Master's thesis

Using Features of Groove in Music Recommendation Systems

A study on analyzing groove in musical items and the effects of groove on musical recommendation

Joseph Clemente

Music, Communication and Technology
30 ECTS study points

Department of Musicology
Faculty of Humanities

Autumn 2023

Joseph Clemente

Using Features of Groove in Music Recommendation Systems

A study on analyzing groove in musical items and the effects of groove on musical recommendation

Supervisors:
Hans T. Zeiner-Henriksen
Olivier Lartillot

Abstract

Music streaming services rely on music recommendation systems to keep users engaged and shape their musical taste. These systems rely on a combination of user and item modeling, and are adept at serving relevant recommendations to users through the analysis of collected data. Streaming services must now focus on combating user feelings of stagnation and listening fatigue associated with not receiving exciting and unique recommendations. This thesis proposes that incorporating elements of groove into a music recommendation system's features can produce higher quality and more surprising recommendations by being genre agnostic while still recommending tracks based on one of the most important characteristics of music. To accomplish this, a beat tracking and onset detection system was used to analyze two varieties of percussive source separated audio to quantify features of groove. These features were then used to sort items into clusters, which were tested in evaluation sessions to determine if groove could influence quality or expectedness of recommendations. While the clusters had little effect on quality of recommendations, participants were consistently reporting items as unexpected and high quality, showing that recommending items based on features of groove could be useful in producing more serendipitous recommendations[1].

[1]A blog post version of this this is available on the MCT blog: https://mct-master.github.io/masters-thesis/2023/12/04/josephcl-groove-thesis.html

Contents

Preface

0.1 Acknowledgements

I'd first like to thank the Music, Communication and Technology at the University of Oslo program for welcoming me and providing me with the skills necessary to complete this thesis, and all of the teachers that helped me along the way. My time in the program would not have been nearly as enjoyable without the camaraderie and friendship my fellow students in the Music, Communication and Technology program provided, tusen takk! This thesis benefited greatly from the guidance of my supervisors, Hans T. Zeiner-Henriksen and Olivier Lartillot, and I'd like to thank the RITMO Centre for Interdisciplinary Studies in Rhythm, Time and Motion and the MIRAGE project for providing me with support throughout my thesis semesters. To everyone that participated in this thesis study, thank you so much for your time and patience. Finally, I'd like to thank my family and friends, both in Norway and abroad, for all the support over these past two and a half years. Ha det, MCT!

Chapter 1

Introduction

1.1 Research Motivation

What began as a way to combat piracy and theft in a digital age, music streaming services have become the predominant way people around the world consume music [1]. Along with the advent of music streaming services, Music Recommendation Systems (MRSs) became crucial to the way these services operate by giving them the ability to "help users filter and discover songs according to their tastes" [2, p. 396].It has also been observed that MRSs play a major role in developing the tastes and listening habits of users [3]. However, implementing a MRS comes with a diverse set of challenges for streaming services. Technical challenges that must be addressed include profiling users, classifying musical items, and finding best practices for serving new musical items to users [2, p. 397]. Streaming services have accomplished much in addressing these relatively straightforward issues, but in recent times, more abstract and complex challenges have arisen. These challenges include combating the fatigue of seemingly infinite choice [4, p. 10] and breaking users out of a "feedback loop," where MRSs serve technically correct, but bland and uninspired recommendations to the user [3]. As MRSs push streaming service users more towards a homogenized listening experience and a socio-technological relationship with the recommendation system itself [3], recommendations that push users into new discoveries are necessary to reward long term engagement. In order to do this, MRSs must constantly evolve by developing new, innovative techniques that utilize different kinds of musical and non-musical features.

MRSs using many different features and techniques have been tested and studied over the years. Some of these features do not involve analyzing the musical content of songs themselves, and instead study features such as the listener's historical behavior [5], song lyrical content [6], user grouping [7], emotions and contextual factors [8], [9], motion data using wearable sensors [9], [10], and mood similarity [11]. Studies that use raw audio data to extract features for a MRS usually focus on Time-Frequency (TF) analysis, using features such as mel-frequency spectral coefficients [6], [12], [13] and spectral centroid [13]. More often though, music streaming services use a hybrid approach that features a healthy mix of musical, non-musical, personal, and social features to serve recommendations [14, pp. 98–99]. Through their Application Programming Interface (API), the twelve music recommendation features used by leading music streaming service Spotify can be viewed [15, p. 238]. These features include metadata such as tempo and time signatures, as well as proprietary calculations for song features including acousticness, danceability, and energy [15, p. 240]. It remains unclear how these values are calculated, but even when describing the characteristics of a song, the basis of these calculations remains overly technical. For instance, danceability is based upon "tempo, rhythm stability, beat strength, and overall regularity," [15, p. 240] while energy is based upon " dynamic range, perceived loudness, timbre, onset rate, and general entropy" [15, p. 240]. More focus on using features in MRSs that describe songs in human-like terms could be the key to breaking MRSs out of their machine-like precision and help them deliver more interesting and surprising recommendations to their users.

The theory that this thesis puts forth is that by using features of groove in a music recommendation system, including pulse, subdivisions, and syncopation [16, pp. 6–9], streaming services could have the ability to serve more human-like recommendations, since people tend to recommend music based upon the features of the music itself rather than metadata. To research this theory, a system was developed that extracted percussive notes from a piece of raw audio and used those percussive notes as a way to analyze and mathematically calculate features of groove. Once these groove features were calculated, a Machine Learning (ML) system that groups similar songs in regards to features of groove was developed to see if this is an effective lens to serve music recommendations to users. Serving music recommendations based on the characteristics of groove can present a novel way of predicting user taste in a way that users would actually be able to understand as opposed to more low level features using TF analysis. Also, focusing solely on the topic of groove allows the opportunity to determine if introducing these features to MRSs can combat the feedback loop that they currently struggle with.

1.2 Research Questions

The research questions for this project are as follows:

- How can characteristics of a groove be used to give individualized music recommendations to listeners?
- What groove characteristics are the most useful for recommending new music to listeners?
- To what extent does recommending music based on groove influence the quality and expectedness of recommended musical items?
- To what extent does recommending music based on groove influence the serendipity of recommended musical items?

1.3 Contributions

The main contributions of this thesis are listed below.

- A comparison of the quality of beat tracking and onset detection between audio processed with harmonic-percussive source separation using median filtering and deep learning drum source separation.
- The development of a system that groups musical items based upon features of groove.
- A study and analysis on if an underlying groove of a musical item affects quality or expectedness of items with similar groove features.

1.4 Manuscript Structure

This manuscript will begin in Chapter 2 with the context and relevant background for the concepts explored in this thesis. The methodology of the research will then be explored in Chapter 3, along with the design and implementation of the developed system in Chapter 4. The results from the development of the system and the evaluation sessions are presented in Chapter 5, concluding with a discussion of the results in Chapter 6 and a conclusive summary of the thesis in Chapter 7.

Chapter 2

Context and Preliminary Background

This chapter will provide the requisite knowledge for understanding the concepts, strategies, and motivations behind this thesis study.

2.1 Groove

This section will explain groove as a concept, exploring different understandings of groove and describing the features of groove that will form the basis of this thesis, as well as mathematical approaches to quantifying one of the most important features of groove.

2.1.1 Understandings of Groove

Câmara and Danielsen [16] define the term “groove” in three ways: as a pattern and performance, as pleasure, and as a state of being. The pattern and performance understanding of groove can be seen as the rhythmic structure of a style of groove, where patterns and interactions between rhythmic onsets create a structure for the groove [16, pp. 2–4]. Groove as pleasure is described as judging the quality of a groove, namely as a “pleasureable drive towards action” [16, pp. 4–5]. Finally, groove as a state of being refers to the euphoric mindset that musicians or listeners find themselves in when they experience a groove they perceive as good [16, pp. 5–6]. For the purpose of this thesis, only groove as pattern and performance is considered due to its more analytical nature as opposed to the other, more abstract understandings of groove.

2.1.2 Groove Characteristics

Câmara and Danielsen [16] break down the characteristics of groove-based music into five features: pulse, subdivisions, syncopation, counter-rhythm, and microrhythm. The pulse of the groove is defined as the steady beat that keeps the groove going, acting as the foundation to add other groove characteristics on top of [16, pp. 6–7]. The subdivisions of a beat are defined as the notes played at faster metrical levels than the beat, which are generally considered necessary to establish a groove and give a sense of drive to the groove [16, pp. 7–8]. Syncopation enhances a groove by temporarily displacing the normal accent of the meter, and is considered to be the most important element in defining a style of groove [16, pp. 8–9]. Counter-rhythm is defined as “momentary instances of cross-rhythm or systematic off-beat rhythm whose ultimate purpose is to destabilize, but not fundamentally challenge, the main pulse” [16, pp. 9–10]. Finally, microrhythm is defined as a slight deviation from the meter in the order of milliseconds that helps to make the groove more dynamic and is more often felt rather than implicitly heard [16, p. 10]. In this thesis, pulse, subdivisions, and syncopation are the characteristics that will be focused on.

2.1.3 Measurements of Syncopation

As perhaps the most defining feature of a groove, measuring syncopation mathematically is an important challenge to express the groove of a rhythmic pattern. Gomez et al. [17] define three mathematical values that can be used to measure syncopation: off-beatness, Keith's measure, and Weighted Note-to-Beat Distance (WNBD).

Off-beatness is defined as when an onset is detected that does not belong in any rhythm in the 12-unit time span [17]. In regards to Table 3.1, an onset would be considered off-beat if it was placed in the 1, 5, 7, or 11 unit bin (see section 3.1.2). Therefore, the measure of the off-beatness of a rhythm is the number of off-beat onsets divided by the total number of off-beat unit bins.

Keith's measure is defined as a way to quantify hesitation, anticipation, and syncopation between different notes [17]. Unlike the off-beatness measure, these three events require both the beginning and the end of a note to calculate the value of Keith's measure [17]. Hesitation is defined as when a note starts on an on-beat and ends on an off-beat, anticipation as a note starting on an off-beat and ending on an on-beat, and syncopation as when the note starts and stops on an off-beat [17]. Each note is then given a value between 0 and 3: 0 for no event, 1 for hesitation, 2 for anticipation, and 3 for syncopation [17].

WNBD is a way to measure syncopation based on distance between notes [17]. To find the WNBD of a note, the T(x) value, which equals the minimum distance between the note and the previous or next pulse as a fraction, must first be defined [17]. If the beat occurs on a pulse, the WNBD is 0 [17]. If not, the WNBD value is then defined in relation to the end of the beat, which is considered to be the start of the next beat [17]. If the beat ends before the next pulse, on the next pulse, or any time after the next two pulses, Equation 2.1 is used [17].

$$WNBD = 1/T(x) \tag{2.1}$$

If the beat ends between after the next pulse, but before or on the pulse after the next pulse, Equation 2.2 is used [17].

$$WNBD = 2/T(x) \tag{2.2}$$

Due to the T(x) value being the denominator for all WNBD values, more weight is given to notes closer to pulses, without occurring on a pulse [17]. Also, the numerator is larger when the end of the note crosses over a single pulse due to the feeling of syncopation being stronger in this case [17].

2.2 Music Recommendation Systems

This section will provide relevant background on MRSs, including how the major streaming services serve recommendations to their users, evaluation techniques and criteria for MRSs, and current challenges in the field of study.

2.2.1 MRS Techniques

In [18], Paul and Kundu outline the five most common forms of music recommendation: collaborative filtering, content-based filtering, metadata-based filtering, emotion-based filtering, and the context-based model. This section will describe each of these techniques and list the pros and cons of using these techniques in a MRS.

Collaborative Filtering

The collaborative filtering method serves recommendations to a user based upon the similarity of their item ratings to other users [18, pp. 280–281]. Items ratings can either be explicit numerical ratings or implicit ratings based upon the behavior of a user on the streaming platform [18, p. 281]. While analyzing user behavior is useful for serving recommendations, this filtering method suffers when there are too few users to analyze and too few ratings on an item, which is known as the cold start problem (see section 2.2.3) [18, p. 281].

Content-Based Filtering

In the content-based filtering method, features are extracted from musical items through either programmatic or manual analysis, and are then used to match items to users who typically rate items with similar features highly [18, p. 281]. To compute similarity, algorithms such as k-means clustering (see section 2.4) can be employed [18, p. 281]. A major advantage of this approach compared to collaborative filtering is that the cold start problem is solved, since it recommends items solely based on the features of an item [18, p. 281]. However, it fails to take into account differences in songs for features that are not analyzed, and it is a challenge to make sure the item model is correct without extensive testing [18, p. 281].

Metadata Based Filtering

The metadata based filtering method uses basic song information to serve recommendations to users without taking into account the user profile [18, p. 282]. This technique generally produces poor results due to the shallow nature of metadata features, but can be improved through using a hybrid approach with other filtering techniques [18, pp. 282–283].

Emotion Based Filtering

The emotion based filtering technique attempts to filter using the emotions that a musical item inspires [18, p. 282]. While the satisfaction of having musical items that match a user's emotions is high and much research is ongoing in this sector, the manual labor to label each song is time consuming and prone to error due to the emotional ambiguity between different users [18, p. 282].

Context-Based Model

The context-based model method focuses on the context surrounding the song and examines how the musical item is perceived by the public and what users in the same area are listening to [18, pp. 282–283]. Recommendations are served based upon a user's information and compares it to what other users like them are saying on social media websites or other users in a similar location [18, pp. 282–283]. In a non-hybrid approach, the context-based model is one of the most effective ways to serve recommendations due to the low amount of data necessary to serve relevant recommendations [18, pp. 282–283].

2.2.2 MRS Components

Song et al. [2] break down the components of a MRS into three elements: users, items, and user-item matching algorithms [2, p. 397]. This section will explain the importance and details of user and item modeling, which are the most relevant to this thesis.

User Modeling

Making a model of a user is important since studies have shown that user personality is strongly linked to musical preference [2, p. 397]. In order to properly profile a MRS user, one must take into account both a user profile and a user experience [2, p. 397]. A user profile model can be simply broken down into demographic, geographic, stable psychographic, and fluid psychographic data, while a user listening experience model attempts to classify a user according to musical expertise as either a savant, enthusiastic, casual, or indifferent user [2, pp. 397–398].

Item Profiling

Similarly to modeling a user, each item in a MRS must be modeled as well. This is typically done by using three styles of metadata: editorial, cultural, and acoustic [2, p. 398]. Editorial metadata is simple information about an item that is acquired by an editor, and is usually employed in metadata information retrieval tasks [2, pp. 398–399]. Cultural metadata comes from external research about patterns in music, and is mainly used in context-based information retrieval [2, pp. 398–399]. Finally, acoustic metadata are features that are retrieved from the audio analysis of an item, and are how most MRSs serve content-based recommendations [2, pp. 398–399].

2.2.3 Challenges in MRS Research

According to Schiedl et al. [14], current challenges in contemporary MRS research include creating a proper evaluation strategy and registering new users in the system. This section will explain these problems as it relates to this thesis, and will discuss approaches towards solving these problems.

