Clustering of Low-Level Acoustic Features
Extracted from Music

Çağatay Çalli
Department of Computer Engineering
Middle East Technical University
Ankara, Turkey
Email: cagatay@ceng.metu.edu.tr

Abstract—In this paper, we define a framework to measure the similarity of sound characteristics in a given music library. Low-level features are extracted from audio signals and clustered by X-means algorithm to learn the number of clusters. This number of clusters and the mapping of song frames to these clusters have a good correlation with musical transitions that a human listener perceives. We present our clustering results, our explanations for these results and the challenges we faced during the implementation of our framework further in the paper.

I. INTRODUCTION

With the improvements on data compression, distribution, storage and music production techniques, music information retrieval (MIR) tools are becoming crucial. Music recommendation (MR) is one of the biggest problems in this domain. The task of music recommendation systems (MRS) is to rank a musical collection according to a query, past listening behavior or similarity with the other songs in the collection.

Over the past decades, a wide range of music recommendation systems utilizing ID3 tag information (e.g. song title, artist, album, genre) together with information gathered from the Web or the environment appeared. These systems remained simple and ineffective because of incomplete ID3 information and the inability to capture the musical meaning and mood of the songs.

Content-based features extracted from the audio signal have to be utilized to build a robust recommendation system. There exist MR systems that employ acoustic features together with ID3 features to compute the similarity between songs. Although these systems have better accuracy over simple ID3-based systems, they cannot be deployed easily because of their computational complexities. Another issue with the current systems is the inability to make use of local feedback (e.g. time of skipping).

With this motivation, we investigate the effectiveness of X-means clustering algorithm to compute musical similarity and detect the nature of musical instruments and characteristic variances among them. We inspect songs by a frame-level analysis to capture local characteristics and significant changes between partitions of any song. The most important aspect of such a method is the possibility of using local feedback to capture partial user tastes for songs. With this approach, we will be able to model cases of liking/disliking particular elements in a song.

II. PROBLEM DEFINITION

Music similarity computation task can be defined in two parts. The first part is extracting the features that correctly represent all prominent sounds in one song. Measurements for a group of songs should be joined in a main file representing all the frames belonging to the group. The second part of the task is clustering these frames with an algorithm that can provide a close approximation to the true value of K, the number of clusters. Such an algorithm is needed because we do not have any ground truth for these audio frames. We want to capture the character of frame series, ordered sequentially in one song.

In this problem our sub-goals are defined as:

• Calculating statistical features that can represent the shape of the signal
• Removing irrelevant features to reduce dimensionality
• Clustering all frame data
• Measuring the correspondence of cluster characteristics with perceptual transitions in our music library

III. APPROACH AND ALGORITHM

The first step in feature extraction from an audio signal is framing. A 44100 kbps stereo audio signal has 44100 samples per second for left and right channels separately. We have to specify a frame size that is sufficient to capture the smallest significant sound (minimum length note) in the song. Since we did not focus on obtaining a beat histogram in our work, we cannot apply a dynamic frame size for different segments of the song. We decided to pick a value that approximately fits the most general cases of minimum length notes. A frame size between 50-150 milliseconds seemed appropriate for our feature extraction algorithm after our observations on related work [1]. We decided to use 2500 samples per frame. A frame size of 2500 samples corresponds to 130.61 milliseconds.

Musical audio signals generally have a temporal characteristic that should be captured by windowing the signal with a Gaussian function such as Hamming window [2] and by overlapping the frames with some rate to avoid disconnected of windowing effects. However, for both speeding up our feature extraction and reducing implementation complexity, we decided not to apply windowing on our inputs.
We decided to extract features that mainly belong to MPEG-7 descriptors. However, we also included a number of statistical descriptors to increase our representation accuracy. These features are explained in Data Characteristics section of our paper.

