Music Recommendation System: Offline Evaluation of Learning Methodologies

(Based on Million Song Dataset Challenge by Kaggle)

Aashish Thite Prakhar Panwaria Shishir Prasad
Computer Sciences Department, University of Wisconsin-Madison
{aashish, prakhar, skprasad}@cs.wisc.edu

***Abstract* —** We designed, implemented and analyzed a music recommendation system for our course project. Using the dataset provided by Kaggle [1] for their Million Song Dataset Challenge [2], we have analyzed various state-of-the-art techniques which can be used to build a music recommendation system. In this paper, we focus on describing different learning algorithms, which we employed in providing music recommendations. Apart from doing offline evaluations and analysis of different solutions, we also describe our experiences and learnings from building a prototype music recommendation system. Our results suggest that ensemble methods applied with user-based collaborative filtering work better than other methodologies for the chosen dataset in generating high quality recommendations for the music lovers.

***Keywords* — recommendation systems, music, collaborative filtering, million song dataset**

1. INTRODUCTION

Currently, there are many music streaming services, like Pandora [3], Spotify [4], Rdio [5] and last.fm [6], which are working on building high-precision commercial music recommendation systems. These companies generate revenue by helping their customers discover relevant music and charging them for the quality of their recommendation service. Thus, there is a strong thriving market for good music recommendation systems.

One thing to note is that all the above mentioned applications use different proprietary techniques to recommend the most relevant music to their customers. For instance, last.fm uses a variant of user-based collaborative filtering [7] where they consider similarity between users based on their shared interests in music. Contrary to that, Pandora uses the similarity between the properties of songs or artists [8] to make appropriate recommendations. Thus, it is an open-ended problem with various possible solutions, and newer techniques to improve precision are being explored in both academia [23] and industry.

It is important to realize that recommending music is a harder problem than other forms of recommendations, for e.g. recommending items on an e-commerce website like Amazon.com. This is because music recommendation should also include the personal context of a user. In e-commerce space, if two items are being frequently bought together, they can be recommended to a user if she already bought one of them before. But, two songs being frequently listened together doesn't imply that a new user would like the other song if she has already listened to one of the songs. While recommending music, it is hard to predict the interests of a user. It may depend on the acoustic properties of a song or her favorite artists. A user may also like songs listened by her friends with similar taste in music

In this paper, we aim to closely analyze various algorithms which can be used to model a novel music recommendation system. As a primary baseline algorithm, instead of just giving random song recommendations, we chose to give overall top N popular songs to every user, where popularity of a song is measured by the number of users who have listened to that song in past. We have explored a diverse set of learning methodologies, for e.g.,

- Instance-based learning algorithms (k-NN)
- Bayesian networks (Naive Bayes)
- Collaborative filtering approaches (both item-based and user-based)
- Ensemble methods (Bootstrap Aggregation) over the aforementioned algorithms.

Before starting with implementation of these algorithms, we made the following hypotheses based on our intuition:

- Collaborative filtering (further referred as 'CF') methods, which are the most popular recommendation algorithms, should work better than other learning methods.

- Ensemble methods should improve performance of a learning algorithm.
- Both item-based and user-based similarity metrics should perform almost equally well for same dataset.

From our experiments, following are some important observations:

- The precision of our results was low in general. This can be attributed to the fact that we ran our experiments on a subset of the original data. It might also be due to the choice of sub-optimal or irrelevant features in the algorithms.
- The accuracy of a specific algorithm depended on the dataset used. For example, for our chosen dataset user similarity metrics based algorithms fared better than the song similarity metrics based algorithms.
- Using ensemble methods for recommendation systems indeed performed better than the corresponding original state-of-the-art algorithms.

1.1 Outline

In the following section, we describe the dataset provided by Kaggle and how we pre-process it to facilitate its consumption by our algorithms. Section 3 describes different learning methodologies and approaches used in our experiments. We evaluate these algorithms offline and analyze the obtained results in Section 4. In Section 5, we describe the related work being done in modeling music recommendation systems. In Section 6, we describe the future work that can be done to further improve the accuracy of our algorithms. We describe our conclusions and our key takeaways from this project in Section 7.