MRS Evaluation Metrics

While accuracy related evaluations are common and well documented, an evaluation strategy for MRSs must include additional, novel measures, such as utility and surprise. Therefore, measures for evaluating MRSs can be defined as either standard Music Information Retrieval (MIR) methods or beyond-accuracy measures, due to their difficulty to describe mathematically [14], [19]. Two standard MIR metrics used to evaluate the prediction power of MRSs, Mean Absolute Error (MAE) and Root-Mean-Square Error (RMSE) calculate the difference between the predicted rating and the actual rating of a specified number of items for a specific user, but RMSE differs from MAE by penalizing larger differences between the predicted and actual rating more than MAE. MAE and RMSE can be calculated in equations 2.3 and 2.4, respectively, where i is the item, u is the user, $r_{u,i}$ is the actual rating, and $\hat{r}_{u,i}$ is the predicted rating.

$$MAE = \frac{1}{|T|} \sum_{r_{u,i} \in T} |r_{u,i} - \hat{r}_{u,i}| \tag{2.3}$$

$$RMSE = \sqrt{\frac{1}{|T|} \sum_{r_{u,i} \in T} (r_{u,i} - \hat{r}_{u,i})^2} \tag{2.4}$$

A scenario Schiedl et al. [14] does not mention is if the prediction power of the MRS needs to be evaluated, but an expected value does not exist. In this scenario, the data can be analyzed using the standard deviation, which is the measure of the variance of a data set [20]. Equation 2.5 shows the equation for the standard deviation of a data set, where x_i is a value in a data set, x_{avg} is the average value of all items in the data set, and n is the number of items in the data set, and ddof is the delta degrees of freedom [21], [22].

$$std = \sqrt{\frac{\sum(x_i - x_{avg})^2}{n - ddof}} \tag{2.5}$$

An example of a beyond-accuracy metric is serendipity, which “aims at evaluating MRS based on the relevant and surprising recommendations” [14, p. 105]. Measuring serendipity according to unexpectedness mixed with the relevance of an item was first proposed by Ge et al. [23, p. 259]. A version of this equation, provided by Schedl et al. [14, p. 105] is calculated using equation 2.6, u is a user, L_u is the list of items recommended to a user, L_u^{unexp} is the list of items labeled unexpected by the user, and L_u^{useful} is the list of items labeled useful by the user. Useful items are usually found by either asking the user or using a rating to interpret the usefulness of an item, while unexpected items are usually found by finding the distance between an item and an expected item, which can be thought of as an item already rated by a user [14, p. 105].

$$serendipity(L_u) = \frac{|L_u^{unexp} \cap L_u^{useful}|}{|L_u|} \tag{2.6}$$

Zhang et al. [24] have proposed a measure of unserendipity based upon the similarity of a new recommendation and a user’s history. However, a measure that appears to be overlooked by literature on the topic is the measure of an item being not useful but expected, henceforth referred to as anti-serendipity. The concept of anti-serendipity has previously been examined by Cooper and Prager [25] in relation to identifying unimportant or irrelevant digital documents. Similarly, a problem in MRSs is the tendency to recommend repetitive content contributing to fatigue and criticism from users [26]. Anti-serendipity can therefore be thought of as the polar opposite of serendipity, measuring how likely a MRS is to recommend irrelevant content that does not surprise. It can be calculated using equation 2.7, where L_u^{exp} is the list of items labeled expected by the user, and $L_u^{notuseful}$ is the list of items labeled not useful by the user.

$$anti - serendipity(L_u) = \frac{|L_u^{exp} \cap L_u^{notuseful}|}{|L_u|} \tag{2.7}$$

Cold Start

A current challenge in MRS design is how to integrate new users or new musical items to the service, which Schiedl et al. [14] defines as the cold start problem. There are four main approaches used by the industry to solve this problem: content-based approaches, hybridization, cross-domain recommendation, and active learning [14, p. 98]. As described in section 2.2.1, content-based approaches serve recommendations solely based on the preferences given by the current user, which only addresses the problem of a new user and not new musical items [14, p. 98], [18, p. 281]. To solve this issue, acoustic features of new items are analyzed either automatically or manually, which are in turn used to predict the preference of a new user [14, p. 98]. A limitation of this approach is the fact that the same acoustic features tend to be used, while studies such as the one performed in [27] show that different acoustic properties affect user taste in individualized ways [14, p. 99]. Hybridization is defined as a dual approach, using a combination of a content-based recommender for analyzing acoustic properties and a collaborative filtering recommender using information from other users [14, pp. 98–99]. While there are usually benefits in a hybrid approach, the approach is often overly complex to understand and computationally heavy [14, p. 99]. Cross-domain recommendation is explained as using information from a different domain, such as a user’s personality, in order to improve results in the MRS domain [14, p. 99]. An obvious problem with this approach is the lack of good quality and easily accessible datasets that could effectively link musical taste to another domain [14, p. 99]. Finally, active learning is a system that learns from its active users to continuously improve and find better recommendations given specific user feedback [14, p. 99]. Major issues with active learning are the lack of a personalized approach to this approach's black box model, and the need to integrate user interfaces and profiles in order to assist in solving this problem, creating more work in the process [14, pp. 99–100].

2.3 Audio Analysis and Processing

This section will describe the process of finding beats and onsets of percussive elements in audio files, including the process of separating percussive elements from a piece of raw audio.

2.3.1 Musical Source Separation

A typical source separation problem involves extracting a set of independent signals from a combination of signals [28]. Musical source separation involves separating independent signals in a piece of raw audio, which presents a more challenging problem since recording, mixing, and production parameters are prone to change between audio files and sometimes within the same audio file [29, p. 31]. This process can be particularly helpful in the process of beat tracking and onset detection due to the ability to remove the harmonic signals that act as interference in these processes [30, p. 148]. This section will outline two methods to achieve musical source separation for those tasks: using Harmonic-Percussive Source Separation (HPSS) and using deep neural network tools.

Harmonic-Percussive Source Separation

Musical sources are usually broken down into harmonic, percussive, or vocal sources [29, p. 32]. The goal of HPSS is to separate the pitched, or harmonic, instruments from the percussive instruments [29, p. 32]. One technique to achieve HPSS involves finding percussive audio by finding a section in an audio file with noisy phase behavior [31]. This section will focus on a more common HPSS method using median filtering, developed by Fitzgerald [32].

The median filtering technique for HPSS was developed in an attempt to create an approach for HPSS that required no pretraining and was less processor and memory intensive than a previously developed approach using tensor factorization [32, p. 1], [33]. The basis of this technique was inspired by Ono et al. [34], who had previously determined that horizontal ridges on a spectrogram corresponded to harmonic components (see Figure 2.1) and vertical ridges corresponded to percussive components (see Figure 2.2) [32, p. 1]. Using this information, a cost function that emphasizes horizontal lines while backgrounding vertical lines for harmonic components, and vice versa for percussive components, was developed [32, pp. 1–2]. Next, a median filtering process takes place [32, p. 2]. Median filters, which are commonly used for signal processing tasks with wide spectrums and sharp edges on signals [35], operate by "replacing a given sample in a signal by the median of the signal values in a window around the sample" [32, p. 2]. By viewing harmonic components as outliers in the percussive spectrum and percussive components as outliers in the harmonic spectrum, median filters are used horizontally and vertically to smooth out spikes in harmonic and percussive events, which finally outputs Harmonic Source Separated (HSS) and Percussive Source Separated (PSS) audio [32, pp. 2–3]. This method is quite efficient as tests have shown that two passes of the median filter, one horizontally and one vertically, are sufficient to output high quality HSS and PSS audio [32, p. 3].

Spleeter

Approaches towards musical source separation using ML and deep learning neural networks have become popular in recent years [36]. One of the best open source tools for musical source

Figure 2.1: Mel-frequency spectrogram of an audio file with only percussive elements.

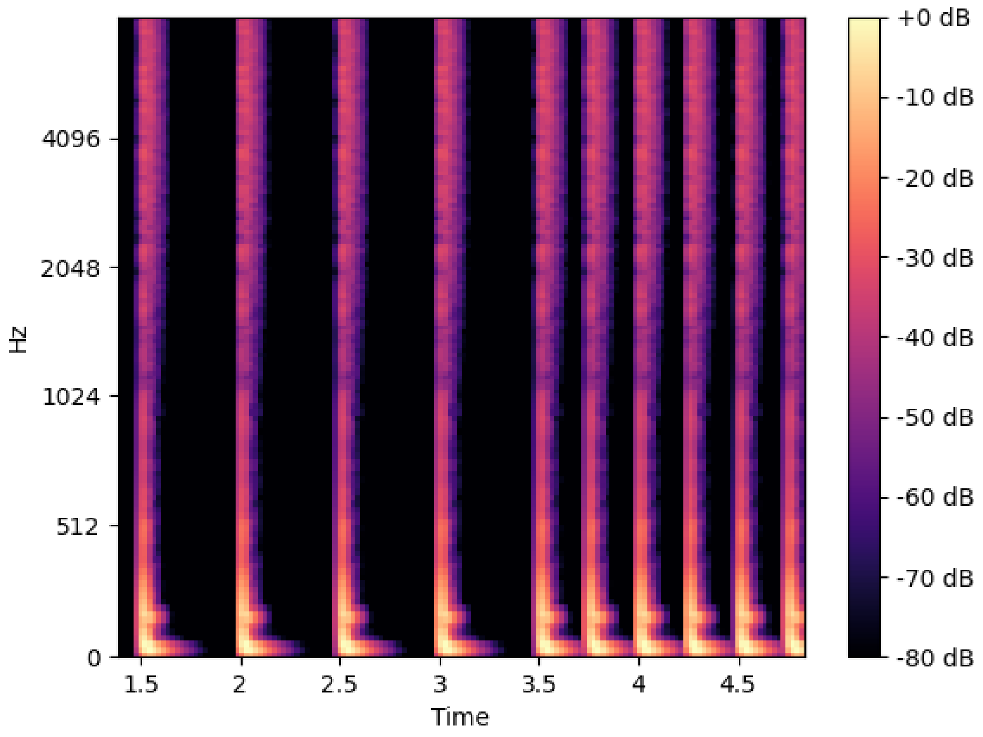
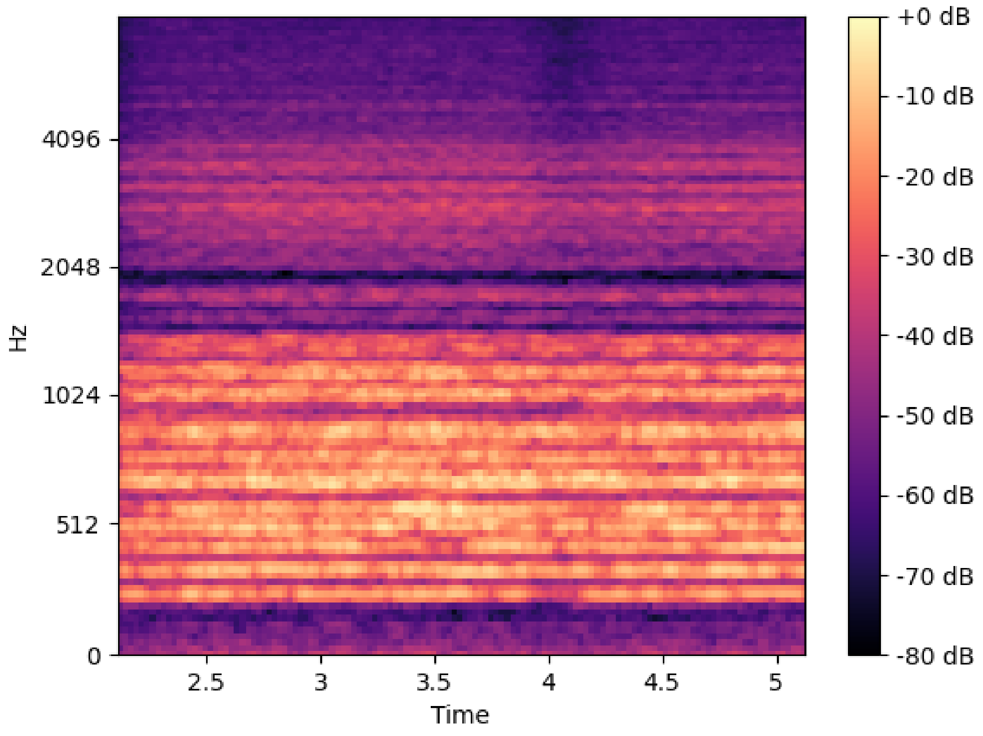


Figure 2.2: Mel-frequency spectrogram of an audio file with only harmonic elements.



separation is Spleeter [36], developed by Hennequin et al. [37]. Spleeter gives users the ability to split audio files into different stems using one of three pre-trained models: a two stem model for vocals and instrumentals, a four stem model for vocals, drums, bass, and other instrumentation, and a five stem model for vocals, drums, bass, piano, and other instrumentation [37]. While using neural networks is more time consuming than an HPSS based approach, an upside of using Spleeter as opposed to other state of the art software such as open-unmix [38] is its relative time efficiency [37]. Spleeter is able to process 100 seconds of audio every second while still producing competitive signal quality in terms of minimizing interference, distortion, and artifacts [37, pp. 2–3].

2.3.2 Onset Detection

Bello et al. [39] describes an ideal single note as containing an attack, transient, onset, and decay. To understand onset detection, the concepts of attacks, transients, and onsets must first be understood. An attack is when the amplitude envelope of a note increases, a transient is when an interval of a signal evolves in an unpredictable way, and an onset is the instance when a transient is detected [39, pp. 1035–1036]. A typical onset detection algorithm processes the original audio signal two or three times [39, p. 1036]. These stages are an optional pre-processing stage to improve performance, a reduction stage which turns the signal into a detection function that highlights the transients, and a peak-picking stage to find the onsets which peak above a certain threshold [39, p. 1036].

2.3.3 Beat Tracking

A beat is defined in this thesis as an estimation of the steady pulse of an audio file. The standard algorithm for beat tracking, developed by Ellis [40], attempts to reconcile the conditions of placing a beat where an onset is detected while maintaining an inter-beat-interval with regular spacing between beats. This is accomplished by first converting audio files into an onset strength envelope, which shows the best candidates for beats where the onset strength is highest [40, pp. 52–53]. Using the onset strength envelope, tempo of the audio file is then estimated using an autocorrelation equation that biases the tempo towards 120 beats per minute [40, pp. 53–54]. To maximize the onset strength and inter-beat-interval consistency, a dynamic programming approach, first developed by Bellman [41], is used [40, pp. 51–53]. It is worth noting that this technique has some limitations. Due to the reliance on a constant tempo for the dynamic programming equation, non-constant tempos are not accommodated in this approach [40, p. 58]. Also, this approach achieves a beat accuracy of around 60% in tests with the state of the art beat tracking training data set, meaning it can still frequently be prone to error despite a relatively high degree of accuracy [40, p. 51].

2.3.4 Audio Analysis Parameters

Audio analysis functions, including onset detection and beat tracking, typically need certain parameters that provide necessary information about the audio and help fine tune the analysis. Two of those parameters are the sampling rate, which is the frequency in Hz of the audio signal, and the hop length, which is the size of the frames that are used to analyze a signal [42, p. 19].

2.4 K-means Clustering

Jain et al. define data clustering as the “unsupervised classification of patterns (observations, data items, or feature vectors) into groups (clusters)” [43, p. 264]. A clustering task must first initiate pattern representation, which references the number and type of patterns in the clustering process [43, p. 266]. To accomplish this, a data clustering algorithm could use feature selection, which determines the best features to cluster the data according to, feature extraction, which transforms input data into new features for the clustering algorithm, or a combination of both [43, p. 267]. Next, a distance function is used to determine the similarity of two different patterns in a process called pattern proximity [43, p. 267]. As the final required step in the clustering process, the data then undergoes grouping using different techniques, including the k-means clustering technique used in this thesis [43, p. 267]. Optionally, the clustering process may include data abstraction, which involves defining a simplified version of a larger data set [43, p. 267]. Analysis of cluster quality can include cluster tendency, which is the analysis of the data itself to determine if clustering is an appropriate method to group data points, and cluster validity, which is the subjective analysis of the quality of clusters produced [43, pp. 267–268].

Clustering algorithms can be defined as partitional, hierarchical, artificial system, kernel-based, and sequential [44]. One of the most predominant categories are partitional algorithms, which aim to acquire a single partition of a data set where all clusters are found at the same time [43, p. 278]. These algorithms stand in contrast to hierarchical algorithms, which produce nested clusters which represent the levels at which cluster groups differ [43, p. 275]. K-means clustering belongs to a sub-group of partitional algorithms known as squared error algorithms, which attempts to minimize the squared error for a cluster [43, p. 278]. This is done using the equation 2.8, where L is a clustering, R is a pattern set, K is the number of clusters, x_i^j is the i^{th} pattern in the j^{th} cluster, and c_j is the centroid of the j^{th} cluster. [43, p. 278].

$$e^2(R, L) = \sum_{j=1}^{K} \sum_{i=1}^{n_j} |x_i^j - c_j|^2 \tag{2.8}$$

The k-means clustering algorithm begins by selecting k cluster centers, assigning each point in a data set to a cluster by finding the nearest cluster center to each point using the aforementioned squared error function, and redefining the center of each cluster based upon the formed clusters [43, pp. 278–279]. Next, these clusters are examined for a convergence criterion, which is usually examining either the decrease of squared error in each cluster or the number of reassignments of data points to new clusters [43, p. 279]. Until the convergence criterion is met, new clusters and cluster centers are found using the previously computed cluster centers [43, p. 279]. The standard algorithm used for k-means clustering is Lloyd's algorithm [45], [46].

