Multi Agent Estimation of Musical Preferences

Milind Jagota, Erik Jones, Zhaoyu (Joe) Lou

***Abstract*—This paper focuses on the problem of "party playlist generation", where the ultimate goal is to generate a playlist for a group based on the listening histories of a group of individuals. We design a classification system which uses both frequency data and song metadata to accurately gauge song similarity. Comparing randomly generated playlists, a purely frequency based approach, and the combined frequency and metadata approach, we found that the addition of metadata resulted in higher quality playlists.**

I. Introduction

Suppose you walk into a party. The music is blaring, and you cringe as you realize your least favorite pop song is playing. Even worse, the whole night is probably going to be filled with songs like this. But you already know there's nothing you can do about it - there's one person who's DJing and their taste happens to not line up with yours. Every college student has been in this sort of situation, seeing as party hosts typically aren't professional DJs. Our project looks to develop a machine learning solution to this problem. Specifically, we want to design a system that can use the individual listening histories of many people to create playlists that will satisfy all of them in some optimal way.

The input of our system is a list of listening histories of some group of n individuals. Each listening history is comprised of some number of songs, with repeats indicating multiple plays. We then use a specially created model to output a new list of m songs, representing the optimized playlist for the group. There are two fundamental components to creating the model necessary to realize this task. First, we have to establish a metric to base song similarity comparisons on. Second, given that metric, we need to determine how to create a playlist that satisfies each member of the group in some optimal way. Previous approaches tended to either focus solely on the frequency data of the song or the metadata of the song, but not both; our system seeks to change that by combining both metadata and raw frequency data to generate the playlist. More precisely, we modeled songs based on 7 metadata features and the MFCC matrix, which contains the preprocessed frequency data. The MFCC matrices were modelled using a Gaussian Mixture Model and distances were defined by approximating the Kullback-Leibler divergence of the resulting distributions. After adding in the metadata features, we used a heuristic distance on a song dataset to get a list of candidates, and then use the full feature set with a more precise distance calculation to select our final playlist.

Our hope is that, ultimately, this project will render the incompetent DJ a remnant of the past. Music can unite people, and this project helps bring that potential into reality.

II. Related Work

Previous work in the area of similarity search and playlist generation has focused primarily on two distinct characteristics: acoustic, frequency based approaches, and subjective, metadata based approaches. We describe both, along with work towards algorithmic optimization, below.

Frequency based approaches generally make use of the Mel Frequency Cepstral Coefficients (MFCCs) of the song's acoustic waveform. Autocouturier and Pachet introduced the idea of discretizing a song into sets of MFCCs, modelling their distribution with a Gaussian Mixture Model, and approximating the Kullback-Leibler (KL) divergence between two songs' distributions using Monte-Carlo sampling to characterize the similarity. [1] Tang *et al.* use a similar approach but apply a different distance metric, the Earth Mover's Distance (EMD), which they claim more accurately approximates the KL divergence. [2] Pampalk *et al.* compare these two to a variety of other frequency based models and find that the EMD outperforms all other models. [3] However, it should be noted that EMD is highly computationally expensive even when optimized, which often limits its' applicability. [4] Berenzwig *et al.* showed that using either distance metric, the EM algorithm to generate the model can be replaced with a modified k-means algorithm followed by covariance calculations to decrease computation time significantly without loss in accuracy. Furthermore, they showed that using diagonal covariances both reduced instances of singular covariances and reduced overfitting. They compared 3 different distance metrics and also found EMD to perform best. [4] In later work, Pampalk proposed a modified distance metric, the Fast Spectral Similarity (FSS), which she shows has slightly worse performance than EMD but is an order of magnitude faster to compute. [5] This metric has become increasingly popular owing to it's high performance and speed.

In regards to the metadata based approaches, many approaches are quite similar to the approach of Cohen and Fan. They characterize a user's listening history according to a variety of human generated metadata features and search for other users who have similar histories. They then recommend songs which have not yet been listened to but are popular among similar users. [6] While frequency based approaches are popular in the academic community, collaborative filtering based on subjective features is generally used in industry applications. It is our goal to unify these two approaches into one.

