Mentors: Abhinav Garg and Jayant Agarwal

Hybrid Music Recommendation System

Because Music Never Hurts

<motivation>

MRS is of great help for music fans, making it easier for them to explore wide collection of songs from a huge ground of versatility and that too based on his/her taste and preferences.

Almost all streaming services such as Apple Music, Spotify, Google Play Music, etc. use recommendation systems to provide new song recommendations to their users.

In a typical Music Streaming Service, there are billions of songs, and millions of users. In a service like this, the main source of expansion is through relevant recommendations.

Spotify Stats

Subscribers: Over 60 million (as of July 2017)

Active users: Over 140 million (as of June 2017)

Number of songs: Over 30 million

Number of playlists: Over 2 billion

Source :- https://press.spotify.com/us/about/

<problem-statement>

For a given user ‘U’, provide at least ‘N’ suggestions most relevant to his previous listening history. The relevance of the suggestions is measured using truncated mAP (mean Average Precision). Aim is to provide new and most relevant recommendations.

Previous Works

Mostly three strategies used

- Collaborative Filtering
    1. Memory Based
       http://ceur-ws.org/Vol-964/paper12.pdf
    2. Model Based
       http://hpac.rwth-aachen.de/teaching/sem-mus-17/Reports/Madathil.pdf
- Content Based Recommendations
  https://papers.nips.cc/paper/5004
- Hybrid Methods
  https://waseda.pure.elsevier.com/en/publications/hybrid-collaborative-and-content

<collaborative-filtering>

- CF has been used extensively in recommender systems. It is still used as a top method among an ensemble of methods, although it suffers from cold-start.
- CF has been proved to be efficient in real-life scenarios as shown in both - Netflix Prize and Kaggle MSD Challenge

Problems	-	Sparsity, Scalability & Speed
Two Approaches	-	Model Based and Memory Based

Model Based

- The problem is approached using Matrix Completion
- Allows us to have a generative model
- Matrix Factorization allows us to learn a generative model
- Recommendation task fast, but difficult to add items or users

Memory Based

- In memory-based CF algorithms unknown entries of User-Item matrix are taken as zero
- The entire user-item matrix is used to generate a prediction
- Generally, given a new user for which we want to obtain the prediction, the set of items to suggest are computed looking at similar users.
- This strategy is typically referred to as user-based recommendation. The recommendations can also be based on item-item similarity

<content-based-recommendation>

- Based purely on description of user and item profiles
- Works on item-item similarity, usually modelled based on user preferences
- User feedback used to increase/decrease weights of certain attributes

<what-we-did>

- Ours was an attempt to make a hybrid recommender, based on exploitation (using CF) and exploration
- We tackle the three problems of CF (SSS) using clustering, prior to running Collaborative Filtering
- Along with this, we aim to make the method of collaborative filtering more scalable and fast using K-Means Clustering on the generated user features as opposed to clustering based on the user-item matrix (*Dakhel et al.*)

Dataset

- We have used the Taste Profile Subset which was previously used in the MSD Challenge (Kaggle)
- The dataset provides the user-song-count triplets, which tells us the number of times the user has listened to the song
- We are only concerned with the recall, and hence we ignore the count
- For training purposes, we have removed users which have listened to less than 50 tracks

