Music Recommendation System

Himanshu Gupta
himanshu.gupta@colorado.edu
CSCI 5502, University of Colorado Boulder
Boulder, Colorado

Jaeyoung Oh
jaeyoung.oh@colorado.edu
CSCI 5502, University of Colorado Boulder
Boulder, Colorado

Youngjin Ko
youngjin.ko@colorado.edu
CSCI 5502, University of Colorado Boulder
Boulder, Colorado

Yutaka Urakami
yutaka.urakami@colorado.edu
CSCI 5502, University of Colorado Boulder
Boulder, Colorado

Figure 1: Obtained from https://www.dezyre.com/project-use-case/music-recommendation-challenge

ABSTRACT

Nowadays, there are various types of platforms (video, music, shopping, etc) that provide recommendation services for users. These recommendation systems not only help novice users who just started on the new platform but also enhance the experience for current users who have been using the platform for quite some time by analyzing past searching history and suggesting content that users are more likely to be interested in. In this project, we aim to make a recommendation system from 'Million Song Data' using tools that we learned in the Data-mining course with Prof. Qin Lv.

KEYWORDS

Data Mining, Data sets, Music Recommendation, Apriori, Collaborative Filtering

1 INTRODUCTION

For decades, thanks to the propagation of the network and connectivity, and popularization of smartphones, music platform subscribers have grown exponentially as can be seen in figure 29. Nowadays, the online music market has became a huge industry. Accordingly, the recommendation system also has become an essential part of it. By using the recommendation system, a customer can come across content that are tailored to his needs and can make a decision more easily. The better quality of recommendation will improve the user experience and the consumers will feel more satisfied with the service and this can lead to them subscribe for longer duration. Therefore, lots of media companies invest in developing excellent recommendation engine.

From 2006 to 2009, Netflix-sponsored competition offered a grand prize of 1,000,000 to the winning team. They provided a data set of over 100 million movie ratings, and the winning criteria was that the new recommendations were at least 10 percent more accurate than the predictions from their existing recommendation system. [13]

There has already been a lot of work done in the field of recommendation systems. Nevertheless, the ground theory of recommendation system is Data-mining. Therefore, we will recursively build the frame of recommendation service with Data-mining tools and apply to a subset of the massive data set 'Million Song Data'.

Through this project, we can figure out how accurate our recommendation system is, comparing with modern commercial recommendation systems and finding out the limitation of our methods. The huge scale of data, ability to mine data from it and solving a real life problem makes it interesting to our group. One of the major challenges is also the huge data set and developing anything using it is going to be computationally expensive and thus difficult. As the size of 'Million Song Data' is substantial, expected processing time is variable and depends on external factors such as the Capacity of RAM, processing size(32bits, 64bits). Also, people have been doing extensive research in this field for a long time now. Devising a system that can improve the existing state of the art recommendation techniques, and comparing results with other techniques, and understanding why one technique works better than the other is not straightforward and hopefully we will try to address that by the end of this project.

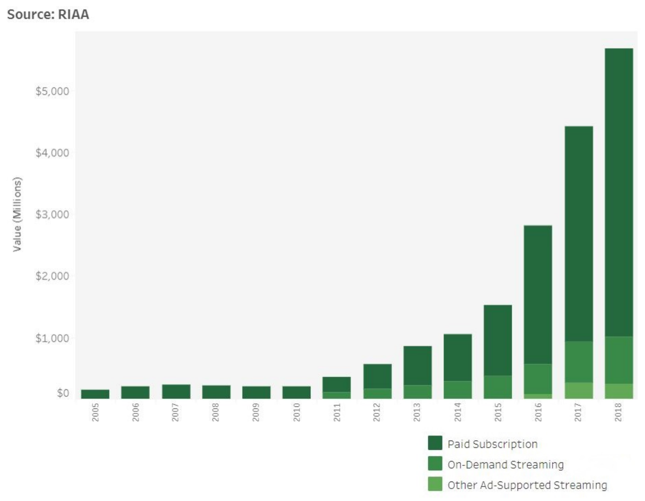


Figure 2: 2005 - 2018 U.S. Music streaming service growth (https://www.riaa.com/u-s-sales-database/).

2 RELATED WORK

Music is prevalent. However, the collection of music that music streaming services have these days is so huge that it is not easy to find and choose the music that a user wishes to listen to. Searching anything on such huge mount of data gives a lot of results, and it is hard to find the right one. This makes a recommendation system essential for finding out the type of music each user will prefer. Recommendation can be considered as an alternative to searching. Every music company provides a personalized playlist with its algorithm[5, 11]. The most frequently used algorithms are mentioned below briefly: [10, 12].

- `Collaborative filtering (CF)` :Collaborate filtering depends on analyzing and gathering a lot of information about user behavior, activities, or preferences, and predicting what users will like based on similarities with other users. One important advantage of this approach is that it does not rely on understanding the meaning of content. Hence, you can recommend multiple items like movies without accurately understanding the item itself. Apple relies on this algorithm to recommend song in iTunes.
- `Context-based filtering` :This algorithm relies on a description of the item and a profile of the user. In the Content-based recommendation system, keywords are used to describe an item, and user profiles define the type that the user likes. This algorithm recommends similar items that users used to like or what they are currently seeing. Specifically, various candidate items are compared with items rated by the user, and the system recommended best-matching items. Internet radio company Pandora relies on this system to make personalized internet radio station.
- `Hybrid approach` :The hybrid approach is a mixture of CF and content-based filtering, which may be more effective in some situations. This method can overcome the common problems in recommendation systems such as cold start and sparsity problems.

There are several scholarly works of literature for music recommendation. They provide novel approaches to improve recommendation accuracy. In the paper[3], they are trying to build a personalized internet radio stream with a new probabilistic collaborative filtering model. They used the playlist of thousands of internet radio stations, which include millions of plays and hundreds of thousands of tracks and artists. They also did cross-source validation. The model trained on the internet radio data was transferred to the Yahoo behavior prediction successfully. Another beneficial approach is matrix factorization.

The paper[10] relies on the method. They improve Yahoo music recommendation accuracy. They suggest the matrix factorization method with temporal analysis of user ratings, and item popularity trends with Yahoo music data set, which include a million users, 600 thousand musical items, and 250 million ratings for a decade. This approach achieves the integration of taxonomy information of items and different music ratings.

The paper[6] suggests the Latent Markov Embedding approach to generate an automatic playlist. The data used in the article are radio playlist from yes.com, which has hundreds of stations in the U.S. They optimized the problem with the regulated maximum-likelihood embedding of Markov chains and solved efficiently.