An optimal k number of clusters for a data set can be found using a variety of techniques, including the elbow and silhouette method [47]. The elbow method is a visual based technique, where each consecutive k value is plot along the x axis, with the y axis being a cost function relating to the distance between data points and cluster centers, such as the sum of squared errors [47, p. 92], [48]. The k value at which the cost function drops dramatically before leveling off is known as the elbow point, and represents the optimal k value to use [47, p. 92]. A downside of the elbow method is that elbow points can sometimes be ambiguous [47, p. 92]. The silhouette method, an approach that compares the distance between points in a cluster with the distance between clusters, can be used either as a more consistent alternative [49] or in conjunction with the elbow method to find the optimal k value [47, p. 93]. This method calculates the silhouette width s(i) using equation 2.9 where i is a piece of data in the data set, a(i) is the average distance between i and all other pieces of data in its cluster, and b(i) is the minimum of the average distances between i and all other pieces of data in every cluster besides the one it belongs to [47, p. 94]. The k value that returns the largest average silhouette width value for all data points can then be declared the optimal number of clusters for the data set [47, p. 94].

$$s(i) = \frac{b(i) - a(i)}{max(a(i), b(i))} \tag{2.9}$$

The k-means algorithm is a popular option for solving clustering problems due to its relative ease of implementation and its computational efficiency [43, p. 278]. In terms of big O notation, which according to Chivers and Sleightholme is a method to communicate the efficiency of an algorithm based upon the data set it works on [50, pp. 359–360], the k-means algorithm achieves O(n), or linear, efficiency [43, p. 278].

Chapter 3

Methodology

This section will describe the design of the ML system used to extract groove features from raw audio, as well as the design of the evaluation of this system.

3.1 ML System

The ML system developed for this thesis is based on detecting onsets and beats in raw audio files and quantifying those values into values representing groove features. Once the groove features were quantified, they were used as features in a k-means clustering algorithm that sorts the audio files into clusters with similar grooves. This section will describe in detail the process to accomplish this system.

3.1.1 Percussive Source Separation and Onset Detection

The first task of developing this system is to process the selected audio files and extract each onset in preparation to analyze the groove characteristics. While harmonic elements may contain elements of groove, only the percussive groove of the musical item was analyzed in order to simplify data. Each analyzed musical item was processed to isolate only the drums using different PSS techniques to ensure that onset detection is as accurate as possible. These different PSS techniques were then compared using a test track with manually added percussive notes to determine which PSS techniques should be tested going forward. The results of this test are reported in section 5.1.

3.1.2 Groove Feature Extraction

Of the aforementioned five features of groove, three of them were focused on when extracting groove features from musical items: syncopation, pulse, and subdivisions. Once the onsets and beats of an item were found, a program was written to assign values to each of the aforementioned components of groove using the calculated beats and onsets. For every beat besides the last beat, the time between the current beat and the next beat is calculated and divided by 12, which represents the total amount of time for each beat bin. Since onsets are the start of a pulse, they have a tendency to fall slightly before the intended beat. Therefore, the beat bin time divided by 2 before and after a beat bin divider is analyzed for each beat bin. If an onset is found within this time, the corresponding beat bin then increases the number of onsets found by 1. This also filters out duplicate onsets in case the onset detection function found multiple onsets within the same time frame of a beat bin. Finally, once all beats are analyzed, the data collection can begin.

	Beat subdivisions in a 12-unit time span											
Rhythm	0	1	2	3	4	5	6	7	8	9	10	11
Fourth Note	x											
Eighth Note	x						x					
Sixteenth Note	x			x			x			x		
Eighth Note Triplet	x				x				x			
Sixteenth Note Triplet	x		x		x		x		x		x	
Off Beat		x				x		x				x

Table 3.1: Rhythms in a 12-unit time span.

Measuring Pulse and Subdivision

Each beat for all musical items in this project is defined in a 12-unit time span, which allows groupings based on two, three, four, and six units. Table 3.1 shows the different rhythms that were analyzed for each item and which unit the onset needs to fall on in order to be on-beat for that specific rhythm, which is represented by an x.

The fourth note rhythm can be considered the pulse of the beat, while all other unit divisions besides those marked "off beat" are the subdivisions. After all the onset beat bin totals were found, the number of times an onset appears in an on-beat bin for each rhythm is divided by the total number of onsets found for all beat bins to normalize their values. These values numerically represent the likelihood of an onset in a rhythm being played across all onsets in the song segment. The normalized values for pulse, eighth notes, sixteenth notes, eighth note triplets, and sixteenth note triplets are then used as features in the ML system.

Measuring Syncopation

As previously mentioned in section 2.1.3, off-beatness was measured by finding the number of pulses in the off-beat bins, corresponding to bins 1, 5, 7, and 11 on table 3.1. WNBD was calculated using the method described in Section 2.1.3, with the beats being the beats calculated by the beat tracking algorithm and the pulses being the onsets detected. Between Keith's measure and WNBD, only WNBD was considered as a feature in the ML system. This is due to the more arbitrary nature of Keith's measure in assigning values of syncopation, as well as a study conducted by Gomez et al. where it was found results of Keith's measure to be effectively similar to a measure of off-beatness through finding that their p-values obtained through rank correlation analysis are very close to each other [51, pp. 8–9]. Therefore, the two features used to represent syncopation were decided to be off-beatness and WNBD.

3.1.3 Clustering

Next, a ML, k-means clustering algorithm was created in order to cluster items together based upon similar groove features. The number of clusters were determined through manual analysis using the elbow and silhouette methods, as well as analyzing the resulting sizes of the clusters to make sure they were relatively even. To determine the cluster validity, the six most central tracks of each cluster were examined in a cluster tendency analysis in order to determine if the percussive elements have groove characteristics in common. This manual analysis is presented in section 5.3.2.

3.2 Evaluation

This section will describe the structure of the evaluation sessions for the ML system as well as how data was collected and analyzed.

3.2.1 Participant Selection

A total of 18 participants were selected for this study. Diverse perspectives on music is desired for the participants in this evaluation to test the robustness of the MRSs. Because of this, personal information was not considered during participant selection for this evaluation in order to focus purely on the musical taste of the participant. Likewise, no personal information was collected in the analysis of the MRSs in an attempt to make this test a purely content-based approach.

3.2.2 Test Procedure

The evaluation session consists of a participant listening to multiple 30 second segments of songs. Each of these items are one of the six closest items to the centroids of either a Librosa PSS cluster or a Spleeter PSS cluster, since these can be considered the most representative items of each cluster. Therefore the number of items each participant listens to is equal to six times the number of Librosa PSS and Spleeter PSS clusters formed by the ML system. All six of the items belonging to the same cluster are played in a row, with the first section of the test playing items from the Librosa PSS clusters and the second section playing items from the Spleeter PSS clusters. In order for participants to know when a musical item started and stopped and to give enough time for the participant to enter their responses, two seconds of silence were added after the end of each track.

The participants then rank these musical items on two different seven point Likert scales [52]. The first scale determines the participant's subjective enjoyment of the track, with 1 corresponding to "extreme dislike" and 7 corresponding to "extreme like." The second scale determines how likely the participant would be to listen to the track, or in other words how expected the item was, with one corresponding to "extremely unlikely" and seven corresponding to "extremely likely." The words "extreme" and "extremely" on each end of the scale were used in order to establish bipolarity within the participant's response [53]. At the end of the test, the participants were asked to state their most and least liked items in the selection in order to represent the extremes of each participant's taste, and finally were asked for additional comments about the test if they had any. Essentially, this test functions as an extended content-based cold start test to determine which clusters match the user's taste the most, but instead of analyzing total quality, cluster item similarity was analyzed, as described in the next section.

3.2.3 Data Analysis

To evaluate the similarity of the quality and expectedness of items in a particular cluster, the standard deviation was calculated between the six evaluated tracks in every cluster for every participant for both quality and expectedness values. Since the goal of the system is to have the quality values of each cluster be relatively the same, standard deviations for the quality ratings that are closer to 0 are more desirable. However, high or low values for the standard deviation of expectedness values are not considered to be good or bad, but rather indicate how consistently clusters serve expected or unexpected items. For both calculations, the delta degrees of freedom was equal to 1, since this has been shown to help correct smaller sample sizes, and with a sample size of six for each cluster the sample size can be considered sufficiently small [54]. Also, the clusters of items that the participant labeled their favorite and least favorite were analyzed to determine if those clusters have higher or lower average quality and expectedness values compared to all other clusters. The accuracy measures MAE and RMSE were considered for analysis, but were not calculated for either quality or expectedness on account of there not being an expected value for any of these tracks for each user.

Furthermore, the beyond-accuracy measures serendipity and anti-serendipity were also analyzed. Tracks that a user rates 1 to 3 on the quality scale were labeled not useful, while tracks rated 5 to 7 on the quality scale were labeled useful. Similarly, tracks rated 1 to 3 on the expected scale were labeled unexpected and tracks rated 5 to 7 were labeled expected. Any item a user rates a 4 on the quality or expected scale were excluded from these measures due to the implied neutrality of the participant. If an item is labeled useful and unexpected, the cluster is marked as containing a serendipitous track, and if an item is labeled not useful and expected, the cluster is marked as containing an anti-serendipitous track. As a supplement, tracks that were labeled not useful and unexpected, henceforth referred to as true negatives, as well as tracks labeled useful and expected, henceforth referred to as true positives, were also labeled.

To analyze these beyond accuracy measures, the total number of accounts of serendipity, anti-serendipity, true positives, and true negatives across all users for each cluster will be examined. This was done to see if either PSS method recommends more serendipitous or anti-serendipitous tracks overall, and to see if there was an overall even distribution of true positive and true negative tracks, which indicates a normal distribution of items. Next, the average quality and expectedness ratings of clusters containing accounts of serendipity or anti-serendipity were then compared against the average quality and expectedness ratings of all tracks for a given participant. This analysis was done to see if containing an item that a participant marks serendipitous or anti-serendipitous has any effect on the average quality or expectedness value of a cluster.

Chapter 4

Design and Implementation

This section will describe how the ML system and evaluation session were implemented and will list the tools and resources used in all areas of this project.

4.1 ML System

This section will report how the ML system was developed and implemented, and elaborates on the choices made for the packages, datasets, and tools used.

4.1.1 Song Selection

The audio files used in this project were taken from the Free Music Archive (FMA), developed by Defferrard et al. [55]. The FMA was developed specifically for MIR tasks due to the lack of accessible large music datasets available for free use [55]. This ML system will use a subsection of the dataset named fma_small featuring 8000 30-second segments of the songs in their library, which other research papers on MRSs have deemed sufficient [56], [57][1]. For this thesis, the FMA was the best fit compared to other datasets due to its large variety of artists which are mainly non-commercial and thus does not introduce any bias caused by participants having heard the song previously.

4.1.2 Percussive Source Separation

Spleeter and the Python package Librosa's HPSS function were used to isolate the percussive elements in each musical item [37], [58]. These two methods were chosen after testing the onset and beat detection system using test audio files, which will be elaborated on in Section 5.1. The code for this initial test can be found in Appendix A[2]. For Spleeter, the 4stems model was used to split the item into vocals, bass, drums, and other stems, and proceeded to use only the drums stem to find beats and onsets. A Python script, which can be viewed in Appendix B[3], was written to call Spleeter for all items in the FMA. For Librosa, the HPSS effect was used, which performs a short-time Fourier transform to provide information in the frequency domain, then performs HPSS based upon Fitzgerald's median filtering approach [32], [59], and finally performs an inverse short-time Fourier transform to return the final percussive audio [60].

[1]While this study uses the fma_medium dataset, the study filters the number of files analyzed down to 7000, making it comparable to the fma_small dataset.

[2]Also available on GitHub: https://tinyurl.com/4jr6s4vc

[3]Also available on GitHub: https://tinyurl.com/m6mmr9v7

4.1.3 Onset Detection and Beat Tracking

Librosa was also used for its beat tracking function, which is inspired by Ellis's beat tracking with dynamic programming approach [40], [58], [61], and for its onset detection function. The hop length of the onset and beat tracking functions are set to 220 samples to maximize both clarity and time efficiency, and the sample rate of the audio files are 44100 Hz. The code for this section can be viewed in Appendix C[4].

4.1.4 Groove Feature Extraction

Groove features were calculated with a Python script, using the beats and onsets previously found to quantify pulse, subdivisions, and WNBD. The code for this section can be viewed in Appendix D[5].

4.1.5 Clustering

The Python package Scikit-Learn's k-means clustering algorithms were used to cluster the data [62], and uses the default Lloyd's algorithm [45]. The number of clusters were first decided by using the elbow and silhouette methods. The elbow method was implemented by repeatedly running the clustering function with an incremental k value between 2 and 10. Each time the clustering function is run, the sum of squared distances for all items from its cluster center is calculated and placed on a graph for manual analysis. For the silhouette method, the silhouette coefficient is calculated using Scikit-Learn's built in function using euclidean distance calculations. The cluster formation process is described in full in section 5.3.1. The code for this section can be found in Appendix E[6].

4.2 Evaluation

The audio file used in the evaluation of this system was created by stitching together song segments from the FMA into a single audio file using GarageBand. Six Librosa PSS clusters and five Spleeter PSS clusters were formed (see Section 5.3.1), and six tracks were taken from each cluster. The overall run time of the evaluation session therefore ran for 35 minutes and 10 seconds[7], and participants were instructed to listen to all of the samples in a row without interruption.

Participants entered their responses on a Google Form, and their responses were later analyzed using a Python script. The analysis script used the Python package NumPy [63] and its functions to calculate important values, including mean values [64] and standard deviation [22], where delta degrees of freedom was equal to 1. The Python analysis script can be found in Appendix F[8]. Once all the responses were collected, the Python library Matplotlib [65] was used to graph various results in a more human readable manner.

[4]Also available on GitHub: https://tinyurl.com/ywbpsn6h
[5]Also available on GitHub: https://tinyurl.com/mpv7j27s
[6]Also available on GitHub: https://tinyurl.com/8kjttcdb
[7]This file can be downloaded on GitHub: https://tinyurl.com/472h8xcv
[8]Also available on GitHub: https://tinyurl.com/29j7dy7s

Chapter 5

Results

5.1 Onset Detection Initial Test Results

Four methods were considered to find beats and onsets: using raw audio, PSS audio with Librosa, PSS audio with Spleeter, and PSS audio with Spleeter and Librosa. To test the accuracy of each method, two audio tracks were created[1]. One track featured only drums playing one measure of notes on the fourth notes, one measure of eight notes, one measure of eighth note triplets, one measure of sixteenth notes, one measure of sixteenth note triplets, and one measure featuring all off beats. This makes for a grand total of 80 onsets for the onset detector to find. The next audio file consists of the same drum pattern, but with additional instrumentation including bass, synth, and a choir sample to simulate vocals. Both files play at a tempo of 120 beats per minute, as this is the tempo that beat tracking systems bias towards [40, pp. 53–54]. The results of each method are found on table 5.1. The tracks with only drums begin with "Just Drums" and the tracks with full instrumentation begin with "Song," with the audio analyzed marked by either "Raw Audio," "Librosa PSS Audio," "Spleeter Audio," or "Librosa PSS and Spleeter Audio" afterwards.

As shown from the table, all methods produced 68.75% to 83.75% accuracy on predicting where onsets were located using detected beats, putting each well within the standard range of Ellis's [40] tests on dynamic beat tracking. The methods that analyzed the tracks with just drums produced between 71.75% and 73.75% accuracy, with the main issue across all methods being the prediction of the off-beat onsets. The raw audio and Spleeter PSS methods produced 71.75% accuracy, while the Librosa PSS and the Spleeter with Librosa PSS produced 73.75% accuracy. This means that for the track with just drums, any method using Librosa PSS was superior, which stands in contrast to the full instrumentation track test where both methods using Spleeter were superior.

