A Comparison of K-means Clustering and Memory-Based Collaborative Filtering for the Million Songs Dataset Challenge

by Alberto Lalama and Felipe Munera

ABSTRACT

This paper describes a preliminary study of two song recommendation approaches. The framework of the study was the Millions Song Data Set challenge, a competition whose objective is to suggest songs to users, given their half their listening histories and a dataset of meta-information of one million songs. We investigated a collaborative based filtering approach, subdivided into user-based filtering and item-based filtering, and a k-means clustering approach, where a subset of features of the million songs were examined. After evaluating these techniques with a mean average precision function, our results indicate that collaborative based filtering yield more accurate recommendations than k-means clustering.

1 INTRODUCTION

The Million Dollar Song Database Challenge (MSDC) is a music recommendation competition, where the goal is to predict the songs a user will listen to provided the listening history. The data for each song is made available by the Million Song Dataset (MSD), a open and free collection of music songs online, powered by Info Chimps.[4] The data used in the MSDC is the Test Profile Subset, which consists of ~48 million triplets of user, song, play count, totaling around 1.1 million users that reference around 380,000 songs from the MSD. It is worth highlighting the magnitude of the challenge, since significant efforts where invested into creating fast and reliable algorithms.

We utilized two approaches to solve this challenge, K-means clustering (KM) and Collaborative Filtering (CF), the comparison was inspired by the work of Fabio Aiolli [1]. CF is a technique that takes advantage of known user preferences over items, in order to filter patterns among users and generate recommendations. The underlying approach stems from the fact that if users ***u*** and **v** both like a given item **i**, then **u** is more likely to like another item **j** that **v** likes that one of a randomly chosen user. The same intuition can be used to filter items based on the users who like them.

K-means clustering is a technique that aims to optimally group similar high dimensional points into clusters, where each cluster is defined by a centroid. The songs are assigned to the cluster that minimizes the euclidean distance between the centroid and the feature vector of the song. Thus, grouping songs into clusters allows recommendation of similar songs to the user, simply by recommending songs that also belong to that cluster.

2 METHOD

2.1 Memory-Based Collaborative Filtering (MBCF)

The approach that we to MBCF is the one suggested by the creators of the MSDC [2] and used by Aiolli[1] Recreating Aiolli's notation, we define ***U*** as the set of ***n*** users and ***I*** as the set of ***m*** items. Then the entries of $\mathbf{R} = \{ r_{ui} \} \in \mathbf{R}^{nxm}$ represent how much user u likes item i. For our purposes $r_{ui} \in \{0,1\}$, since a user either listened to a song or not. The goal is to predict a set of τ items $I_u \subset I$ that user u will like the most (disjoint from items already rated by u).

For MBCF , we attempted to reproduce the result of Aiolli's two strategies. The first, known as user similarity-based recommendation, recommends items to a user based on items rated by similar users. The second, known as item similarity-based recommendation, recommends items based on items already rated by the user. We define the similarity between users **u** and **v** by using a modified cosine similarity of their set of items with an additional parameter $\alpha \in [0,1]$ to tune. Likewise, we define the similarity between two items as i and j as the cosine similarity of the users who have rated them.[1]

$$w_{uv} = \frac{|\mathcal{I}(u) \cap \mathcal{I}(v)|}{|\mathcal{I}(u)|^{\alpha}|\mathcal{I}(v)|^{1-\alpha}} \quad w_{ij} = \frac{|\mathcal{U}(i) \cap \mathcal{U}(j)|}{|\mathcal{U}(i)|^{\alpha}|\mathcal{U}(j)|^{1-\alpha}}.$$

Then, we define a scoring function that aggregates the weights for all items or users. Where q is a parameter that enforces the locality of our scores.[1]

$$f^{U}_{ui} = \sum_{v \in U(i)} (w_{uv})^{q} \quad f^{I}_{ui} = \sum_{j \in I(u)} (w_{ij})^{q}$$

Given our limitations in computing power, we selected a subset of 100 users to recommend 50 songs. The results are included in section 3. In order to evaluate our results we used the same metric as described in the challenge guidelines: mean average precision truncated at 100 songs[1].

1. For more information on mAP consult section 4.2 from [2].

2.2 K-means Clustering

Our implementation of K-means clustering used a max of 10 features to cluster the data: Table 1 provides an example our features and the corresponding data that each feature had.