2. DATASET DESCRIPTION

The Million Song Dataset (MSD) Challenge hosted by Kaggle [1], a platform for predictive modeling and analytics competitions, was used as the primary dataset for our experiments. The raw data consisted of listening history of a million users in `<user_id, song_id, play_count>` format.

2.1 Preprocessing Data (Files to Database)

Raw data from files was extracted and loaded into a local MySQL database for easier consumption and subsequent analysis, using simple Python scripts. The very long string-based user and song identifiers were converted to monotonically increasing integers to reduce both memory and CPU-processing. Table [1] shows the normalized schema of the database used for our project and some random examples.

user_id	song_id	play_count
1	1	3
1	2	6

[1a]: msd_data

kaggle_user_id	user_id
00000b722001882066dff9	1
00004fb90a86beb8bed1e9	2

[1b]: msd_users_map

kaggle_song _id	song_id
SOBQJJX12A6D4F7F01	1
SOUBEXV12AB01804A	2

[1c]: msd_songs_map

Table [1]: Database Schema of MSD

2.2 Low-level Dataset Description

The original MSD consists of 1 million users and songs. Processing such a large dataset is highly memory and CPU-intensive and requires a dedicated distributed system, for e.g. a Hadoop [20] cluster with 10 server machines. In the absence of such a powerful system, we opted to work on a random subset of the MSD. Table [2] shows the detailed summary of our randomly chosen subset of original data.

Total Users	Total Songs	Total Records	Avg. Recs/User	Avg. Recs/Song
10000	105042	479270	47.92	4.56

Table[2]: Statistical summary of chosen subset

3. ALGORITHMS

We developed a few baseline algorithms which would serve as the benchmark for evaluating the effectiveness of our main algorithms. Table [3] describes a summary of all the algorithms being used in our case.

3.1 Baseline Algorithms

3.1.1 Top N Most Popular

This is a simple benchmark which works by computing the most popular songs i.e. the songs which have been listened by the maximum number of users. Such songs are then ranked in the decreasing order of popularity and top N songs are returned as recommendations.

Algorithm	Summary
Top N Most Popular	Return the top N most popular songs
k-NN	Return the top N songs from K most similar users
Naive Bayes	Return top N probable songs conditioned on listening history
Item-based CF	Return the top N songs from all aggregated similar songs from song-similarity matrix
User-based CF	Return the top N songs from all the aggregated songs of similar users in user-similarity matrix
Bagging	Bagging approach with above learning algorithms

Table [3]: Summary of Algorithms

3.1.2 k-Nearest Neighbors

K-Nearest Neighbor algorithm was employed on the user listening history. For each test user, the set of closest K users in the training set were found. Each song was used to build a feature vector by weighing it according to the play count of that song for the user [9]. Closeness between the users is computed by determining the cosine distance between their feature vectors. The final recommendations were obtained by merging the songs from K-closest users, ranking them based on number of song repetitions across those users and returning the top N results.

3.1.3 Naive Bayes

It is our hypothesis that the conditional probability of a song (S) listened by a user, given the listening history (LH), would be a good measure to recommend a song to the user. We assume a song in the listening history is independent of other songs in the same set. Assuming this independence, we can construct a Naive Bayes network. We compute:

$$P(S|LH) \propto P(LH|S) \times P(S)$$
$$P'(S|LH) = P(LH|S) \times P(S)$$
$$= P(S1, S2, \ldots|S) \times P(S)$$
$$= P(S1|S)P(S2|S) \ldots\ldots \times P(S)$$
$$= (\prod_i P(Si|S)) \times P(S)$$

As mentioned in [10], there is a problem with using plain conditional probabilities. These conditional probabilities could be high not because the song to be recommended was similar to other songs in the listening history, but may be because it was popular. A less popular song will be more reliable in giving recommendations than the one which is among the most popular. Work done in [10] gives a way to address such a problem by normalizing for popularity. To do this, we divide the conditional probability by the frequency of the song. This frequency can be scaled by a parameter α. With α=1, we get same result as the plain conditional probability.