For the test track with full instrumentation, the best approach was using both Spleeter and Librosa HPSS at 83.75% accuracy, with the pure Spleeter approach close behind at 78.25% accuracy. Compared to the pure drum track test, the approaches using Spleeter maintained around the same level of accuracy for on-beat onsets while fairing much better at detecting off-beat onsets. The approach using Librosa PSS fared slightly worse than the Spleeter approaches, scoring only 68.75% accuracy. In fact, the Librosa HPSS approach was less accurate than even the raw audio approach at 76.25% accuracy. The main difference between these two approaches was that Librosa HPSS registered slightly more offsets in incorrect locations than the raw audio approach. However, the Librosa HPSS approach found a total of 77 onsets while the raw audio approach found a total of 71 onsets, which makes the Librosa HPSS approach's total onset count closer to the target than the raw audio approach. Therefore, the two methods selected for analyzing beats and onsets were using Librosa PSS audio and using Spleeter PSS audio due to

[1]These files can be downloaded on GitHub: https://tinyurl.com/mr4bvc8a

target	Just Drums Raw Audio	Just Drums Librosa PSS Audio	Just Drums Spleeter Audio	Just Drums Librosa PSS and Spleeter Audio	Song Raw Audio	Song Librosa PSS Audio	Song Spleeter Audio	Song Librosa PSS and Spleeter Audio
20	23	23	23	23	23	23	21	20
4	0	0	0	0	0	0	0	0
4	6	6	6	6	4	6	6	6
4	5	4	5	4	4	4	4	5
8	7	8	7	8	8	9	9	8
4	2	2	2	2	2	1	2	4
12	13	13	13	13	13	15	14	12
4	2	2	2	2	2	2	2	2
8	9	9	9	9	7	9	9	9
4	5	5	5	5	5	5	5	5
4	3	3	3	3	3	3	3	3
4	0	0	0	0	0	0	3	3
Accuracy (%)	71.25	73.75	71.25	73.75	76.25	68.75	78.25	83.75

Table 5.1: Initial onset detection and beat tracking test.

the comparable accuracy and to compare and contrast both PSS methods individually in terms of grouping items with similar grooves together.

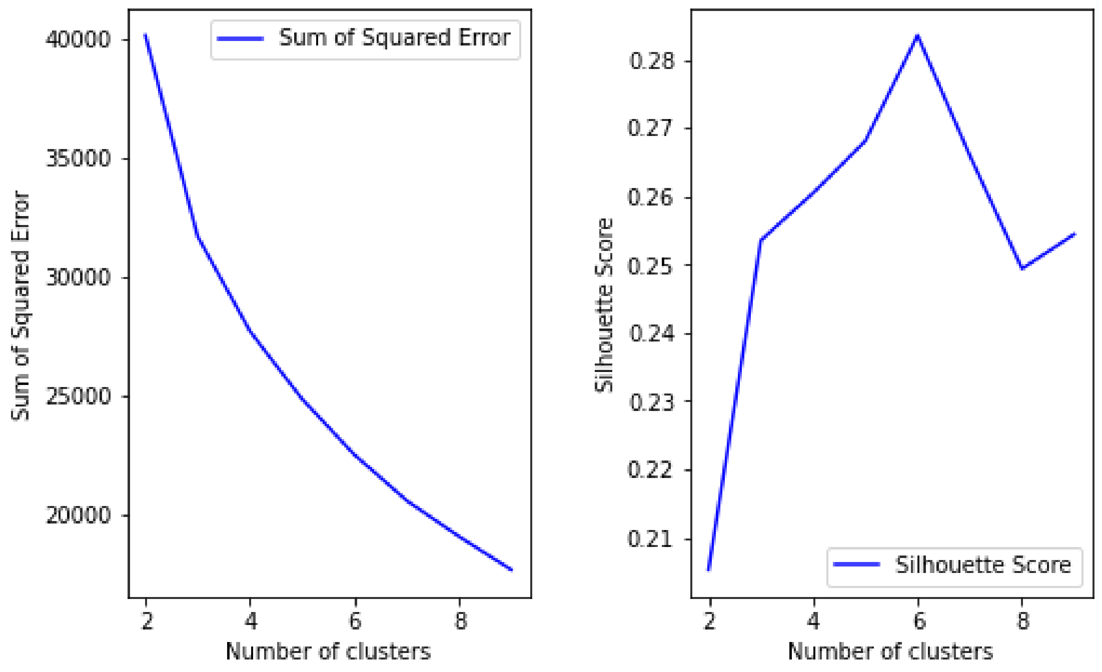
5.2 Beat Tracking Manual Analysis

While using the test audio tracks described in Section 5.1 helped to fine tune the onset detection and beat tracking system, using pieces of audio with various recording qualities, production styles, and percussive elements can prove to be challenging for beat tracking systems. As was shown from Ellis's study [40, p. 51], state of the art beat tracking systems are still only around 60% accurate. In order to analyze the robustness of the beat tracking system using PSS audio from both methods, manual analysis of a representative sample of tracks in the FMA archive was performed. The remainder of this section will review this manual analysis to show the different types of successes and failures in analyzing beats for tracks in the archive[2].

Tracks that feature steady, loud, and predominant drum recordings, such as tracks 135365, 135368, and 135372, all do a great job at finding steady, consistent beats for both PSS methods. However, tracks that do not have these sorts of drums result in all different kinds of undefined behavior depending on which PSS method was used. For instance, using the Librosa PSS method, item 135010 features a fairly steady pulse with only a moment of instability in the middle of the track, tracks 13092 and 135338 speed up and slow down the steady pulse throughout the duration of the track, item 135054 features no beat but the beat tracking system still finds one, and item 135363 features a fairly steady beat until vocals appear. Compared to the Spleeter PSS method, tracks 135010, 135092, and 135363 feature basically no beat at all, while tracks 135338 feature some semblance of a beat, but very off-kilter, and item 135054 has the same behavior as Librosa PSS method where there is no beat but one is still found, albeit in a completely different place. Based on the manual review, Librosa PSS handles beat tracking for tracks without strong drum recordings much better than Spleeter PSS, most likely due to the fact that Spleeter filters out all non-drum and weak drum sounds in each item so a groove is impossible to analyze. However, the results for both methods are nowhere near perfect and present a significant challenge for groove feature extraction moving forward.

[2]The tracks featured in this manual analysis with clicks that represent the beats detected can be found on GitHub: https://tinyurl.com/27y626e3 and https://tinyurl.com/2s4kh9za

Figure 5.1: The elbow and silhouette method charts for Spleeter PSS clusters.



5.3 Clustering

This section will describe the process of determining the number of clusters formed for both of the PSS audio methods, and will contain manual analysis of the six closest tracks to the center of each cluster to determine if there are certain patterns in groove style that emerge in each cluster.

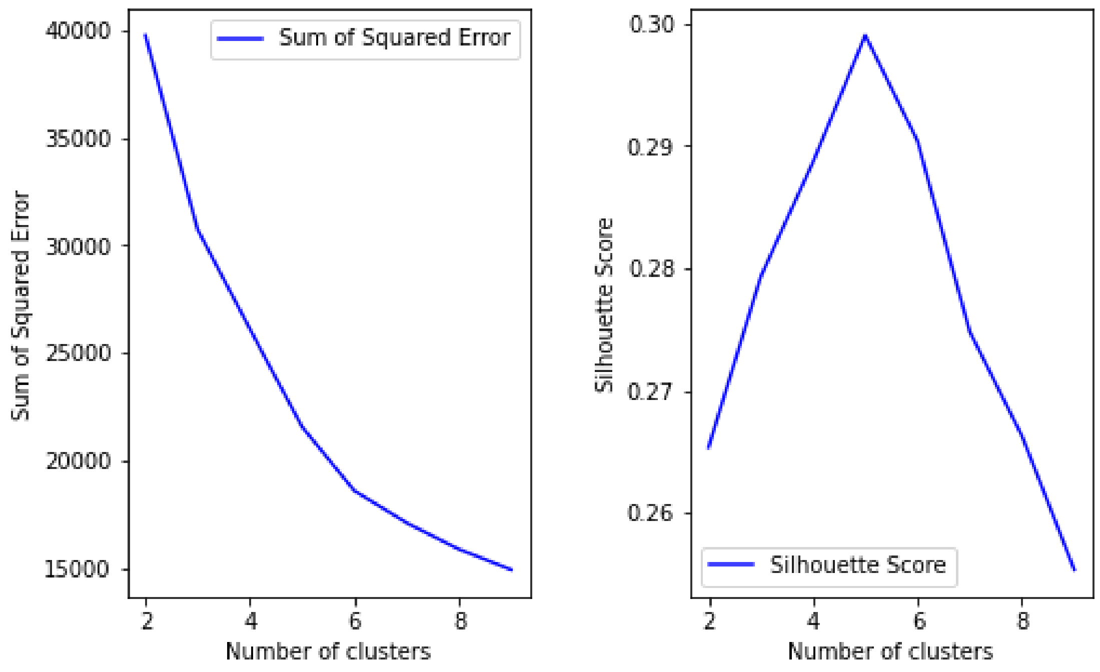
5.3.1 Cluster Formation

The number of clusters for both methods were determined through manual analysis using the elbow and silhouette methods, as well as analyzing cluster size to confirm that no cluster is too small or too large. Note that the cluster sizes reported represent the sizes of the clusters during one run through of the k-means algorithm with a given k value and that cluster sizes will vary each time it is run, but only by small amounts and with clusters mostly trading outliers, which were not important to the testing of these clusters.

The results of the elbow and silhouette methods for the Spleeter PSS audio approach can be seen in Figure 5.1. While the elbow method in Figure 5.1 shows no clear elbow points, the silhouette method shows a clear peak at 6 clusters. When the k-means algorithm is run with a k value of 6, the sizes of the clusters range from 78 to 1738 items. 78 was determined too small of a cluster size, so the next highest silhouette values, where k is 5 and 7, were analyzed next. When the k-means algorithm is run with a k value of 7, the sizes of the clusters range from 78 to 1818 items, making this approach worse for cluster size distribution than when k is 5. When the k-means algorithm is run with a k value of 5, the sizes of the clusters range from 830 to 2151 items, which is by far the most even distribution of clusters among the k values tested. Therefore, the Spleeter PSS audio was grouped into a total of 5 clusters.

The results of the elbow and silhouette methods for the Spleeter PSS audio approach can be seen in Figure 5.2. The elbow method in Figure 5.2 shows a subtle elbow point when the k value is 5 and a more dramatic elbow point when the k value is 6, while the silhouette method shows a peak when the k value is 5, with a slight drop off when the k value is 6. When the k-means algorithm is run with a k value of 5, the sizes of the clusters range from 753 to 2761

Figure 5.2: The elbow and silhouette method charts for Librosa PSS clusters.



items, while when the k-means algorithm is run with a k value of 6, the sizes of the clusters range from 676 to 1829 items. While both of these methods are suitable, a k value of 6 was ultimately chosen for grouping Librosa PSS audio due to the more dramatic elbow point and the negligible difference between silhouette values between the k values of 5 and 6.

5.3.2 Cluster Tendency Analysis

After the Spleeter and Librosa PSS clusters were formed, the six closest items to the centroid of each cluster were extracted. In a cluster tendency analysis, these items were examined to determine if the k-means clustering algorithm had successfully grouped items based upon groove characteristics. The written analysis can be found in the bullet list below.

- Librosa PSS cluster #1: The musical styles are incredibly varied, but the common groove style between most of these pieces is a lack of syncopation, with the percussive elements often remaining eighth and fourth notes.
- Librosa PSS cluster #2: The groove styles in these tracks contain much faster percussive notes than the first cluster. While the majority of percussive elements still occur on fourth and eighth notes, most of these tracks include some kind of drum fill or faster rhythmic pattern.
- Librosa PSS cluster #3: The average sound of the items in these clusters are much more abstract than the previous clusters, with sparse and unique production choices often not featuring any percussive elements for some amount of time. This could be seen as a sort of “misfit cluster” where items that were difficult to cluster ended up.
- Librosa PSS cluster #4: The predominant groove in this cluster is closer to sixteenth and sixteenth triplet notes, with some light syncopation on a few of the tracks that give these songs a driving, yet groovy feeling.
- Librosa PSS cluster #5: This cluster contains the most amount of songs that belong in the same genre compared to the other clusters, with all but one song in the “indie rock”

genre. The main driving groove of this cluster is eight notes, with sixteenth note fills at the end of some measures.

- Librosa PSS cluster #6: This is by far the most syncopated cluster of the bunch, and definitely feels the most "groovy." The sonic palette of the cluster is reminiscent of "lofi hip hop," characterized by slightly off beat drum hits to emphasize the groove of a song [66, p. 43].
- Spleeter PSS cluster #1: This cluster contains dense, rhythmic grooves with plenty of triplets and sixteenth notes. While the tracks come from many different genres, most of the tracks have the same level of moderate groove. The cluster is very similar to the Librosa PSS cluster #2.
- Spleeter PSS cluster #2: This cluster is slightly disjointed in terms of groove, but the most obvious link is that the groove is mostly dependent on eighth and sixteenth notes.
- Spleeter PSS cluster #3: This is another "misfit cluster" like the Librosa PSS cluster #3. The percussive elements of these songs are either arithmetic or prone to change in tempo and speed throughout the segment.
- Spleeter PSS cluster #4: This is the most disorganized cluster of the bunch, with many of the tracks not containing drums at all, and one item containing possibly the densest collection of drum onsets in this entire collection. A possible explanation for this is that Spleeter filtered these songs in such a way that either gave way too many offsets to tracks with no drum onsets or way too few offsets to the item with a dense collection of drum onsets.
- Spleeter PSS cluster #5: There is very little in common in the groove styles of these six tracks. This may be a different style of "misfit cluster" where items are grouped because few other items in the collection contain a similar groove style. Clusters like these could have possibly been prevented by adding more clusters.

An overall assessment of the quality of these clusters and each technique's ability to cluster items with similar groove qualities is available in Section 6.1.1.

5.4 Evaluation Session Results

This section will begin by reporting the average and standard deviation values of the quality and expectedness per participant in Librosa and Spleeter PSS clusters. Following this, the total number of true positives, true negatives, accounts of serendipity, and accounts of anti-serendipity will be reported.

5.4.1 Quality Results

The standard deviation and average values of the subjective quality of all items in each Librosa PSS cluster for every participant can be viewed in Figure 5.3. The average standard deviation between all clusters is 1.42 with standard deviations between 1.13 and 1.69, and the average quality values between all clusters is 4.25 with values between 3.9 and 4.67.

The standard deviation and average values of the subjective quality of all items in each Spleeter PSS cluster for every participant can be viewed in Figure 5.4. The average standard deviation between all clusters is 1.54 with standard deviations between 1.44 and 1.7, and the average quality values between all clusters is 3.52 with values between 2.9 and 4.11.

Figure 5.3: Standard deviation and average values of quality values per participant in Librosa PSS clusters.

