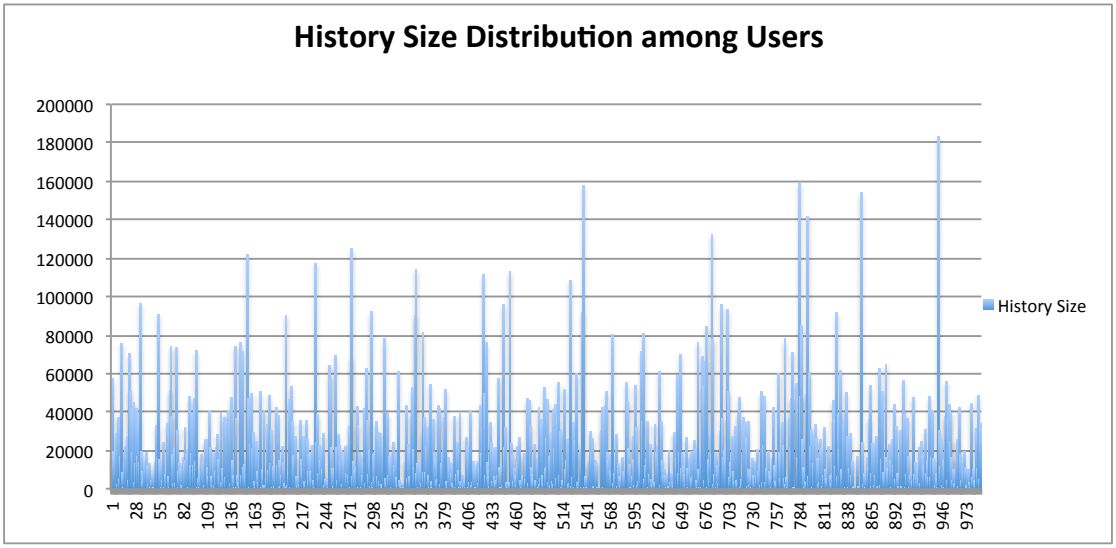

Figure 2.1: History Size Distribution among Users

music. Secondly, we aim to recommend new songs to each user, so for the user with a small history size, we won't have enough test data to predict the new songs he will listen to. We decided to filter users with history sizes smaller than 10K, leaving us with only 542 "valid" users out of 1K as our sample set.

In the raw dataset file, we have more than 100K artists and 1 million songs. Since putting them all in the user-item matrix is not computationally feasible, we analyzed the popularity of artists and songs and chose the most popular m artists and n songs to construct our dictionaries. For now, we've chosen $m = 3000$ and $n = 10000$, and we've fixed this number through the different experiments discussed in Secction 4.

We wrote a 200-line *Java* program to analyze the raw datafile and do the above two main things. The format of one history record in the raw datafile is as follows:

<timestamp, user_id, artist_id, artist_name, song_id, song_name>

After our program, we output two dictionary files for both artists and songs. Also, for each user, we create two vectors v_a and v_s corresponding to user-artists and user-songs:

$$v_a =< a_1, a_2....a_n >$$
$$v_s =< s_1, s_2....s_n >$$

where a_i and s_j are the times a user listens to the artist i and song j.

3 Implemented Method

3.1 Similarity

We determined the similarity of users x and y using simply the cosine of their vectors:

$$sim(x,y) = \cos(\vec{x},\vec{y}) = \frac{\vec{x}\cdot\vec{y}}{||\vec{x}||_2 \times ||\vec{y}||_2} \tag{3.1}$$

In actuality, each user had two vectors, one for the songs they had listened to, and the other for the artists. We considered adding these together with learned, optimized weights, but because we are using a significantly different method for learning in the future, we simply added the two cosines together.

3.2 Recommendations

The goal is to provide user x with a set of songs that he or she will probably like. This set, X_R, must be derived from the set $X_W = S - X_\ell$, where S is the set of all known songs and X_ℓ is the set of songs x has listened to.

For each song $S_i \in X_W$, we project a recommendation score r_{xi} based on the number of times other users U have listend to S_i, weighted by their similarity to x:

$$r_{xi} = \sum_{u \in U} sim(x,u) \times \frac{\ell_{ui}}{\sum_{j=1}^{|S|} \ell_{uj}} \tag{3.2}$$

ℓ_{ui} is the number of times user u listened to S_i. It is normalized by sum of all the times u has listened to any song, essentially giving the percent that S_i constitutes u's history.