$$P'(S|LH) = \left(\prod_i P(Si|S)\right) \times P(S)/P(S)^{\beta}$$
$$= (\prod_i P(Si|S)) \times P(S)^{1-\beta}$$
$$= (\prod_i P(Si|S)) \times P(S)^{\alpha}$$

3.2 Main Algorithms

3.2.1 Collaborative Filtering

We used memory-based CF algorithms which work by building an in-memory similarity matrix on the complete dataset prior to recommendations.

3.2.1.1 User-based CF

The intuition behind this algorithm is that similar users listen to similar songs. Thus, if we know user u is similar to user v, we can recommend v's listened songs to u. The main steps of the algorithm are:

- Build a user-to-user similarity matrix using cosine similarity.
- For any test user u, find the set of all train users similar to it from the user similarity matrix. Merge all the listened songs of the users in the training set as possible recommendations, excluding songs already listened by the test user. Then, rank all the songs in the set based on their aggregated scores and return the top N songs to the test user.

To determine the similarity between any two users u and v, we use a variant of cosine similarity with a weighing factor α [9], which ensures that all users are not weighted equally in cosine distance calculations. A user who listens to lot of songs does not add much information, so she should have a lower contribution in the final recommendations and vice-versa.

$$sim(u,v) = \frac{\#common\ items(u,v)}{\#items(u)^{\alpha}\ .\ \#items(v)^{1-\alpha}}\ , \alpha \in [0,1)$$

To calculate the score of each train song c to be recommended for a test user u, we sum the user-similarity score for each user v in the set of train users V who has listened to this train song c. To further emphasize the impact of high weights and minimize for low weights, we also added a normalization coefficient γ[9].

$$w_c = \sum_{v \in V} (sim(u,v))^{\gamma}$$

3.2.1.2 Item-based CF

The intuition behind this algorithm is that a user is most likely to listen to a song which is similar to songs which she has already listened. Thus, if we know that songs i and j are similar and the test user has already listened to song i, we can recommend song j to her. The main steps of the algorithm are as follows:

- Build a song-to-song similarity matrix using cosine similarity.
- For every test user u and for every training song c not listened by u, find how closely c resembles to the listening history of u by summing the similarity score for c with each song i listened by u. The final recommendations consist of the top N train songs with highest aggregated sums.

To determine the similarity score between songs i and j, we again use cosine similarity [9].

$$sim(i,j) = \frac{\#common\ users(i,j)}{\sqrt{\#items(i).\#items(j)}}$$

To calculate the score of each training song c (not listened by the test user) to be considered for recommendation for a test user u, we sum the song-similarity of c with all the songs in the listening history H of the test user. The top N songs with highest scores are returned as recommendations for the test user u.

$$w_c = \sum_{i \in H} sim(i,c)$$

3.2.2 Ensemble Methods

In general, ensemble methods [22] are learning methodologies, which generate a set of classifiers on uncorrelated datasets and then classify test instances based on the predictions made by those classifiers. Ensemble approaches like bagging, boosting, and random forests are generally used for classification tasks. Since our algorithms generate a set of recommendations rather than performing classifications, we have used a variant of bagging, where instead of getting a plurality vote among the generated models, we do smart aggregation of the recommendations made by different models and return the top N most relevant songs for a particular test user.

3.2.2.1 Modified Bootstrap Aggregation

Bootstrap Aggregation (or Bagging) approach is known for reducing the variance of the learning algorithms which ultimately helps in avoiding overfitting. Since, ensemble methods are independent of the learning algorithms used, we have used our modified approach with all the base algorithms, i.e. k-NN, Naive Bayes, and CF approaches.