Source :- https://labrosa.ee.columbia.edu/millionsong/

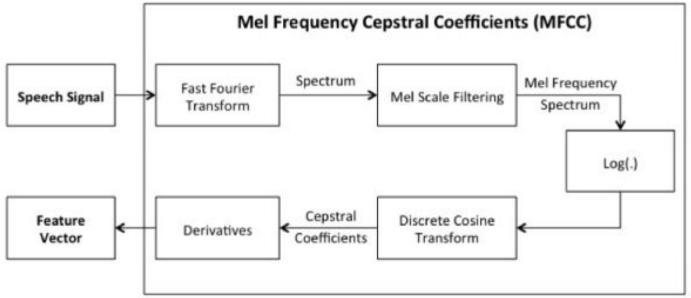
Exploitation

because money

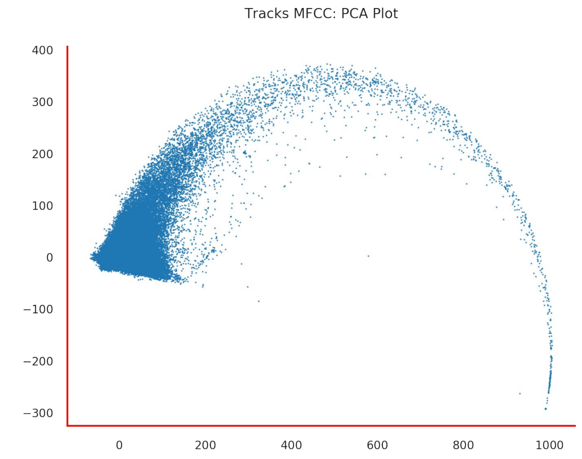
1. Cluster items/tracks on MFCC features using soft-assignment with diagonal covariance matrix
2. Compute user features as a “soft” bag of items (*Yoshii et al.*) using “weights of item clusters”
3. Cluster users using K-Means with diagonal covariance matrix
4. Use collaborative filtering to recommend items

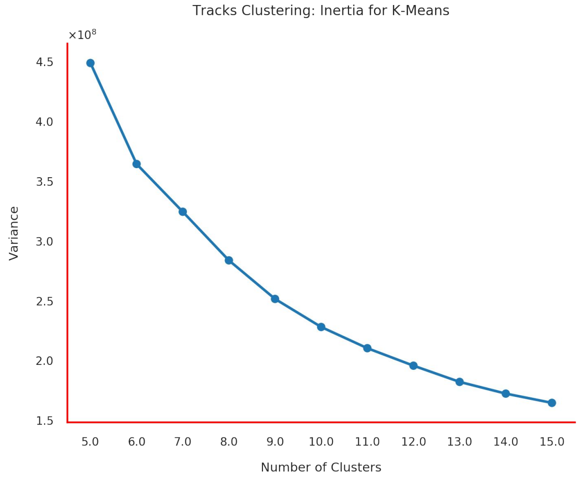
Items Clustering using MFCC

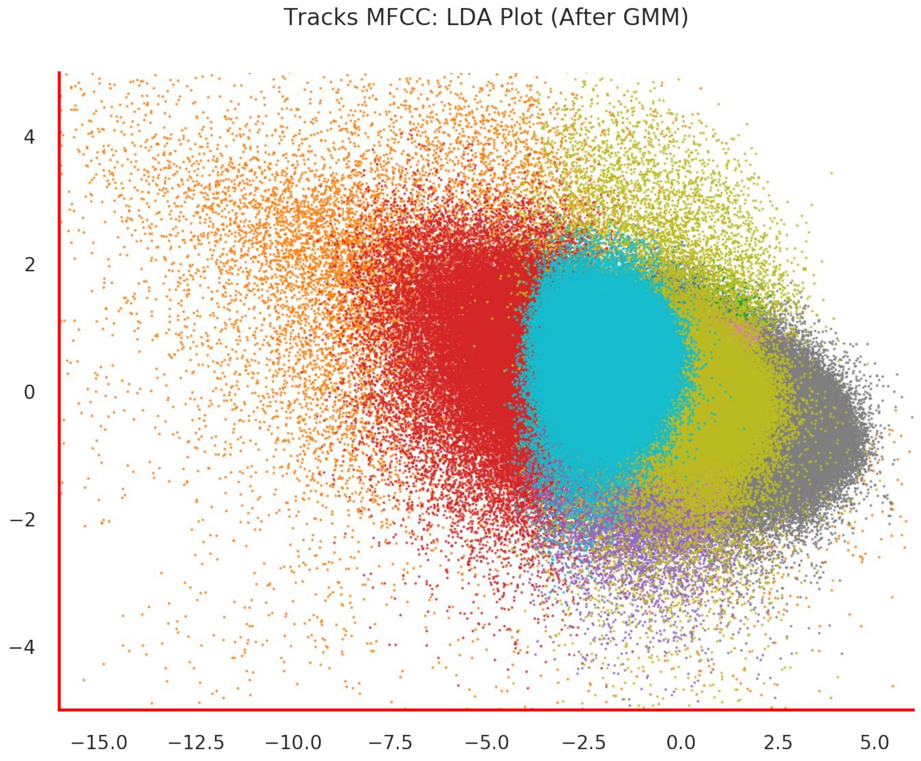
- Items are clustered using MFCC (Mel-Frequency Cepstral Coefficients) features extracted from the sound tracks
- MFCC mimics the logarithmic perception of loudness and pitch of human auditory system and tries to eliminate speaker dependent characteristics
- MFCC have been showed as a good approach to music classification (genres) (*Aucouturier et al.*)
- Number of Clusters = 10