The last paper[8] shows a context-aware music recommendation system. The data is human-compiled music playlists, which include 7,051 playlists and 21,783 songs. They provide a topic modeling with the latent topics generated from the most frequent tagged songs. The method has a more accurate recommendation result than a conventional system that relies on CF or content-based filtering.

3 DATASET(S)

We plan to use two existing datasets: Kaggle's competition, Million Song Dataset Challenge[9] and Million Song Dataset. [14]

The data set in the Kaggle competition comprises of : 1) the 386,213-song IDs; 2) the 110,000-user IDs; 3) the listening histories of the 110,000 users; 4) the mapping from song IDs to the track IDs.

The Million Song Dataset uses the same song and track IDs and has detailed information about those tracks. It includes 1) the mapping from the song IDs to the song title and artist name; 2) the mapping from the track IDs to the track descriptions (artist name, danceability, duration, energy measure, key, loudness, year, tempo, and so on); 3) the mapping from the artist IDs to his/her locations (latitude, longitude, city, and state)

4 PROPOSED WORK

In this section we discuss about the strategies that we wished to implement at the start of the project and the motivation behind them.

4.1 Apriori Algorithm

Apriori algorithm [15] is used for mining frequent item sets and determining strong association rules over relational databases. It proceeds by identifying all the frequent 1-item sets in the database. It then generates all the possible 2-item set candidates by combining these 1-item sets and choose those item sets that appear sufficiently often in the database. Apriori uses a "bottom up" approach, where frequent subsets are extended one item at a time. The algorithm terminates when no further successful extensions are possible. The frequent item sets determined by Apriori can be used to determine association rules which can be used to generate recommendations or determine general trends in the database. Amazon uses this algorithm to recommend users similar items for purchasing and it has worked really well for them. The same Apriori algorithm can also be used to determine frequent song-sets and songs from the same set can be used for recommendations to all the users. We use the algorithm to find frequent itemsets. Our dataset provides us the listening history of users in this format- user ID, song ID, number of listenings.

We perform data pre-processing and convert this into the desired format, which is

UserID - song ID$_1$, $songID_2, \ldots songID_N$

for all the users and then apply Apriori to it.

This approach doesn't take into consideration the number of times a particular user listened to any song. We plan to use this information in our algorithm for better recommendations and try to solve the ranking problem at the same time.

One other approach is to find frequent user-sets. The data can be pre-processed and converted into the following format,

SongID - User ID$_1$, $UserID_2, \ldots UserID_N$

for all the songs and then apply Apriori to it.

This allows us to find similar users. Songs listened by a user are then recommended to similar users.

Similar work [4] using Apriori algorithm has already been done to solve the cold start problem.

Here is an example of this algorithm. Let the data set be Table 1 and the minimum support be 0.5. After generating candidates from the first scan, 5 song sets (Table 2) are found. D was pruned because the support is less than the minimum support. After getting candidates from the second scan, 6 song sets (Table 3) are found. The two sets, {A, B}, {A, E}, are pruned because of the support. After generating candidates from the third scan, 1 song (Table 4) set is found. There are no more frequent song sets that can be generated, so the process is stopped.

4.2 FP-Growth

FP-Growth is an algorithm for extracting frequent itemsets with applications in association rule learning that emerged as a popular alternative to the established Apriori algorithm. This Algorithm was proposed by Han in, and became famous for its faster performance than other known methods. Its basic concept of finding frequent pattern is almost similar to that of Apriori algorithm. However, FP-Growth is different when it come to generating frequent itemsets. FP-Growth uses FP-tree to overcome the problems of candidate generation and multiple scans of Apriori algorithm. As a result, this algorithm performs better for large data sets. As our project dataset is huge, we also use FP-Growth to extract frequent patterns.

Table 1: Data set

User	Song Set
1	A, C, D
2	B, C, E
3	A, B, C, E
4	B, E

Table 2: Candidates from the first scan

Song Set	Support
A	0.50
B	0.75
C	0.75
D	0.25
E	0.75

Table 3: Candidates from the second scan

Song Set	Support
A, B	0.25
A, C	0.50
A, E	0.25
B, C	0.50
B, E	0.75
C, E	0.50

Table 4: Candidate from third scan

Song Set	Support
B, C, E	0.50

4.3 Collaborative Filtering

The most basic models for recommendations systems are collaborative filtering models[7] which are based on assumption that people like things similar to other things they like, and things that are liked by other people with similar taste.

We plan to use the matrix factorization technique to generate user vector embeddings and song vector embeddings. For any given song, similar songs are then computed using cosine distance. Similarly, for any given user, similar users can be figured out using cosine distance.

We know that collaborative filtering is the favored method throughout related research. We decide to use collaborative filtering as a comparison result. And, the winner of the million song dataset employed memory-based collaborative filtering because the method is an efficient way to deal with the large dataset. We adapted the code[1] and approach from the winner of the million song dataset challenge for comparison purposes. [2].

The memory-based CF uses the entire user-item matrix to predict the recommended songs. The system builds a recommendation list based on the similarity between the existing and the new user. This one is a user-based recommendation. The other approach is an item-based recommendation. The system computes the similarity of songs and makes the recommendation list. They employed the simple weighted sum method to collect the information[2]. The popular scoring method is cosine similarity, and it is good to be symmetric. However, we already know the favor of the user, and we need to calculate the support of a user. It is a conditional probability. Hence, they used the conditional probability to get the similarity score[1].

4.4 Clustering

Clusters of similar songs can be generated using various clustering algorithms such as k-means algorithm or DB Scan algorithm. The song vector embeddings obtained using Matrix Vectorization are used to generate these song clusters. For any given songs, songs from the same cluster are then recommended to a user.

4.5 Feasibility

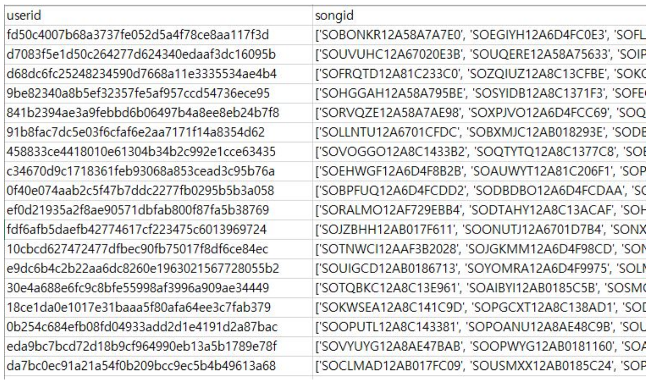
The work requires a lot of literature review from our side and understanding the best practices for our proposed work. Since we are graduate students, I believe we will be able to do this. The subtask of figuring out similar songs using matrix factorization might run into problems because of insufficient computation power of our systems. We might have to find an alternative in that case.