III. Dataset and Features

A. Initial Scraping

We used a subset of the Million Songs Dataset [7] as our foundation for data generation. The Million Songs Dataset contains songs and a large number of quantitative and more qualitative features corresponding to each song. To access songs in the dataset and create sample listening histories, we modified code from [8]. To create song histories, we hand selected eighty songs from the subset that we felt would be somewhat recognizable for the general population. This recognizability was critical to get reliable survey results, but was nontrivial since the dataset was created more than six years before this project was implemented. We also created a dataset with 100 songs based on similar constraints to be used in the playlist generation step. Though there were many potential features in the million songs dataset, we ended up selecting eight: Tempo, Duration, Loudness, Familiarity, Hotness, Danceability, Energy, and the MFCC matrix. The first seven of these features are scalar metadata, while the eighth is a matrix containing frequency data.

B. MFCC Matrix

Frequency data is represented in the form of Mel Frequency Cepstral Coefficients (MFCCs), which have been shown previously to work well for audio classification. [9] [3] MFCCs are calculated by first taking the discrete time Fourier transform of a signal, then mapping the log powers of the spectrum onto the mel scale, which approximates human perception of acoustic power. Finally, the MFCCs are obtained by taking the discrete cosine transform of the result. We were provided the final results of this calculation on each song in the dataset in the form of a 12 x n matrix, where n is proportional to the duration of the song. Each column has the cepstral data for a single time interval spanning on the order of a few milliseconds. Because humans generally do not perceive audio at the millisecond level, we used the standard approach for processing these MFCCs espoused by the European Telecommunications Standards Institute (ETSI), in which consecutive MFCCs in a small sliding window spanning about 25 ms are concatenated into a single vector. [9] The entire song is thus represented by a set of these MFCC vectors, each of which consists of cepstral data over 25 ms. A heat map of one such MFCC matrix is shown in Figure 1. The rectangle shows one such range of MFCC coordinates that is concatenated after the aforementioned prepossessing.

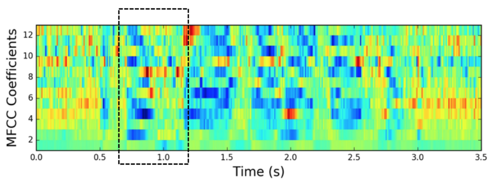


Fig. 1: A heat map of a sample MFCC matrix. The dotted black box shows consecutive MFCC vectors that we concatenate.

Ultimately, we want to convey the information in the MFCC matrix as a scalar distance metric. The process for doing so is detailed in the methods section.

C. Normalization

The eight features we selected have very different scales. Some features, like Hotness, operate on a 0 to 1 scale while others, like Tempo, are in the 100s. We needed to normalize the features for the Euclidean distance to be reasonable. To normalize we followed the following process:

1) Extract average feature values for different sets of songs
2) Normalize with respect to the inverse some distance metric within features

This gave us a good starting point. From this point, we manually tuned the weights further by systematically adjusting them based on outputs on a training example and iteratively testing different combinations.

IV. Methods

At a high level, we implemented k-means clustering using a weighted euclidean distance on the songs. The scalar metadata features lent themselves easily to this approach; however, it was considerably more difficult to define a sensible scalar distance between MFCC matrices. The majority of the methods we developed were aimed at addressing this problem.
Motivated by previous work by Autocourtier and Pachet [1], we chose to model the distribution of the MFCC vectors using a Gaussian Mixture Model. Given the model for two songs, we could then calculate the KL divergence between their two distributions as the distance between their MFCC matrices. However, no closed form solution or efficient algorithm for calculating the KL divergence between two Gaussian Mixture Models is currently known, so it was necessary to look for reasonably accurate alternatives.

