ADAPTIVE SHUFFLE: DYNAMIC AUDIO PLAYBACK BASED ON USER BEHAVIOR AND MUSIC SIMILARITY

Madhukesh Ayyagari
Georgia Institute of Technology
mayyagari7@gatech.edu

Jeremy Sparks
Georgia Institute of Technology
jsparks23@gatech.edu

ABSTRACT

In this paper, we present and evaluate a system for dynamically changing the playback order of songs in a playlist based on the user's behavior and the similarity between songs. Spectral and rhythmic features are extracted from each file and used to determine song similarity. The user is allowed to "Like" or "Dislike" each song, creating a two-class feature space. The system must then determine whether to play or not play the next song in the playlist. The K-Nearest Neighbors algorithm is utilized as the distance measure to determine the similarity between songs. An evaluation of the system was conducted and compared to a system where the songs in the playlist are simply shuffled randomly. The results of this evaluation are discussed and the validity of the system is examined.

1. INTRODUCTION

Music is a very personal and ever-changing field. Popular songs come and go, people change their preferences often, and the way music is created is constantly evolving. One way that people attempt to create some structure out of this chaos is by making playlists. These playlists contain songs grouped by theme, similarity, or any number of other factors specific to each user. However, problems can arise wherein a single playlist has songs that are very diverse in style, and it would be tedious to group these songs into other, smaller playlists. Our goal with this system is to create a means to play a diverse mix of shuffled music in an order that is tailored to the mood and time-dependent preferences of the user.

2. RELATED WORK

Playlist generation is becoming a popular topic in the age of digital music. Apps like Spotify and Pandora are becoming a primary means of music listening and discovery. With this, many methods of automatically generating playlists have come about. One popular method is with the use of one or more "seed songs". These songs are considered to be representative of the entire playlist and are used to find the other songs that will be played. The system presented in [10] uses a dynamic clustering algorithm and one seed song to suggest new songs to the user. An even simpler method is presented in [3], which simply takes the N nearest songs to the seed song as the playlist.

Another technique involves setting user-defined constraints on the playlist outcome and choosing a group of songs that meet the criteria. This method can be seen in [7] and [8]. These models involve looking at an entire database of songs in order to find the best group and order that meet the input criteria. Related to this is the method proposed in [9], which takes expertise from DJs to create a "music landscape", or indexed database, and attempts to navigate this space in an automatic and optimal way. The expertise from the DJs and the user input place rules, or constraints, on the outcome of the system and determine the playlist structure. This method, however, was found not to be satisfactory in automatically creating playlists, wherein it was found that manual input was needed to optimize the system.

The final method we will discuss here is proposed in [2] and describes a system that only looks at the listening habits of the user and ignores all the metadata for each song. This system uses two algorithms – one to track the listener's habits and one to form a "listener model" from the data. This model is then used to formulate a playlist that is connected to the specific listener model that is determined by the system. However, it was not successful in that it was not able to determine what types of songs the user preferred simply based on the user's input.

3. PROBLEM STATEMENT

Most of the systems shown above involve the use of a database and a large amount of metadata to function. This is good for complex, large-scale systems, but lacks the ability to have a simple and localized approach. A large amount of time and resources go into creating a database of musical metadata. Our goal is to create a simple, yet robust system that is able to change the playing order of a playlist based on a single user's input. This means that there is very limited data available to the system. It also means that we do not expect it to work as well as commercial, large-scale software. The vision for this system is a passive algorithm that a user could activate and deactivate within a music player, such as iTunes, and would relearn the user's time-dependent preferences each time they open the app.

4. SYSTEM OVERVIEW

The system we present here is not strictly automatically creating a new playlist. It is an algorithm that changes the play order of a predefined playlist based on the user's input and the similarity of songs. The user is allowed to either "Like" or "Dislike" each song and, in doing so, classifies two classes of songs. The algorithm will then measure the similarity of the songs that are up next in the playlist to these two classes and play a song when it is found to be more similar to the "Like" class.

Our system is broken up into three separate phases. The first phase involves taking the user-selected playlist, shuffling it into a random order, and extracting audio features from each of the audio files. For this model, we use the typical feature set that is used in many similarity measure and genre classification tasks. This set includes spectral and rhythmic features. After the playlist is uploaded, the music is ready to play as a typical music player. We have implemented a primitive app in MATLAB to use as a music player. It provides the basic functions the user needs to work the system. However, it also has latency while extracting the audio features from each of the files. It takes about one minute to read and extract all the features from 90 files.