To use bagging, the algorithm is given a learning method along with the TRAIN dataset as well as the TEST_VISIBLE dataset (see Section 4.1 for more details). First, we generate datasets by randomly drawing instances from the original dataset with replacement. For all randomly generated datasets, we build models of the learning algorithms. For each model, we then get recommendations (list of songs with their scores) for the users in the TEST_VISIBLE dataset. After collecting the list of song recommendations from all the models, we aggregate the songs in a single data structure, while adding up the score of a song if it is present in multiple recommendations, and finally return the top N songs in decreasing order of their scores.

Bagging generally works well with unstable learning algorithms, which are sensitive to the dataset, for e.g., k-NN and CF approaches. In Section 4, we'll observe the performance of Bagging used with different learning algorithms.

4. EVALUATION

4.1 General Methodology

The general evaluation methodology that we used for each algorithm is as follows:

1. Load the MSD from MySQL database.
2. Perform M-fold Cross-Validation (mostly with M=20)
3. Run the algorithm 'M' number of times. For each run:
   a. Generate three datasets from the cross-validation folds
      - **TRAIN** dataset: Complete listening history of existing users.
      - **TEST_VISIBLE** dataset: Partially known listening history of test users. It is used by algorithms to make recommendations.
      - **TEST_HIDDEN** dataset: Remaining partial listening history of test users, which needs to be predicted by algorithms.
   b. Generate learning model using TRAIN dataset.
   c. Generate recommendations using TEST_VISIBLE dataset and the learned model. If algorithm is not able to achieve N recommendations, bootstrap the results with the top N overall popular songs.
   d. Evaluate using the TEST_HIDDEN dataset and compute precision for this run.
4. Return average precision for this algorithm across all the runs.

4.2 Experimental Setup

All the experiments were run on a single machine in the mumble lab in CS department, whose configuration was 8GB RAM, Intel(R) Core 2 Duo @2.66 GHZ.

4.3 Accuracy Metrics

The metric we chose for computing the accuracy of our algorithms was the precision value of song recommendations. Specifically, it was equal to the number of matched songs in the TEST_HIDDEN dataset for a user (true positives) divided by the total number of songs recommended for the user (predicted positives). Our reason for choosing precision as the accuracy metric is two-fold. First, this is the metric used for determining the accuracy on Kaggle for MSD challenge [2]. Second, precision is much more important than recall in a recommendation system because false positives can lead to a poor user experience.

4.4 Results

In this section, we evaluate different learning algorithms mentioned in Table [3].

4.4.1 k-NN: Finding Optimal K

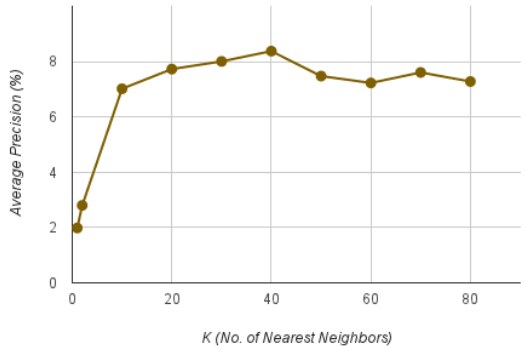
In this experiment, we are trying to find an optimal value of `K` in k-nearest neighbors algorithm which gives best performance on the TEST_HIDDEN dataset. We observed that the accuracy first increases up to a certain point (when `K` = 40) and then decreases. With the value of `K` being low, the reason of getting a low accuracy may be susceptibility of the algorithm to the noise in the training data. And, the reason of getting a low accuracy with the value of `K` being too high may be inclusion of recommendations from the less similar users. We can see from Figure [1], that the optimal value of `K` for our dataset comes to be around 40.

4.4.2 User-Based Collaborative Filtering: Finding Coefficients

Here, we try to find the variation of precision for user-based CF algorithm as we vary α across different values of γ and vice-versa [9]. From our experiments as shown in Figure[2, 3], we observed that the maximum precision is achieved for α=0.8 and γ=8.0.

4.4.3 Naive Bayes: Finding Model Parameters

Here, we try to find an optimum value for the value α, parameter used for normalizing the popularity of a song to be recommended. We conducted experiments with different values of α varying from 0.0 to 1.0. As indicated in Figure [4], we observed that the maximum precision was achieved for α =1.