After generating these frame-based measurements, we needed an algorithm to learn the nature of this data. Since we did not have any labels for our data and all our data consisted of numerical values (expect timestamps of frames), we decided that clustering was the best option for this task.

We simplified our clustering problem to attribute-based clustering. The most popular algorithm to approximate the number of clusters K is X-means algorithm [3] so we decided to use this algorithm.

A. Clustering Acoustic Features

Measuring the similarity of the song frames and clustering this large dataset is not a trivial task. Traditional clustering algorithms like K-means or K-medoids is not very suitable for our data because the number of centroids may always change and not initially known. These centroids identify groups formed by an unknown criteria (such as genre, instrument label, vocal type, tempo etc.) most of the time. We think that it is best to use X-means algorithm in such a case of unidentified cluster characteristics. Since x-means algorithm is an extension over k-means algorithm, we are going to provide a definition of k-means algorithm first.

1) K-means Algorithm: First described in 1967 by J. MacQueen [4]. k-means algorithm is one of the simplest and most effective method to analyze metric data. Here is the algorithm in its basic form:

Algorithm 1 K-means Algorithm

1: \{Make initial guesses for the means \( m_1, m_2, \ldots, m_k \)\}
2: while any mean \( m \) has changed do
3: \{Use the estimated means to classify the samples into clusters\}
4: for \( j \) from 0 to \( k \) do
5: \{Replace \( m_j \) with the mean of all of the samples for cluster \( j \)\}
6: end for
7: end while

Initialization is an important step that we did not define in our pseudo-code. Most popular initialization method is choosing \( k \) samples from the data at random. Variations of k-means algorithm using different initialization strategies exist. k-means algorithm is known to converge to a local minimum. It is also considered to be too slow to use with practical databases. A bad initialization can also cause poor results in k-means algorithm since it is strictly deterministic after initialization phase.

2) X-means Algorithm: In [3], X-means algorithm is proposed to deal with user supplied number of clusters and local minima problem. It uses Bayesian Information Criterion (BIC) score comparisons to estimate the number of clusters. X-means algorithm is a simple yet very effective algorithm. Here is the pseudo-code of the algorithm:

Algorithm 2 X-means Algorithm

1: while \( k \) changed and \( k < k_{\text{max}} \) do
2: \{Run K-means algorithm until convergence\}
3: for each cluster do
4: \{Split centroid into 2 children\}
5: \{Move centroids to a random opposite direction to each other, by a distance proportional to the size of the cluster\}
6: \{Run a local k-means with \( k=2 \) inside cluster\}
7: \{Calculate BIC\( (k=1) \) and BIC\( (k=2) \)\}
8: if BIC\( (k=1) < \) BIC\( (k=2) \) then
9: \{Use the previously calculated 2 child clusters and their centroids to replace the original cluster and its centroid\}
10: else
11: \{Use the original centroid for that cluster\}
12: end if
13: end for
14: end while
15: return The best clustering among all \( k \) values by comparing BIC scores of k-means clusterings

Equations 1 and 2 are used in the BIC score calculation for one centroid, where \( I_j(D) \) is the log-likelihood of the data according to the \( j^{th} \) model and taken at the maximum likelihood point, and \( p_j \) is the number of parameters in \( M_j \). Number of free parameters \( p_j \) is the sum of \( k-1 \) class probabilities, \( M \times K \) centroid coordinates, and one variance estimate. \( R \) is the total number of points which belong to the centroids under consideration. The maximum log likelihood estimate for the variance \( \sigma^2 \) is spherical since K-means algorithm assumes data varies equally in every direction. Equation 3 defines \( \sigma^2 \) mathematically.

\[
\hat{\sigma}^2 = \frac{1}{R-K} \sum_i (x_i - \mu(i))^2
\]

Fig. 3. Equation (3)

Aside from its main advantage of automatically choosing the whether to increase the number of centroids and preventing
exhaustive test of every possible k value; another advantage of this approach is its strength against local minima problem. Regional k-means with 2 centers is found to be less sensitive to local minima.