TABLE1

Features	Value
Artist Popularity	0.4933483788
Danceability	0.0
Duration	186.48771
Energy	0.0
Loudness	2
mode	-6.874
Key	1
Song popularity	0.2279329367
tempo	127.034
time signature	5
Year	1995

We also clustered based on just some combination of features that we estimated to be useful, such a Artist and Song Popularity, Tempo and Danceability, Key and Energy. We prioritized vectors with smaller dimensions, as they ran faster and usually provided more optimum results. For each song the user had listened to, we picked a song in the same cluster and suggested it to the user.

To simplify the clustering process, we utilized the open library SciPy[3], that allowed us to call an already optimized KM algorithm. We were allowed to specify the number of iterations to run KM.

We determined the ***k*** number of clusters based on the approximation:

$$k=\sqrt{\frac{n}{2}}$$

where n is the number of songs to cluster.

2.3 Optimizations

The data is delivered through an HDF5 format, a highly compressed form of data that is then processed using pyTables[6]. However, the pyTables interface increases the latency of our algorithms. To rapidly adjust parameters and enforce correctness in our algorithms, we were forced to cache intermediary results to text files that we could subsequently read off and load into RAM. As a simple comparison loading the 1M songs from the pyTable takes around 45 minutes, reading an already processed file with this information takes a couple of minutes.

3 RESULTS

3.0 Baseline results

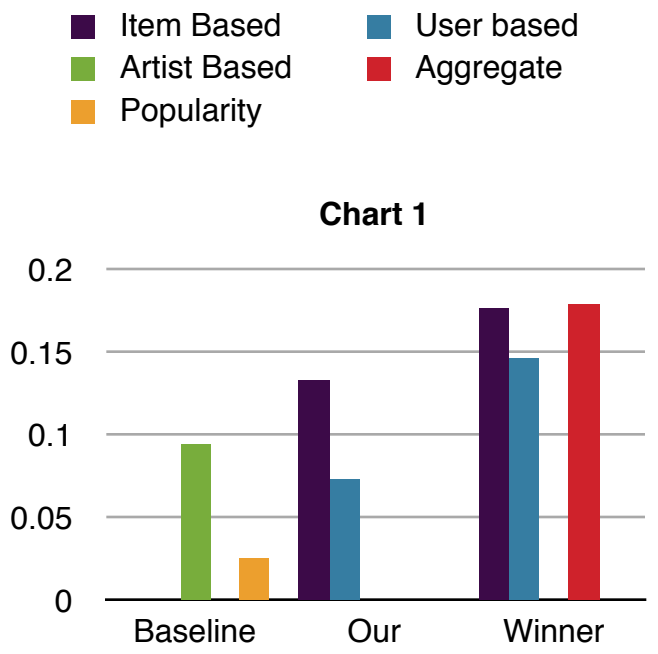
The baseline for the challenge using some metrics were as follows:

	mAP
Popularity	0.026
Same artist	**0.095**
BPR-MF	0.031

The challenge winner obtained an mAp of 0.17712.

3.1 MBCF

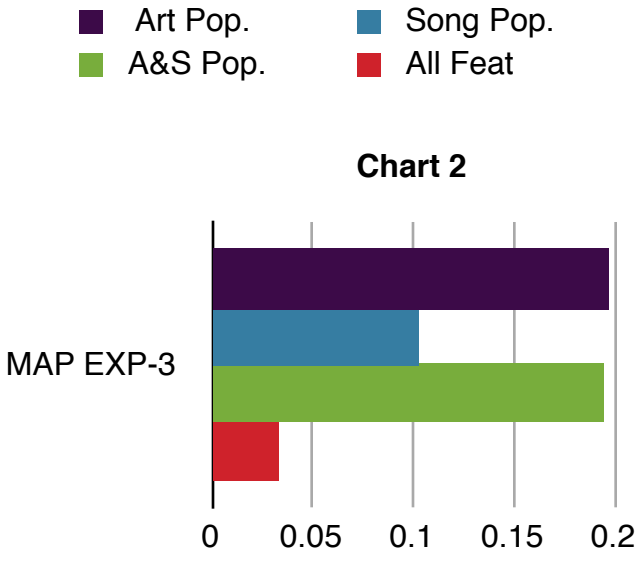
Our collaborative-based filtering approach performed better than any of the metrics used in the baseline: 0.133769 item-based (IB), 0.073899 (UB). Again, because of computation limitations we consulted external resources [1] to define the parameters of our scoring functions rather than testing them. As our testing data we used recently published data that contains the same users, but is composed by two files: a hidden part of the listening history and a visible part. The MAP for this data are shown on **Chart 1**.