Figure[1]: Avg. Precision for different values of K

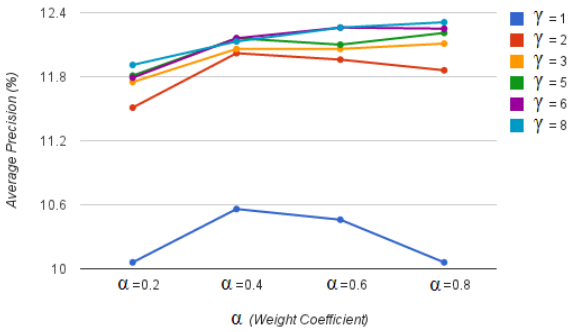


Figure [2]: Avg. Precision vs α for User-based CF

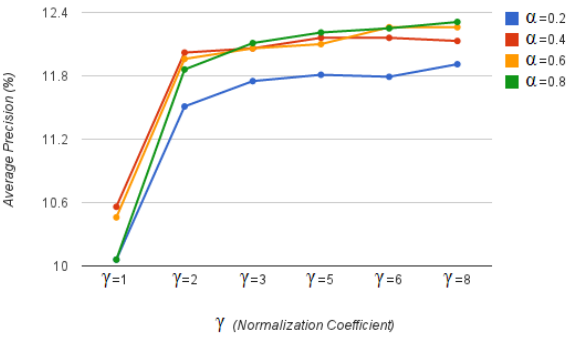


Figure [3]: Avg. Precision vs γ for User-based CF

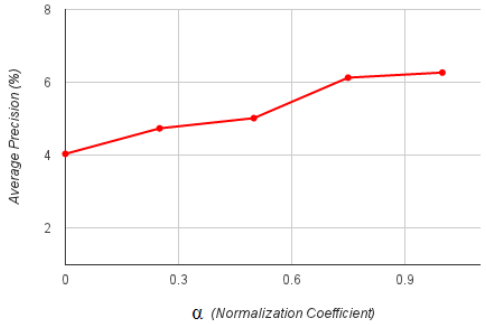


Figure [4]: Avg. Precision vs α for Naive Bayes

4.4.4 Impact of Bagging

In this experiment, we used 20-fold cross-validation to generate 20 different datasets, and ran our experiments to get scatter plots showing the impact of bagging on different algorithms. In general, we found that accuracy of unstable learning algorithms improves with bagging.

4.4.4.1 Bagging with k-NN

Figure[5] shows a scatter plot between k-NN algorithms (with K = 40) with and without the application of bagging. We can see that k-NN performed slightly better when bagging is applied, which may be because k-NN is an unstable algorithm, thus, applying bagging might have resulted in different uncorrelated models, reducing overall variance, and, hence, overfitting.

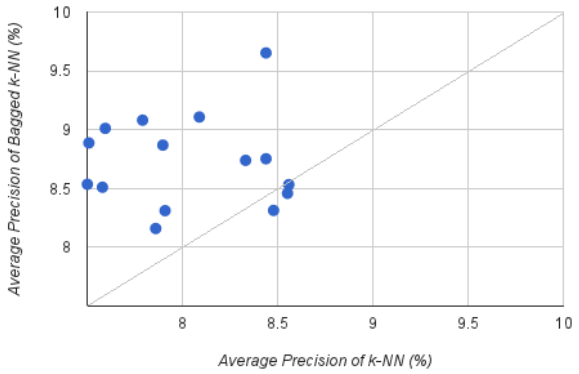


Figure [5]: Impact of Bagging on k-NN

4.4.4.2 Bagging with Naive Bayes

We also applied bagging with Naive Bayes (with α=1), but as we can see from the scatter plot in Figure [6], algorithm did not perform well with bagging. This may be because model parameters of Naive Bayes exhibit little variation across different datasets generated by Bagging, which might not have helped in improving the accuracy.