Features obtained from
http://www.ifs.tuwien.ac.at/mir/msd/



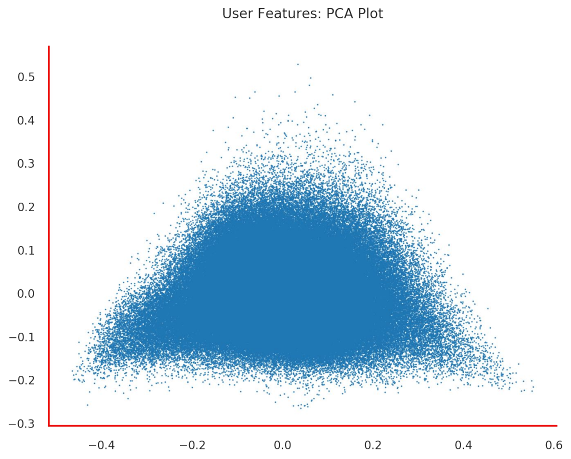


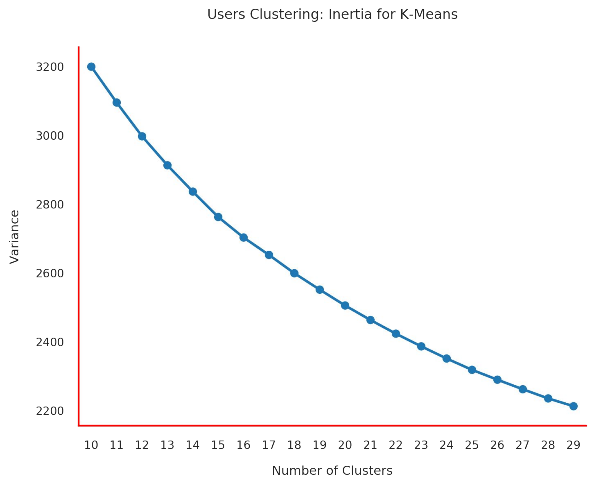


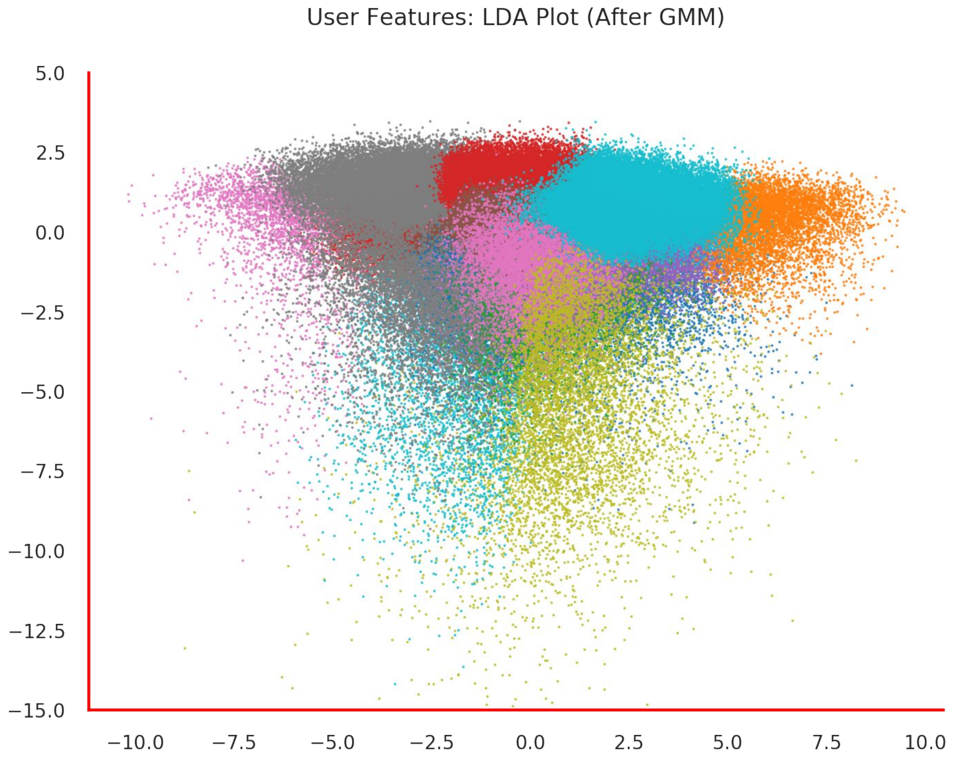
Choosing Number of Clusters - Items

User Clustering

- User features are computed by adding the cluster assignment probabilities of different
- Users are clustered using GMM as well
- Number of Clusters = 20







