CS229 Project Report
Offline Music Recommendation

Emilien Dupont, Isabelle Rao, William Zhang

{edupont, isarao, wzhang4}@stanford.edu

Abstract— **The goal of this project was to recommend songs to users based solely on their listening histories, with no information about the music. We applied various Collaborative Filtering methods to achieve this: user-user neighborhood models, item-item neighborhood models and latent factor models. We achieved the best results with item-item cosine similarity. The code for this project can be found here.**

I. INTRODUCTION

Companies like Spotify and Pandora recommend songs to their users based on user listening histories and on meta-data about the music. In this project we attempt to make recommendations based solely on user history, using the Million Song Dataset [5]. This dataset is comprised of a large number of user listening histories used for training. Separate from this is a test set comprised of users whose listening histories did not appear in the training set. It is split into two parts: a visible one, consisting of half of each unseen user's history; and a "hidden" one, consisting of the true, other half of the same users' listening histories. Using various collaborative filtering methods, we attempt to predict the hidden half, given the visible half.

II. RELATED WORK

The methods and procedures in our recommendation system are used widely, not only in music, but in various other areas such as movies, news and e-commerce. Companies like Facebook, Twitter and LinkedIn also use such methods to recommend friends/followers/connections [4]. As such, there is a large amount of literature relating to this subject. Perhaps most famously, Netflix [1] challenged competitors to come up with an algorithm to recommend movies to their users based on their viewing history. The winning entry used a combination of various methods, but one of the most succesful algorithms was based on Latent Factor Models, which we will explore in this project.
As another famous example, Amazon uses both item-item and user-user correlations to recommend products to customers [6]. We will attempt to emulate this approach using cosine similarity.
As mentioned by Hu et al. [3], one of the most common approaches to collaborative filtering is that of neighborhood models (see below for more explanation). The underlying assumption is that users with similar ratings on some items will have similar ratings on the others (an analogous assumption is made for items that share similar ratings for many users). Another set of methods that has shown promise recently relies on low-rank matrix factorization, which seeks to uncover the most important factors governing song choices. These two approaches are the ones we will be focusing on in the rest of this paper.

III. DATASET & FEATURES

The data is in the form of (user, song, play count) triplets. For example:

- (Isabelle, Hey Jude, 6)
- (Isabelle, Shake it Off, 12)
- (William, Whole Lotta Love, 15)
- (Emilien, Shake it Off, 9)

The training set contains 48 million such triplets, corresponding to 1.2 million users and 380K songs. The test set consists of 100K users and 157K songs. There are a few interesting statistics we can note about the data:

- There are 780 songs in the test set which do not appear in the training, so we will never be able to predict those

- The average number of play counts for a user is 142
- The average number of unique songs for each user is 50

If we let U denote the number of users and I the number of items (or songs), then we can store the data as a $U \times I$ matrix of play counts with entries

$$C_{u,i} = \text{number of times } u \text{ has listened to song } i$$

A. Count to Rating

One of the main challenges for this project was the fact that we wanted to apply methods for explicit feedback (e.g. a IMDb user's rating for a movie is explicit), whereas our data was implicit in the form of song counts. Most collaborative filtering methods rely on each entry of the user-item matrix being a "rating" (e.g. 1-5 stars). We have therefore experimented with several different ways of defining a "rating" $R(u,i)$ of user u on item i from play counts:

- $R(u,i) = C(u,i)$ (counts)
- $R(u,i) = \begin{cases} 1 & \text{if } C(u,i) > 0 \\ 0 & \text{otherwise} \end{cases}$ (binary counts)
- $R(u,i) = \frac{C(u,i)}{\max_i C(u,i)}$ (max normalized)

We have experimented with using these ratings as features and our results are shown in the sections below.

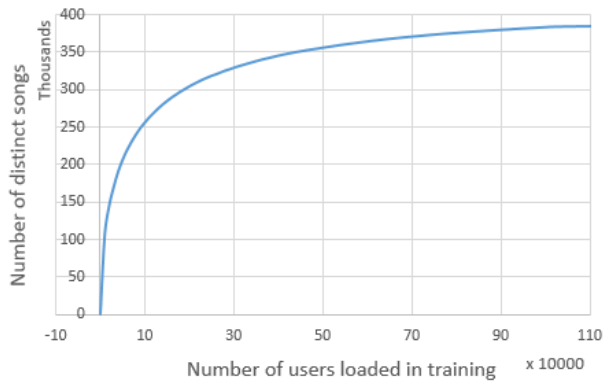

Fig. 1. Number of distinct songs listened to by all the users, as a function of number of users in the dataset.

