Music Classification - Pick a beteer title later

Christian Johnson
Electrical Engineering Department
United States Coast Guard Academy
New London, Connecticut 06320
Email: Christian.S.Johnson@USCGA.edu

Daniel Nusraty
Electrical Engineering Department
United States Coast Guard Academy
New London, Connecticut 06320
Email: Daniel.Y.Nusraty@USCGA.edu

Joshua West
Electrical Engineering Department
United States Coast Guard Academy
New London, Connecticut 06320
Email: Joshua.C.West@USCGA.edu

***Abstract*—This is an abstract**

I. Introduction

A. Motivation

With the advent of music subscription services, many people have become accustomed to automatically generated playlist, tailor-made to their own personal taste. These services provide such playlists at the push of a button, analyzing the user's listening history to continuously recommend similar songs. Providers such as Pandora, Spotify, and LastFM maintain massive databases containing context information on millions of songs in order to present the best recommendations. For those who prefer to maintain their own local music library however, the options for tailored music recommendations are dramatically reduced. In this paper, we seek to explore the application of digital signal processing techniques in order to implement similar music recommendation functionality that will work with local music files.

B. Similar Work

II. Theory

A. Background

The majority of music analysis is based on a family of functions known as Fourier Transforms. A Fourier Transform is responsible for transforming time-domain audio information into a frequency-domain representation. In the time domain, sound is represented as amplitude as a function of time. In the frequency domain, sound is instead represented as a function of magnitude based on frequency. What is important to recognize, is that in both cases the signal is still continuous, meaning unbroken. A continuous signal contains a great deal of information that makes it difficult to perform operations on. In order for a computer to be able to process such a signal, that amount of information must be reduced through an operation known as *sampling*. Sampling replaces the continuous signal with discrete representation; an array of values at (typically) evenly spaced indeces of time. The normal Fourier Transform cannot operate on discrete signals however, which is where a specific type of Fourier Transform known as the Discrete Fourier Transform (DFT) comes in. The DFT is comprised of a single summation operation, which will produce an array of complex number representations.

$$X(k) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)e^{-j\frac{2\pi}{N}kn} \tag{1}$$

Each value in the DFT output $S[k]$ is a complex number that represents a point in an N dimensional vector space. The magnitude and phase of that point represent the content of the time-domain signal at frequency $\frac{2k\pi}{N}$, where N represents the length of $S[k]$. These values depend a great deal on a concept known as *sampling frequency*, which refers to the time between each sample taken of the original time-domain signal. Each 'bin' of $S[k]$ represent a collection of frequencies, $\frac{-F_s}{2N} + \frac{k*F_s}{N} < f < \frac{F_s}{2N} + \frac{k*F_s}{N}$. As such, it is important to recognize that $S[1]$ does not directly correspond to a specific frequency value, and in order to find the DFT output for, say, 20Hz, one must determine which bin this frequency will fall into.

1) The FFT: In its original form, the DFT is quite computationally expensive and complex, comprised of $\mathcal{O}(n^s)$ operations. An algorithm known as the Fast Fourier Transform (FFT) dramatically simplifies this complexity, reducing the expense to $\mathcal{O}(n*log(n))$ operations. It does this by exploiting symmetry within the DFT. Equation (1), the basic representation for the DFT, is periodic about N, i.e. $X_{k+l*N} = X_k$ for any integer l. The FFT uses this relationship to break the DFT into even and odd components

$$\begin{aligned} X_k &= \sum_{n=0}^{N-1} x(n)e^{-j2\pi kn/N} \\ &= \sum_{m=0}^{N/2-1} x(2m) * e^{-j2\pi km/(N/2)} \\ &+ \sum_{m=0}^{N/2-1} x(2m+1) * e^{-j2\pi km/(N/2)} \end{aligned} \tag{2}$$

This particular implementation is known as the Cooley-Tukey FFT, and is one of the most widely used algorithms in the world.

B. Analyzing Music

In order to analyze and classify music, it is important to understand how it is structured. Fundamentally, music is simply

composed of air molecules vibrating at different frequencies and amplitudes. This means that there are a theoretically infinite number of ways to structure and divide these different frequencies, but in western music, the most common method uses what are known as major scales. The major scales are composed of 7 unique notes, A through G order based on their pitch interval. While advanced music theory is outside of the scope for this paper, it is important to recognize that a single major scale is composed of 7 notes, with a predetermined order of pitch difference. These pitch differences are somewhat unintuitive however, since music is arranged on a logarithmic scale, meaning that the note typically known as A0 (The note A in the first recognized octave) is defined as 27.5 Hz, A1 is 55 Hz, A2 is 110 Hz, and so on. This is significant, because it means that the most commonly used frequencies in music will typically be grouped towards the lower half of the established musical range. In general, musical notes are defined from C0 at 16.35 Hz all the way to B8 at 7902.13 Hz. Most types of music will use notes toward the center of this range, as the extremes are considered somewhat unpleasant; however, different genres of music may use varying distributions of these frequencies. For instance, an EDM or electronic artist may choose to use frequencies in the 0 or 1 octave in order to produce a more physical feeling. These frequencies are considered "super-low", since they are so low that, when combined with other tones, they are more often *felt* than heard.

III. Experimental Procedure

Using this information, it becomes relatively simple to start processing audio files and comparing them to each other. There are 3 key steps to this process; audio information should be read from a file, then that audio information should be compressed and generalized in order to compare two songs. This generalized audio information should be collated into some structure that will allow simple access later. The simplest application for this concept is to compare a collection of songs to a single different audio file and return a list of songs from the collection that are most similar to that audio file.