A. Fast Spectral Similarity

The most exact approach to approximating the KL divergence between the two Gaussian Mixture Models for two songs would be to compute the "generation probability", the probability that the songs were generated from each others' distributions. More precisely, we would take all the MFCC vectors in song A and compute the log likelihood that those vectors were drawn from the model for B, add the log likelihood that the vectors for B were drawn from the model for A, and normalize by subtracting the log likelihood of the vectors for A and B being drawn from their own models. Letting S^A be the set of MFCC vectors for song A, we are thus computing

$$\begin{aligned} GP(M^A, M^B; S^A, S^B) = \ell(S^A|M^B) + \ell(S^B|M^A) \\ -\ell(S^A|M^A) - \ell(S^B|M^B) \end{aligned}$$

We let x_i^A be the i-th MFCC vector in song A, N_A be the total number of MFCC vectors, k_A is the number of Gaussians in model A, P_j^B be the prior probability of the j-th Gaussian in model B, and $\mathcal{N}(x|M_j^B)$ to be the probability of drawing x from that j-th Gaussian of model B. Then we can use Bayes rule to define

$$\ell(S^A|M^B) = \frac{1}{N^A} \sum_{i=1}^{N_A} \sum_{j=1}^{k_B} P_j^B \mathcal{N}(x_i^A|M_j^B)$$

However, given the number of vectors in each song ($10^5 - 10^6$), it is impractical to calculate these likelihoods. Therefore, it is necessary to sample some of the MFCC vectors and approximate the generation probability using that sample. But a random sample has a small probability of being a poor representation of the true distribution, which would cause large error in the calculated distance.

A solution to this sampling problem is presented by Pampalk. He proposes a modified version of this method in which the same general formula is used but instead of sampling randomly, the means of the model are used. [5] Thus, we have

$$\begin{aligned} FSS(M^A, M^B; S^A, S^B) = \ell(S^A|M^B) + \ell(S^B|M^A) \\ -\ell(S^A|M^A) - \ell(S^B|M^B) \end{aligned}$$

Where, by applying the Law of Total Probability twice and defining μ_i^A as the i-th mean in model A, we have

$$L(S^A|M^B) = \sum_{i=1}^{k_A} P_i^A \log \sum_{j=1}^{k_B} P_j^B \mathcal{N}(\mu_i^A|M_j^B)$$

FSS is fast to compute in comparison to other methods (in particular the Earth Mover's Distance discussed in [2]), and is still a good approximation to the distance.

A sample heatmap of FSS distances between 5 songs is shown in Figure 2. The songs are:

0) Rihanna: Don't Stop the Music
1) Kanye West: Through The Wire
2) Nirvana: Heart-Shaped Box
3) Kanye West / Adam Levine: Heard 'Em Say
4) Rihanna: Music Of The Sun

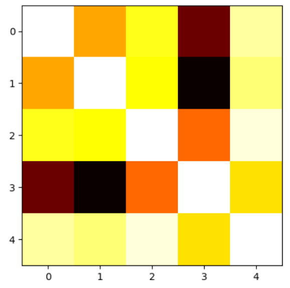


Fig. 2: A sample heatmap of the Fast Spectral Similarity matrix for five songs.

Those familiar with these songs will recognize that these distances are reasonable; the major outlier is Heard 'Em Say, which makes sense given it's unique beat and high frequency piano component. This demonstrates that our work to convert MFCC matrices into a usable feature has been successful.

B. Centroid Distance

Another approach which yields a fast and simple approximation to the KL divergence between two Gaussian Mixture Models is the centroid distance, which is defined simply as the sum of distances between all means in song A to all means in song B. Mathematically, the centroid distance between models M^A and M^B is defined

$$CD(M^A, M^B) = \sum_{i=1}^{k_A} \sum_{j=1}^{k_B} ||\mu_i^A - \mu_j^B||$$

While very fast, this approach suffers from some inaccuracy since it discards information about the covariances of the data. We thus only use it as a heuristic in our system.

C. Collapsing K-means

Ordinarily the parameters of the Gaussian Mixture Model would be fit using the EM algorithm; however, the EM algorithm is fairly computationally heavy so a faster alternative was desired. Berenzwig *et al.* found that a k-means algorithm can be used as a faster alternative to estimate the means of the Gaussians. However, K-means clustering has the inherent shortcoming that one must specify K, the number of clusters, and there is no way to know *a priori* how many different clusters of tones should be in a song. Imposing a single value for all songs is therefore a large constraint. Motivated by the work of Chen [10], we decided to use a modification called the collapsing k-means algorithm for mean selection. To implement collapsing k-means we run standard k-means with a large number of clusters, but after each iteration if two centroids are closer to each other than a certain threshold we merge their clusters. Merged clusters can't be merged again in the same iteration.