Finally, we should note that when the data is initially loaded in, a lot of new songs (as shown in figure 1). Once a few users have been loaded in, you will start to see more songs you have already seen. In order to make accurate predictions we need to have encountered a significant amount of the songs in the training set, which means we need to train on a large number of triplets.

IV. METHODS

We used two different types of **Collaborative Filtering** methods: Neighborhood Models and Latent Factor Models.

A. Baseline Model

As a baseline, we recommended the 500 most popular songs to every user.

B. Neighborhood Model

We implemented two types of neighborhood models. The assumption behind neighborhood models is that if two users u and v are similar (in a sense that will be made precise), then user u will like songs in user v's listening history. We use Cosine Similarity (as seen for instance in [2]) to define the similarity between 2 users:

$$\text{similarity}(u,v) = \frac{u^T v}{\|u\|\|v\|}$$

where u is the row corresponding to user u in the rating matrix. Item-item similarity is defined in a similar way

$$\text{similarity}(i,j) = \frac{i^T j}{\|i\|\|j\|}$$

where i is now a column of the rating matrix. Intuitively, two users are similar if they have listened to a lot of the same songs (the dot product will be over a lot of the same song indices). Similarly, two songs are similar if they have been listened to by many of the same users. Now given our similarity matrix, how do we make recommendations? To do this, we define a score function which represents our guess of how much a user u will enjoy a song i. Let $U(i)$ be the set of users who have listened to song i. Then we define [4]

$$\text{Score}(u,i) = \sum_{v \in U(i)} f(\text{similarity}(u,v))$$

where $f(x)$ is some scoring function. A similar definition can be given in terms of item-item

similarities. To make a recommendation for a user u, we then simply pick the k songs with the highest score and recommend those.

C. Latent Factor Model

Latent factor models are another approach we considered during the course of this project. Specifically, we considered an approach inspired by the Singular Value Decomposition. The idea is to approximate the ratings matrix R as the product of two rank k matrices: $R \approx X^T Y$, where X is a $k \times U$ matrix and Y is a $k \times I$ matrix. The hope is to be able to respectively summarize each user and item by the k-dimensional vectors x_u and y_i, where the k components capture the salient latent factors behind the ratings matrix. Intuitively, we would like the product $X^T Y$ to be as close to R as possible (minimize for instance the Frobenius norm of their difference). However, this is likely to overfit the data on the observed ratings. One way to avoid this is to include regularization terms, and the optimization problem can thus be formally stated as below:

$$\min_{X,Y} \sum_{r_{u,i} \in \text{observed}} (r_{u,i} - x_u^T y_i)^2 + \lambda_x \|x_u\|^2 + \lambda_y \|y_i\|^2$$

Gradient descent can be used to optimize the objective function; however, it is non-convex because of the dot product $x_u^T y_i$, and it turns out that gradient descend is often slow, requiring several iterations. Another optimization routine is Alternating Least Squares, which, as its name suggests, alternatingly treats X and Y as constants and optimizes for the other variable (see Algorithm 1).

Algorithm 1 Alternating Least Squares

Initialize X, Y (using SVD, for example)

Repeat until convergence:

- for $u = 1...U$ do

$$x_u \leftarrow \left(\sum_{r_{u,i} \in r_{u*}} y_i y_i^T + \lambda_x I_k \right)^{-1} \sum_{r_{u,i} \in r_{u*}} r_{u,i} y_i$$

- for $i = 1...I$ do

$$y_i \leftarrow \left(\sum_{r_{u,i} \in r_{*i}} x_u x_u^T + \lambda_y I_k \right)^{-1} \sum_{r_{u,i} \in r_{*i}} r_{u,i} x_u$$

V. RESULTS & DISCUSSION

A. Evaluation metric: Mean Average Precision

Before discussing results we need to define our evaluation metric. The predictions are evaluated by means of the *mean average precision*, as described in the AdMIRe'12 paper [5]. Define the *feedback* matrix $M \in \{0,1\}^{U \times I}$, where $M_{u,i} = 1$ if song i appears in user u's unseen listening history, 0 otherwise, and y_u a prediction for user $u \in \{1,...,U\}$, where $y_u(j) = i$ indicates that song i is ranked at position j for user u. It's assumed that y_u omits items already known to be played by u. Three steps are necessary to compute the MAP:

- For any k, define the *precision-at-k* P_k as the proportion of correct recommendations within the top-k of the predicted ranking:

$$P_k(u,y) = \frac{1}{k} \sum_{j=1}^{k} M_{u,y(j)}$$

- The next step is to take the average of the previously computed precisions:

$$\text{AP}(u,y) = \frac{1}{n_u} \sum_{k=1}^{\tau} P_k(u,y) \times M_{u,y_u(k)}$$

  where τ is a threshold that represents how many of the top predictions in y_u to include, and n_u is the minimum of the number of hidden songs in a test user's history, and τ.
- The final step is to average the previous quantity over all m users:

$$\text{MAP} = \frac{1}{m} \sum_{u} \text{AP}(u, y_u)$$

B. Results and analysis

In all of the announced results that follow, we determined the parameters using k-fold cross-validation, where $k = 5$, on a subset of the data that spanned 10k users in the training set and 1k users in the test set (chosen randomly). In figure 2 we show the MAP for our various algorithms on a set of 50K training users and 5K test users (corresponding to 208K songs and 2.5 million triplets) . We used (mostly) binary ratings for the song counts (1 if song has been listened to and 0 otherwise). As can be seen, the item-item Cosine similarity model performs the best, while user-user

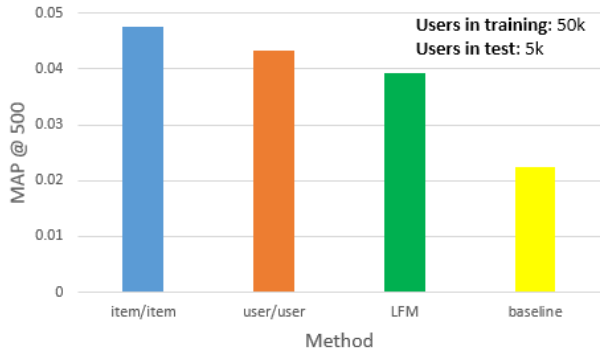

Fig. 2. Performance of various methods on a 50K user training, 5K test training. Best mAP is 4.76%.

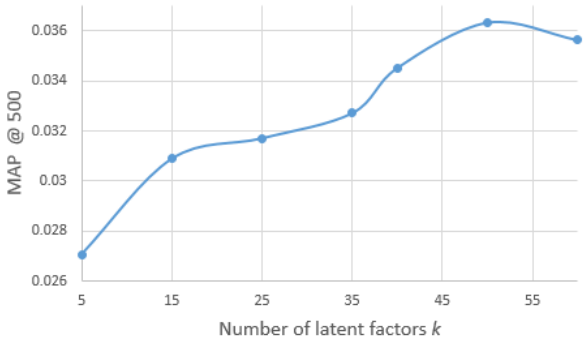

Fig. 4. Performance vs number of factors on 10K training users, 1K test users

similarity gives similar results. While outperforming the baseline, the Latent Factor Model did not perform as well as the other two models.

C. Tuning the models

1) Cosine similarity: For the similarity models, we experimented with tuning a parameter α in the range [0,1] on the similarity measure

$$\text{similarity}(u,v) = \frac{u^T v}{\|u\|^{2\alpha}\|v\|^{2(1-\alpha)}}$$

Results are shown in figure 3.

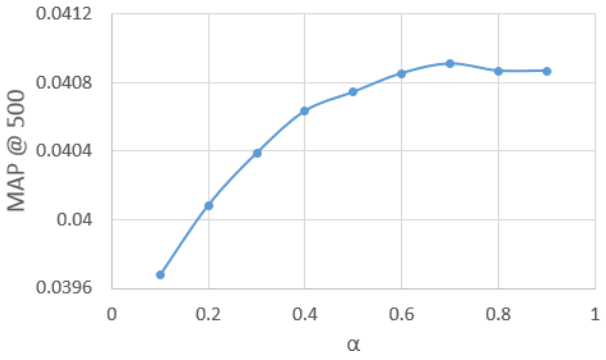

Fig. 3. Performance of user-user collaborative filtering on a 20K user training, 2K test training for various values of α

As can be seen, the best choice for α is approximately 0.7. The "standard" choice for $\alpha = 0.5$ gives very similar results, but we do obtain slight improvements (order of 0.1% in MAP).

2) Latent Factor Model: For the Latent Factor Model, there were several parameters we could tune: the number of factors k and the regularization parameters λ_x and λ_y.

The optimal values of the regularization parameters depended on the size of the datasets. For 10k training user, the regularization parameter was chosen to be 21 (which is also the value used in figure 4). As can be seen, increasing the number of factors improves the performance of the Latent Factor Model. This is to be expected as we will capture more "information" about each user and item with larger vectors. However, when using a very large number of factors (55 on the plot) the performance starts to dip, a likely indication of overfitting.