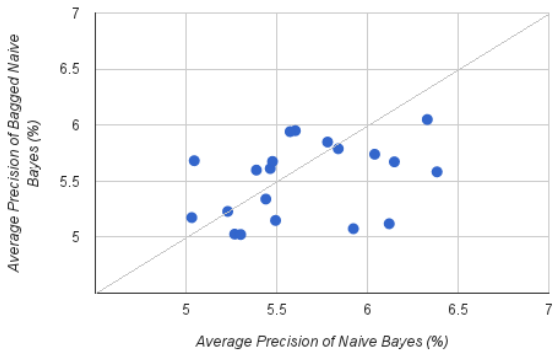


Figure [6]: Impact of Bagging on Naive Bayes

4.4.4.3 Bagging with Collaborative Filtering

As shown in Figure [7] and [8], we find that both the algorithms (for user-based CF, α=0.8 and γ=8.0) perform better with bagging. Its reason may be same as that for k-NN, as both the CF algorithms are sensitive to data.

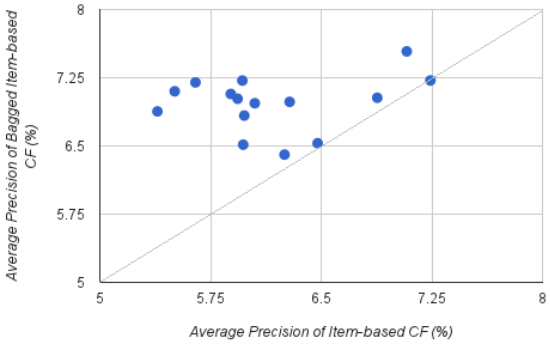


Figure [7]: Impact of Bagging on Item-based CF

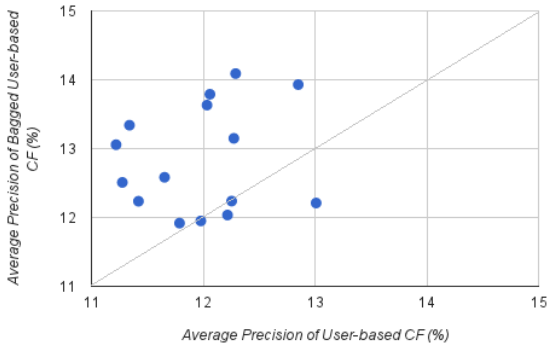


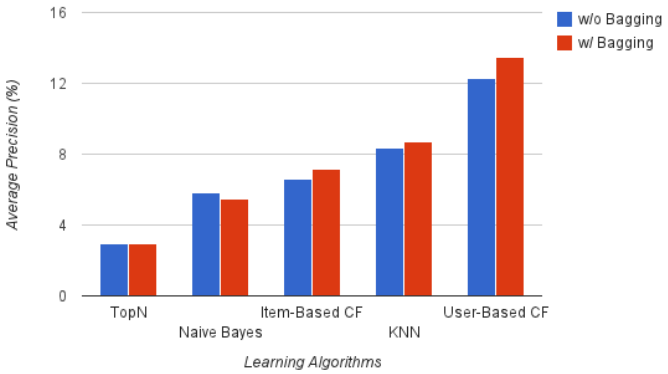
Figure [8]: Impact of Bagging on User-based CF

4.4.5 Comparison of Algorithms

In this experiment, we chose optimal parameters for all the algorithms (K=40 for k-NN, α=0.8 and γ=8.0 for user-based CF, and α=1 for Naive Bayes), and using the same dataset generated from 20-fold cross-validation, we evaluated average accuracy of all the algorithms as shown in Figure [9]. Following are results of this experiment:

- Comparing between our baseline algorithms, k-NN performed better than TopN algorithm. This is probably because k-NN learner uses similarity between the users as its criteria, whereas TopN algorithm just recommends N-most popular songs which a user might not like.
- Figure [9] also shows that algorithms using user-based similarity metrics have performed better than item-based similarity metrics. For instance, both user-based CF and k-NN showed better performance than Naive Bayes and Item-based CF.
- Finally, we observed that user-based CF was more accurate than k-NN. Its reason may be that in k-NN,

we are considering songs from only 40 most-similar users giving equal weights to each of them irrespective of their distance. On the contrary, in user-based CF, we consider all the users in training dataset and give their songs a weight corresponding to their distance from the test user in question.