The algorithm proceeds as follows, given a training set $\{x^{(1)}, x^{(2)}, ...x^{(n)}\}$ to cluster and a merging threshold α.

1) Initialize k cluster centroids $\{\mu_1, \mu_2, ...\mu_k\}$ randomly.
2) Repeat until convergence:
    a) For every *i*, set
    $$c^{(i)} := \arg\min_j (||x^{(i)} - \mu_j||^2)$$
    b) For every *j*, set
    $$\mu_j := \frac{\sum_{i=1}^n 1\left\{c^{(i)} = j\right\} x^{(i)}}{\sum_{i=1}^n 1\left\{c^{(i)} = j\right\}}$$
    c) For every *k*,*l*, if
    $$||\mu_k - \mu_l||^2 < \alpha$$
    then for all *i* such that $c^{(i)} = l$, set $c^{(i)} = k$. Then set
    $$\mu_k := \frac{\sum_{i=1}^n 1\left\{c^{(i)} = k\right\} x^{(i)}}{\sum_{i=1}^n 1\left\{c^{(i)} = k\right\}}$$
    and remove μ_l.

D. Implicit k-means

We required an algorithm to cluster a list of songs, using a list of scalar features as well as the fast spectral similarities between each songs. Standard k-means fails here since we don't have a way to create an MFCC matrix for a cluster centroid. We developed implicit k-means as a solution to this challenge. The algorithm proceeds as follows, given a training set $\{x^{(1)}, x^{(2)}, ...x^{(n)}\}$ to cluster.

1) Calculate the *n*x*n* matrix *S* of fast spectral similarities.
2) Initialize k cluster centroids $\{\mu_1, \mu_2, ...\mu_k\}$ randomly by choosing k of the MFCC vectors.
3) Initialize k cluster MFCC distance vectors $\{s_1, s_2, ...s_k\}$ by choosing the columns of S of the corresponding indices of the mean initialization.
4) Repeat until convergence:
    a) For every *i*, set
$$c^{(i)} := \arg\min_j(||x^{(i)} - \mu_j||^2 + ||s_j||^2)$$
    b) For every *j*, set
$$\mu_j := \frac{\sum_{i=1}^n 1\left\{c^{(i)} = j\right\} x^{(i)}}{\sum_{i=1}^n 1\left\{c^{(i)} = j\right\}}$$
$$s_j := \frac{\sum_{i=1}^n 1\left\{c^{(i)} = j\right\} S_{,i}}{\sum_{i=1}^n 1\left\{c^{(i)} = j\right\}}$$

The distortion function is changed to
$$J = \sum_{i=1}^n ||x^{(i)} - \mu_{c^{(i)}}||^2 + ||s_{c^{(i)}}||^2$$
Where $S_{,i}$ denotes the *i*th column of *S*. In words, we approximate the Fast Spectral Similarity between a centroid and each song by the mean of the Fast Spectral Similarities between every song in the cluster and that other song. We use this approximation at every step in the k-means algorithm. While we didn't prove that this is strict coordinate descent on our new J, the algorithm reliably converged on our test cases.

V. System Design and Results

A. Models

Using the algorithms described in the methods section, we created three different models to evaluate: a baseline model which randomly selected songs, a frequency based approach which used only MFCC data, and our proposed model incorporating MFCCs and metadata. The design of the latter two systems is described in more detail below and shown in Figure 3.

1) Take in *N* user histories (lists of songs) and combine them into a weighted single song list. Weighting is accomplished by duplicating each history by the least common multiple of all the history lengths.
2) Transform each song's MFCC matrix into a series of concatenated vectors and run collapsing k-means to obtain centers for the mixture of Gaussians model.
3) Assuming a diagonal covariance matrix, calculate variances for each cluster and maximize priors to obtain a full mixture of gaussians model.
4) Run implicit k-means on the weighted list of songs using the scalar features from the dataset and the FSS between songs' GMMs or only the FSS between the GMMs, depending on the model. This k-means clustering outputs a series of centroids representing the interests that the combined song list was generated from.
5) Taking in a song database, narrow down the database to a list of candidates using centroid distance as a heuristic. Specifically, for every song in the database, calculate the distance to every centroid of the combined song list, using the mean of centroid distances for the last component. Pick the $1.5m$ songs with the lowest distances as candidate songs.
6) To select the final set of songs, repeat the above process on the list of candidates using the fast spectral similarity for the last component and choose the m songs with the lowest distance.