5 COMPLETED TASKS

This section covers the lists of tasks that we finished from our proposed work list and some additional tasks that we performed for the completion of the project.

5.1 Data Preprocessing

- Information about the data: There are three major files that we are using, ”*kaggle_songs.txt*” (has mappings of *song_ids* to numbers), "*kaggle_users.txt*" (has mappings of uids to numbers) and "*kaggle_visible_evaluation_triplets.txt*" (has triplets in each row which tells how many times a user listened to a particular song).
- Approach: To utilize this data for our Apriori and FP growth algorithm, we had to process the row data into transaction form. We wrote python code for this transformation. The approach itself is very simple. We construct a dataframe with 2 columns, one corresponds to the UserID and the other to the the UserID. We then traverse through all the raw-data from top to bottom. In the beginning we ran into memory issues but we quickly realized that this naive approach won't give us results with limited computation power. We then came up with a smarter approach of sorting the data and appending a new transaction whenever we notice a change in UserID. The sample of result looks like follow figure
  We generated two types of datasets.
  The first type is:



userid	songid
fd50c4007b68a3737fe052d5a4f78ce8aa117f3d	['SOBONKR12A58A7A7E0', 'SOEGIYH12A6D4FC0E3', 'SOFL
d7083f5e1d50c264277d624340edaaf3dc16095b	['SOUVUHC12A67020E3B', 'SOUQERE12A58A75633', 'SOIP
d68dc6fc25248234590d7668a11e3335534ae4b4	['SOFRQTD12A81C233C0', 'SOZQIUZ12A8C13CFBE', 'SOKC
9be82340a8b5ef32357fe5af957ccd54736ece95	['SOHGGAH12A58A795BE', 'SOSYIDB12A8C1371F3', 'SOFE
841b2394ae3a9febbd6b06497b4a8ee8eb24b7f8	['SORVQZE12A58A7AE98', 'SOXPJVO12A6D4FCC69', 'SOQ
91b8fac7dc5e03f6cfaf6e2aa7171f14a8354d62	['SOLLNTU12A6701CFDC', 'SOBXMJC12AB018293E', 'SODE
458833ce4418010e61304b34b2c992e1cce63435	['SOVOGGO12A8C1433B2', 'SOQTYTQ12A8C1377C8', 'SOE
c34670d9c1718361feb93068a853cead3c95b76a	['SOEHWGF12A6D4F8B2B', 'SOAUWYT12A81C206F1', 'SOP
0f40e074aab2c5f47b7ddc2277fb0295b5b3a058	['SOBPFUQ12A6D4FCDD2', 'SODBDBO12A6D4FCDAA', 'SC
ef0d21935a2f8ae90571dbfab800f87fa5b38769	['SORALMO12AF729EBB4', 'SODTAHY12A8C13ACAF', 'SOH
fdf6afb5daefb42774617cf223475c6013969724	['SOJZBHH12AB017F611', 'SOONUTJ12A6701D7B4', 'SONX
10cbcd627472477dfbec90fb75017f8df6ce84ec	['SOTNWCI12AAF3B2028', 'SOJGKMM12A6D4F98CD', 'SON
e9dc6b4c2b22aa6dc8260e1963021567728055b2	['SOUIGCD12AB0186713', 'SOYOMRA12A6D4F9975', 'SOLN
30e4a688e6fc9c8bfe55998af3996a909ae34449	['SOTQBKC12A8C13E961', 'SOAIBYI12AB0185C5B', 'SOSMC
18ce1da0e1017e31baaa5f80afa64ee3c7fab379	['SOKWSEA12A8C141C9D', 'SOPGCXT12A8C138AD1', 'SOD
0b254c684efb08fd04933add2d1e4191d2a87bac	['SOOPUTL12A8C143381', 'SOPOANU12A8AE48C9B', 'SOU
eda9bc7bcd72d18b9cf964990eb13a5b1789e78f	['SOVYUYG12A8AE47BAB', 'SOOPWYG12AB0181160', 'SOA
da7bc0ec91a21a54f0b209bcc9ec5b4b49613a68	['SOCLMAD12AB017FC09', 'SOUSMXX12AB0185C24', 'SOP

Figure 3: Dataset after Processing

  UserID - $songID_1, songID_2, \ldots songID_N$
  The second type is:
  SongID - $UserID_1, UserID_2, \ldots UserID_N$
  The UserIds and SongIds were later replaced by their corresponding numerical values. The transformed dataset has 163206 unique songs and 110000 unique users in it.
- Observations: We tried applying Apriori algorithm on the entire dataset. Our machines were not powerful enough to run the computationally expensive Apriori algorithm on all the data. We thus observed the distribution of the frequency of each song, so as to run Apriori on only some meaningful subset of the data and not all of it. The interesting thing that we observed was that around 154985 songs have their frequency in the range from 1 to 100. Out of these songs, 108054 have their frequency less than 10. This is a huge number. We can get rid of these songs because they are going to be infrequent and test our data on songs that have their frequencies in the range of 11 to 5000. There are 51808 such songs and we have now reduced the dataset by almost a factor of 3.
  We have attached all the interesting plots showing the distribution of song frequencies at the end of the report.
- Generate reduced dataset: We filtered out all the songs whose frequency was not in the range [11,5000]. This reduced the size of each transaction considerably.
- Generate dataset with High/Medium or Low labels: Our novel idea is to take into consideration the frequency with which each user listens to a song while generating frequent itemsets. After careful analysis of each transaction, we decided to use the label Low or L when the frequency of a song in a transaction is less than 5, use the label Medium or M when the frequency of a song in a transaction is greater than or equal to 5 and less than 10, and use the label High or H when the frequency of a song in a transaction is greater than or equal to 10. For example, if SongID 13222 occurred 3 times in a transaction, then the songid is modified to 13222L. We did this transformation for both the original dataset and the reduced dataset.
- Major Problems faced: We have huge amount of data with limited computation power. It is difficult to even generate

transactions in the form of two itemsets mentioned above, let alone run an algorithm for generating frequent itemsets.