Fig. 4. (a) Splitting original centroids into 2 children (b) Applying 2-means in parallel (c) Result after all parallel local 2-means terminate (d) Surviving configuration after applying BIC tests

Steps of a sample run of X-means algorithm is presented in Figure 4. In Figure 4(a) every original centroid is splitted into 2 children in random directions. The distances of these children from the original centroid varies according to the region that they cover. In Figure 4(b), local 2-means is applied in parallel to all three regions and the location of the children change according to the local K-means execution. Next in Figure 4(c), all parallel 2-means have converged and BIC values for the children are calculated for every region. Finally in Figure 4(d) the output of this X-means execution is shown.

IV. DATA CHARACTERISTICS

Frame-based acoustic attributes used in our system are as follows:

Arithmetic Mean Mean value of the raw audio input signal.
Variance Variance of the raw audio input signal.
Standard Deviation Standard deviation of the raw audio input signal.
Skewness A measure of the asymmetry of the raw audio input signal (the third standardized moment about the mean).
Zero Crossing Rate The rate of sign changes along the audio input signal.
Magnitude Mean Mean value of the magnitude spectrum.
Magnitude Variance Variance of the magnitude spectrum.

Magnitude Standard Deviation Standard deviation of the magnitude spectrum.
Magnitude Skewness A measure of the asymmetry of the magnitude spectrum.
Spectral Centroid It represents whether the signal is dominated by low or high frequency signals. It is computed with the formula below where $P_x(k)$ is a spectrum coefficient.

$$C = \sum_n \log_2\left(\frac{f(n)/1000}{\sqrt{\sum_n P_x(n)}}\right)$$

Fig. 5. Spectrum Centroid Formula

Flatness It provides a measure of signal flatness by geometric mean/arithmetic mean ratio of the signal within a frequency bin. It is computed with the formula below. $c(i)$ is the mean value of a group of spectrum coefficients. $ih(b)$ and $il(b)$ represent the highest index and lowest index of the group $b$ respectively.

$$SFM_b = \frac{1}{ih(b) - il(b) + 1} \sqrt{\prod_{i=il(b)}^{ih(b)} c(i)}$$

Fig. 6. Spectrum Flatness Formula

Roll-off It is a measure of spectral shape which is used to distinguish between voiced and unvoiced speech.[5] The frequency below which %85 of the accumulated magnitudes of the spectrum is concentrated. It is computed with the formula below:

$$\sum_{k=1}^{K} |X(k)| \leq C \cdot \sum_{k=1}^{K} |X(k)|$$

Fig. 7. Spectrum Roll-off Formula

Harmonic Mean The arithmetic mean of the harmonic spectrum.
Harmonic Variance Variance of the harmonic spectrum.
Harmonic Standard Deviation Standard deviation of the harmonic spectrum.
Harmonic Centroid This is like Spectral Centroid, it is computed for the harmonic spectrum in this case.
Harmonic Flatness This is like Flatness, it is computed for the harmonic spectrum in this case. It computes this by using the coefficients of the harmonic spectrum to find the geometric mean/arithmetic mean ratio.
Harmonic Roll-off This is like Roll-off, it is computed for the harmonic spectrum in this case.

Tristimulus-1 This measures the relative weight of the first harmonic to the others. [6] It is computed with the formula below:
\[ Tr_1 = \frac{A_1}{\sum_{n=1}^{N} A_n^2} \]

\[ Fig. 8. \text{ Tristimulus-1 (tr}_1\text{) Formula} \]

Harmonic spectrum is a collection of frequency coefficient defining the shape of the signal. The frequencies it maps with the coefficients are multiples of the fundamental frequency of the signal.