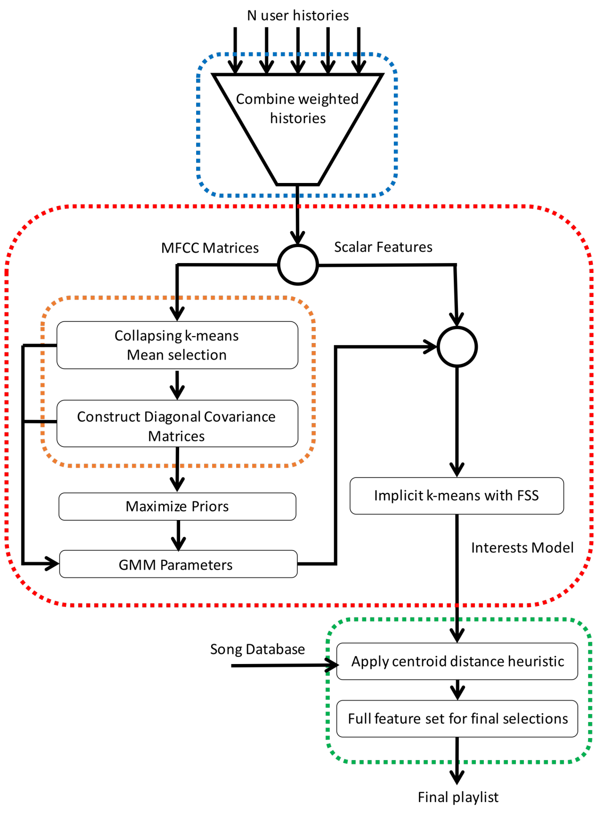

N user histories
Combine weighted histories
MFCC Matrices
Scalar Features
Collapsing k-means Mean selection
Construct Diagonal Covariance Matrices
Maximize Priors
GMM Parameters
Implicit k-means with FSS
Interests Model
Song Database
Apply centroid distance heuristic
Full feature set for final selections
Final playlist

Fig. 3: Flowchart showing organization of system.

B. Hyperparameter selection

Our system had numerous hyperparameters to tune. To select the max number of clusters and collapsing threshold for the collapsing k-means algorithm, we estimated that a song should have around one to ten clusters in the concatenated MFCC vectors space. This number was estimated intuitively by considering how many different moods a typical song has, and how much sound varies within a single mood. We therefore set a ceiling of fifteen clusters and tuned the collapsing threshold so that most songs finished with one to ten clusters.

The k-means on the combined song list was run with four clusters. We chose this value because we found having more genres than this in a single playlist made the playlist incohesive.

We also had to determine a feature weighting for the k-means on the combined song list - this is discussed in the normalization subsection of the dataset and features section.

C. Testing procedures and evaluation metrics

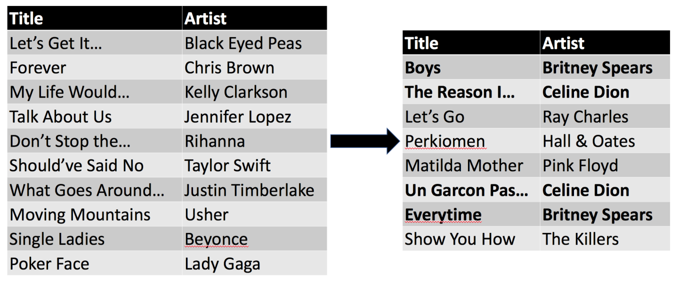
To test our system, we developed four different sample song histories of twenty songs each, each representing a group of users with different listening trends. We used our three models (random generation, frequency based, and frequency + metadata) to generate playlists for each history.