5.2 Explored Cloud platforms

While applying and playing around with Apriori, we soon realized that our computer is not powerful enough to run the algorithm on the entire dataset. We reduced the number of songs in every transaction by filtering high and low frequency songs. The dataset was still too large for our machines to handle. This problem became very challenging and interesting to our group. In order to overcome this, we even reduced the number of transactions by only considering those that had at least 10 songs in it. The problem still prevailed. This forced us to try our hands on cloud computing which gave us access to more powerful machines than our local systems. The two cloud platforms that we explored were GCP and AWS. Since none of us had prior experience with cloud computing, we spent a considerable amount of time exploring both the platforms. We realized that both are at par when it comes to performance but we chose GCP because we found it more user-friendly. We even set up our own jupyter server on those remote machines which gave us the ability to run out Jupyter notebooks on the remote server. With the help of our GCP, we were able to play around with smaller values of MIN_SUP and generate frequent itemsets. We chose a machine with 60GB of RAM. However, we still ran into memory issues occassionally for MIN_SUP values smaller than a particular threshold. This made us realize how it is not at all practical to use Apriori on huge amounts of real life data. We couldn't choose more powerful remote servers because they were too expensive. However, smaller execution time and better results can be obtained by more powerful machines.

5.3 Implemented Apriori algorithm

We tried to implement the Apriori algorithm on the entire dataset using existing python library, "apyori". We obtained results for higher values of MIN_SUP but the generated itemsets were not enough in quantity to build a recommendation system and were only 2-itemsets and 3-itemsets. For smaller values of MIN_SUP, we ran into memory issues(even on GCP). We decided to run the algorithm on two datasets, reduced dataset with HML format and reduced dataset without HML format. We experimented with different values of MIN_SUP and MIN_CONF and generated various plots which will be discussed in the Observations section.

5.4 Implemented FP-Growth algorithm

From the lecture slides, we knew that Apriori takes so much time because of candidate generation and multiple database scans. On the other hand, the FP growth algorithm overcomes these problems and is much quicker than Apriori. Hence, we decided to apply FP growth. This algorithm was very effective and generated frequrent itemsets pretty quickly on our local machines. We used the existing python library, "pyfpgrowth". We experimented with various different values of MIN_SUP and MIN_CONF and generated interesting plots which will be discussed in the Observations sections. We implemented the FP algorithm on four different datasets, the original dataset, the original dataset with HML labels, the reduced dataset and the reduced dataset with HML labels.

5.5 Implemented Collaborative filtering

We initially proposed to use the ALS Matrix factorization technique for CF. We processed the data into the required format and tried to implement it. However, the computation power we had (even with a 60GB CPU) was not enough to obtain user and song vectors from this technique.

We needed the output from CF to compare this method and our proposed work. We explored other technoques for CF and found that the winner of the million song dataset challenge employs the same method. We adapt and modify the previous work form the winner[2][1].

The winner of the million song dataset challenge employed the following algorithm[1]. Let $U(i)$ be the set of users who have listened to a song i. The winner defined the similarity between two songs i, j as

$$W_{ij} = \frac{|U(i) \cap U(j)|}{|U(i)|^{\alpha} |U(j)^{1-\alpha}|}$$

, where $[0, 1]$ is a parameter to tune. When $= 0$, the conditional probability $P(u|v)$ is obtained. When $\alpha = 0.5$, the cosine similarity is obtained. The winner used 0.15 as α.

We adapted this algorithm for getting all the similar songs of each song[1][2]. Executing the algorithm on the entire dataset took a huge amount of time. To generate a list of similar songs for each song, it took us around 55 hours. It takes around 3 seconds for every song to generate a list of similar songs. We used five dedicated computers on GCP to finish this work(16 vCPUs, and 60 GB Memory).

5.6 Design the Recommender System

Once we were able to generate frequent itemsets from our data using the FP Growth algorithm, we wrote the results to a file. We generated frequent itemsets on two types of dataset, original dataset with HML labels and reduced dataset with HML labels. Now, the system takes a SongID as input and generates recommendations by using frequent itemsets if it is present in any frequent itemsets or else it just returns the most popular songs on the platform.

Similarly, we also have a system to generate recommendations from CF method. It takes the SongID as input and outputs the list of recommended songs genenrated by CF method. If no recommendations exist for that song, then it just returns NULL.

6 EVALUATION

In a recommendation system, there is no labeled data using which one can check their system's performance. The general practice is to release your model and see the user's reaction to the recommendations that they get. Collaborative filtering is one of the the state of the art techniques for generating recommendations. We implemented CF so that the results generated from it can be compared with our system's recommendations.

For the purpose of this project, we designed a new metric called average overlap percentage. For any new recommendation system's prediction, this metric gives how much is the average intersection percentage between the predictions of the new system and the baseline CF model for all the songs. We designed this metric to observe how different our recommendations are. For our model, these are the values we obtain :

- Percentage Overlap of recommendations from FP growth on all dataset with HML and CF wrt to recommendations from is CF 0.10414541734942243. This tells us that for every recommendation list, on an average number of recommendations predicted by our model are around 0.1% of the number of recommendations predicted by CF. This implies that it can help in overcoming the problem of popularity bias.
- Percentage Overlap of recommendations from FP growth on all dataset with HML and CF wrt to recommendations from FP growth is 4.4344022439838. This tells us that for every recommendation list, on an average 4.4% of the number of recommendations predicted by our model are same as that of CF. This means it is generating new recommendations which is different from that of CF. It is hard to say if they are good or bad. We just know that they are different. This implies that it can probably help in discovering more kinds of songs.
- Percentage Overlap of recommendations from FP growth on reduced dataset with HML and CF wrt to recommendations from CF is 0.07911922213407353. This tells the same thing as point 1 but for a different dataset (Reduced dataset with HML)
- Percentage Overlap of recommendations from FP growth on reduced dataset with HML and CF wrt to recommendations from FP growth is 3.3688131950774842. This tells the same thing as point 2 but for a different dataset (Reduced dataset with HML)

7 RESULTS AND CONCLUSION

We observed that the recommendations generated by our model are either different from that generated by the CF method or are at very low priority/similarity when it comes to recommendations. We believe that this can help in overcoming the popularity bias problem of CF method. We can't really comment if the recommendations are good or bad but they are different. Also, generating recommendations for all the songs using CF method took us around 55 hours. Generating similar recommendations using frequent itemsets took us around half an hour which is a significant improvement. Thus, we believe our model is at least worth trying in the real world and then analysis should be performed to observe the correctness of its recommendations.

8 OBSERVATIONS

In the report we are only displaying plots for three systems, System1 - FP on All Dataset with HML labels, System2 - FP on Reduced dataset with HML labels, System3- Apriori on Reduced dataset with HML labels. However, more plots are attached with the source code.

8.1 Plot of number of itemsets generated vs MIN_SUP

(Figure 4, Figure 5 and Figure 6) The plots are in accordance with what is expected. Number of itemsets decreases as the min support value increases for all the three models.