Choosing Number of Clusters - Users

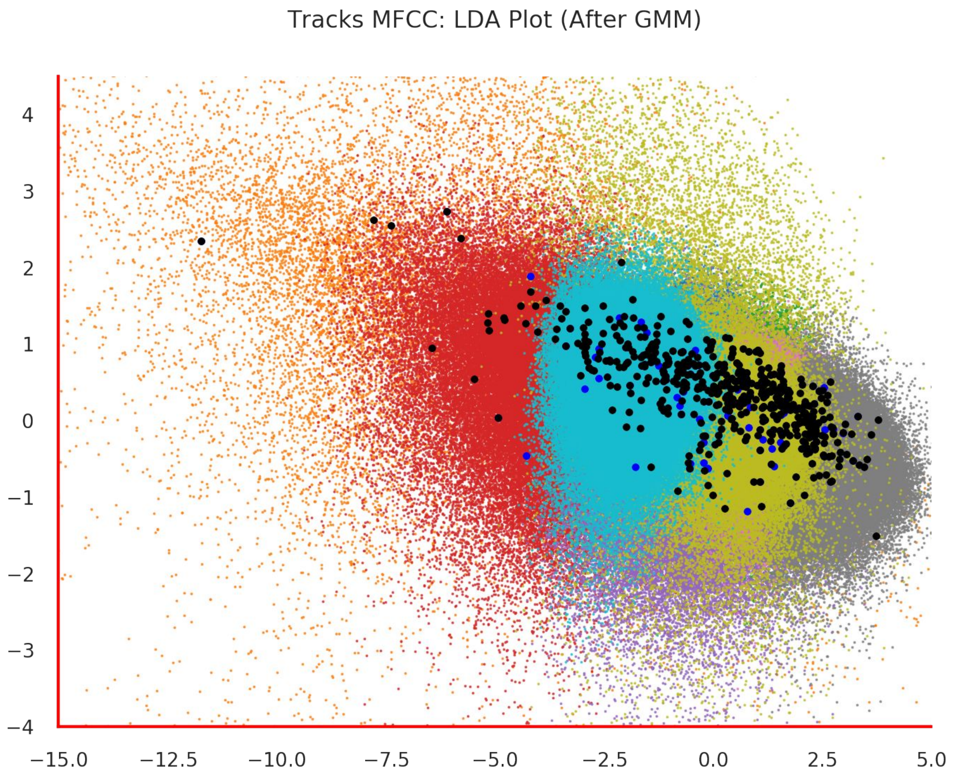
Recommendation

- The recommendations are made using Collaborative Filtering using User-User similarity
- The similarity metric used is Cosine Similarity (popularly used for memory based CF)

$$\text{similarity}(u_1, u_2) = \frac{\mathcal{U}_1 \cdot \mathcal{U}_2}{|\mathcal{U}_1||\mathcal{U}_2|}$$