\section{V. Implementation}

\subsection{A. Feature Extractor}

Our feature extraction code is implemented in C. It uses GStreamer multimedia framework \cite{7} for translating any song in MPEG-1 Layer 3 audio format into a 2-channel 32-bit floating point data. GStreamer decodes mp3 files using libmad, an MPEG audio decoder library.\cite{8} Our GStreamer pipeline is the following: (filesrc ! mad ! audioconvert ! audioresample ! capsfilter ! fakesink). We added a pad buffer probe to src pad of audioresample element. Using a fakesink instead of an alsasink, which is the output sink to hear the sound of currently played audio file, is gathering audio data faster. During playback, data is intentionally streamed slowly through the pipeline, waiting the sound card to convert digital data to analog sound signals.

We are using libxtract to extract the audio features from audio frames streamed through the GStreamer pipeline.\cite{9} Initially we included other features than the ones in Data Characteristics section. However, as we are going to explain in Challenges section, we had to strip down irrelevant features to speed up clustering and more importantly, be able to place all of the data to memory.

After computing the features defined earlier, we are saving an ARFF or CSV file (we worked with both kind of files during the project) for each song. We are appending the timestamp of every frame to these files as the starting column. Other than these individual files, we have a main file where we store extracted data from all song frames. We are appending the song name and timestamp of every frame to this file as the two starting columns respectively.

\subsection{B. X-Means Clusterer}

In this project, we mainly used Dan Pelleg’s X-Means implementation \cite{10} integrated into WEKA.\cite{11} We were able to customize parameters such as cut-off factor, maximum number of iterations, minimum and maximum number of clusters using this implementation. Since we recorded additional marker attributes such as song name and timestamp to our ARFF and CSV files as attributes, we could not use this clusterer directly from the console. We needed to make WEKA ignore these attributes before applying X-means. By using X-Means in WEKA on a single ARFF or CSV input from any song, we could always get significant results quickly. To give an idea about the running time, a 6 minute long rock song is clustered within 1 minute and 40 seconds, yielding 48 clusters.

As we are going to explain later in more detail, we faced problems when we tried to process our main file, holding all data about the collection. Since conventional data mining tools such as WEKA and RapidMiner were designed with the assumption that they can have access to any amount of memory if they need, we could not run X-Means from WEKA on our main files.

To succeed in processing our large main files, we have tried various methods. We are going to give more detail in Challenges section but these efforts included a K-Means implementation supporting disk I/O in MATLAB \cite{12} and in Java \cite{13}. We have tested these K-Means implementations because a working K-Means implementation that is able to process at least half of the original data is needed to implement X-Means. After observing the ineffectiveness of these implementations (and the disk I/O approach itself), we abandoned these programs and generated our results with WEKA.

\section{VI. Challenges}

The biggest challenge of this project was coping with the size of extracted data. When a sample rate of 44100 kbps and a channel size of 2 is considered, an ordinary 5 minute long audio signal has $\frac{(88200 \times 300)/2500}{10584}$ frames. $N \times 10584 + 6 = N \times 63504$ bytes are needed to store the feature extraction result of this song, where $N$ is the number of extracted features. In our project, minimum value of $N$ was 19. Approximately 19 $\times$ 63504 = 1206576 bytes, close to 1.2 MBytes are needed for one song. When we multiply this value with a small personal music library size of 1000, it makes 1200 Mbytes, approximately 1.2 GB in size!

First we tried to modify WEKA’s Instance API (in weka.core package) to add a caching mechanism for such cases. Since the API was designed with “all data fit into memory” assumption in mind, we realized it was a hard task and we changed our mind. We succeeded in implementing a version of K-Means supporting disk I/O in MATLAB and Java. Both version included a caching mechanism that would help disk I/O to be optimized for sequential reading. However, when we observed the running times, we understood that both our implementations had failed miserably. We also thought that our failure could be related with optimization of file access and indexing so we also tested these methods with an SQLite database and we failed again.

