Clustering of Last.fm Music

Jordan Rosenblum, Justin Law & Erin Grand

Data Science Institute, Columbia University, New York, NY 10027

May 7, 2015

ABSTRACT

We attempt to build a recommendation system for a subset of the Million Song Dataset. We explore various algorithms including matrix factorization, user-based collaborative filtering, and item-based collaborative filtering. In each case we recommend the top 500 songs that our algorithm returns to each user and compare against the testing set of what the user actually listened to. Matrix factorization gives the worst results, only slightly above the popularity baseline, while artist-based popularity and user-based and item-based collaborative filtering methods yield better results.

Contents

1. Introduction

In the online world, recommendation systems are becoming increasingly important in various industries including in retail, entertainment, and even dating. Because of their popularity and usefulness, there are numerous incentives to implement a good recommendation system. In the music industry, companies use recommendation systems to provide a better service to their listeners. With the music industry shifting away from a more traditional distribution model of physical CDs or records, having an efficient and well performing recommendation system is very important.

In building a recommendation system, we had to take into account various trade-offs in the type of models we chose to fit. For example, some methods take more information about the user and the song into account (i.e. content-based) while others use a collaborative filtering approach without taking into account user or song characteristics such as age, genre, location, etc. There is no set similarity measure between two songs, as evident by the fact that songs by the same artist are often very different. As such, there are many different types of algorithms to explore.

For our project, we explored various song recommendation systems algorithms such as popularity recommendations, artist based recommensations, matrix factorization, and user and item based similarity measures.

1.1. Data

The main source of data for this project was the Million Song Data Set (MSD) (Bertin-Mahieux et al. 2011). We used the sample data set from the Kaggle competition which contains tuples of user id, song id, and play count.

The sample data set had 110000 users, 163206 songs, and the number of play counts for each user and song pair. The scale of the data was too large to explore on our computers, so we subset the data down to a size easily handled by an average laptop.

We choose to subset the data to users who have listened to more than 27 songs and songs which were listened to by more than 22 users. This resulted in a datset which was 10% the size of the original. Despite subsetting the data, we are still dealing with a matrix of 8130 users by 11861 songs, so some of our algorithms will take a fair amount of time to run.

1.1.1.   Data Statistics

- Sparsity .0016086 -

SONG PLAY COUNT
mean = 28
max = 35432
min = 1
std = 215.826789

USER PLAY COUNT
mean = 42
max = 1305
min = 5
std = 53.31547

1.2.   Data Analysis

In order to perform the analysis, we had to fill in training and testing matrices which contained play counts for user and song pairs. We did a random selection of data elements to split between the training (80%) and testing (20%) sets. Next, we implemented various algorithms on the dataset in order to recommend songs to users, including matrix factorization, user-based collaborative filtering, and item-based collaborative filtering.

1.3.   Testing

We tested our algorithms against each other using a mean average precision score for each method. The mean average precision (MAP) calculation came directly from the Kaggle MSD challenge (kag 2012). The score was used to rank the competition entries.

Specifically, we used MAP@500 scores which take into account the first 500 recommendations given to each user. Generally, the score looks at the precision between a list of recommendations for a user and that user's test set. This was done by calculating the average precision at each element in the list of recommended items (i.e. percentage of correct items among first k recommendations) and averaging these for the first 500 recommendations (i.e. for each $k\epsilon 1, ..., 500$, calculate the precision and average them together). We then averaged the score over all users. Intuitively, the score looks at the percentage of recommendations that were in-line with the test set of songs actually listened to but also takes order into

account. So it is preferable to recommend the songs in the user's test set earlier in the list of 500. The MAP score is similar to the AUC of the ROC curve discussed in class except it takes ordering of the recommendations into account, which is an important factor to consider in a recommendation system.

We used two different baseline benchmarks to compare our MAP scores against. The first was to simply recommend the top 500 most popular songs to every user. This naive method resulted in a MAP score of 0.0138 (top 500 songs based on number of users who listened to the song) or 0.0126 (top 500 songs based on number of plays the song had). The second method was to first recommend a user the most popular songs of artists the user already had already played (but different songs from that artist). If this did not result in enough recommendations, we would then recommend the overall most popular songs as well at the end (similar to baseline method 1). Although this method is conservative in that it does not go beyond the tastes the user already has in the training set, it resulted in a significant improvement in the MAP score to 0.0448.

