CSCI 5502
Music Recommendation System

Youngjin Ko (yoko2606@colorado.edu)
Yutaka Urakami (yutaka.urakami@colorado.edu)
Himanshu Gupta (himanshu.gupta@colorado.edu)
Jaeyoung Oh (jaeyoung.oh@colorado.edu)

Why recommend music?

- We all use various platforms for streaming music, and with lots of options to choose from, people sometimes feel overwhelmed.
- This raises the need for an efficient music recommender system for customers.
- With the help of such systems, companies can improve users' satisfaction and ensure high retention of users.
- A large scale, open and transparent Million Song Dataset gives us an opportunity to work on a music recommendation system that is centered around users and their behaviour (which is a hot topic these days).

Why this problem and challenges?

- 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.
- 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 is a challenging task. Comparing results with other new techniques from other similar domains can come in handy.

Data

We plan to use two existing datasets- Kaggle's dataset and Million Song Dataset.

Kaggle dataset includes:

- 386,213 song IDs and 110,000 user IDs
- each user's listening history (user_id, song_id, #listening) in kaggle_visible_evaluation_triplets.txt
- relationship b/w song_id and track_id(s) in taste_profile_song_to_tracks.txt

Million Song Dataset includes:

- each track information (tempo, loudness, energy and so on) in track_id.h5
- List of all artist ID (artist_id, artist, mbid, track_id, artist name) in unique_artists.txt
- And so on!

Previous Work

- Build Your Own Music Recommender by Modeling Internet Radio Streams, WWW 12, [1]
  - Goal: Personalized music recommendation
  - Data: playlists of thousands of Internet radio stations (Millions of plays, hundreds of thousands of tracks and artists)
  - Method: a probabilistic collaborative filtering (CF) model

- Yahoo! Music Recommendations: Modeling Music Ratings with Temporal Dynamics and Item Taxonomy, RecSys 11, [2]
  - Goal: Improve Yahoo music recommendation accuracy
  - Data: Yahoo music dataset, million users, 600 thousand musical items, 250 million ratings for decade
  - Method: matrix factorization with temporal analysis of user ratings, and item popularity trends

Previous Work

- Playlist Prediction via Metric Embedding, ACM KDD 12, [3]
  - Goal: automated playlist generation
  - Data: radio playlist from yes.com (hundreds of stations in the U.S)
  - Method: Latent Markov Embedding (LME)

- Context-Aware Music Recommendation Based on Latent Topic Sequential Patterns, Recsys 12, [4]
  - Goal: Context-aware music recommender system
  - Data: human-compiled music playlists (7,051 playlists, 21,783 songs)
  - Method: Topic modeling with the latent topics generated from the the most frequent tagged songs.

Proposed work and techniques

- We have already seen how identifying frequent itemsets helps is generating item recommendations for users. Why not use the same algorithm for generating song recommendations?
- Tackle the popularity bias problem of Collaborative Filtering technique.
- If possible, use audio content features to cluster similar songs and then generate relevant recommendations from that.

Completed Tasks

- Data Pre Processing (Successful)
- Data Analysis (Successful)
- Data Reduction + Transformation (Successful)
- Cloud Platforms (Successful)
- Apriori Algorithm (Unsuccessful)
- FP Growth (Successful)
- Matrix Factorisation (Unsuccessful)
- Collaborative Filtering (Successful)
- Recommendation System (Successful)
- Evaluation (Successful)
- Interesting Plots (Successful)