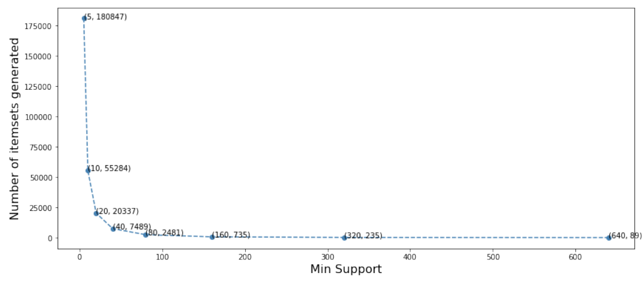


Figure 4: FP with All Dataset and HML labels

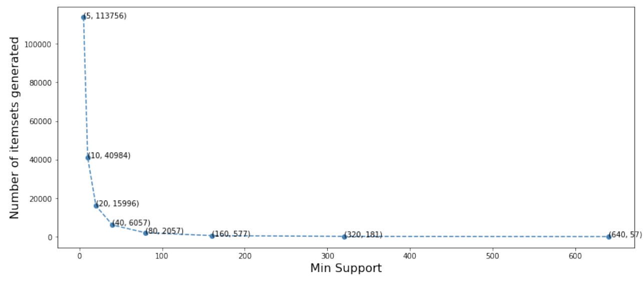


Figure 5: FP with Reduced Dataset and HML Labels

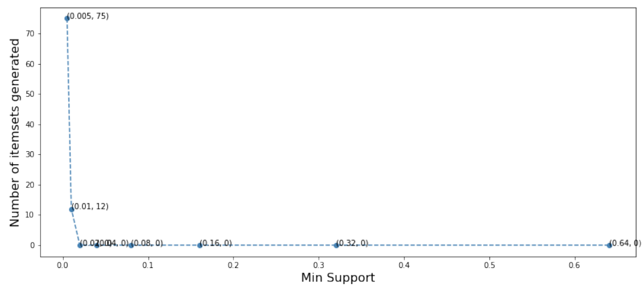


Figure 6: Apriori with Reduced Dataset and HML labels

8.2 Plot of time taken for generating frequent itemsets vs MIN_SUP

(Figure 7, Figure 8 and Figure 9) The plots are in accordance with what is expected. Number of itemsets decreases as the min support value increases for all the three models. Also, the time taken decreases continuously for System 1 and System 2 as MIN_SUP increase. However, that is not the case for System3 which is interesting.

8.3 Plot of Number of Association rules vs MIN_SUP

(Figure 10, Figure 11 and Figure 12) The plots are in accordance with what is expected. Number of association rules decrease as the min support value increases for all the three models. For System 3, since we had limited computation power, we couldn't choose a MIN_CONF value lower than 0.5, and so there are almost no association rules.

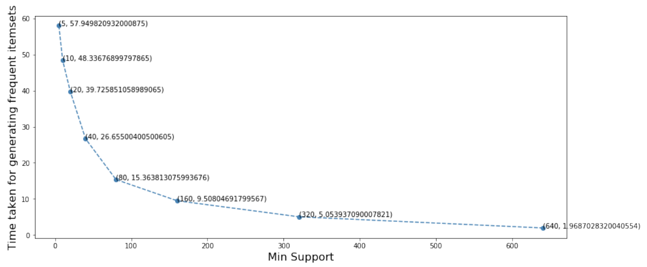


Figure 7: FP with All Dataset and HML labels

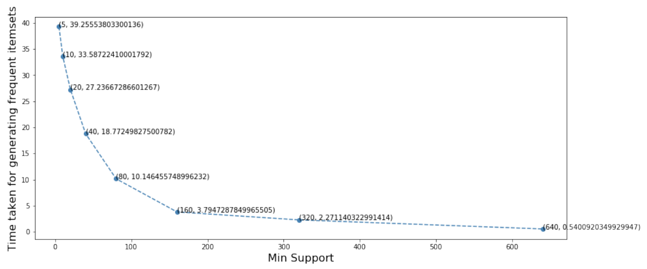


Figure 8: FP with Reduced Dataset and HML Labels

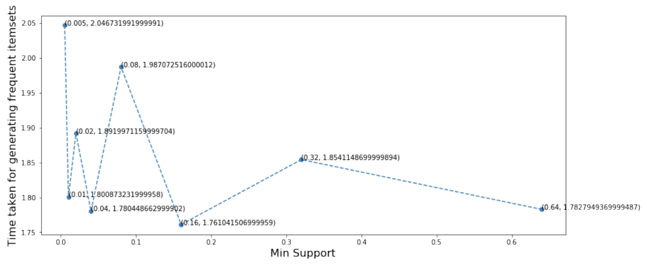


Figure 9: Apriori with Reduced Dataset and HML labels

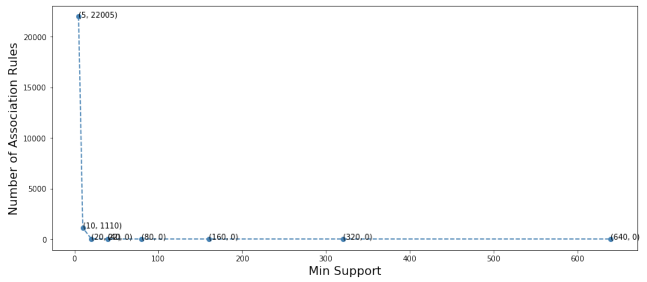


Figure 10: FP with All Dataset and HML labels

8.4 Plot of Number of Association rules vs MIN_CONF for a fixed MIN_SUP=20

(Figure 13, Figure 14 and Figure 15) The plots are in accordance with what is expected. Number of association rules decrease as the min conf value increases for all the systems. For System 3, since we had limited computation power, we couldn't choose a MIN_CONF value lower than 0.5, and so there are almost no association rules.