The next phase is the data collection phase. The app functions as a typical music player until there are at least three songs in both the "Like" and "Dislike" classes. We acknowledge here that there is a possibility that the user may not dislike or like any song. However, we believe this circumstance is very unlikely in a real-world context.

Once the initial data has been collected, the third phase begins and the similarity measure and adaptive algorithm takes place. It is interesting to note that, at this stage, there is little to no noticeable latency in the system playing and fetching the next songs. Each time the user likes or dislikes a song the training data changes and grows, hopefully giving better results as the user continues to use the app. Additionally, in gathering data in this way, the system relearns the user's preferences each time they use the app and uses a similarity measure based on the current behavior of the user.

5. AUDIO FEATURES

In audio signal analysis, features are mathematically calculated attributes of an audio file that attempt to describe the musical attributes of the file. While not all features directly correspond to a perceptual attribute, many features together can almost completely characterize a song. In this system, we are interested in using features that will help describe how similar songs are to each other. We will be using common similarity and genre classification features, as presented in [5], along with two novel rhythmic features we have developed. In this section, we will briefly describe these features and what perceptual quality they roughly represent.

5.1 Spectral Features

Spectral features are used to describe the timbral quality of the audio signal. Timbre is a quality of sound that is hard to describe explicitly, but allows people to recognize different instruments and voices. The specific spectral features we use in this system are also used in [5], and we utilize the equations found in [1] to calculate each feature. Each of these features involves transforming the signal to the frequency domain via the Fourier Transform and performing calculations on separate, small blocks of the data. For our system, we use the standard block and hop sizes of 1024 and 512 samples, respectively. The following are the spectral features that we extract from each audio file:

5.1.1 Spectral Centroid

The spectral centroid is defined as the center of gravity of the magnitude spectrum. It correlates to the brightness of the audio, or the energy in the high frequencies.

5.1.2 Spectral Rolloff

Spectral rolloff is defined as the frequency at which the accumulated magnitude reaches a certain percentage of the overall sum of magnitudes. The standard percentage to use is 85%, which is what is utilized here. This feature relates to magnitudes of higher vs. lower frequencies.

5.1.3 Spectral Flux

Spectral flux is defined as the average difference between successive frames of the magnitude spectrum. This represents the amount of change in the spectral shape of the audio signal.

5.1.4 MFCCs

Mel-frequency cepstral coefficients are a commonly used feature to describe an audio file because they are a compact representation of the total spectral envelope of an audio signal. The number of coefficients to use is dependent on the application, and can range from the first 3 to the first 20 coefficients. For our system, we are using the first 3 coefficients to minimize the amount of dimensions we have for our small-scale algorithms.

5.1.5 Zero Crossing Rate

Although not strictly a spectral feature, zero crossing rate is important in describing the timbral quality of an audio file. It measures the number of times the audio signal crosses zero. This is a simple and efficient way of describing the noisiness of the signal.

5.2 Rhythmic Features

The rhythm of a piece of music is a fundamental and important element to describe the style of a song. The features presented here are high-level features that are calculated from large 4-second blocks with 2-second overlaps. The tempo is a necessary attribute for music similarity, but there are also other elements that should be taken into consideration in the context of rhythmic analysis. A beat histogram is a useful method of finding more detailed rhythm features. It gives us a graphical representation of the overall rhythmic content of an audio file. It not only gives us the tempo of the song, but also the energy of every bpm present in the song. We use the methods found

in [5] to calculate the beat histogram, find the tempo of the song, and then use this information to derive other rhythmic features of the audio.

5.2.1 Tempo

One of the most important and perceptually relevant features we can take from an audio signal is the tempo. This is the "foot-tapping" rate of the music and is calculated by detecting periodicities throughout the whole file.

5.2.2 Peak Ratio

The peak ratio refers to the ratio of the heights of the first and second highest peaks in the beat histogram. This ratio can tell us the strength of the downbeats to the off beats in the song.

5.2.3 Relative Amplitude

Relative amplitude is the amplitude of the highest histogram peak divided by the sum of the histogram across all bpms. This can tell us the beat strength of the main tempo versus the "noisy" micro-tempos across the audio file.