A. Processing a Single Audio File

Digital audio files are simply a collection of amplitude information arranged based on time. This information is sampled from the original continuous audio source, typically at a rate of 44,100 Hz (Meaning that there should be 44,100 samples making up a single seconds worth of audio information). This format is incredibly useful for a computer which must play recorded audio, since the data will simply tell a computer the *amplitude* at which to vibrate a speaker diaphragm, and the time at which to do so. It is less applicable to examining and classifying the underlying composition of the audio however, and must be tranformed using the FFT to a more applicable format. Taking the FFT shifts the sampled data from amplitude in terms of time to amplitude in terms of frequency, as discussed earlier.

B. Interpreting Data

The FFT output is formatted in a much more accessible fashion than the time domain data. It provides an array of complex bin values that represent a range of frequencies as discussed earlier. In order to use this information to compare two song files, it is important to generalize the information somewhat. Comparing the raw FFT of two files can hold some value, but they contain such a high volume of information that such a comparison could result in too high a degree of specificity. One method for generalizing this data is to separate the frequency spectrum into so-called 'bands'. These bands can be chosen based on a variety of different factors, but it seems most logical to divide them based on musical structure. Based on the scale structure discussed earlier in this paper, a reasonable set of frequency breakpoints could be $0-60$ Hz to represent 'superlow' frequencies, $61-240$ Hz for lows, $241-500$ Hz for mids, $501-2000$ Hz for highs, and $2001-8000$ Hz to represent 'superhighs'. Since the FFT bin indeces do not correspond to actual frequencies, it is necessary to convert these frequencies in order to determine the closest bins that these frequencies correspond to. Calculating the bin into which a specific frequency will fall relies upon the relationship between sampling frequency and sequence length. If N represents the number of samples in an FFT signal, f represents the frequency value, and r represents sampling rate then bin number can be found from the following equation:

$$\text{Bin Index} = \frac{f * N}{r} \tag{3}$$

Using this equation, it becomes a trivial operation to determine the necessary breakpoint indeces, then to segment the FFT data into a set of bands. This segmented FFT groups together several series of frequencies which make up similar ranges. Finding the bin with the largest magnitude within each segment presents a way to identify which frequencies are most prominent within a given audio file. Given the segments defined earlier, this compresses 44,100 samples per second of frequency information to an array of 5 complex numbers of the form $a+bj$, which can each be rewritted as polar coordinates. Polar coordinate, in general, represent a point or vector on a two-dimensional plane. The distance between the endpoints of two vectors can be computed using the Euclidean Distance Algorithm, wherein each polar coordinate is converted to cartesian coordinates of the for (x, y). The Euclidean Distance is simply the length of the line segment connecting the two coordinates, using the following relationship:

$$\text{Distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{4}$$

Thus, the similarity between two songs can be approximated from each song's array of 5 polar coordinates. The distance between each element is found and concatenated into an array of 5 distance values. The average of these values can be approximated as a single similarity metric to compare two songs.

C. Storing Comparison Data

Above, it is shown how a song can be characterized through its frequency data and compared to a single other song. This comparison takes the form of a distance metric, inversely representing the proximity of the two songs' primary component frequencies. The larger this distance metric, the less similar the two songs are. In order to construct a system which can handle more than two songs, it is necessary to construct a method for storing and accessing data. The storage and access functions, while related, can be constructed seperately. The system can 'take in' new audio files, performing the FFT, segmenting, and gathering an array of polar coordinates for each file. These polar arrays are stored as values in a dictionary structure, with the filename as the key. This dictionary structure allows extremely for an extremely fast lookup time when accessing values by key, and serves as persistant storage for frequency data.

Upon a new entry into the dictionary, the new entry is compared to every other entry in the dictionary, producing a three element array with the first two elements being the names of the two songs being compared, and the third value being the comparison metric; the distance between the two songs. This 3 element array is concatenated into a second datastructure - a list comprised of three element arrays. This list serves as storage for every possible comparison. In order to produce a list of 'similar' songs, the system can take a file as input, checking whether that file is already present in the dictionary of frequency data. If the file is not present, the FFT is computed, the information is added to the dictionary, and comparison data is added to the list. If the data is already present, the FFT is skipped. The system searches for all elements of the list that contain the name of the song being searched for and removes that song to produce a list of two element arrays. These arrays are sorted by distance metric, to produce an ordered list of songs. The songs at the top of the list have the smallest distance (and therefore are most similar to the baseline song).

IV. Results

V. Bibliography

We will move all these sources to a .bib file later and import using BibTex. Until we figure out how to do that, they are referenced here by section.

A. Similar Work

https://www.music-tomorrow.com/blog/how-spotify-recommendation-system-works-a-complete-guide-2022 https://www.toptal.com/algorithms/shazam-it-music-processing-fingerprinting-and-recognition

B. Theory

https://dsp.stackexchange.com/questions/5915/what-is-the-meaning-of-the-dft https://dsp.stackexchange.com/questions/26927/what-is-a-frequency-bin https://stackoverflow.com/questions/10754549/fft-bin-width-clarification

C. FFT

https://pythonnumericalmethods.studentorg.berkeley.edu/notebooks/chap Fast-Fourier-Transform.html

D. Analysing Music

https://muted.io/note-frequencies/ https://rwu.pressbooks.pub/musictheory/chapter/major-scales/ https://people.umass.edu/gmhwww/382/pdf/12-musichttps://www.onlinepianocoach.com/common-music-scales.html https://www.petervis.com/hi-fi-info/jvc-sea-graphic-equalizers/frequency-spectrum-of-common-musical-instruments.html