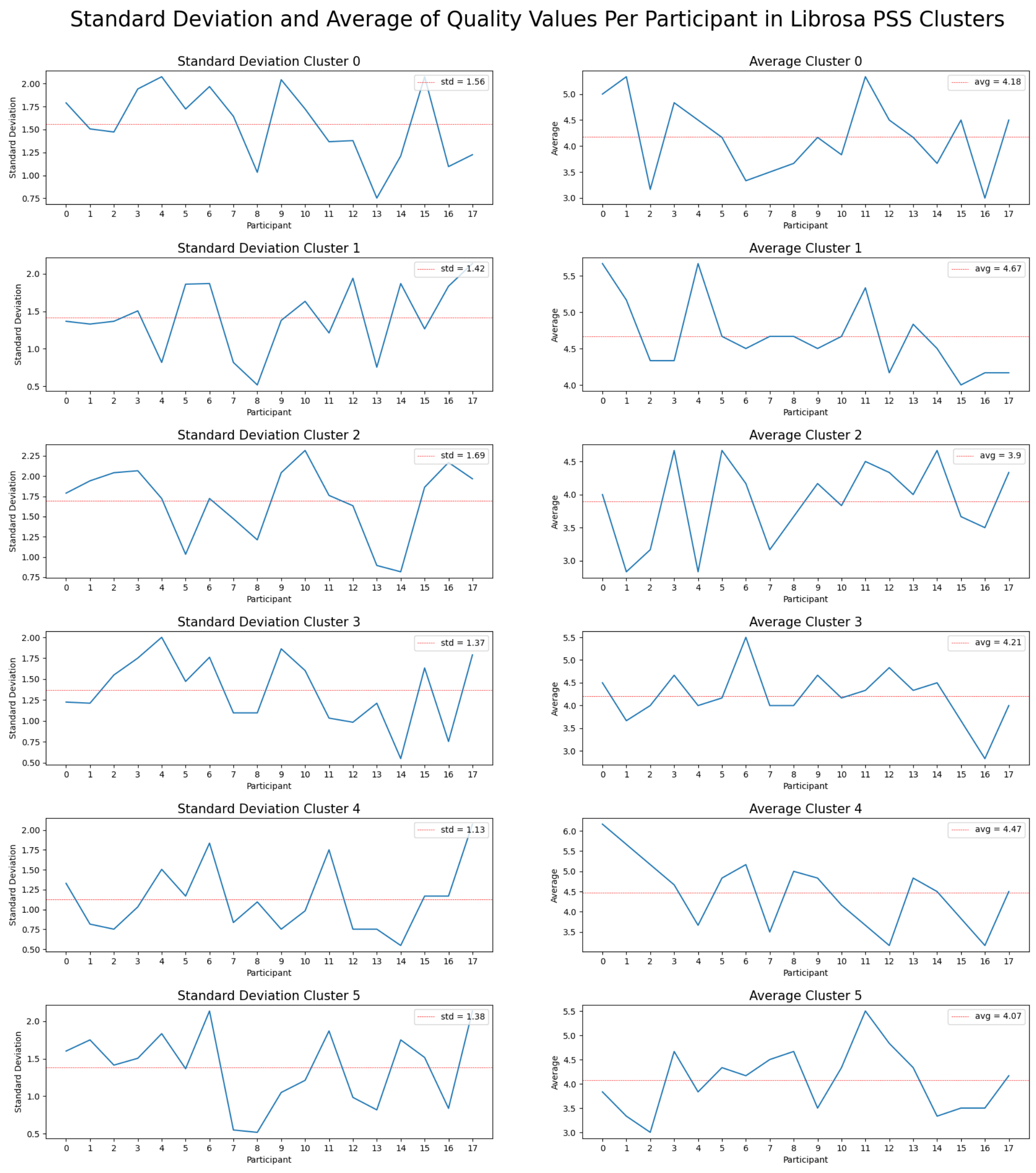
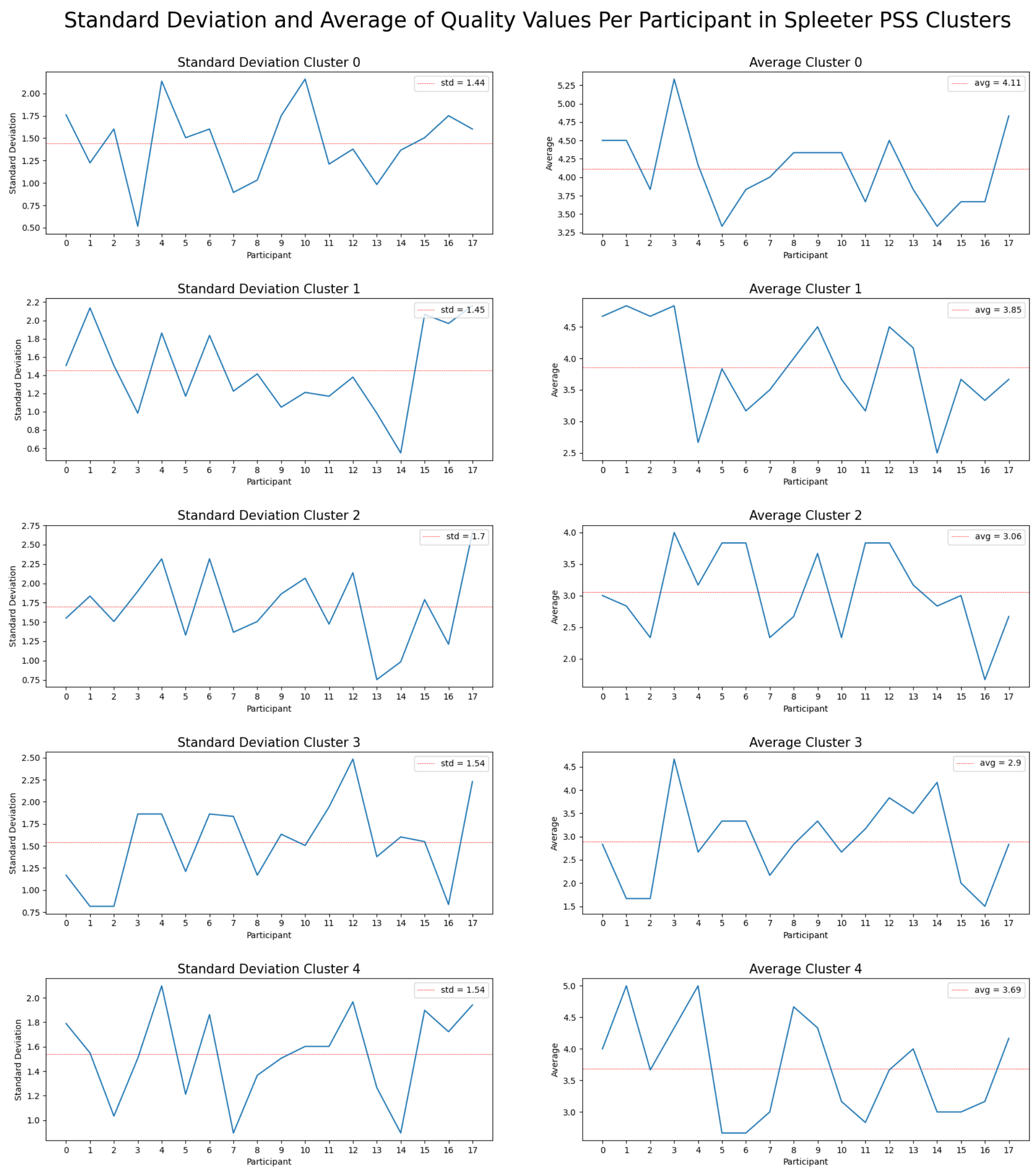


Figure 5.4: Standard deviation and average values of quality values per participant in Spleeter PSS clusters.



5.4.2 Expectedness Results

The standard deviation and average values of the expectedness of all items in each Librosa PSS cluster for every participant can be viewed in Figure 5.5. The average standard deviation between all clusters is 1.61 with standard deviations between 1.34 and 1.89, and the average quality values between all clusters is 3.37 with values between 2.98 and 3.97.

The standard deviation and average values of the expectedness of all items in each Spleeter PSS cluster for every participant can be viewed in Figure 5.6. The average standard deviation between all clusters is 1.56 with standard deviations between 1.4 and 1.69, and the average quality values between all clusters is 2.91 with values between 2.43 and 3.47.

5.4.3 Serendipity and Anti-Serendipity Per Cluster

The following section will report the total number of accounts of serendipity and anti-serendipity found across each participant in every cluster. As there were 18 participants and 66 tracks, there are a total of 1188 possible accounts of serendipity and anti-serendipity, with 648 possible accounts in the Librosa PSS clusters and 540 possible accounts in the Spleeter PSS clusters.

The total number of tracks reported as serendipitous across all participants per Librosa PSS cluster can be viewed in Figure 5.7. There were a total of 91 accounts of serendipity reported (14.04% serendipity), which equals a total of 15.17 accounts of serendipity per cluster.

The total number of tracks reported as anti-serendipitous across all participants per Librosa PSS cluster can be viewed in Figure 5.8. There were a total of 10 accounts of anti-serendipity reported (1.54% anti-serendipity), which equals a total of 1.67 accounts of anti-serendipity per cluster.

The total number of tracks reported as serendipitous across all participants per Spleeter PSS cluster can be viewed in Figure 5.9. There were a total of 60 accounts of serendipity reported (11.11% serendipity), which equals a total of 12 accounts of serendipity per cluster.

The total number of tracks reported as anti-serendipitous across all participants per Spleeter PSS cluster can be viewed in Figure 5.10. There were a total of 20 accounts of anti-serendipity reported (3.7% anti-serendipity), which equals a total of 4 accounts of anti-serendipity per cluster.

5.4.4 True Positives and True Negatives Per Cluster

To compare against the serendipitous and anti-serendipitous items, instances of true positive and true negative items across all users in each cluster were examined. As in the previous section, there are 648 possible true positives and negatives in the Librosa PSS clusters and 540 possible true positives and negatives in the Spleeter PSS clusters.

The total number of tracks reported as true positives across all participants per Librosa PSS cluster can be viewed in Figure 5.11. There were a total of 175 true positives reported (27.01% true positives), which equals a total of 29.17 true positives per cluster.

The total number of tracks reported as true negatives across all participants per Librosa PSS cluster can be viewed in Figure 5.12. There were a total of 177 true negatives reported (27.31% true negatives), which equals a total of 29.5 true negatives per cluster.

The total number of tracks reported as true positives across all participants per Spleeter PSS cluster can be viewed in Figure 5.13. There were a total of 81 true positives reported (15% true positives), which equals a total of 16.2 true positives per cluster.

The total number of tracks reported as true negatives across all participants per Spleeter PSS cluster can be viewed in Figure 5.14. There were a total of 230 true negatives reported (42.59% true negatives), which equals a total of 46 true negatives per cluster.

Figure 5.5: Standard deviation and average values of expectedness values per participant in Librosa PSS clusters.

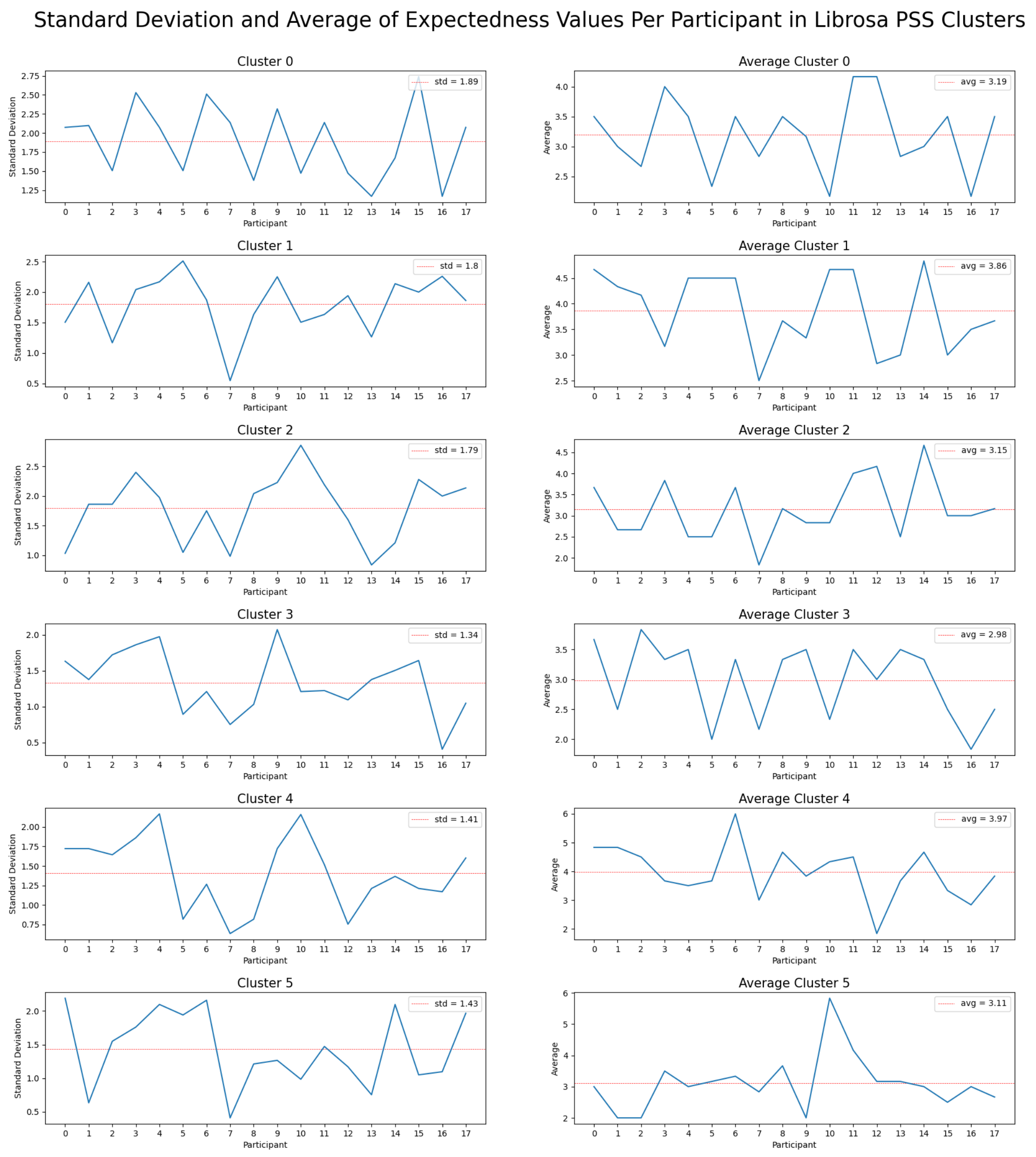


Figure 5.6: Standard deviation and average values of expectedness values per participant in Spleeter PSS clusters.

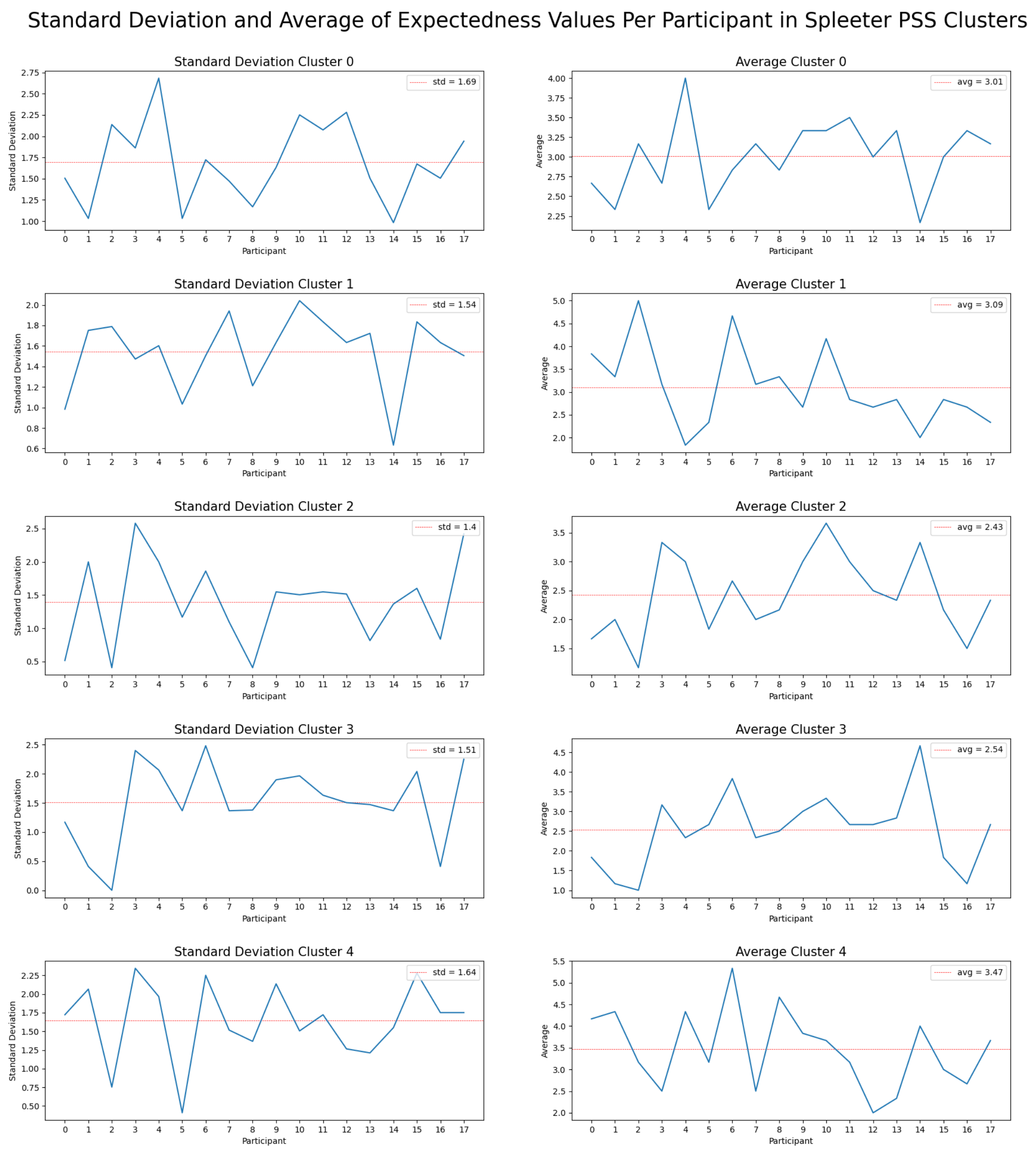


Figure 5.7: Total serendipity across all participants per Librosa PSS cluster.