5.2.4 Amplitude Sum

The amplitude sum is simply the amplitudes of the beat histogram summed up for all bpms. This directly relates to overall beat strength of the song.

5.3 Feature Post-Processing

Once we have the features extracted from the audio files, we must process them into a format that is compact and efficient. For this, we must aggregate the features, or calculate individual numbers that will represent the audio file as a whole. For the spectral features, we take the mean and standard deviation across all blocks of audio to give us a 14-dimensional space. When we add the beat features, we get just one number for each feature per file, so the 4 beat features we extract correspond to 4 new features in our feature space. Altogether, we extract a total of 18 features from each audio file for use in the similarity measurements. After these features are aggregated into one number per feature per file, we use z-score normalization in order to balance the numbers for our similarity measure.

6. SIMILARITY MEASURE

In order to determine what songs to play and skip, we must have a way of measuring how similar songs are to each other. There are many methods of doing this, and one of the most common ways is by using an algorithm called K-Nearest Neighbors. This is described in [1] and simply finds the k number of other audio files that are nearest to the file in question. In our system, we use the Euclidean distance to measure the similarity of each file, but there are other measures that can be chosen, depending on the applications. With a multidimensional space and a small amount of data being used, kNN is an efficient means of measuring the similarity between files.

In our system, we have a two-class problem involving a "Like" feature space and a "Dislike" feature space. These multidimensional spaces hold the features of each song that is either liked or disliked by the user. When the next song is ready to play, the system will measure the distance between this song and each of the files in both feature spaces. It will then determine the k minimum distances, thus finding k closest files to the next song. If there are more "Like" songs than "Dislike" songs within these k neighbors, the song will play. If there are less "Like" songs than "Dislike" songs, the song will be skipped and be placed at the end of the playlist. If there is a tie in the number of "Like" and "Dislike" neighbors, the app will play the song as well. The k value starts at 1 and increments every other time a song is played, up to a maximum of 9. This strategy is implemented so that the number of neighbors will increase with respect to how many data points are in the available feature space.

This method of similarity measure is very simple. Yet, we believe that, given such a small amount of data to work with, a simpler system is better to work quickly and efficiently with the data it has.

7. EVALUATION

It is difficult to evaluate systems based on similarity of music because similarity is a subjective measure that can vary per person. However, genre is a classification system that humans have created to attempt to objectively separate songs into groups based on similarity. Therefore, to determine the performance of our system, we use genre as a means to determine the objective ground-truth similarity of songs in a playlist.

7.1 Dataset

The dataset we used for evaluating our system is a very popular music genre classification set used in [7]. It consists of 10 different genres with 100 songs per genre, with each song represented by a 30 second clip. Since we are dealing with small, local systems here, we used only 50 songs from each of these genres.

7.2 Method

Our method of evaluating our system is similar to that used in [4]. In their evaluation, they measure the amount of skips it takes for the user to listen to a set number of specific songs, given a seed song and a music database. Similarly, we took the 10 different genres in our dataset and created 10 different playlists, one for each genre. In each of these test playlists, we have 90 songs, half of which are of a "target" genre and the rest of which are evenly split between the other genres. For example, in the rock playlist, we have 45 rock songs and 5 songs of each of the other 9 genres, making 90 songs total.

In order to evaluate whether our system functions as it should, we ran tests on each playlist to see whether the target genre was played more often than the other genres. Our goal here is to determine if the system will play more songs "similar" to each other, based on the genres.

The tests involved counting how many plays it took to play half (23 in this case) and all (45) of the target songs in the given playlist. We repeated this test 3 times for each playlist and took an average of the playback numbers. We then compared these numbers to what is expected if the system played the songs randomly. In this

case, since this is a two-class problem, we would expect the system to play the target and non-target songs each half the time. Given this, we can calculate a ratio of the number of target songs played to the total number of songs played.

7.3 Results

The resulting data from our evaluation represents the average ratio of target songs to non-target songs while running our system. In a totally randomly shuffled system, the expected result would be 0.5. In other words, if the songs were played randomly, target and non-target songs would each play half of the time. This ratio can tell us the performance of our system compared to a randomly shuffled playlist. Our system's performance can be seen below.