Figure[9]: Comparison of Algorithms

4.5 Discussion

Analysis of our observed results has given us a deeper understanding of our proposed hypotheses and the actual results obtained. Here is the comparison of our proposed and actual results:

1. We hypothesized that item-based CF should work equally well as the user-based CF algorithm. From the results, we see that user-based CF performed as per our expectations, but item-based CF algorithm underperformed for our dataset. This is because the song-similarity matrix was very sparse for our chosen dataset. On an average, 1 user listened to 47 songs, whereas 1 song was listened by only 4 users. From this, we infer that item-based similarity is well suited for the case where number of items is much lesser than the number of users. For example, Pandora has 0.8 million songs and 80 million users [8]. Thus, their song-similarity matrix would generally not be sparse. This is confirmed by the fact that Pandora uses content-based recommender systems [8].
2. Bagging indeed performed better than original models.

Other lessons that we learned from our experiments are:

1. The success of the chosen algorithm depends a lot on the underlying dataset. This indicates that we should ideally use a hybrid approach where we choose different underlying algorithms based on nature of the data. For example, if the dataset has fewer songs in proportion to the dataset size, item-based similarity metrics should be used, otherwise user-based similarity metrics should be preferred.
2. Music recommendation in general is a tough problem. Our best algorithm could only manage 13.37% precision. We need to provide additional signals to our algorithms to simulate user's interests. Additional metadata, like song's tempo or lyrics, and user's preferences, like genre, can help further improve the algorithms' accuracy.
3. Recommendation algorithms are computationally intensive and are well suited to be run on distributed systems for fast parallel processing.
4. The precision accuracy is dependent on the size of data available for learning. This is commonly known as Cold Start problem [11], where the initial recommendations are poor because of insufficient user preference data. We believe our algorithms would give better accuracy when run on the complete MSD on a distributed cluster.

5. RELATED WORK

Music recommender systems are currently a hot research topic in both academia and industry. Amazon has been at the forefront of item-based CF [17] and relevant algorithms like Slope-one [12] have been very effective. In our project, we experimented with the listening history of a user and the corresponding play counts as the signals to our algorithms. But in real-world systems, various other signals related to song metadata, as mentioned before, are employed to further augment the basic algorithms.

Research in this field also suggests that hybrid approaches which combine multiple models generally outperform any individual method [13]. For example, Hybrid CF which combines both Memory-based CF and Model-based CF has been shown to perform much better. But hybrid algorithms are expensive and difficult to implement. The current research focus has been to reduce the runtime complexity of hybrid algorithms, to make them feasible for real-world systems [14].

Another major area of focus has been the parallelization of recommendation algorithms. Approaches like CF create a huge similarity matrix that cannot fit in memory on a single machine, when dealing with real-world workloads. For example, Spotify currently has 10 million users, and thus, would need a user-similarity matrix of size $10^6 x 10^6$ to implement user-based CF [15]. Thus, both the data computation and processing has to be parallelized using distributed system clusters (e.g. Hadoop) to enable such large-scale

computation. Active research is happening in both academia [16] and industry on this front.

6. FUTURE WORK

In this course project, our goal was to implement state-of-the art recommendation techniques and some new algorithms learnt in our course like Naive Bayes, ensemble methods etc. In the future we plan to extend our work in the following directions:

1. Add song metadata from other data sources like MusixMatch [18], last.fm [19] etc., as additional signals to the recommendation algorithms and test its accuracy. We believe including the context of the song into our algorithms would boost the test data accuracy.
2. Run the algorithms on a distributed system, like Hadoop [20] or Condor [21], to parallelize the computation, decrease the runtime and leverage distributed memory to run the complete MSD. We believe our accuracy results would improve further when run with the complete MSD.
3. Test additional ensemble methods, like AdaBoost [22], and compare their accuracy with the above recommendation algorithms.