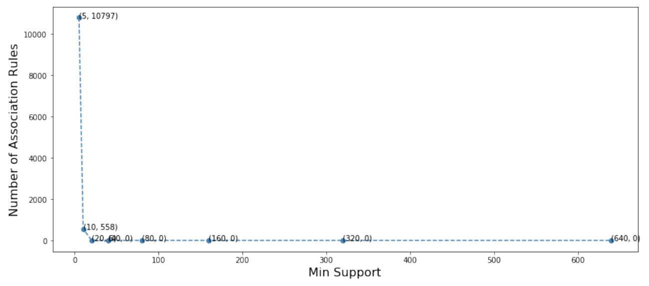


Figure 11: FP with Reduced Dataset and HML Labels

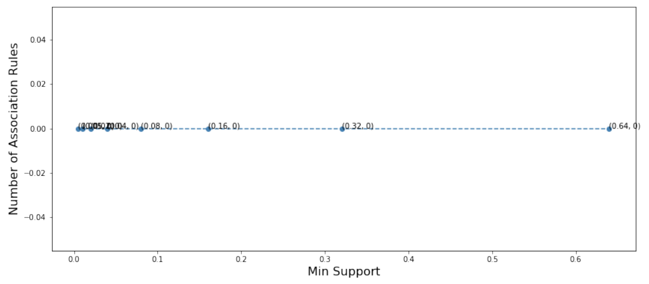


Figure 12: Apriori with Reduced Dataset and HML labels

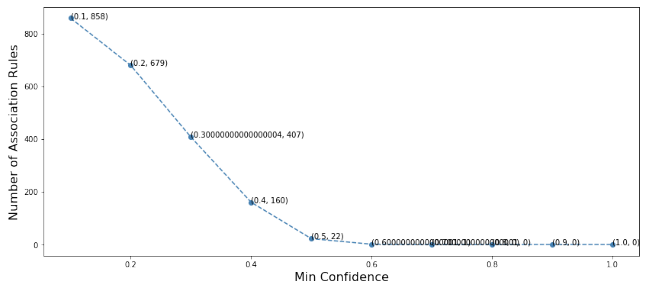


Figure 13: FP with All Dataset and HML labels

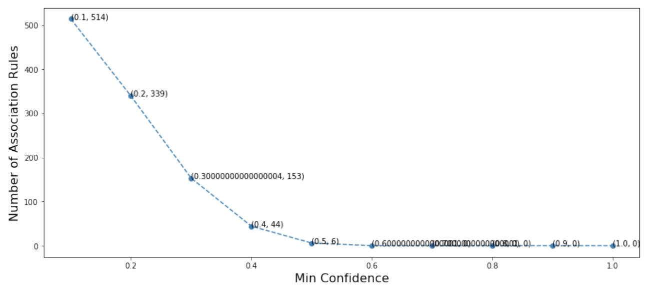


Figure 14: FP with Reduced Dataset and HML Labels

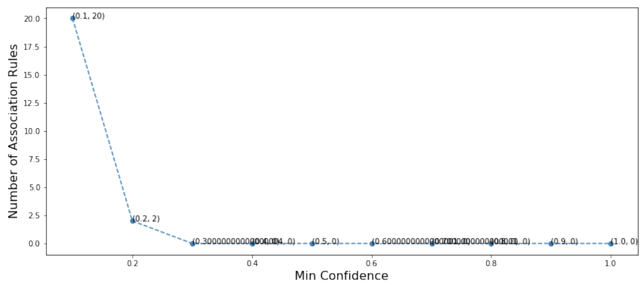


Figure 15: Apriori with Reduced Dataset and HML labels

8.5 Plot of Time taken for generating Association rules vs MIN_CONF for a fixed MIN_SUP=20

(Figure 16, Figure 17 and Figure 18) The plots are not at all in accordance with what we expected. We expected either a gradual increase or decrease but instead we obtain a zig zag pattern for all the systems which is difficult to explain.

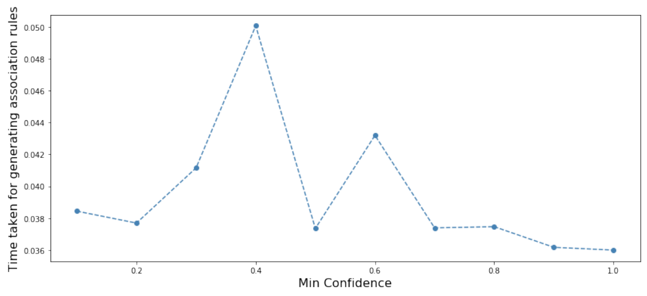


Figure 16: FP with All Dataset and HML labels

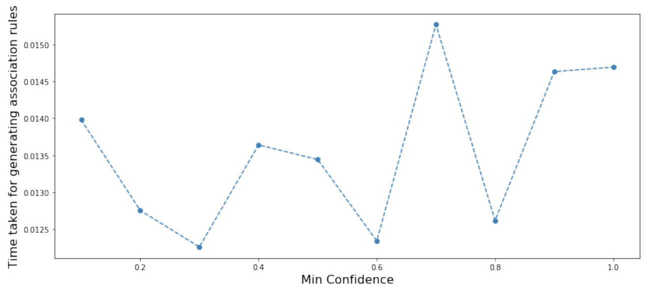


Figure 17: FP with Reduced Dataset and HML Labels

8.6 Plot of Number of Association rules vs MIN_CONF for different MIN_SUP

(Figure 19, Figure 20) The plots are in accordance with what we expected. For small values of MIN_SUP, we get high number of association rules which is expected and it keeps on decreasing as the MIN_CONF value increases.

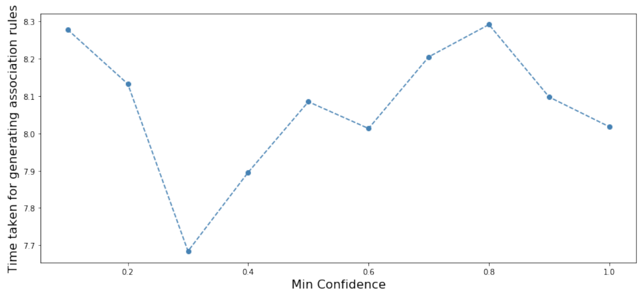


Figure 18: Apriori with Reduced Dataset and HML labels

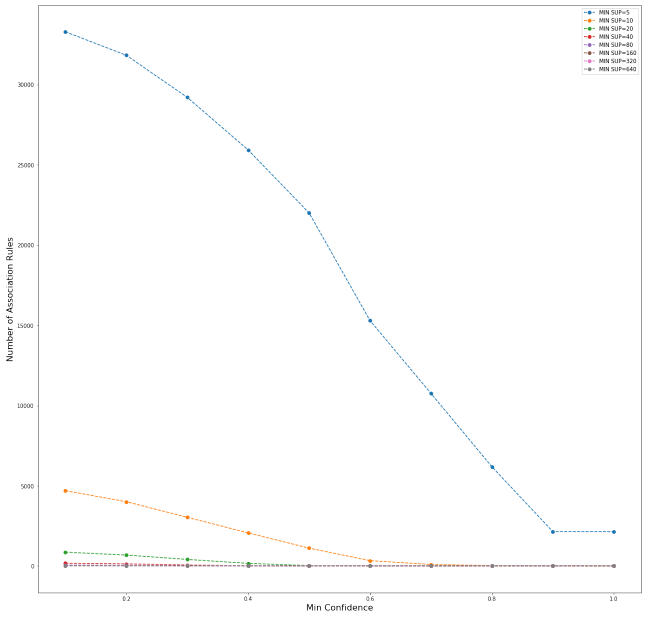


Figure 19: FP with All Dataset and HML labels

8.7 Plot of time taken for generating Association rules vs MIN_CONF for different MIN_SUP

(Figure 21, Figure 22 and Figure 23) The plots are not at all in accordance with what we expected. We expected either a gradual increase or decrease but instead we obtain a zig zag pattern for all the systems which is difficult to explain. The ziz zag pattern is occurring across different values of MIN_SUP as well.