D. Different types of ratings

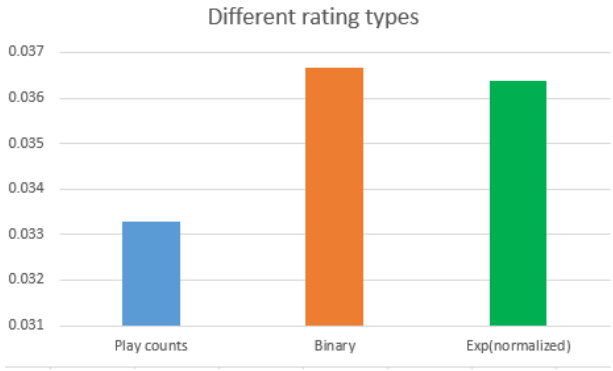

Fig. 5. Performance of various rating types on 50K training users, 5K test users for user-user cosine similarity.

We also experimented with using different count to rating functions as described earlier. Surprisingly, the best results were obtained when we used binary ratings, i.e. $R(u,i) = 1$ if user u has listened to song i and 0 otherwise. This means we throw away all information about how many times the

user has listened to the song. While this could be useful information, for our model it creates a large unwanted bias towards highly played songs. We experimented with using log's of play counts as well to reduce this, but binary ratings still had the best performance.

E. Number of predictions

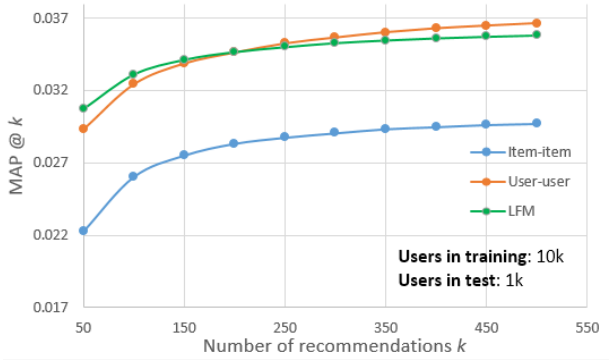

Fig. 6. MAP @ τ vs number of recommendations τ

Unsurprisingly, increasing the number of predictions made, increases the MAP. However, there is a tradeoff between recommending a large amount of songs and having a high MAP: firstly, it does not make sense to recommend 10,000 songs to a user, secondly, after some threshold the increase in the MAP becomes so small it is almost negligible. As can be seen in figure 6, 500 recommendations is a good threshold (which is also the one chosen by the initial paper [5])

F. Computational Limitations

As mentioned previously, in order to make accurate predictions, we need to load in many user histories. Indeed, initially we will see a lot of new unseen songs and as the number of users increases we will see fewer new songs. As can be seen in figure 1, the number of new songs added to the count matrix stops increasing significantly after we load in about 500K users, which corresponds to about 350K songs. This means our count matrix would be 500K by 350K, which is very large (even though we, of course, use sparse matrices). Both Cosine Similarity and Latent Factor Models are very slow on matrices of this size. As we were limited to using a single machine to run our algorithms, we could not run our models on more than 50K users without the computations becoming prohibitively slow. We believe one of the reasons for not obtaining higher MAP scores is related to this, and it would have been very interesting to experiment with our models on larger matrices.

VI. Conclusion & Future Work

In this project, we studied and compared the performance of two types of collaborative filtering models using implicit feedback. Both the neighborhood model and the latent factor model significantly outperformed the baseline of recommending the most popular songs to every user.
The procedure which performed the best was item-item cosine similarity for which we obtained a MAP of 4.76 %. We used this in combination with Binary Ratings for which we also obtained the best results.
The main challenge in getting these procedures to yield a satisfactory MAP is that collaborative filtering methods have been shown to work well in the context of recommendation. However, our task was somewhat different: we are trying to predict what other songs a user already has listened to. In future work, we would like to explore variants of the latent factor model, some of which also try to include an additional weights matrix representing the confidence in observing a certain song for a particular user. Furthermore, we believe that running our models on larger matrices would have yielded a significant improvement in performance.

References

[1] James Bennett and Stan Lanning. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35, 2007.

[2] Michael D Ekstrand, John T Riedl, and Joseph A Konstan. Collaborative filtering recommender systems. *Foundations and Trends in Human-Computer Interaction*, 4(2):81–173, 2011.

[3] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *In IEEE International Conference on Data Mining (ICDM 2008*, pages 263–272, 2008.

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

[5] Brian McFee, Thierry Bertin-Mahieux, Daniel PW Ellis, and Gert RG Lanckriet. The million song dataset challenge. In *Proceedings of the 21st international conference companion on World Wide Web*, pages 909–916. ACM, 2012.

[6] J Ben Schafer, Joseph Konstan, and John Riedl. Recommender systems in e-commerce. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 158–166. ACM, 1999.