2. Algorithms

2.1. Matrix Factorization

Matrix factorization is a type of collaborative filtering algorithm that can be used to build a recommendation system for users based on user feedback of objects (in this case songs). This allows us to recommend songs to users based on their listening history without the need for content based approaches. Training and testing matrices were constructed, of which both have N_1 users (rows denoted by u_i) and N_2 songs (columns denoted by v_j). As users have only listened to a fraction of all the songs available, the matrices are sparse, containing zeros in all entries except for those in which a user (a user representing a single row) has listened to. The goal of matrix factorization is to factor the training matrix into the product of two matrices U and V. The matrix U is $N_1 \times d$ and the matrix V is $d \times N_2$. Another goal of matrix factorization is typically to learn a low-rank factorization of the original matrix while restricting the patterns that are generated in the product of the factorized matrix (e.g. if a user likes a song in a specific group of songs, the user may also like other songs in that group of songs). The choice is d can be subjective but 20 is a common value to start with. In the factorized matrices, the predicted rating will be $\hat{M}_{ij} = u_i^T v_j$ Using a coordinate ascent algorithm over 100 iterations, each row (u_i) and column (v_j) of the training matrix is then updated (equations 1 and 2) in order to maximize the log joint likelihood (equation 3).

$$u_i = \left(\lambda \sigma^2 I + \sum_{j \in \Omega_{u_i}} v_j v_j^T \right)^{-1} \left(\sum_{j \in \Omega_{u_i}} M_{ij} v_j \right) \tag{1}$$

$$v_j = \left(\lambda \sigma^2 I + \sum_{i \in \Omega_{v_j}} u_i u_i^T \right)^{-1} \left(\sum_{j \in \Omega_{v_j}} M_{ij} u_i \right) \tag{2}$$

$$\mathcal{L} = \sum_{(i,j) \in \Omega} \frac{1}{2\sigma^2} ||M_{ij} - u_i^T v_j||^2 - \sum_{i=1}^{N_1} \frac{\lambda}{2} ||u_i^2|| - \sum_{i=1}^{N_2} \frac{\lambda}{2} ||v_j^2|| + \text{constant} \tag{3}$$

Note: $\lambda = 10$ was used and Ω is the set of all indices in the matrix which have an observation. We keep track of the root mean square error (RMSE) versus the testing set (i.e. how close our prediction is as compared to the actual normalized play count in the testing set) and the log joint likelihood of the training set as a function of iteration (see Figures **??** and **??**). There are several hyper parameters that can be tweaked in order to achieve a better mAP result using matrix factorization. The main ones that was varied were the variance, the rank d of the factorization, and the iterations. As the project was being done using Python and the main computational package (*scikit-learn*) did not have probabilistic matrix factorization functions, we decided to apply the algorithm that we had wrote ourselves for a different class.

After testing around with a few parameters, it was soon discovered that matrix factorization results were significantly below the popularity baseline and changing the hyperparameters around did not help much. The best MAP value of 0.0032 were achieved was using a high rank d of 80 over 100 iterations with a variance of 0.0001. The log joint likelihood chart can be used to verify that the probabilistic matrix factorization algorithm is running properly in that the values are monotonically decreasing which corresponds to a decreasing training error. The plot shows that there is more room for improving the training error, but the MAP plot that was taken from the same run shows that there is marginal difference in terms of improving the MAP score after the initial 5 iterations. Due to this result, further exploration of the probabilistic matrix factorization algorithm was limited to 30 iterations to reduce computational time.

One explanation for the very low MAP scores using PMF was that the number of plays that were used are different from what PMF is typically used for which is bounded ratings. Using implicit user feedback (number of plays) instead of explicit user feedback (ratings) have been found in literature to create various issues. One of the main issue is that implicit feedback is inherently noisy does not measure negative feedback. Users may have played a song to simply find out that the dislike the song. The other main issue is that the exact number of plays has could have significantly different meaning measured between different users. For example, an avid user who has played a song five times could have different meaning to a user that has at most only played songs five times.

In order to tackle the issue of using implicit user feedback, we tried four different scaling schemas independently. These include normalizing user play counts by the total number of plays for the user, normalizing song play counts by the total number of plays for the song,

making the data binary (changing all non-zero play counts to 1), and also applying tfidf. While testing out these schemas, we continued to vary the hyperparameters d and variance in order to determine if they changed the MAP results significantly.

From the experiments, it was found that the rank d did not contribute significantly to the MAP values especially in settings where the MAP values were close to the baseline. Higher variance values were found to actually significantly increase the MAP values. In terms of scaling schemas, making the data binary produced the best result, with the second best being normalizing the data against the total number of plays for each user. Perhaps it can be said that using number of plays introduces more noise than simply considering if the user has played the song or not. Interestingly enough, normalizing the data against total number of plays for each song performed the worst and this most likely relates to the fact that number of plays is more meaningful relative to each user rather than to each song.

Regardless, the best result achieved for PMF in the end was 0.0143 which was only slightly above the popularity baseline. Based on literature review, we have found that this is better or in line with what other people have achieved with using PMF on play counts. Nevertheless, this indicates that PMF is not suitable for the MSD dataset and this can be attributed to the implicit feedback issue or simply that the dataset is just too sparse. However, it is still possible to apply matrix factorization techniques to this dataset through using metadata and this has been demonstrated to be viable by Li et al. (2012) which achieved MAP values exceeding 0.1.