7. CONCLUSION

Our course project has given us a keen insight into the recommendation systems for music domain. The low precision of 13.37% has made us realize that predicting songs for a listener is a fundamentally tough problem and is dependent on many intrinsic (song's metadata like tempo, lyrics etc.) and extrinsic factors (user's personality etc.). We believe that including additional features like song's metadata would help in improving the overall accuracy. We also learnt that recommendation algorithms are highly CPU and memory-intensive and are well suited to run on large distributed systems for faster processing.

Our results indicate that ensemble methods tend to give better precision as compared to the original algorithms because of their model averaging approach. Also, the success or failure of various algorithms depends on the dataset involved. Our dataset had a very sparse song similarity matrix, which led to lower precision for song-based metrics as compared to user-based metrics.

REFERENCES

[1] Kaggle. http://www.kaggle.com/.

[2] McFee, B. and Bertin-Mahieux, T. and Ellis, D. and Lanckriet, G. "The million song dataset challenge" *Proc. of the 4th International Workshop on Advances in Music Information Research (AdMIRe*), 2012

[3] Pandora.http://www.pandora.com/.

[4] Spotify. http://www.spotify.com/.

[5] Rdio. http://www.rdio.com/.

[6] last.fm. http://www.last.fm/.

[7] Collaborative Filtering (Source: Wikipedia). http://en.wikipedia.org/wiki/Collaborative_filtering

[8] Pandora Radio (Source: Wikipedia) http://en.wikipedia.org/wiki/Pandora_Radio

[9] Li, Y., Gupta, R., Nagasaki, Y., Zhang, T. "Million Song Dataset Recommendation Project Report". 2012. Unpublished Manuscript.

[10] Karypis, George. "Evaluation of item-based top-n recommendation algorithms."*Proceedings of the tenth international conference on Information and knowledge management*. ACM, 2001.

[11] Lam, Xuan Nhat. "Addressing cold-start problem in recommendation systems." *Proceedings of the 2nd international conference on Ubiquitous information management and communication*. ACM, 2008.

[12] Lemire, Daniel, and Anna Maclachlan. "Slope one predictors for online rating-based collaborative filtering." Society for Industrial Mathematics 5 (2005): 471-480.

[13] Song, Yading, Simon Dixon, and Marcus Pearce. "A Survey of Music Recommendation Systems and Future Perspectives." 9th International Symposium on Computer Music Modelling and Retrieval. June 2012.

[14] Das, Abhinandan S., et al. "Google news personalization: scalable online collaborative filtering." Proceedings of the 16th international conference on World Wide Web. ACM, 2007.

[15] Spotify: Music Recommendations Algorithm. http://www.slideshare.net/erikbern/collaborative-filtering-at-spotify-16182818

[16] Aiolli, Fabio. "A Preliminary Study on a Recommender System for the Million Songs Dataset Challenge". Unpublished Manuscript.

[17] Linden, G., Smith, B., & York, J. (2003). Amazon. com recommendations: Item-to-item collaborative filtering. *Internet Computing, IEEE*, *7*(1), 76-80.

[18] Musixmatch dataset. http://labrosa.ee.columbia.edu/millionsong/musixmatch

[19] last.fm dataset. http://labrosa.ee.columbia.edu/millionsong/lastfm

[20] Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." *Communications of the ACM* 51.1 (2008): 107-113.

[21] Raman, Rajesh, Miron Livny, and Marvin Solomon. "Matchmaking: Distributed resource management for high throughput computing." *High Performance Distributed Computing, 1998. Proceedings. The Seventh International Symposium on*. IEEE, 1998.

[22] Dietterich, Thomas G. "Ensemble methods in machine learning." *Multiple classifier systems*. Springer Berlin Heidelberg, 2000. 1-15.

[23] Sarwar, Badrul, et al. "Item-based collaborative filtering recommendation algorithms." *Proceedings of the 10th international conference on World Wide Web*. ACM, 2001.