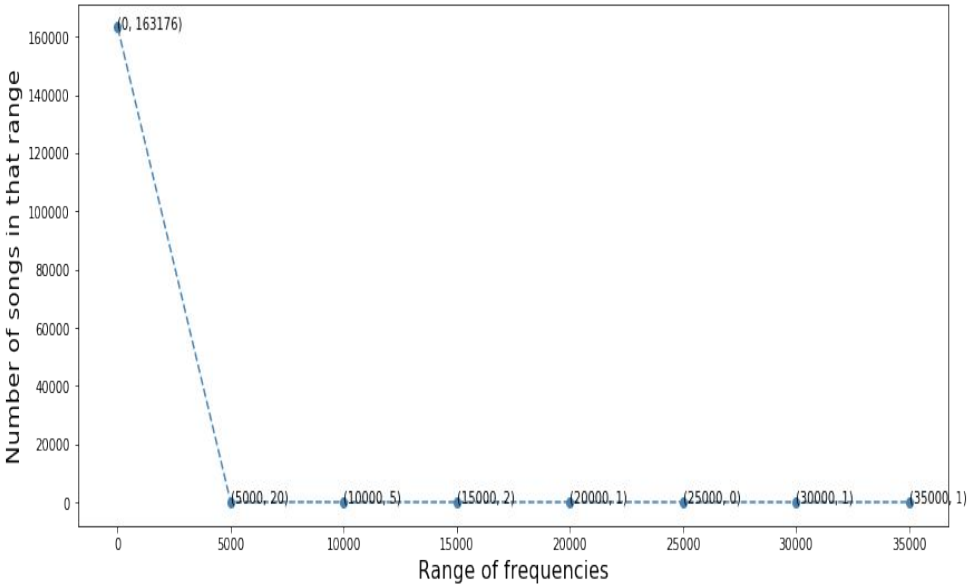
Data Processing

- Raw dataset is one to one transaction b/w UserID and SongID.
- Transform this to one to many transactions b/w UserID and SongIDs
- Transform this to one to many transactions b/w SongID and UserIDs.

Challenges

- Memory issue -> solved by initially sorting the row data
- Huge amount of data with limited computation power.
    - #Unique songs = 1.6M
    - #Unique users = 1.1 M

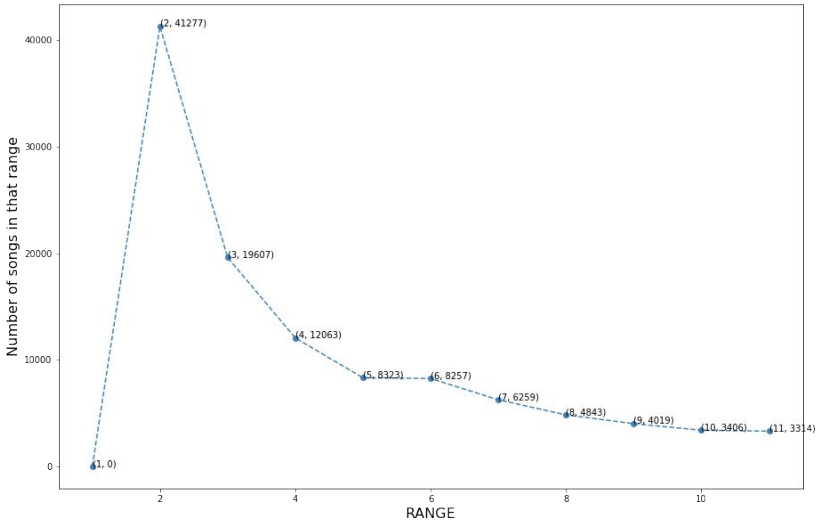
Data Analysis (1)



- We tried applying Apriori algorithm on the entire dataset. Our machines were not powerful enough to run the computationally expensive Apriori algorithm.

- 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.

Data Analysis (1)



- 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.

Data Reduction + Transformation

- We filtered out all the songs from each transaction that had frequency <=10 or >5000

- Generated 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 < 5,
  - label Medium or M when the frequency of a song in a transaction >= 5 and < 10
  - label High or H when the frequency of a song in a transaction >=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.
  - We have 4 datasets now
    - All Dataset; All Dataset with HML
    - Reduced Dataset; Reduced Dataset with HML

Implementing Apriori Algorithm

- We used existing Python libraries:
  - apriori (https://pypi.org/project/apyori/)
  - mlxtend.frequent_patterns

- Problems
  - Huge dataset and limited computation power (once again).
  - These libraries seem to have some issue when the item names are random strings(as is in the dataset).