The only slightly successful solution was data compression by loss of detail. We converted our floating point outputs from 10 decimal places to 4 decimal places to reduce file size. Then we eliminated irrelevant features that generated missing values on most of the inputs such as harmonic skewness, average deviation, tonality etc. We have also tried Principal Component Analysis (PCA) to reduce dimensionality. However, this method did not work as expected in our data and degraded the quality of our measurements during dimension reduction.

Other challenging aspects of the project include the difficulty in visualizing and inspecting the clustering result together with real song data. General classifications of songs exist but they are useless for this task. Since our task is
defined as detecting the nature of audio frames, no labeled data was available to test our results automatically. Evaluating the effectiveness of our approach required a significant amount of time. WEKA adds up to the complexity of this task by not saving visualization output together with the clustered data instances. This means that you have to inspect and label your results as soon as possible and you lose your results if WEKA crashes when you are inspecting.

Additionally, when we include features such as harmonic skewness, missing values appear frequently because of the computation result being set to Inf or NaN. We believe that this is caused by libxtract library's behavior of continuing all processing with float values or without any change on the float variables to avoid this. If we implement our feature extractor functions, we believe that we can fix these kinds of errors.

VII. RESULTS AND EVALUATION

A. Local Clustering Results

These results summarize that an acceptable segmentation of a song is produced by X-Means clustering. It yields cluster membership changes that map to significant perceptual transitions between different sound segments. For any song, this result can be observed easily.

Here is an example clustering output of a pop song: Justin Timberlake - What Goes Around...Comes Around

In Figure 9, we can observe that in a,b and c groups share common clusters among them, occurring again when their periods repeat. a represents the vocal bridge starting with "Don’t wanna talk about it...", b represents the part starting with "This is the way it's going down..." and c represents the chorus starting with "What comes around...". d represents the part where string instrument plays solo for a while and an electronic music style comes in with very different vocals from the entire song. e part is the most different partition from the entire song.

It can be observed that the purple cluster, 5th from the top represents a sound belonging to the repeated string instrument partition in the song. It can also be observed instantly that the cluster membership structure of the vocals of Justin Timberlake between 0:31 and 0:57 is very different than a,h,c part. These parts may seem similar at first. However, the gaps seen between the cluster memberships are significant, often ranging from a duration of 1 second to 10 seconds.

We can observe the interchanging process between sounds. At any given time, a sound may become more prominent than another and have less gaps. However, we also note that an interleaving, zig-zagging occurs. Since our frames are very small, we can falsely think that a given sound frame have become a member of more than one cluster but this is just an illusion.

We also observed a problem in our current approach. Human vocals get lost behind other instrument sounds and we cannot correctly detect human vocal transitions. Even though Justin Timberlake's vocal has many transitions in this song, we do not have an idea of most of the transitions in the clustering.

Here is an example clustering output of a metal song:

Dementia - Nuklear

Important state transitions are marked with lines throughout the song. It can be seen that especially sounds inside the rectangles interchange roles and define the structure of the song. In fact, 1:51-2:33 is the first guitar solo and 2:52-3:33 is the second guitar solo in this song. This solo guitar tone is the green cluster in rectangles. The other sound in cyan cluster marked with rectangles is the guitar tone from the main riff of the song. It can be checked that open hi-hats mixed with snare drums and cross drums are also clustered cleanly by listening this song. We think that there is good amount of correspondence between musical transitions and cluster membership transitions. We marked several matches with a red circle to give an example.

Our method not always work good, especially for human vocals as we discussed a little above in Justin Timberlake case. Now we present the clustering result of a jazz song which is based on a deep female vocal. Here is the output of clustering Madeleine Peyroux - Dance Me To The End of Love:

As it can be seen from the Figure 11, the black line shows where the vocal starts in the song. When we observe the clustering, we understand that our approach cannot detect any transition that signals the addition of a human vocal to the
song. So we can say that currently our approach fails miserably for human vocals.