The inherent subjectivity of music similarity combined with the fact that we were using an unsupervised learning algorithm with no ground truths made evaluation difficult. Indeed, we found no clear consensus in the research literature on the best way to evaluate metrics of song similarity. However, Berenzweig *et al* examined multiple different popular evaluation metrics and found that surveying independent music listeners about artist similarity is an acceptable evaluation metric. [4] We decided to follow this idea and survey about song similarity to evaluate our system.

We surveyed twenty people about each of our test outputs, asking which songs in the output were similar to the seed. Because the dataset we used is relatively old (published 2011), most of the people we surveyed were only familiar with two of the four seeds, consisting of major pop and rock songs. As a consequence, we were only able to report results for those two.

D. Results and Discussion

The first ten songs of the pop seed and the playlist generated are shown in figure 5 as an example of the outputs from our system. Bold songs in the output were labeled as fitting with the seed by more than half of people surveyed.



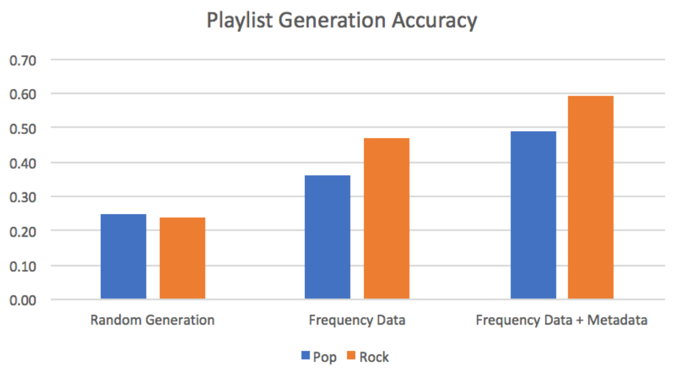
Title	Artist
Let's Get It...	Black Eyed Peas
Forever	Chris Brown
My Life Would...	Kelly Clarkson
Talk About Us	Jennifer Lopez
Don't Stop the...	Rihanna
Should've Said No	Taylor Swift
What Goes Around...	Justin Timberlake
Moving Mountains	Usher
Single Ladies	Beyonce
Poker Face	Lady Gaga

Title	Artist
Boys	**Britney Spears**
The Reason I...	**Celine Dion**
Let's Go	Ray Charles
Perkiomen	Hall & Oates
Matilda Mother	Pink Floyd
Un Garcon Pas...	**Celine Dion**
Everytime	**Britney Spears**
Show You How	The Killers

Fig. 4: Pop seed and corresponding output. Bold songs were indicated to fit seed by survey.

Our full survey results for the two genres where we could collect enough survey data are shown in Figure 5. By using only frequency data, our increase in accuracy over random selection is similar to that observed by Berenzweig *et al* (slightly under two times) [4]. Furthermore, adding in metadata gave noticeable improvements for both seeds over only using frequency data. These results support our initial hypothesis that including metadata adds valuable information about high level song structure that can improve performance of song similarity measures.

We also manually analyzed the outputs of our other two seeds (old rap and new rap/EDM) by listening to the outputted songs to look for any obvious problems. Our analysis illuminated one shortcoming of our system: there was a very small amount of old rap present in our song database. While our system selected a good percentage of the old rap that was available for it to choose from, this still made up only a small percentage of the final output playlist. This issue would likely not be as large if we used a larger song database; however, even at scale, this shows that our system may not work well for users with very narrow or niche listening histories.



Genre	Seed Size	Output Size	Random Generation	Frequency Data	Frequency Data + Metadata
Pop	20	8	0.25	0.36	0.49
Rock	20	8	0.24	0.47	0.59

Fig. 5: Fraction of output songs that fit seed based on survey.

VI. Conclusion and Next Steps

We developed a system to generate party playlists given a set of user song histories. We used various unsupervised learning algorithms, including two modified versions of the k-means algorithm and approximations of the EM algorithm. We found the accuracy of using our system with only frequency information to be on par with previously published song selection methods. Adding high level metadata gave a noticeable increase in performance, showing the importance of including the high level structural information that the metadata captures.