- Proposed Solutions
    - Use a numerical mapping for encoded Uid and Song_id.
    - Use FP-Growth instead or get powerful machines

Explore Cloud Platforms

- Apriori gave errors on reduced dataset as well.
  - Failed even when number of transactions were reduced by putting a lower bound of 10 songs/ transaction.
  - This problem was quite challenging and fascinating to our group.

- This forced us to try cloud computing which gave us access to more powerful machines than our local systems.
  - We explored AWS and GCP.
  - We chose GCP because it is more user friendly.
  - Set up Jupyter server on remote machine to run our jupyter notebooks

- We used instance with 60GB of memory.
  - Apriori still ran out of memory for MIN_SUP < 0.001 on Reduced Dataset.
  - We concluded - Apriori is really not practical. No point in trying its variants.
  - Need to try different algorithm.

Apriori on dataset using GCP

- We used instance with 60GB of memory.
    - Apriori ran out of memory for MIN_SUP < 0.01 on All dataset.
    - Apriori still ran out of memory for MIN_SUP < 0.001 on Reduced Dataset.
    - We concluded - Apriori is really not practical. No point in trying its variants.
    - Need to try different algorithm.
- We ran Apriori on Reduced Dataset with HML for MIN_SUP >= 0.005 and obtained various different plots.
    - Discuss at the end.

FP Growth

- Applied FP growth using the “pyfpgrowth” python library.
- It generated results on the entire dataset without any errors on our local systems with limited computational power.
  - It was so much more effective than Apriori.
- We ran FP Growth on all the 4 datasets that we generated.
  - Generated various interesting plots.
  - Generated and stored frequent itemsets from all the datasets.

Collaborative Filtering

- CF is one of the state of the art techniques for recommendation systems.
- We planned to implement ALS matrix factorisation to generate user vectors and song vectors.
- Ran into memory issues on both our local system and GCP.
  - Not possible to get more powerful machines.
- Looked for alternate CF techniques.
  - Came across the approach adapted by the winner of MSD challenge.

Implementing collaborative filtering

- Algorithm
  - We used the codes from the winner of the challenge [5],[6]
  - Let U(i) be the set of users who have listened to a song i
  - The similarity between two song i and j is defined by

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

  - We used 0.15 as α.
- Problem
  - Huge dataset and limited computation power (once again).
  - When using 16 vCPUs and 60G memory (GCP), getting all similarities of each song for all data sets takes 55 hours.

Build Recommendation System

- **Using Frequent itemsets**
  - Take a songid as input
  - Look for itemsets that have the songid.
  - Recommend all items from those itemsets.
  - How do you decide the order?
    - Song with label H > song with label M > song with label L
    - This gives a ranking approach as well.

- **Using CF**
  - Take a songid as input
  - Return the recommendation list corresponding to that songid.

Evaluation (1)

- We plan to compare and evaluate our model against the state of the art CF technique and compare the recommendations.

- We defined a metric to calculate the average percentage overlap between our model's prediction and CF's predictions (Average over all the songs).

- What we got?

    - 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.

Evaluation (2)

- We plan to compare and evaluate our model against the state of the art CF technique and compare the recommendations.

- We defined a metric to calculate the average percentage overlap between our model's prediction and CF's predictions (Average over all the songs).

- What we got?

    - 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.