	Ratios		
Genre	23 Target Songs (Half)	45 Target Songs (All)	Total Ratio (Half and All)
Blues	0.67	0.62	0.65
Classical	0.68	0.73	0.70
Country	0.64	0.63	0.63
Disco	0.64	0.52	0.58
Hip Hop	0.53	0.51	0.52
Jazz	0.57	0.59	0.58
Metal	0.65	0.66	0.66
Pop	0.73	0.59	0.66
Reggae	0.59	0.62	0.61
Rock	0.53	0.57	0.55
Total	0.62	0.60	0.61

Table 1. Ratios the number of target genre songs played versus the total number of songs played before the target number was achieved.

7.4 Discussion

The results of the evaluation of our system are statistically significant. Across all genres, we saw an increase in the number of target songs to non-target songs. Overall, our system played target songs 11% more often than non-target songs. An interesting thing to note is that some genres performed better than others. Our best performing genre overall was classical, with an average of 7 out of 10 songs played being of the target genre. The worst performing genre was hip-hop, showing only a 0.52 ratio, which is not significant compared to the expected 0.5 of a random playlist.

Considering this is meant to be a passive system that handles minimal data, we can say that these results are promising. Although genre is not an absolute measure when it comes to similarity, it is the best objective measure we have to determine groups of similar songs. Since this system plays more of a certain genre that we are wanting, we can say that it achieves its goal to a degree. Further subject testing would need to be conducted in order to fully understand the behavior of the system.

8. CONCLUSION

In this paper, we have presented a system that takes user input and music similarity measures to dynamically change the playback order of a predefined playlist. We used timbral and rhythmic features to calculate similarities between songs, and the user was allowed to input whether they liked or disliked the song played. This is a simple system, but a novel take on using small-scale data to find similarities in songs. The results show that the system plays more songs similar to the songs that are "Liked" by the user, therefore showing that the system achieves its goal to some extent. In the context of being used as a passive system in a working music player, this small-scale algorithm works well in dynamically altering the playback order of a playlist based on the user's preferences. We believe these results prove this is valid system, but future subjective tests would tell if the system is suited for real-world use.

9. REFERENCES

[1] E. Author: "The Title of the Conference Paper," *Proceedings of the International Symposium on Music Information Retrieval*, pp. 000–111, 2000.

[2] A. Someone, B. Someone, and C. Someone: "The Title of the Journal Paper," *Journal of New Music Research*, Vol. A, No. B, pp. 111–222, 2010.

[3] A. Lerch: *An Introduction to Audio Content Analysis: Applications in Signal Processing and Music Informatics*, IEEE, Piscataway, NJ, 2012.

[4] A. Andric and G. Haus: "Automatic Playlist Generation Based on Tracking User's Listening Habits". *Multimedia Tools and Applications* Vol. 29, No. 2, pp. 127–151, 2006.

[5] B. Logan: "Content-Based Playlist Generation: Exploratory Experiments". *Proceedings of the International Symposium on Music Information Retrieval*, 2002.

[6] E. Pampalk, T. Pohle, and G. Widmer: "Dynamic Playlist Generation Based on Skipping Behavior". *Proceedings of the International Symposium on Music Information Retrieval*, 2005.

[7] G. T. and P. Cook: "Musical genre classification of audio signals". *IEEE Transactions on Speech and Audio Processing*, Vol. 10, No. 5, pp. 293-302, 2002.

[8] G. Dubey, K. Kumar Budhraja, A. Singh, and A. Khosla: "User Customized Playlist Generation Based on Music Similarity". *2012 National Conference On Computing And Communication Systems*, 2012.

[9] J. Aucouturier and F. Pachet: “Scaling up Music Playlist Generation”. *Proceedings of IEEE International Conference on Multimedia and Expo*, 2002.

[10] M. Ghoniemy and A. Tewfik: “A Network Flow Model for Playlist Generation”. *IEEE International Conference on Multimedia and Expo*, 2001.

[11] M. Crampes, J. Villerd, A. Emery, and S. Ranwez: “Automatic Playlist Composition in a Dynamic Music Landscape”. *Proceedings of the 2007 International Workshop on Semantically Aware Document Processing and Indexing*, 2007.

[12] S. Pauws and B. Eggen: “Realization and User Evaluation of an Automatic Playlist Generator”. *Journal of New Music Research*, Vol. 32, No. 2, pp. 179–192, 2003.