B. Global Clustering Results

Our main goal was to find the similarity (clustering) among all the songs in a music library. Since we had problems with clustering big amounts of data, we tried to test our expectations by clustering an album. Here is the clustering result from *Americana* album of the famous punk-rock band *The Offspring*.

In this result, it can be observed that the bonus track found in *Pay the Man*’s ending at 9:17 is detected successfully. We observed that a significant different appeared between *Pay the Man* and other songs in the album at the cluster in the red circle. We also see that natural differences between the songs in an album appear, such as the difference between the songs marked with the blue circle (not having these clusters) and the other songs having frames belonging to clusters at that level. As a side note, our clustering algorithm shows the richest song encapsulating *The Offspring*’s style in this album as *Walla Walla*. In our opinion, this is a correct observation.

As our final global clustering test, we created a mixed folder having one song from different musical genres. This folder consisted of songs belonging to *acoustic*, *rock ballad*, *rocky metal - hard rock*, *classic rock*, *metal*, *classical*, *punk*, *jazzy hybrid* and *pop* genres.

Here is the song list:
- Acoustic: Rodrigo y Gabriela - Tamacun
- Rock Ballad: Aerosmith - Cryin’
- Hard Rock: Metallica - The Unforgiven
- Classic Rock: Dire Straits - Sultans of Swing
- Metal: Metallica - The Four Horsemen
- Classical: Beethoven - Symphony No.3
- Punk: The Offspring - Staring at the Sun
- Jazzy Hybrid: Jamiroquai - Virtual Insanity
- Pop: Michael Jackson - Billie Jean

Here is the clustering result:

In this test, we observed that every song had a cluster that did not belong heavily to another song. To summarize, the songs dominated the cluster related to their genre. Metal vs. Hard Rock (Rocky Metal) comparison may provide misinformation to a reader not knowing Metallica. We intentionally put two *Metallica* songs in this collection as a false positive test. However we have seen that aside from instrumental similarities, there exist various significant clustering differences in these two Metallica songs. Our clustering approach worked as we expected in this case.

VIII. Future Work

We did not compute Mel Frequency Cepstral Coefficients (MFCC) in our current work. With MFCCs we will be able to change our pitch mapping to a perceptually correct one (*Mel scale*). The pitch difference between notes do not stay the same in upper or lower octaves. Mapping our frequencies to Mel scale will provide us a better measurement tool about similarity.

Adding to that, we did not compute any tempo features. Our clustering results can be furtherly improved by calculating a beat histogram and measuring the frequency of the peaks. Without tempo features, it is very hard to discriminate genres from each other so this is important. We can utilize local tempo measurements are additional information.

Moreover, we are going to find a solution to detect human vocal events correctly. Applying local normalization on the audio signal may provide good results since the fundamental problem about human vocals is the weakness of the signal most of the time.
We should optimize our sample selection and arrange our framework to be able to use a dynamic, adaptive frame size. When used together with a beat histogram, we can guarantee that we will capture the shortest note in the song.

In our current work, we did not apply a windowing function on our frames. Indeed, most of the musical instruments have temporal characteristics. Past frames affect the present frame. This may also help us to solve human vocals problem.

Finally, as our main job, we are going to implement a highly-parallel clustering algorithm using MPI and solve this large data size problem. First, we need to investigate current literature on incremental clustering algorithms to complete this task.

IX. CONCLUSION

In this paper, we defined a framework to measure the similarity of sound characteristics in a given music library by using X-means algorithm to learn the number of clusters and observe the nature of the data automatically. We have discussed our approach to music similarity problem, our implementation and challenges we faced throughout our project. We presented an analysis on current state of the approach and we stated possible ways to improve this project. When a parallel clustering algorithm that can process more features in less time is implemented, our approach is going to be more powerful by the addition of MFCC and tempo calculations, together with normalization operations. However, even in its current form this approach can serve as a baseline to split a song into meaningful segments.

REFERENCES