Results and Conclusion

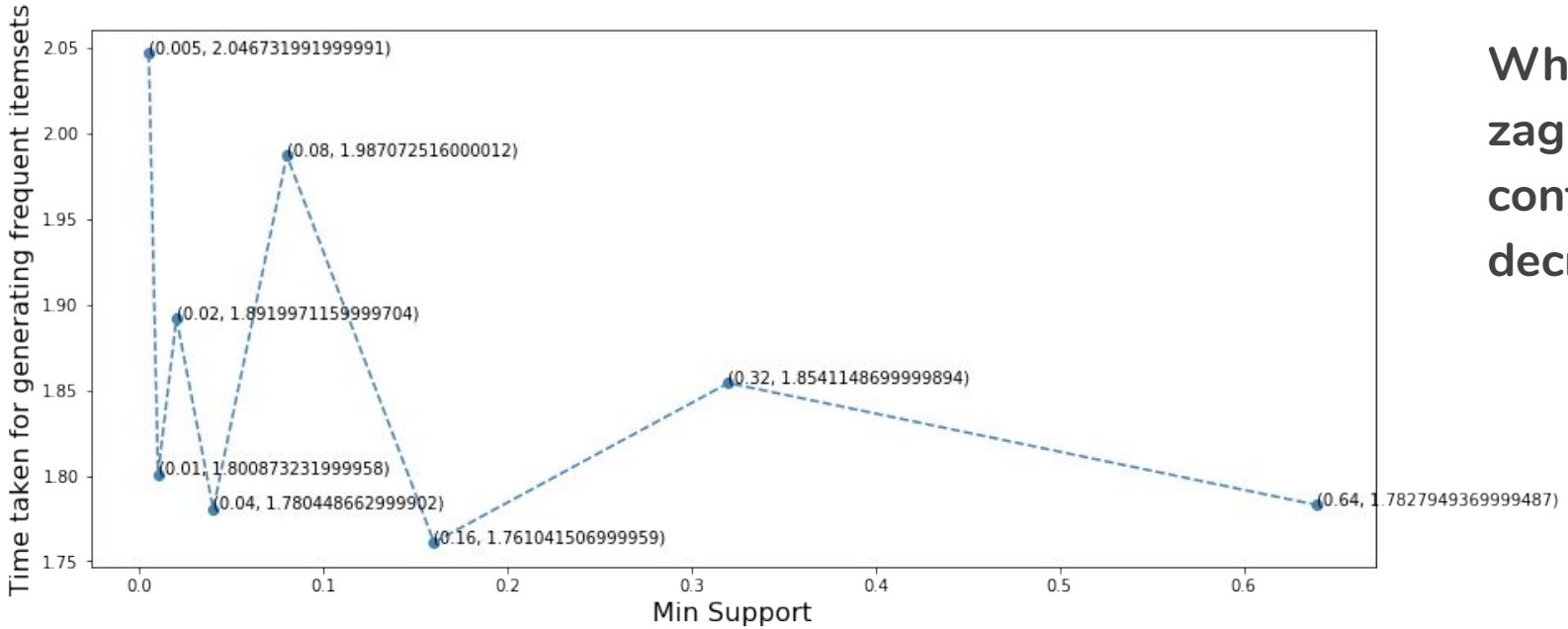
- We observed that the recommendations generated by our model are either different from that generated by the CF method or are at very low priority/similarity when it comes to recommendations.

- 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.

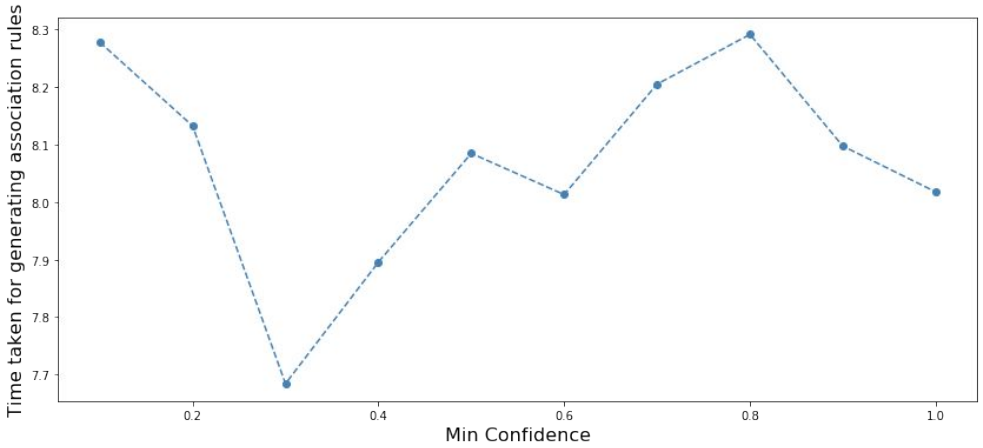
- Also, generating recommendations for all the songs using CF method took us around 55 hours. Generating similar recommendations using frequent itemsets took us around half an hour which is a significant improvement. Thus, we believe our model is at least worth trying in the real world and then analysis should be performed to observe the correctness of its recommendations.