In the future, we'd like to examine multiple evaluation metrics and see how consistent the accuracy of our system is. We'd also like to try out our system on a more modern dataset to make evaluation easier and allow the system to be more usable. The system is still computationally expensive and it would be nice to further optimize both the machine learning algorithms and the system structure, again with the goal of coming closer to a usable product. Finally, it would be interesting to integrate with the Spotify or Apple Music APIs to see what other types of data might be available.

VII. Contributions

A. Erik

Erik developed the project idea along with Milind. He was responsible for both selecting appropriate songs for the final song dataset and most of the experimental sets, and he modified and wrote new code to extract them from the Million Songs Dataset, and write them into files. Erik also did research on how to appropriately normalize and weight different features for inter-song clustering, and then normalized the weights enabling productive optimization. Along with Milind, he implemented the reducing k version of k-means for clustering a song's MFCC cloud vector and, along with Joe he both identified and debugged the notorious singular covariance bug. The vast majority of this project was spent in long group sessions where the project-team brainstormed, tediously ran code, and debugged. Erik, along with everyone else in the group, was an active participant.

B. Milind

Milind developed the project idea along with Erik, found the dataset, and designed overall structure of the system (clustering of songs, followed by an initial and intensive search through the database). He helped out Joe a little with researching ways to process the MFCC matrix. Along with Erik, he implemented the reducing k version of k-means for clustering a song's MFCC clouds vectors. He also identified the problem of clustering songs while including the MFCC matrix and came up with the implicit k-means algorithm as a solution. He then implemented and tested this algorithm. He developed the final test sets that we report results for. He participated in all the coding sessions towards the end of the quarter where the system was tested and debugged extensively, and helped with parameter optimization.

C. Zhaoyu (Joe)

Joe researched methods for modeling songs based on their MFCC coefficients and was the one to propose the mixture of Gaussians model. In the process of researching this model he also found and proposed the modified k-means algorithm as an efficient alternative to the full EM algorithm. He was the one to evaluate the various different distance metrics and the one who learned about and decided on the Fast Spectral Similarity method. On the implementation front, Joe wrote the code for calculating both the Fast Spectral Similarity and the heuristic centroid distance of two GMMs as well as the code for formatting the MFCC matrix into a point cloud. He also wrote and debugged the code for the aggregation of user histories and the generation of a playlist based on the cluster model. In the later stages of the project he managed the codebase and refactored code as necessary to maintain readability and make debugging easier. Like Milind and Erik, he participated in all the coding sessions and helped with debugging and parameter search.

References

[1] J.-J. Aucouturier, F. Pachet *et al.*, "Music similarity measures: What's the use?" in *ISMIR*, 2002, pp. 13–17.

[2] Y. Tang, Y. Cai, N. Mamoulis, R. Cheng *et al.*, "Earth mover's distance based similarity search at scale," *Proceedings of the VLDB Endowment*, vol. 7, no. 4, pp. 313–324, 2013.

[3] E. Pampalk, S. Dixon, and G. Widmer, "On the evaluation of perceptual similarity measures for music," in *of: Proceedings of the sixth international conference on digital audio effects (DAFx-03)*, 2003, pp. 7–12.

[4] A. Berenzweig, B. Logan, D. P. Ellis, and B. Whitman, "A large-scale evaluation of acoustic and subjective music-similarity measures," *Computer Music Journal*, vol. 28, no. 2, pp. 63–76, 2004.

[5] E. Pampalk, "Speeding up music similarity," *2nd Annual Music Information Retrieval eXchange, London*, 2005.

[6] W. W. Cohen and W. Fan, "Web-collaborative filtering: Recommending music by crawling the web," *Computer Networks*, vol. 33, no. 1, pp. 685–698, 2000.

[7] T. Bertin-Mahieux, D. P. Ellis, B. Whitman, and P. Lamere, "The million song dataset," in *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, 2011.

[8] tbertinmahieux, "Msongsdb," https://github.com/tbertinmahieux/MSongsDB, 2011.

[9] E. Standard, "Speech processing, transmission and quality aspects (stq); distributed speech recognition; front-end feature extraction algorithm; compression algorithms," 2003.

[10] H. Chen, *Collapsing K-means Clustering*, 2016. [Online]. Available: https://books.google.com/books?id=nMGjnQAACAAJ