9 APPENDIX

9.1 Honor Code Pledge

- On our honor, as University of Colorado Boulder students, we have neither given nor received unauthorized assistance.

9.2 Individual Contribution

- Himanshu Gupta worked on the entire code for Apriori and FP growth, Recommender system, designing the evaluation

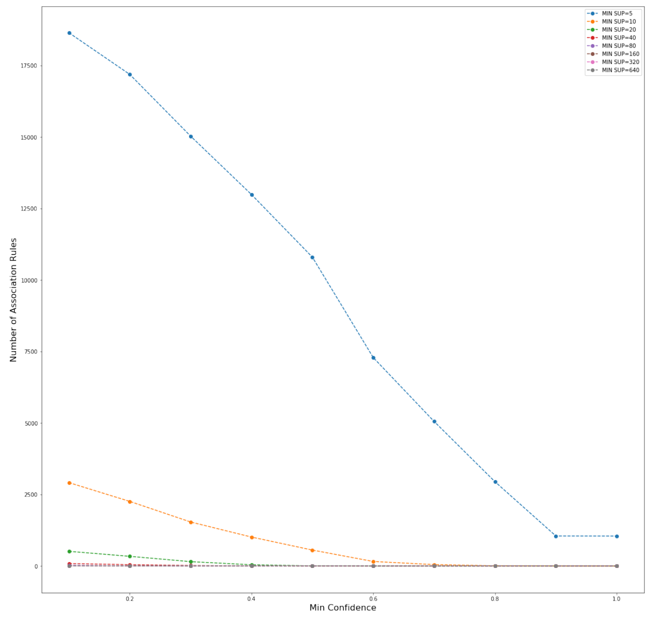


Figure 20: FP with Reduced Dataset and HML Labels

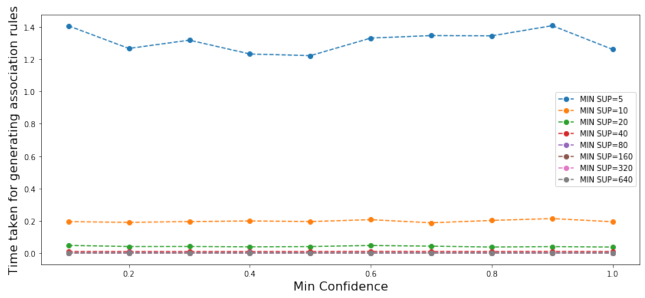


Figure 21: FP with All Dataset and HML labels

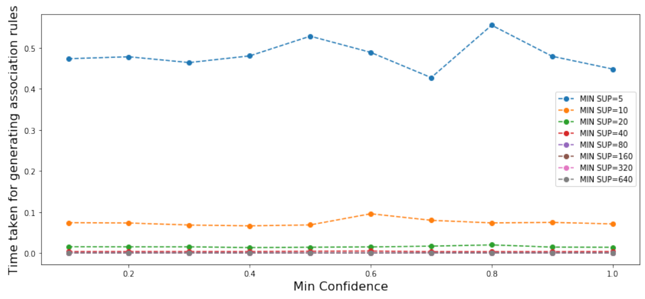


Figure 22: FP with Reduced Dataset and HML Labels

metric and generating various different interesting plots.He also performed data analysis for reducing the dataset and worked on transforming dataset to different formats for execution.

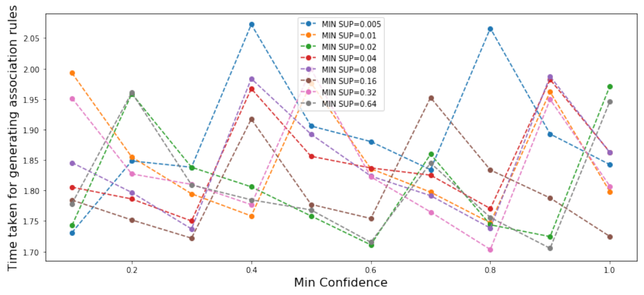


Figure 23: Apriori with Reduced Dataset and HML labels

- Jaeyoung Ho worked on CF method and exploring other dataset for finding interesting patterns. He couldn't find any.
- Youngjin Ko worked on Data Preprocessing and handled our GPU tasks.
- Yutaka Urakami worked on Data Preprocessing, implemented CF on GPU, explored AWS for us.

10 SOME INTERESTING PLOTS

(Figure 24 - Figure 29)

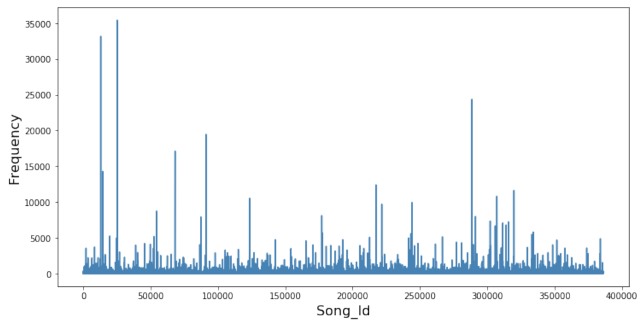


Figure 24: Plot of Song id with frequency

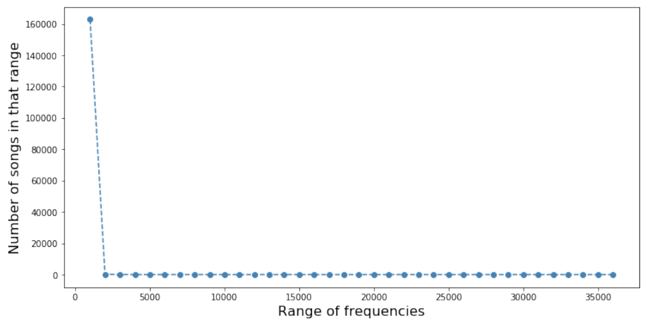


Figure 25: Plot of range of frequency vs number of songs in that range (bucket size=1000)