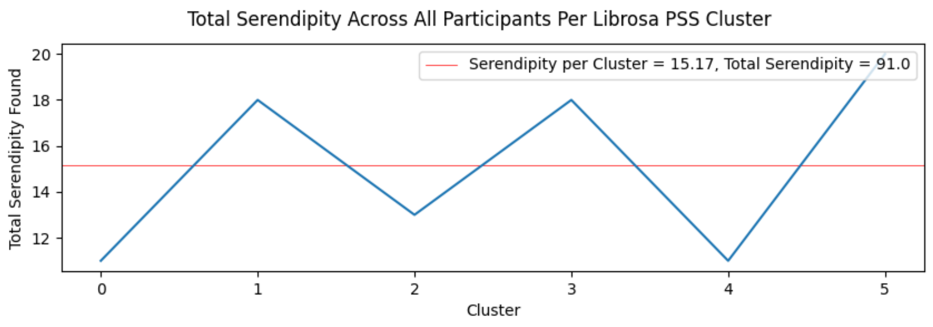


Figure 5.8: Total serendipity across all participants per Spleeter PSS cluster.

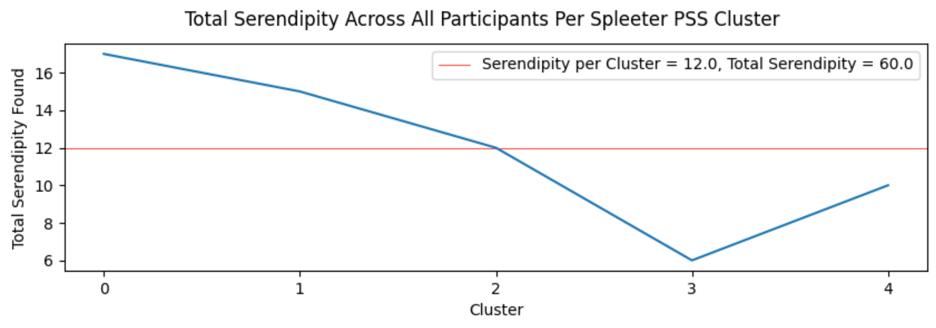


Figure 5.9: Total anti-serendipity across all participants per Librosa PSS cluster.

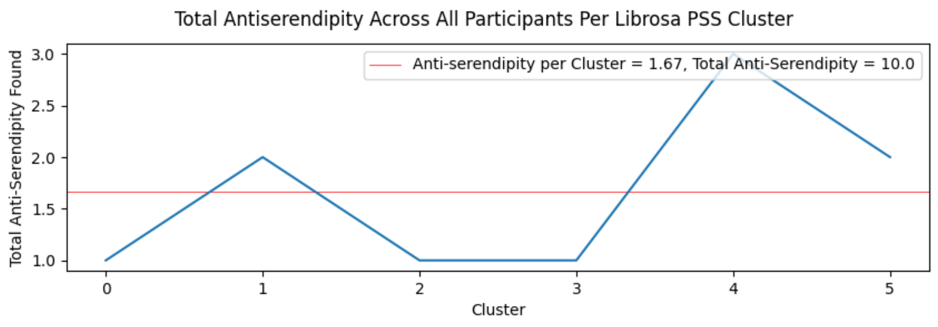


Figure 5.10: Total anti-serendipity across all participants per Spleeter PSS cluster.

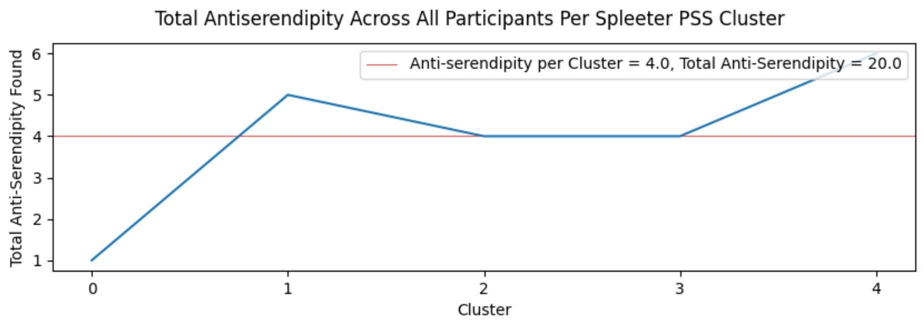


Figure 5.11: Total true positives across all participants per Librosa PSS cluster.

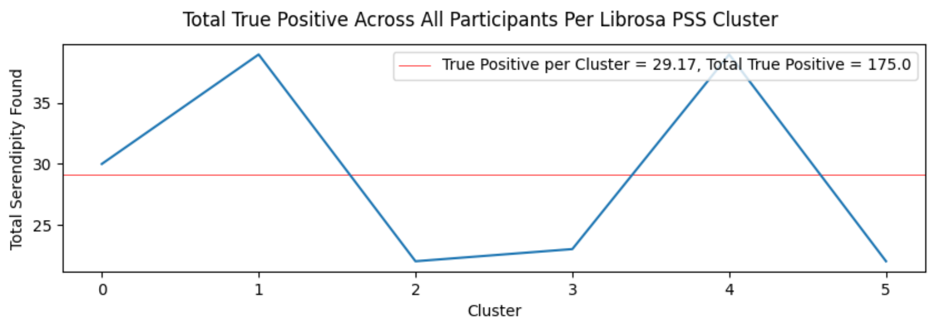


Figure 5.12: Total true negatives across all participants per Librosa PSS cluster.

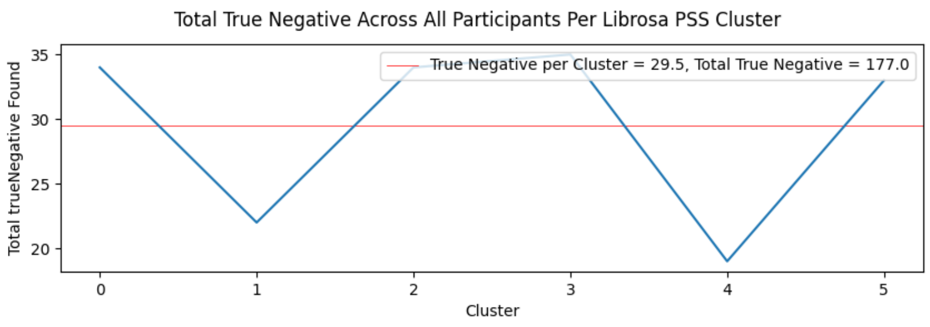


Figure 5.13: Total true positives across all participants per Spleeter PSS cluster.

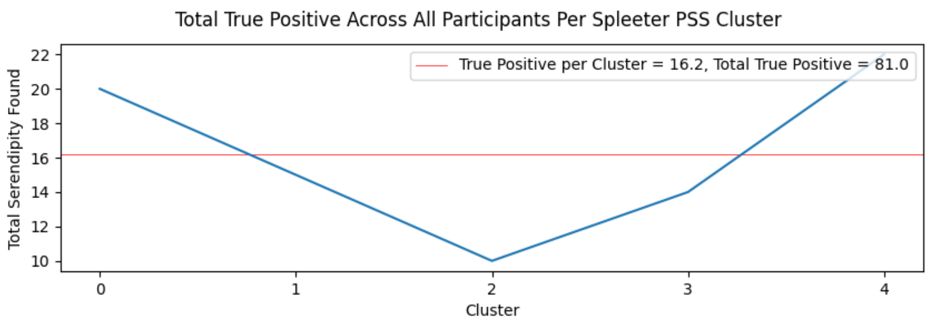
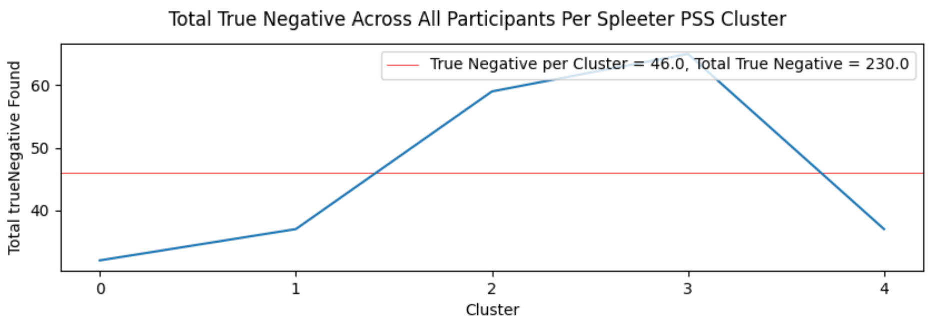


Figure 5.14: Total true negatives across all participants per Spleeter PSS cluster.



5.4.5 Effects of Serendipity and Anti-Serendipity Against Average Cluster Quality and Expectedness

This section will examine if having at least one item in a cluster with an account of serendipity or anti-serendipity has any effect over the average quality or expectedness of an entire cluster. This is done by comparing the average quality and expectedness values for a cluster with an account of serendipity or anti-serendipity against the average quality and expectedness values of all other clusters from the same PSS method. A total of 108 Librosa PSS clusters and 90 Spleeter PSS clusters were analyzed across all users.

There were a total of 62 Librosa PSS clusters that contained at least one instance of serendipity (57.41% of all Librosa PSS clusters). Of those clusters with serendipity, 34 had an average quality value higher than average across all Librosa PSS clusters for a given participant (54.84% of Librosa PSS clusters with serendipity), and 28 had an average quality value lower than average for the across all Librosa PSS clusters for a given participant (45.16% of Librosa PSS clusters with serendipity). There were also 21 clusters that had above average expectedness across all Librosa PSS clusters for a given participant (33.87% of Librosa PSS clusters with serendipity) and 41 clusters that had below average expectedness across all Librosa PSS clusters for a given participant (66.13% of Librosa PSS clusters with serendipity).

There were a total of 42 Spleeter PSS clusters that contained at least one instance of serendipity (46.67% of all Spleeter PSS clusters). Of those clusters with serendipity, 24 had an average quality value higher than average across all Spleeter PSS clusters for a given participant (57.14% of Spleeter PSS clusters with serendipity), and 18 had an average quality value lower than average for the across all Spleeter PSS clusters for a given participant (42.86% of Spleeter PSS clusters with serendipity). There were also 20 clusters that had above average expectedness across all Spleeter PSS clusters for a given participant (47.62% of Spleeter PSS clusters with serendipity) and 22 clusters that had below average expectedness across all Spleeter PSS clusters for a given participant (52.38% of Spleeter PSS clusters with serendipity).

There were a total of 8 Librosa PSS clusters that contained at least one instance of anti-serendipity (7.41% of all Librosa PSS clusters). Of those clusters with anti-serendipity, 5 had an average quality value higher than average across all Librosa PSS clusters for a given participant (62.5% of Librosa PSS clusters with anti-serendipity), and 3 had an average quality value lower than average for the across all Librosa PSS clusters for a given participant (37.5% of Librosa PSS clusters with anti-serendipity). There were also 5 clusters that had above average expectedness across all Librosa PSS clusters for a given participant (62.5% of Librosa PSS clusters with anti-serendipity) and 3 clusters that had below average expectedness across all Librosa PSS clusters for a given participant (37.5% of Librosa PSS clusters with anti-serendipity).

There were a total of 15 Spleeter PSS clusters that contained at least one instance of anti-serendipity (16.67% of all Spleeter PSS clusters). Of those clusters with anti-serendipity, 4 had an average quality value higher than average across all Spleeter PSS clusters for a given participant (26.67% of Spleeter PSS clusters with anti-serendipity), and 11 had an average quality value lower than average for the across all Spleeter PSS clusters for a given participant (73.33% of Spleeter PSS clusters with anti-serendipity). There were also 10 clusters that had above average expectedness across all Spleeter PSS clusters for a given participant (66.67% of Spleeter PSS clusters with anti-serendipity) and 5 clusters that had below average expectedness across all Spleeter PSS clusters for a given participant (33.33% of Spleeter PSS clusters with anti-serendipity).

5.4.6 Correlation Between Favorite and Least Favorite Items and Average Cluster Quality and Expectedness

To represent the most extreme poles of a participant's taste, they were asked at the end of the evaluation session for their favorite and least favorite musical items. The clusters that the participant's favorite and least favorite items belong in were then analyzed to observe their effect on quality and expectedness values. Unlike the previous results, the results are not separated into Librosa and Spleeter PSS clusters but instead are treated as a whole. However, the total number of user reported favorite and least favorite items in Librosa and Spleeter PSS clusters will be reported. Out of the 18 participants, 17 properly reported their favorite items and 18 properly reported their least favorite items.

A total of 82.35% of participants (14 participants) reported that their favorite item was in a cluster with a higher than average quality value, while 17.65% of participants (3 participants) reported that their favorite item was in a cluster with a lower than average quality value. A total of 64.71% of participants (11 participants) reported that their favorite item was in a cluster with a higher than average expectedness value, while 35.29% of participants (6 participants) reported that their favorite item was in a cluster with a lower than average expectedness value. Finally, 70.59% of participants reported that their favorite item was in a Librosa PSS cluster (12 participants), while 29.41% of users reported that their favorite item was in a Spleeter PSS cluster (5 participants).

A total of 27.78% of participants (5 participants) reported that their least favorite item was in a cluster with a higher than average quality value, while 72.22% of participants (13 participants) reported that their least favorite item was in a cluster with a lower than average quality value. A total of 16.67% of participants (3 participants) reported that their least favorite item was in a cluster with a higher than average expectedness value, while 83.33% of participants (15 participants) reported that their least favorite item was in a cluster with a lower than average expectedness value. Finally, 22.22% of participants reported that their least favorite item was in a Librosa PSS cluster (4 participants), while 77.78% of users reported that their least favorite item was in a Spleeter PSS cluster (14 participants).

Chapter 6

Discussion

This section will expand upon the key findings through the information provided in the results, reflect upon the limitations in this study, and give recommendations for future research in this field.

6.1 Key Findings

The key findings in this study come from the analysis of the PSS audio clusters, the beat tracking and onset detection system, and interpretations of the data collected from the evaluation sessions.

6.1.1 Clustering Method Quality

The discussion on the quality of the Librosa and Spleeter PSS clustering methods will be broken down into two parts: musical similarity within clusters and groove similarity within clusters. Note that low musical similarity within a cluster is neither a positive or negative attribute, since this could present more opportunities for serendipitous items, but low groove similarity is a negative attribute because this implies the ML system did not cluster items with similar groove styles as it was intended.

Musical Similarity

Of the content analyzed, the musical similarity between items in the same Librosa PSS cluster struck a strange balance between consistent and wildly different musical styles. For example, Librosa PSS cluster #1 contains musical styles including rap, instrumental acoustic guitar, marching band music, and orchestral arrangements, while every item except for one in cluster #5 is in the indie rock genre. These are the most dramatic examples, but the other Librosa PSS clusters' set of analyzed items usually feature two or three items within the same genre, with the other three or four items belonging to completely different genres. On the other hand, the musical similarity between items in the same Spleeter PSS cluster is nearly nonexistent. Every cluster features tracks with barely any musical attributes in common, akin to Librosa PSS cluster #1, with the possible exception of Spleeter PSS cluster #4 which could be described as experimental, drone, or ambient music. Therefore, based purely on manual analysis, the Spleeter PSS approach is the most likely to serve the most unexpected items, but both approaches have the ability to serve items in completely different musical styles. Whether users will experience serendipity with these different musical styles will be elaborated on in the discussion on the evaluation sessions.

Groove Similarity

After a manual analysis of the six closest tracks to the centroid of each cluster, it can be reasonably assumed that the Librosa PSS audio is more suited to clustering than Spleeter PSS audio since it was much harder to find common groove styles in Spleeter PSS clusters compared to Librosa PSS clusters. One possible explanation for this phenomenon is the difference in quality of the PSS audio between each PSS method. Spleeter PSS specifically tries to extract the drums from each track, while Librosa PSS filters the percussive elements, which could include drums, piano hits, or a particularly sharp acoustic guitar strum, among others. Not only does this mean that mean that Librosa PSS can analyze a wider variety of items, as is evident by the much larger number of invalid files found in Spleeter PSS audio analysis compared to Librosa PSS audio, but it also shows that the artifacts that exist in Spleeter PSS separated audio could possibly confuse the onset detection and beat tracking algorithms, resulting in inconsistent groove patterns within clusters. This could also explain the aforementioned randomness of musical styles within the Spleeter PSS clusters. Therefore, based purely on manual analysis, the Librosa PSS approach is the most likely to serve items with a similar groove style.