Some interesting plots!

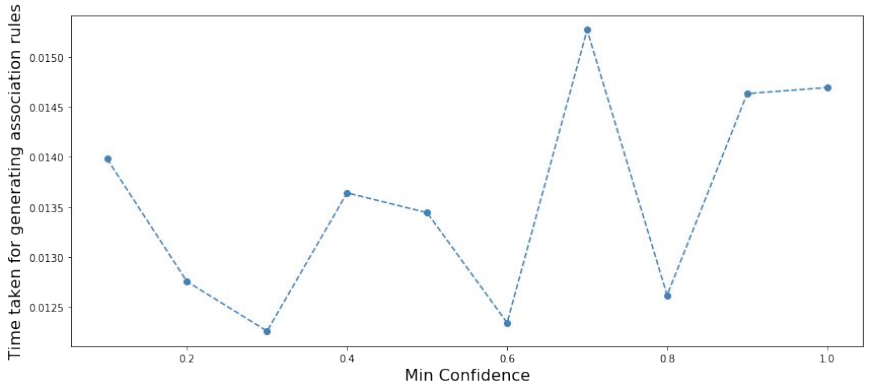
Apriori with Reduced Dataset and HML labels



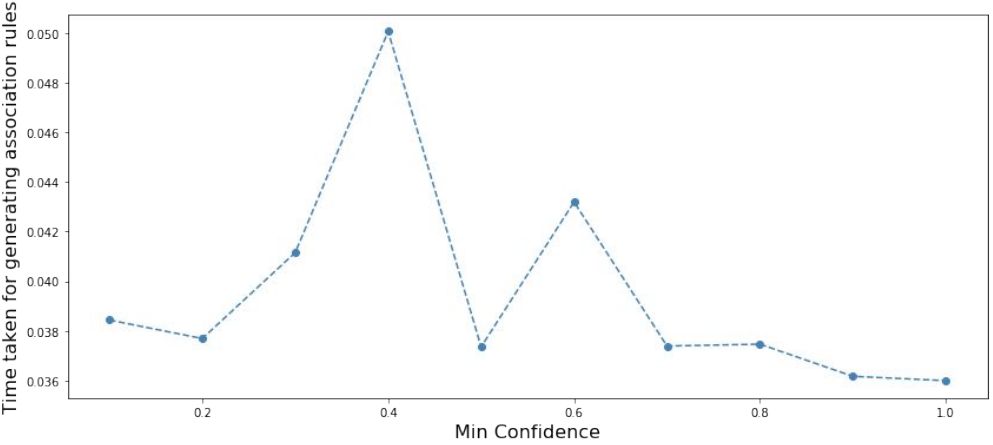
Why is it zig zag and not continuously decreasing?