- Also used localization ranking (*Dhanjal et al.*) which provided much better suggestions
- Accuracy tested using Precision&Recall Method as named by the MediaLite Library, which computes the mAP @ N (used often in RecSys), that is computed as the sum of precisions over the first *k* recommendations, averaged over the total N recommendations

Localization Exponent	mAP@500
1	0.032539860136
3	0.134488305391
6	0.275001780514



Choosing Number of Clusters - Users

Exploration

because more money

Opposite of what Multi-Armed Bandits does, which utilizes repeated purchases

1. Choose (sample) clusters of items (instead of items) using Multinoulli
2. For each cluster, rank the items based on cluster weights (probability) and popularity, and recommend the top item
3. Repeat until sufficient recommendations generated

Recommendation

- The cluster is chosen / sampled using a Multinoulli distribution, with the probability weights as

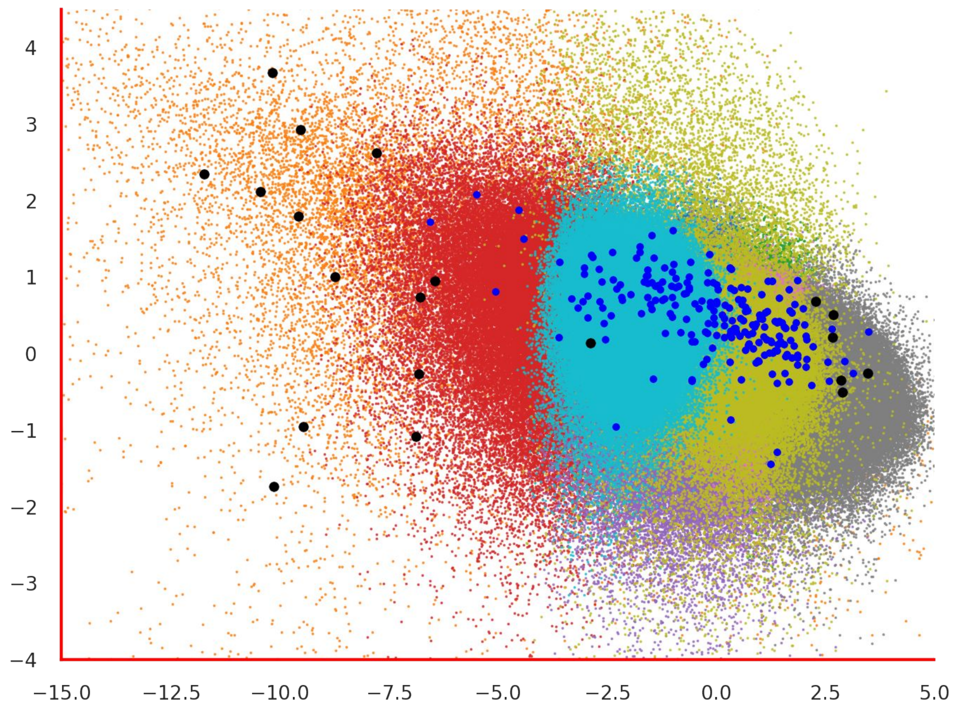
  sqrt(#(tracks in cluster) / # (tracks listened by the user belonging to this cluster))

- Within each cluster, the items are sorted (decreasing) according to the popularity and cluster weight

  #(users who listened to this track) X Prob [tracks belonging to this cluster]

- From each cluster, the top track, which hasn't been recommended or listened to yet is recommended

Tracks MFCC: LDA Plot (After GMM)



Choosing Number of Clusters - Users

Possible Extensions

- ➔ Use Content Based Recommendations to remove Cold-Start Problem
- ➔ Artist Level recommendations to improve exploration as well as an attempt to remove cold-start problem
- ➔ Use landmarking within user clusters, and perform collaborative filtering on them (could be highly scalable and fast)

Complete Python Notebook here

Fin!