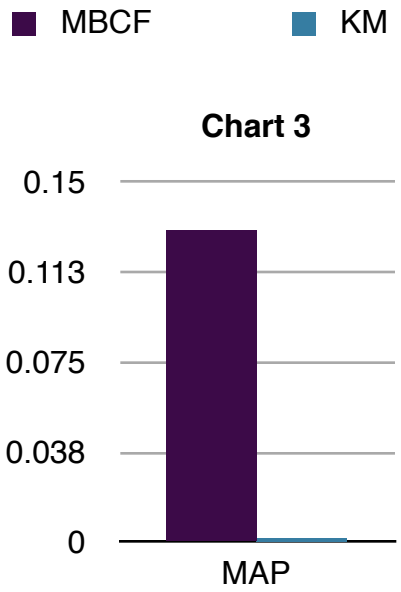
The winner of the competition aggregated both his results from IB and UB to achieve a maximum of 17.896.

3.2 KM

Our KM approach preformed worse than any other metric. We experimented with different feature vectors to attempt to optimize our results. The results of the top metrics are displayed on **Chart 2.**



All of these MAP values are scaled down by 3 orders of magnitude. The optimal vector for clustering is simply the artist popularity. A comparison between both KM and MBFC is provided in **Chart 3**.



4 DISCUSSION

4.1 MFBC

It was surprising to see positive results given that we did not make use of the available million song data. We concluded, that it might be enough to explore relationships between users and items in order to predict songs. It was also interesting to compare our user-based and item-based approaches. Given that our item-based approach fared better, it is tempting to conclude that items carry more implicit information in their intertwining.

4.2 KM

The optimum values for our KM was to when we included the artist popularity, as it yielded the best results. Counterintuitively, utilizing more features does not improve clustering, in fact, it makes our results worse. Given the features of a song, the most important factor in determining what songs the users will listen to next is the artist popularity even more important than songs with the same popularity.

A possible extension to this approach is to try and cluster users together, attempting to recommend to a user what others in the same cluster listened to.

4.3 MFBC vs KM

Clearly, MBFC outperforms K-means by almost two orders of magnitude in terms of MAP. However, this is

counter intuitive to think this way as one would believe that the features of the songs are what define and thus predict the musical interests of a person, not simply the connections between the songs. We are certain that there are better ways of clustering this data, but it seems that predicting the patters among listening histories is a better approach. We speculate that the reason behind this is because the patterns among the song data are so complex that more elaborate algorithms are needed that the ones employed in this study. However, if we abstract this pattern recognition abilities to the human mind and base our recommendation on human patterns we will achieve better results. We can even think of users themselves as hyper-intelligent clusters of data indicating relevant songs together, thus clustering users to create meta clusters could be a viable approach. However, we wouldn't cluster users based on their listening history (since that is defeating the purpose), we would try and cluster users based on *other* data such as age, gender, country of origin and even results to personality tests (such as the Myers-Briggs) to then recommend songs this way.
Finally, we have to stress the computational power needed to accomplish all of these tasks. None of these tasks are embarrassingly parallel in nature, and thus modern multicore processors are rendered useless against these tasks. In order create a system with very good results both more processing power is needed, as well as a redefinition of some algorithms that allow them to be concurrently computed.

5 CONCLUSION

The MSDC is a challenge that outlines the problems that many industries that depend on recommendation face today, such as e-commerce, news, entertainment and even online dating. Furthermore, with the rapid creation of data, the need to filter through and extract the context relevant content is increasingly becoming a complex and fruitful issue. Hopefully, research will continue to blossom from the private sector to enhance the experience of users in the online realm.

6 REFERENCES

[1] Aiolli, Fabio. 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. 2012.

[2] McFee et al. The Million Song Dataset Challenge. http://www.columbia.edu/~tb2332/Papers/admire12.pdf. 2010.

[3] SciPy Python Library. www.scipy.org.

[4] The Milllion Song Dataset Challenge. https://www.kaggle.com/c/msdchallenge/data. 2012.

[5] The Million Song Dataset. http://labrosa.ee.columbia.edu/millionsong/. 2010.

[6] PyTables Python Library. http://www.pytables.org/moin.