Apriori with Reduced Dataset and HML labels



FP Growth with Reduced Dataset and HML labels

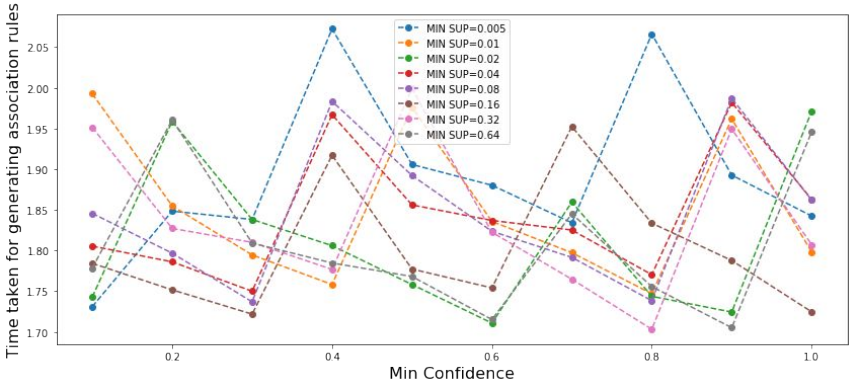


FP Growth with All Dataset and HML labels

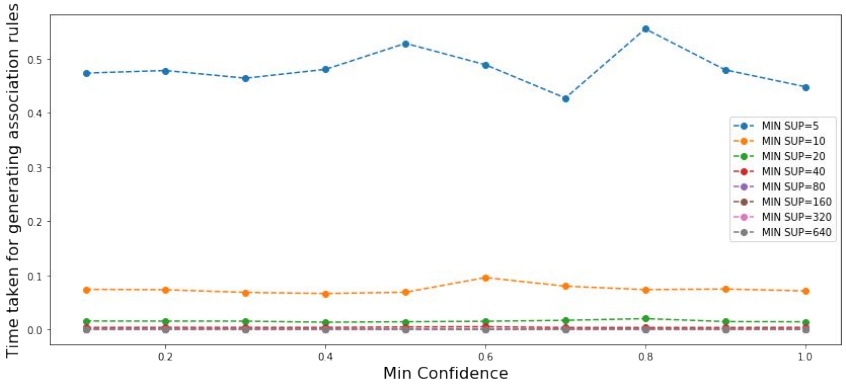
Why are they not continuously decreasing?

Is it just for this value of MIN_SUP?

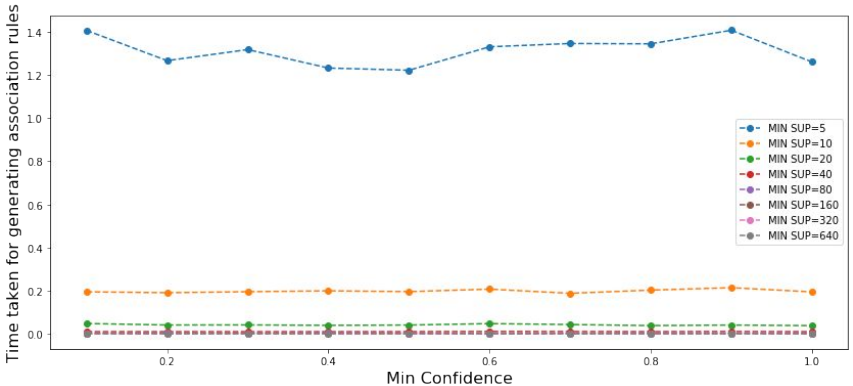
(MIN_SUP = 20)



Apriori with Reduced Dataset and HML labels



FP Growth with Reduced Dataset and HML labels



FP Growth with All Dataset and HML labels

Why are they not continuously decreasing?

Well, clearly the behaviour is persistent across different MIN_SUP values.

References

1) 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.
2) 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
3) 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.
4) Negar Hariri, Bamshad Mobasher, and Robin Burke. 2012. Context-aware music recommendation based on latent topic sequential patterns. In *Proceedings of the sixth ACM conference on Recommender systems* (RecSys '12). ACM, New York, NY, USA, 131-138
5) Fabio Aiolli. 2012. 2012 MILLION SONGS DATASET Challenge CODE. https://www.math.unipd.it/~aiolli/CODE/MSD/
6) 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

THANK YOU!