6.1.2 Percussive Source Separation Comparison

As the previous key findings showed, the Librosa and Spleeter PSS methods differ quite a bit in the way that beats are found and clusters are formed. To continue the differences between these two methods, the Librosa PSS method had far fewer invalid files than the Spleeter PSS method. A file is classified as “invalid” if there were no beats or onsets detected in the file, resulting in the item not being able to be analyzed by the groove feature extraction program. Out of 8000 audio files, the Librosa PSS method had 50 invalid tracks (0.625% invalid), while the Spleeter PSS method had 876 invalid tracks (10.95% invalid). While the obvious downside of using Spleeter PSS for onset detection is that there are fewer tracks to test with, a larger issue was a fundamental misunderstanding of Spleeter’s purpose. Spleeter in fact does not perform percussive source separation, but drum source separation. While the assumption in this thesis was that these methods would be somewhat interchangeable, this turned out not to be the case, as the model that Librosa was trained on was very effective at filtering non-drum sounds. In order to analyze the groove of a musical item without any drums, a PSS method such as Librosa that takes into account all percussive noises is absolutely necessary. Spleeter in groove analysis could be useful in analyzing the grooves of different instruments if a data set can guarantee that the items contain a particular instrument. However, Spleeter offers no advantages in drum groove analysis that Librosa PSS does not already, due to the relative processing efficiency of Librosa PSS and the ability to detect all manner of percussive events in a musical item.

6.1.3 Relevant Evaluation Session Comments

After the evaluation session was completed, participants were asked formally (on the Google Form) and informally (through conversation) on their thoughts about the evaluation session. This list below will summarize and discuss the key points found in these comments. A full set of the relevant, formal additional comments can be found in Appendix G[1].

- A large percentage of participants reported listening fatigue as the test went on, stating that there were simply too many tracks to listen to. Furthermore, a few participants had trouble keeping count of the current item due to the repetition.
- Some participants commented negatively on the quality of the tracks as a whole, stating that the amateurish quality of some of the tracks left an overall negative view of the

[1]The full set of participant responses, including item quality and expectedness ratings, can be found on GitHub: https://tinyurl.com/4k7whtxz

tracks. While the poor recording quality is a clear flaw of some of the tracks in the FMA, an overall negative attitude towards the tracks may actually simulate a streaming service fairly accurately. If a truly representative sample of the music on a streaming service was tested, one could surmise that the average listener would be off-put and react negatively to a majority of the music.

6.1.4 Evaluation Session Results

The discussion on the results of the data analysis from the evaluation session will be split into six parts. These include discussions on the overall quality and expectedness values per cluster, standard deviation per cluster, serendipity and anti-serendipity per cluster, true positives and true negatives per cluster, the effects on serendipity and anti-serendipity on cluster quality and expectedness, and finally the effects of favorite and least favorite items on cluster quality and expectedness.

Average Quality and Expectedness Values

To begin the comparison of Librosa and Spleeter PSS clusters, the average quality values will be examined. Assuming a normal distribution for quality and expectedness ratings, the average quality and expectedness values across all tracks for both PSS methods should be around 4 [67]. Across all participant ratings for all items of the evaluation session, an average quality rating of 4.25 in the Librosa segment suggests a fairly normal distribution, while an average quality rating of 3.52 suggests that other factors were at play that pushed the average rating further from 4. This could be because there were not enough ratings for a normal distribution, due to only having 5 Spleeter PSS clusters with 540 total ratings compared to 6 Librosa PSS clusters with 648 total ratings. Another explanation for this discrepancy is that the overall quality of the items served in the Spleeter system may have been less desirable than the items served by the Librosa system, which will be discussed further while analyzing true positives and true negatives later in this section.

In terms of the average expectedness, both Librosa and Spleeter PSS clusters have lower than average values, with the Librosa method's average value being 3.37 and the Spleeter method's average value being 2.91. This is mostly likely on account of the content of the FMA, which includes more experimental and left field musical items than streaming services. While streaming services contain plenty of experimental music themselves, it also contains nearly the entire history of popular music, which could result in a more normal distribution of expectedness. With lower than average expectedness values, it is also expected that there will be more serendipity and true negatives found compared to anti-serendipity and true positives, respectively.

Standard Deviation Per Cluster

Standard deviation of quality values per cluster is analyzed to determine if solely using features of groove in a MRS can serve relevant recommendations to users. Assuming a normal distribution for quality and expectedness ratings, around 68% of the ratings are within one standard deviation of the mean and 95% of the ratings are with two standard deviations of the mean [67]. Therefore, if the average standard deviation across all clusters is below 1.5 then it can be reasonably assumed that the recommendation system is better than random guessing.

With an average quality rating standard deviation per cluster of 1.42 for Librosa PSS clusters and 1.54 for Spleeter PSS clusters, it can be concluded that participants were more likely to receive relevant recommendations using the Librosa method, but both methods were fairly close to random guessing. Therefore, for the goal of suggesting high quality items to users, using only features of groove does not produce relevant results. However, the slightly lower standard deviation per cluster average that the Librosa method produces suggests that with further fine tuning and possibly more features, a more refined recommendation system could be developed. It is also worth noting that the average standard deviation of expectedness ratings per cluster for both Librosa and Spleeter methods are fairly similar and above 1.5, suggesting a truly random distribution of expectedness values per cluster.

Serendipity and Anti-Serendipity Per Cluster

Participants reported higher than expected accounts of serendipity, with both Spleeter and Librosa PSS items having higher than 10% serendipity accounts. This could be attributed to the diverse catalog of the FMA, or a general tendency for users to encounter positive surprises when exploring a wide variety of music. However, participants tended not to report instances of anti-serendipity. This could be a result of the combination of lower than average expectedness values for both Librosa and Spleeter PSS clusters and the aforementioned accounts of the low quality of some items in the FMA. The perceived low quality may have resulted in lower rated items receiving equally low expectedness values due to the intensely negative reaction they elicited in some participants. However, since items near the centroid of Spleeter PSS clusters produced more than double the instances of anti-serendipity compared to Librosa PSS clusters, it can be stated that Librosa PSS clusters do a better job at combating stagnation and steering users away from low quality items in genres they are familiar with. Overall, the high values of serendipitous items versus the much lower values of anti-serendipitous items shows that if music streaming services expanded their horizons by introducing new styles of music to listeners that do not have a history listening to that style of music, they would be much more likely to find positive surprises rather than negative surprises, and that achieving serendipity through the lens of groove could be a worthwhile approach to address this issue.

True Positives and True Negatives Per Cluster

Tracks from the Librosa PSS clusters contained a fairly even amount of true positive ratings (27.01% of all ratings) and true negative ratings (27.31% of all ratings), further suggesting a fairly even set of musical items in terms of quality and expectedness for each participant. Comparatively, items from the Spleeter segment of the evaluation session contained a significant number of true negatives (42.59% of all ratings), compared to only 15% of all ratings being marked as true positives. This is backed up anecdotally by some users pointing out the the end of the test, when the items from the Spleeter clusters were played, contained more strange and experimental music.

Effects of Serendipity and Anti-Serendipity On Average Cluster Quality and Expectedness

To begin the discussion of the effects of serendipity and anti-serendipity on average cluster quality and expectedness, it is worth noting that the number of clusters containing anti-serendipity were minuscule and therefore did not produce statistically significant results. However, there were many instances of serendipity across both Spleeter and Librosa PSS clusters, which gives a better insight into how serendipity affects quality and expectedness values. In general, Librosa PSS clusters had a much better spread of clusters that contained serendipitous items than Spleeter PSS clusters. However, both Librosa and Spleeter PSS clusters that contained accounts of serendipity only had higher than average quality values a little over half of the time. While this suggests at least a little correlation between accounts of serendipity and an increase in quality, this could also be explained by the fact that an account of serendipity guarantees that at least one item in the cluster was rated above a 4 for quality. This could be all that a cluster needs to boost its average above what was typical for a user, so it can be stated that correlation between serendipity and quality was not significant for either PSS method. Similarly, average expectedness in Spleeter PSS clusters with accounts of serendipity was below average a little over half of the time, suggesting low correlation. However, a significant number of Librosa PSS clusters with accounts of serendipity had below average expectedness values, with 66.13% of the clusters having below average expectedness values. Therefore, the most significant finding in this analysis is that for the Librosa PSS clusters, a cluster that contains an instance of serendipity has some correlation with a lower expectedness value across all items in the cluster.

Effects of Favorite and Least Favorite Items on Average Cluster Quality and Expectedness

In the analysis of quality values against participants' reported favorite and least favorite items, the results were similar to what was expected. A very high percentage of favorite items were in clusters with above average quality values, while a very high percentage of least favorite items were in clusters with below average quality values. While this trend could be explained with the same logic that influences the average quality of tracks with accounts of serendipity and anti-serendipity, the high percentages suggest that there is some correlation between a favorite item being in a group of items with above average quality values for a participant. Another expected result was the fact that Librosa PSS clusters contained a large majority of favorite items compared to Spleeter PSS clusters. However, according to the previous analysis of the quality of Librosa and Spleeter PSS cluster items, it was slightly surprising to not see a higher percentage of Librosa PSS items being marked as favorite items. The effect of favorite and least favorite items on expectedness produced somewhat expected results as well. A somewhat high percentage of favorite tracks were in clusters with above average expectedness values, while a very high percentage of least favorite tracks were in clusters with below average expectedness values. These results match well with what was observed so far in terms of the lowest quality items also having low expectedness values, but it also shows that participants tended to pick favorite tracks that were in clusters that felt more comfortable and expected. Overall, not much information was gained from this analysis, most likely due to the fact that there was little correlation in quality between items within clusters.

6.2 Limitations

While this experiment attempted to replicate the recommendations of a music streaming service, the resources available were far fewer than that of a typical streaming service. This, as well as the inherent unpredictability of analyzing raw audio files, leads to limitations in this study, which include but are not limited to:

- The state-of-the-art beat tracking system is still only around 60% accurate according to Ellis's study [40, p. 51].
- Audio files provided by the FMA vary in sound quality, instrumental makeup, and have no guarantee of a constant tempo.
- The FMA data set used contained 8000, 30 second audio files as opposed to a typical streaming service which contains millions of full length audio files [15, p. 238].
- Only 18 participants were used in this study, compared to the millions of users on a leading streaming service [15, p. 239].
- Only the six closest items to the center of each cluster were manually analyzed and used as test data for the evaluation sessions. For manual analysis, analyzing more items close to the cluster could produce entirely new perspectives on the groove styles within a cluster. For the evaluation sessions, this limited analysis may produce different results, making each clustering method either more or less accurate in predicting user taste. As a real world example, popular music streaming services have access to the ratings and behavior surrounding every song a user listens to, giving their MRSs many more items and features to analyze.

- As a result of using songs most participants would not be aware of, it was not possible to compare the prediction power of other streaming services compared to the developed cluster due to the fact that most of these songs were not on Spotify.

6.3 Recommendations

This section will list recommendations for each component of this thesis experiment in order to improve the quality of results and investigate new research questions in future studies.

6.3.1 Beat Tracking And Onset Detection

Based on the initial test results and the results of the evaluation session, a series of changes to the beat tracking and onset detection system is recommended for future experiments.

- Having 12 subdivisions for each beat may have been too precise of a measurement. Since analyzing with smaller subdivisions could exclude different rhythms and produce inaccurate groove analysis, a more sophisticated onset detection system would be required for further pulse and subdivision analysis.

- Tracks selected for future studies should have a defined tempo and should be recorded to a click track, thus eliminating the need for beat tracking and ideally making onset detection more accurate. Tracks with MIDI note values [68] available could also be considered, which would also remove the need for onset detection, make groove detection 100% accurate, and could allow the ability to add microtiming as a groove feature. While some MIDI datasets exist for more specialized tasks, such as the expanded groove MIDI dataset from Callender et al. [69], a major downside of this approach is the lack of quality datasets with MIDI values for popular styles of music.

- Other forms of PSS and beat tracking should be tested besides Librosa that use novel techniques to improve the quality of HPSS.

6.3.2 Evaluation Session

With the responses of the participants and anecdotal evidence as a basis, there are a series of changes to the evaluation session that are recommended in the future.

- Instead of a single, 35 minute audio file, musical items should be placed within the form itself so users know exactly what item they're listening to and can listen back as many times as they want.

- Google Forms should not be used because it does not support embedding tracks in the form, the editor is clunky and inefficient for this style of evaluation, and the loading of the form online takes a significant and unnecessarily long time to load.

6.3.3 Future Studies

While it was shown that recommending tracks solely based on groove has only some merit, combining groove features with other MRS techniques or test data could improve test data. Below is a list of changes to this study that could be performed in future studies to further evaluate the effectiveness of including groove features in serving musical recommendations.

- Instead of clustering items solely based on groove characteristics, items could be classified using the groove characteristics and state-of-the-art features, such as the features in the Spotify API [15]. This should then be compared to item clustering using only the state-of-the-art features to observe if there was a noticeable shift in clusters. This way, a solid baseline in terms of inter-cluster quality would be established due to the well tested nature of the state-of-the-art features.
- A separate study on how expectedness relates to quality between clusters should be performed separately using state-of-the-art features, as there has not been a baseline established for self-reported expectedness between items recommended by a MRS.
- Focusing on recommending content based on a participant's predefined preferences, rather than examining how closely items in a single cluster are rated, could be a more effective lens to examine the quality of a recommendation system.
- Combining a collaborative filtering or a context-based model with groove characteristic features could help improve the quality of recommendations.
- Instead of using tracks within a free music archive, popular songs on streaming services could be used instead to serve recommendations closer to what a music streaming service would recommend. A popular song could be defined as either a song that has a certain number of plays, or a song from an artist with a certain number of monthly listeners. The biggest challenge of implementing this would be to find artists and songs that are popular, but the participant has not heard before, as this could introduce bias in the test results.
- More tracks should be analyzed in future tests in order to more closely simulate the library of a music streaming service.

Chapter 7

Conclusion

Characteristics of groove can be broken down mathematically to be used as features in ML systems used to recommend music in MRSs. This ML system first requires a sophisticated beat tracking and onset detection system to be developed in order to properly analyze the groove of raw audio data. To make the groove analysis more consistent, PSS must be performed in order to separate the percussive elements from the harmonic elements of a track. The two PSS methods that were tested in this thesis were using Librosa HPSS and using the drum source separation from the tool Spleeter. Beats and onsets were then extracted from an archive of copyright-free musical items. In the end, using Librosa HPSS was clearly the superior method for beat tracking and onset detection due to Spleeter filtering out everything except for drum sounds, which completely ignores the other percussive noises in a musical item.

Next, groove features were calculated for both PSS methods using the beats and onsets found. The features that were deemed most useful for recommending musical items due to the ease of calculation and overall representation of an item's groove included the pulse of the beat, the subdivisions of the beat, and two methods for calculating overall syncopation of the item: off-beatness and WNBD. These features were then used as the basis of a k-means clustering algorithm for each PSS method, with the number of clusters determined through manual analysis of the elbow method, silhouette method, and uniformity of cluster size. Through a review of the six items closest to the centroid of each cluster, it was determined that Librosa PSS clusters had much more distinct and clearly defined grooves compared to Spleeter PSS clusters.

An evaluation session was then performed with 18 participants to determine if there was a correlation between groove styles and quality or expectedness. Using a normal distribution, it was found that grouping musical items using features of groove produced quality and expectedness values fairly similar to random guessing. This study found little to no correlation between the quality or expectedness values in a given cluster using either PSS method, although the Librosa PSS clusters were marginally better. Also, Librosa PSS clusters had a fairly even number of items marked expected and high quality compared to items unexpected and low quality, while Spleeter PSS clusters had many more unexpected and low quality items than expected and high quality items.