2.2.   User Based and Item Based Collaborative Filtering

A simpler set of methods for determining which songs to recommend to individual users is by using user- or item/song- based similarity (both also considered collaborative filtering). These only consider whether or not a user listened to a song and does not take into account the play count. Similarity scores between users (i.e. user-based) and songs (i.e. item-based) were calculated using cosine similarity. Intuitively, they work by recommending users songs that other similar users like (user-based) or by recommending songs which are similar to the songs the user has already listened to (item-based). The downside to this method is the computational complexity. Given the quadratic runtime of these algorithms, they are more difficult to scale to a larger dataset. As for storage requirements, we will have to store all pairs of similarity scores for users in the case of user-based and songs in the case of item-based. The details are described below.

2.2.1. *User-Based Collaborative Filtering*

For user-based recommendation, we first calculated the similarity score between every pair of users, u and v, using the equation below:

$$sim(u, v) = \frac{\# \text{ common items}(u, v)}{\# \text{ items}(u)^{1/2} \times \# \text{ items}(v)^{1/2}} \tag{4}$$

Then, for each user, we looked at each song in the dataset and found all other users who have listened to that song. Next, we added up the similarity score of each of those users, v, with the original user, u, and got the weight for a particular song, i:

$$w_i = \sum_{v \epsilon V} sim(u, v)$$

The sum is a proxy for how likely the user is to like that particular song. We went through that process for all songs and recommended the user the top 500 songs which had the highest scores. This method led to a MAP value of 0.0377.

2.2.2. *Item-Based Collaborative Filtering*

For item-based recommendation, which turned out to be our best algorithm in terms of a MAP value, we first calculated the similarity score between every pair of songs, i and j, using the equation below and saved the most similar songs to each song in the process:

$$sim(i, j) = \frac{\# \text{ common users}(i, j)}{\# \text{ items}(i)^{1/2} \times \# \text{ items}(j)^{1/2}} \tag{5}$$

Then, for each user, we found all the songs listened to. Next, we got the most similar songs to each of those songs listened to by the user. If the same similar songs came up multiple times, we added the weights together. In other words, for each song, b, that was found to be similar to one of the songs a that the user listened to, we calculated the score for that song, b:

$$w_b = \sum_{a \epsilon A} sim(a, b)$$

Lastly, we sorted the songs by scores and recommended the user the top 500 songs which had the highest scores. This method led to a MAP value of 0.0479.

3. Conclusion

After comparing the various algorithms, the simpler models gave us the best results. Our most complicated model, matrix factorization, gave us results barely above the popularity baseline. On the other hand, simply recommending users songs by artists they have already listened to (the artist-based baseline) resulted in a significantly better recommendation system. Our song-based collaborative filtering technique produced the best results despite being a relatively simple algorithm as compared to matrix factorization. Our final results for each technique employed are summarized in the table below.

Algorithm	MAP score
Artist Based Baseline	0.0448
Top 500 Songs by Count Baseline	0.0138
Top 500 Songs by Plays Baseline	0.0126
User Based CF	0.0377
Item Based CF	0.0479
Matrix Factorization	0.0143

4. Notes

Sparsity of $8.08199453451 \times 10^{-5}$ (needs updating)

29 - users 34 - songs 60 %

Interesting plots: - Cumulative Distribution of number of songs with a given play count

To Reference a paper:

The Million Song Dataset Challenge: (McFee et al. 2012)
Million Song Dataset Recommendation Project Report: (Li et al. 2012)
codebook-based scalable music tagging with poisson matrix factorization: (Liang et al. 2014)
Matrix Factorization Techniques for Recommender Systems: (Koren et al. 2009)
A Preliminary Study on a Recommender System for the Million Songs Dataset Challenge: (Aiolli 2013)

REFERENCES

(2012). Million song dataset challenge.

Aiolli, F. (2013). A preliminary study on a recommender system for the million songs dataset challenge. `http://www.ke.tu-darmstadt.de/events/PL-12/papers/08-aiolli.pdf`.

Bertin-Mahieux, T., Ellis, D. P., Whitman, B., and Lamere, P. (2011). The million song dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*.

Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix factorization techniques for recommender systems. `http://www2.research.att.com/~volinsky/papers/ieeecomputer.pdf`.

Li, Y., Gupta, R., Nagasaki, Y., and Zhang, T. (2012). Million song dataset recommendation project report. `www-personal.umich.edu/~yjli/content/projectreport.pdf`.

Liang, D., Paisley, J., Ellis, D. P., et al. (2014). Codebook-based scalable music tagging with poisson matrix factorization. `http://www.columbia.edu/~jwp2128/Papers/LiangPaisleyEllis2014.pdf`.

McFee, B., Bertin-Mahieux, T., Ellis, D. P., and Lanckriet, G. R. (2012). The million song dataset challenge. `https://www.ee.columbia.edu/~dpwe/pubs/McFeeBEL12-MSDC.pdf`.

5. Appendix

- Link to our github repository: `https://github.com/eringrand/musicanalysis`

- The MSD Kaggle challenge: `https://www.kaggle.com/c/msdchallenge`

This preprint was prepared with the AAS LATEX macros v5.2.