This function was based off of one for a standard dataset, where a user's relationship with an item is deteremined by a rating, not a history. These ratings were normalized by subtracting the average of all the user's ratings [2].

$$r_{xi} = \bar{r}_x + k \sum_{u \in U} sim(x,u) \times (r_{ui} - \bar{r}_u) \tag{3.3}$$

Ratings and histories differ in their range of values, so normalizing using the original function was not effective. Also, because we are not predicting a rating, we do not need to incorporate the user's average rating or the normalization constant k.

The percent-based normalization we've developed worked adequately, but percentage was basically a low hanging fruit. It is highly probable that a better normalization scheme exists for this type of data.

4 Evaluation

Using the first n songs that X listened to, X_n, to find recommendation scores for $R = S - X_n$, we tried to predict X_p, the m most listened to songs in $X - X_n$. R_m is the m highest scoring songs in R. We define success rate z as:

$$z = \frac{\sum_{r \in R \text{ and } r \in X_p} 1}{m} \tag{4.1}$$

We ran our program in *Matlab* with different parameters and recommend songs to each of users. Then we calculate the average accuracy $Accu$ to evaluate our algorithm.

$$Accu = \frac{\sum_{u \in U} z_u}{|U|} \tag{4.2}$$

where U is the user set containing 542 valid users.

This evaluation scheme suits the context of our problem better than the more general approach of matrix completion, because we are not attempting to estimate user ratings. We do not have ratings in our dataset, we have listening history, so it would make sense to try to predict future listening behavior.

Furthermore, we believe this to be a more aggressive goal. With this evaluation, we are attempting to predict, out of the songs that they have not yet heard, which ones the users will listen to the most. Discovering a user's favorite songs before the user does is a much more meaningful goal than predicting ratings for songs the user might never listen to.

4.1 Results with different numbers of recommend songs

We choose different number of songs, from 5 to 30, as the size of recommended songs to each user and we observe the change of the average accuracy $Accu$.

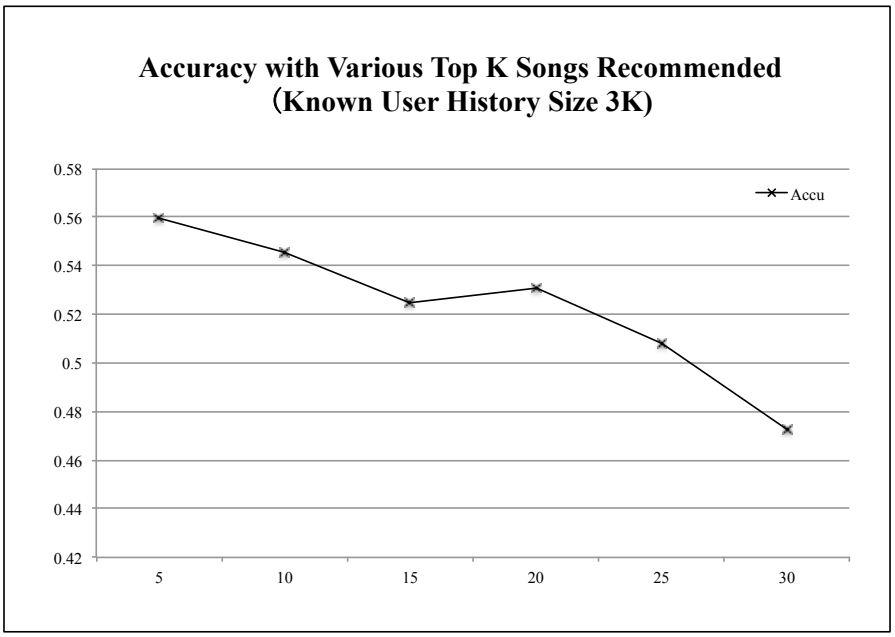

Figure 4.1: Accuracy with Various Top K Songs Recommended

The result in Fig 4.1 shows that the accuracy goes down when we have a larger number of recommended songs. It is quite reasonable that when we make a larger number of prediction with the same size of training data, the accuracy would surely goes down.