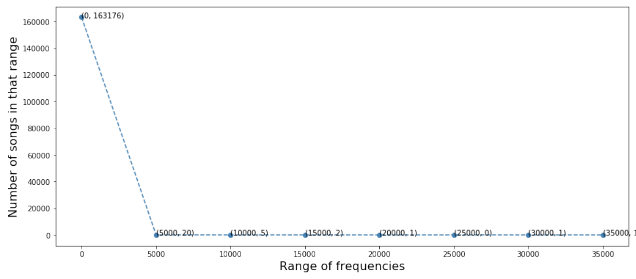


Figure 26: Plot of range of frequency vs number of songs in that range (bucket size=5000)

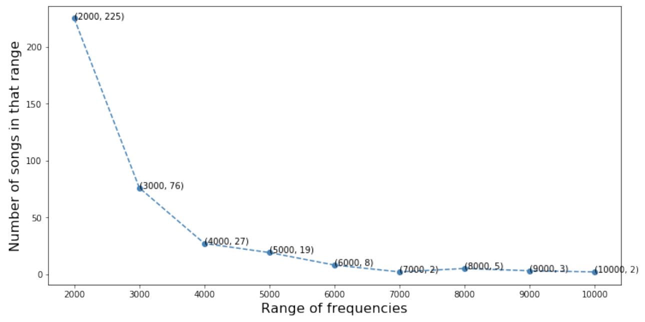


Figure 27: Plot of range of frequency vs number of songs in that range from 2nd to 10th bucket (bucket size=1000)

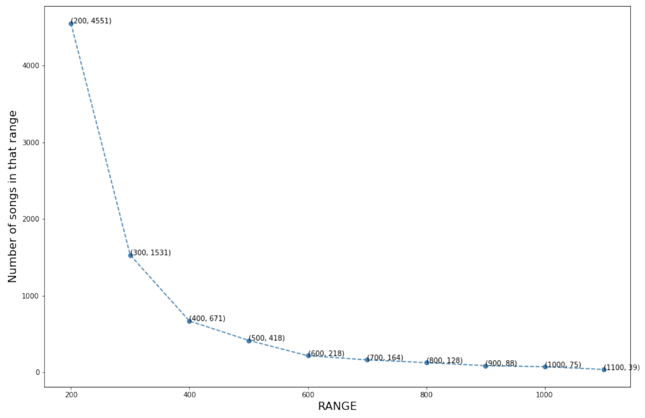


Figure 28: Plot of range of frequency vs number of songs in that range from 2nd to 10th bucket (bucket size=100)

REFERENCES

[1] Fabio Aiolli. 2012. *2012 MILLION SONGS DATASET Challenge CODE.* https://www.math.unipd.it/~aiolli/CODE/MSD/

[2] Fabio Aiolli. 2013. Efficient Top-n Recommendation for Very Large Scale Binary Rated Datasets. In *Proceedings of the 7th ACM Conference on Recommender Systems (RecSys '13).* ACM, New York, NY, USA, 273–280. https://doi.org/10.1145/2507157.2507189

[3] Natalie Aizenberg, Yehuda Koren, and Oren Somekh. 2012. Build Your Own Music Recommender by Modeling Internet Radio Streams. In *Proceedings of the 21st International Conference on World Wide Web (WWW '12).* ACM, New York, NY, USA, 1–10. https://doi.org/10.1145/2187836.2187838

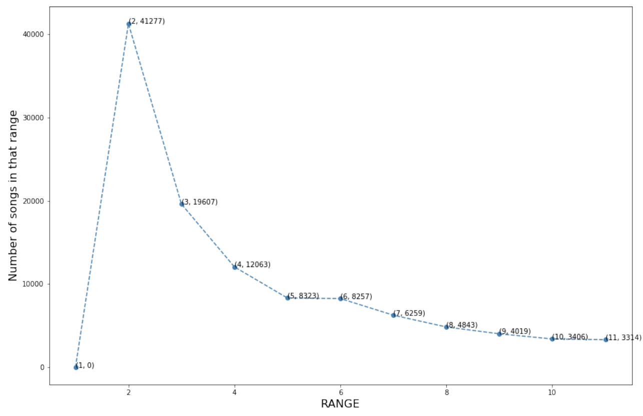


Figure 29: Plot of range of frequency vs number of songs in that range from 1st to 11th bucket (bucket size=1)

[4] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. 2011. The Million Song Dataset. In *IJECS Volume 3 Issue 7 July, 2014 Page No.6855-6858.*

[5] Ò. Celma. 2008. *Music Recommendation and Discovery in the Long Tail.* Ph.D. Dissertation. Universitat Pompeu Fabra, Barcelona.

[6] Shuo Chen, Josh L. Moore, Douglas Turnbull, and Thorsten Joachims. 2012. Playlist Prediction via Metric Embedding. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '12).* ACM, New York, NY, USA, 714–722. https://doi.org/10.1145/2339530.2339643

[7] Prince Grover. 2017. *Details about Collaborative Filtering.* Retrieved October 10, 2019 from https://towardsdatascience.com/various-implementations-of-collaborative-filtering-100385c6dfe0

[8] Negar Hariri, Bamshad Mobasher, and Robin Burke. 2012. Context-aware Music Recommendation Based on Latenttopic Sequential Patterns. In *Proceedings of the Sixth ACM Conference on Recommender Systems (RecSys '12).* ACM, New York, NY, USA, 131–138. https://doi.org/10.1145/2365952.2365979

[9] Kaggle. 2012. *Kaggle Dataset.* Retrieved October 10, 2019 from https://www.kaggle.com/c/msdchallenge/data

[10] Noam Koenigstein, Gideon Dror, and Yehuda Koren. 2011. Yahoo! Music Recommendations: Modeling Music Ratings with Temporal Dynamics and Item Taxonomy. In *Proceedings of the Fifth ACM Conference on Recommender Systems (RecSys '11).* ACM, New York, NY, USA, 165–172. https://doi.org/10.1145/2043932.2043964

[11] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2011. Introduction to recommender systems handbook. In *Recommender systems handbook*. Springer, 1–35.

[12] Markus Schedl, Peter Knees, and Johannes Kepler. [n.d.]. Context-based Music Similarity Estimation. In *in Proceedings of the 3rd International Workshop on Learning the Semantics of Audio Signals (LSAS 2009.* 2009.

[13] Lohr Steve. 2009. *A 1Million Research Bargain for Netflix, and Maybe a Model for Others.* Retrieved September 21, 2009 from https://www.nytimes.com/2009/09/22/technology/internet/22netflix.html

[14] V.Manvitha1 and M.Sunitha Reddyand. 2014. Music Recommendation System Using Association Rule Mining and Clustering Technique To Address Cold start Problem. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2014).*

[15] Wikipedia. 2019. *Details about Apriori Algorithm.* Retrieved October 10, 2019 from https://en.wikipedia.org/wiki/Apriori_algorithm