4.2 Results using different size of training data

Additionally, we ran the program with different size of training data set. We used different sizes of training data (the known history size) and observed the change of the average accuracy $Accu$. The size we used varied from 0.5k to 5k. The number of songs we recommended is fixed at 20.

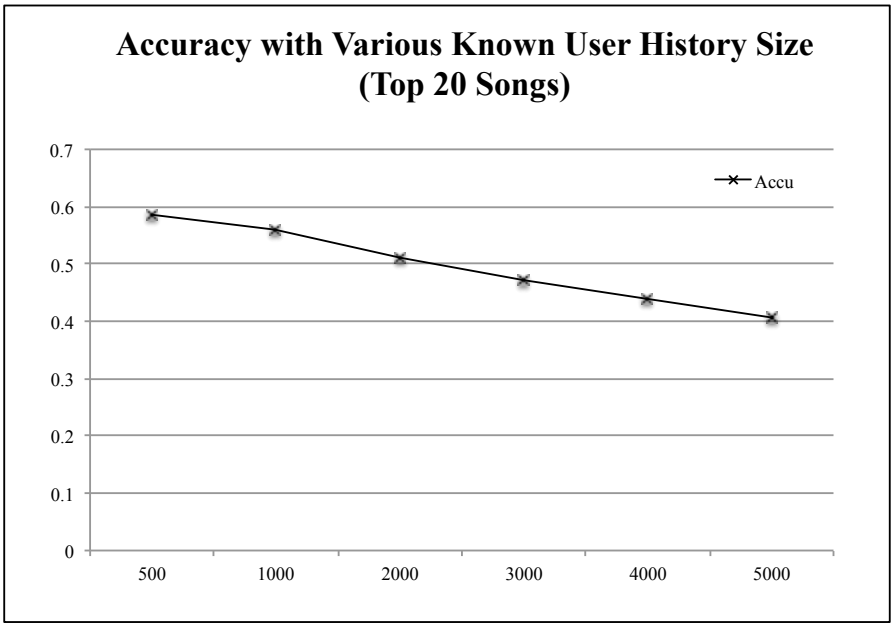

Figure 4.2: Accuracy with Various Known User History Size

Fig 4.2 shows that the accuracy goes down when we have a larger history set. The result is quite counter-intuitive at first glance, but after further analysis we think it is a reasonable result. We want to recommend songs that the user hasn't heard before. However, when the size of listening history becomes larger, the user may tend to listen to some "old" songs instead of many new ones since a kind of "listening pattern" is formed. Hence, the new songs they choose to listen may not reflect the listening habit very well and it's harder for us to predict the new songs. Also, with larger known history, there is a smaller unknown history for us to predict.

5 Future Work

5.1 Combination with more features

We have other user profile features that we have not yet incorporated into the analysis, and there remains the possibility of using additional song features from another dataset.

5.2 Matrix factorization with SVD

Matrix factorization with SVD is a popular choice for recommender systems, but it also has some particularly appropriate applications for our data.

The standard model, as given in the Recommender Systems Handbook [4], has a rating prediction function:

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T p_u \tag{5.1}$$

and a learning function:

$$\min_{b_*, q_*, p_*} \sum_{(u,i) \in \mathcal{K}} (r_{ui} - \mu - b_i - b_u - q_i^T p_u)^2 + \lambda(b_i^2 + b_u^2 + ||q_i||^2 + ||p_u||^2) \tag{5.2}$$

Here again we are presented with certain challenges in adapting our dataset. There is no clearly appropriate value for μ, and as we are not predicting ratings, but producing a set of recommendations, it is unclear whether we need μ at all.

5.3 Further refinement

Our data is rich with temporal information. Intuitively, it makes perfect sense that musical preferences shift with time and the user's mood, and that song biases will rise and fall as songs fade in and out of public consciousness. Matrix factorization lends itself to modeling these temporal shifts, but it will involve changing the user and item biases b_u, b_i into real valued functions that change over time. This will not be trivial, and will require further refinement of both our preprocessing as well as our matrix factorization model.

References

[1] last.fm music recommendation dataset. `http://last.fm`. Accessed: 2015-02-17.

[2] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, 17(6):734–749, 2005.

[3] O. Celma. *Music Recommendation and Discovery in the Long Tail*. Springer, 2010.

[4] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B Kantor. *Recommender systems handbook*, volume 1. Springer, 2011.