Serendipity, or the reporting of a musical item being unexpected and high in subjective quality, and anti-serendipity, or the reporting of a musical item being expected and low in subjective quality, was also analyzed from the results of the evaluation session. Overall, the Librosa PSS clusters contained many more accounts of serendipity and Spleeter PSS clusters contained many more accounts of anti-serendipity. In analyzing the effects of serendipity and anti-serendipity, little to no correlation between an account of serendipity in a cluster and the average quality of all items in a cluster for either method was found. There were also too few accounts of anti-serendipity found to produce a statistically significant analysis of the effects of anti-serendipity on quality and expectedness. However, the analysis showed statistically significant effects of an account of serendipity on the expectedness of items in Librosa PSS clusters, showing that clustering items based upon groove using Librosa HPSS can help streaming service users fight feelings of stagnation in their recommendations by serving unexpected items with similar styles of groove.

Using Librosa HPSS was found to be superior to using Spleeter in nearly all scenarios, including finding beats and onsets, analyzing groove, and clustering items with similar grooves together. Even though recommending musical items solely based on groove will not produce higher quality recommendations for the average streaming service user with either PSS method, the findings in this study could be used to fuel future research on this topic. Combining features of groove into other state-of-the-art MRSs could help fight feelings of stagnation by helping to recommend music outside of a streaming service user's typical consumption. Future studies should focus on building a more robust beat tracking system, testing different styles of HPSS, combining groove features with other MRS features, using different music archives, and expanding the scope of evaluation sessions with more participants, musical items, and a better system for evaluation sessions.

Chapter 8
Bibliography

[1] K. Swanson, “A case study on Spotify: exploring perceptions of the music streaming service,” MEIEA J., vol. 13, no. 1, pp. 207–231, Jan. 2013.

[2] Y. Song, S. Dixon, and M. Pearce, “A Survey of Music Recommendation Systems and Future Perspectives”.

[3] S. Freeman, M. Gibbs, and B. Nansen, “‘Don’t mess with my algorithm’: Exploring the relationship between listeners and automated curation and recommendation on music streaming services,” First Monday, Jan. 2022, doi: 10.5210/fm.v27i1.11783.

[4] E. Moore, “Infinite content and interrupted listening”.

[5] W. Feng, T. Li, H. Yu, and Z. Yang, “A Hybrid Music Recommendation Algorithm Based on Attention Mechanism,” in MultiMedia Modeling, vol. 12572, J. Lokoč, T. Skopal, K. Schoeffmann, V. Mezaris, X. Li, S. Vrochidis, and I. Patras, Eds., in Lecture Notes in Computer Science, vol. 12572. , Cham: Springer International Publishing, 2021, pp. 328–339. doi: 10.1007/978-3-030-67832-6_27.

[6] M. Vystrčilová and L. Peška, “Lyrics or Audio for Music Recommendation?,” in Proceedings of the 10th International Conference on Web Intelligence, Mining and Semantics, Biarritz France: ACM, Jun. 2020, pp. 190–194. doi: 10.1145/3405962.3405963.

[7] H.-C. Chen and A. L. P. Chen, “A Music Recommendation System Based on Music and User Grouping,” J. Intell. Inf. Syst., vol. 24, no. 2–3, pp. 113–132, Mar. 2005, doi: 10.1007/s10844-005-0319-3.

[8] W. G. Assuncao, L. S. G. Piccolo, and L. A. M. Zaina, “Considering emotions and contextual factors in music recommendation: a systematic literature review,” Multimed. Tools Appl., vol. 81, no. 6, pp. 8367–8407, Mar. 2022, doi: 10.1007/s11042-022-12110-z.

[9] D. Ayata, Y. Yaslan, and M. E. Kamasak, “Emotion Based Music Recommendation System Using Wearable Physiological Sensors,” IEEE Trans. Consum. Electron., vol. 64, no. 2, pp. 196–203, May 2018, doi: 10.1109/TCE.2018.2844736.

[10] H.-G. Kim, G. Y. Kim, and J. Y. Kim, “Music Recommendation System Using Human Activity Recognition From Accelerometer Data,” IEEE Trans. Consum. Electron., vol. 65, no. 3, pp. 349–358, Aug. 2019, doi: 10.1109/TCE.2019.2924177.

[11] I. Andjelkovic, D. Parra, and J. O’Donovan, “Moodplay: Interactive music recommendation based on Artists’ mood similarity,” Int. J. Hum.-Comput. Stud., vol. 121, pp. 142–159, Jan. 2019, doi: 10.1016/j.ijhcs.2018.04.004.

[12] X. Wang and Y. Wang, “Improving Content-based and Hybrid Music Recommendation using Deep Learning,” in Proceedings of the 22nd ACM international conference on Multimedia, Orlando Florida USA: ACM, Nov. 2014, pp. 627–636. doi: 10.1145/2647868.2654940.

[13] A. Elbir and N. Aydin, “Music genre classification and music recommendation by using deep learning,” Electron. Lett., vol. 56, no. 12, pp. 627–629, Jun. 2020, doi: 10.1049/el.2019.4202.

[14] M. Schedl, H. Zamani, C.-W. Chen, Y. Deldjoo, and M. Elahi, "Current challenges and visions in music recommender systems research," Int. J. Multimed. Inf. Retr., vol. 7, no. 2, pp. 95–116, Jun. 2018, doi: 10.1007/s13735-018-0154-2.

[15] R. Panda, C. Gonçalves, and R. P. Paiva, "HOW DOES THE SPOTIFY API COMPARE TO THE MUSIC EMOTION RECOGNITION STATE-OF-THE-ART?," 2021.

[16] G. S. Câmara and A. Danielsen, "Groove," in The Oxford Handbook of Critical Concepts in Music Theory, A. Rehding and S. Rings, Eds., Oxford University Press, 2020, pp. 270–294. doi: 10.1093/oxfordhb/9780190454746.013.17.

[17] F. Gomez, A. Melvin, D. Rappaport, and G. T. Toussaint, "Mathematical Measures of Syncopation".

[18] D. Paul and S. Kundu, "A Survey of Music Recommendation Systems with a Proposed Music Recommendation System," in Emerging Technology in Modelling and Graphics, vol. 937, J. K. Mandal and D. Bhattacharya, Eds., in Advances in Intelligent Systems and Computing, vol. 937. , Singapore: Springer Singapore, 2020, pp. 279–285. doi: 10.1007/978-981-13-7403-6_26.

[19] M. Kaminskas and D. Bridge, "Diversity, Serendipity, Novelty, and Coverage: A Survey and Empirical Analysis of Beyond-Accuracy Objectives in Recommender Systems," ACM Trans. Interact. Intell. Syst., vol. 7, no. 1, pp. 1–42, Mar. 2017, doi: 10.1145/2926720.

[20] H. Boone and D. Boone, "Analyzing Likert Data," J. Ext., vol. 50, no. 2, Apr. 2012, doi: 10.34068/joe.50.02.48.

[21] D. K. Lee, J. In, and S. Lee, "Standard deviation and standard error of the mean," Korean J. Anesthesiol., vol. 68, no. 3, pp. 220–223, May 2015, doi: 10.4097/kjae.2015.68.3.220.

[22] "numpy.std — NumPy v1.26 Manual." Accessed: Nov. 09, 2023. [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.std.html

[23] M. Ge, C. Delgado-Battenfeld, and D. Jannach, "Beyond accuracy: evaluating recommender systems by coverage and serendipity," in Proceedings of the fourth ACM conference on Recommender systems, Barcelona Spain: ACM, Sep. 2010, pp. 257–260. doi: 10.1145/1864708.1864761.

[24] Y. C. Zhang, D. Ó. Séaghdha, D. Quercia, and T. Jambor, "Auralist: introducing serendipity into music recommendation," in Proceedings of the fifth ACM international conference on Web search and data mining, Seattle Washington USA: ACM, Feb. 2012, pp. 13–22. doi: 10.1145/2124295.2124300.

[25] J. W. Cooper and J. M. Prager, "Anti-serendipity: finding useless documents and similar documents," in Proceedings of the 33rd Annual Hawaii International Conference on System Sciences, Maui, HI, USA: IEEE Comput. Soc, 2000, p. 8. doi: 10.1109/HICSS.2000.926691.

[26] P. Snickars, "More of the Same – On Spotify Radio," Cult. Unbound, vol. 9, no. 2, pp. 184–211, Oct. 2017, doi: 10.3384/cu.2000.1525.1792184.

[27] A. Novello, M. M. F. McKinney, and A. Kohlrausch, "Perceptual Evaluation of Inter-song Similarity in Western Popular Music," J. New Music Res., vol. 40, no. 1, pp. 1–26, Mar. 2011, doi: 10.1080/09298215.2010.523470.

[28] J.-F. Cardoso and B. H. Laheld, "Equivariant adaptive source separation," IEEE Trans. Signal Process., vol. 44, no. 12, pp. 3017–3030, Dec. 1996, doi: 10.1109/78.553476.

[29] E. Cano, D. FitzGerald, A. Liutkus, M. D. Plumbley, and F.-R. Stoter, "Musical Source Separation: An Introduction," IEEE Signal Process. Mag., vol. 36, no. 1, pp. 31–40, Jan. 2019, doi: 10.1109/MSP.2018.2874719.

[30] J. Park and K. Lee, "HARMONIC-PERCUSSIVE SOURCE SEPARATION USING HARMONICITY AND SPARSITY CONSTRAINTS," 2015.

[31] C. Duxbury, M. Davies, and M. Sandler, "SEPARATION OF TRANSIENT INFORMATION IN MUSICAL AUDIO USING MULTIRESOLUTION ANALYSIS TECHNIQUES," 2001.

[32] D. Fitzgerald, “Harmonic/Percussive Separation Using Median Filtering,” 2010.

[33] D. Fitzgerald, M. Cranitch, and E. Coyle, “Using Tensor Factorisation Models to Separate Drums from Polyphonic Music,” 2009.

[34] N. Dno, K. Miyamoto, J. L. Roux, H. Kameoka, and S. Sagayama, “SEPARATION OF A MONAURAL AUDIO SIGNAL INTO HARMONIC/PERCUSSIVE COMPONENTS BY COMPLEMENTARY DIFFUSION ON SPECTROGRAM,” 2008.

[35] T. Nodes and N. Gallagher, “Median filters: Some modifications and their properties,” IEEE Trans. Acoust. Speech Signal Process., vol. 30, no. 5, pp. 739–746, Oct. 1982, doi: 10.1109/TASSP.1982.1163951.

[36] L. A. C A and A. C A, “Research on DNN Methods in Music Source Separation Tools with emphasis to Spleeter,” Int. Res. J. Adv. Sci. Hub, vol. 3, no. Special Issue 6S, pp. 24–28, Jun. 2021, doi: 10.47392/irjash.2021.160.

[37] R. Hennequin, A. Khlif, F. Voituret, and M. Moussallam, “Spleeter: a fast and efficient music source separation tool with pre-trained models,” J. Open Source Softw., vol. 5, no. 50, p. 2154, Jun. 2020, doi: 10.21105/joss.02154.

[38] F.-R. Stöter, S. Uhlich, A. Liutkus, and Y. Mitsufuji, “Open-Unmix - A Reference Implementation for Music Source Separation,” J. Open Source Softw., vol. 4, no. 41, p. 1667, Sep. 2019, doi: 10.21105/joss.01667.

[39] J. P. Bello, L. Daudet, S. Abdallah, C. Duxbury, M. Davies, and M. B. Sandler, “A tutorial on onset detection in music signals,” IEEE Trans. Speech Audio Process., vol. 13, no. 5, pp. 1035–1047, Sep. 2005, doi: 10.1109/TSA.2005.851998.

[40] D. P. W. Ellis, “Beat Tracking by Dynamic Programming,” J. New Music Res., vol. 36, no. 1, pp. 51–60, Mar. 2007, doi: 10.1080/09298210701653344.

[41] R. Bellman, “The theory of dynamic programming,” Bull. Am. Math. Soc., vol. 60, no. 6, pp. 503–515, 1954, doi: 10.1090/S0002-9904-1954-09848-8.

[42] B. McFee et al., “librosa: Audio and Music Signal Analysis in Python,” presented at the Python in Science Conference, Austin, Texas, 2015, pp. 18–24. doi: 10.25080/Majora-7b98e3ed-003.

[43] A. K. Jain, M. N. Murty, and P. J. Flynn, “Data clustering: a review,” ACM Comput. Surv., vol. 31, no. 3, pp. 264–323, Sep. 1999, doi: 10.1145/331499.331504.

[44] U. Kutbay, “Partitional Clustering,” in Recent Applications in Data Clustering, IntechOpen, 2018. doi: 10.5772/intechopen.75836.

[45] S. Lloyd, “Least squares quantization in PCM,” IEEE Trans. Inf. Theory, vol. 28, no. 2, pp. 129–137, Mar. 1982, doi: 10.1109/TIT.1982.1056489.

[46] G. Hamerly and J. Drake, “Accelerating Lloyd’s Algorithm for k-Means Clustering,” in Partitional Clustering Algorithms, M. E. Celebi, Ed., Cham: Springer International Publishing, 2015, pp. 41–78. doi: 10.1007/978-3-319-09259-1_2.

[47] T. Kodinariya and P. Makwana, “Review on Determining of Cluster in K-means Clustering,” Int. J. Adv. Res. Comput. Sci. Manag. Stud., vol. 1, pp. 90–95, Jan. 2013.

[48] M. A. Syakur, B. K. Khotimah, E. M. S. Rochman, and B. D. Satoto, “Integration K-Means Clustering Method and Elbow Method For Identification of The Best Customer Profile Cluster,” IOP Conf. Ser. Mater. Sci. Eng., vol. 336, p. 012017, Apr. 2018, doi: 10.1088/1757-899X/336/1/012017.

[49] K. R. Shahapure and C. Nicholas, “Cluster Quality Analysis Using Silhouette Score,” in 2020 IEEE 7th International Conference on Data Science and Advanced Analytics (DSAA), sydney, Australia: IEEE, Oct. 2020, pp. 747–748. doi: 10.1109/DSAA49011.2020.00096.

[50] I. Chivers and J. Sleightholme, “An Introduction to Algorithms and the Big O Notation,” in Introduction to Programming with Fortran, Cham: Springer International Publishing, 2015, pp. 359–364. doi: 10.1007/978-3-319-17701-4_23.

[51] F. Gomez, E. Thul, and G. Toussaint, “AN EXPERIMENTAL COMPARISON OF FORMAL MEASURES OF RHYTHMIC SYNCOPATION”.

[52] A. T. Jebb, V. Ng, and L. Tay, “A Review of Key Likert Scale Development Advances: 1995–2019,” Front. Psychol., vol. 12, p. 637547, May 2021, doi: 10.3389/fpsyg.2021.637547.

[53] A. T. Jebb, V. Ng, and L. Tay, “A Review of Key Likert Scale Development Advances: 1995–2019,” Front. Psychol., vol. 12, 2021, Accessed: Oct. 31, 2023. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fpsyg.2021.637547

[54] E. H. Livingston, “The mean and standard deviation: what does it all mean?,” J. Surg. Res., vol. 119, no. 2, pp. 117–123, Jun. 2004, doi: 10.1016/j.jss.2004.02.008.

[55] M. Defferrard, K. Benzi, P. Vandergheynst, and X. Bresson, “FMA: A Dataset For Music Analysis.” arXiv, Sep. 05, 2017. Accessed: Sep. 23, 2023. [Online]. Available: http://arxiv.org/abs/1612.01840

[56] “Content-based Music Recommendation System.” Accessed: Sep. 23, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/9435533/

[57] Adiyansjah, A. A. S. Gunawan, and D. Suhartono, “Music Recommender System Based on Genre using Convolutional Recurrent Neural Networks,” Procedia Comput. Sci., vol. 157, pp. 99–109, Jan. 2019, doi: 10.1016/j.procs.2019.08.146.

[58] B. McFee et al., “librosa/librosa: 0.10.1.” Zenodo, Aug. 16, 2023. doi: 10.5281/ZENODO.591533.

[59] “Harmonic-percussive source separation — librosa 0.10.1 documentation.” Accessed: Nov. 07, 2023. [Online]. Available: https://librosa.org/doc/latest/auto_examples/plot_hprss.html

[60] “librosa.effects.hpss — librosa 0.10.1 documentation.” Accessed: Nov. 07, 2023. [Online]. Available: https://librosa.org/doc/main/generated/librosa.effects.hpss.html

[61] “librosa.beat.beat_track — librosa 0.10.1 documentation.” Accessed: Nov. 07, 2023. [Online]. Available: https://librosa.org/doc/latest/generated/librosa.beat.beat_track.html

[62] O. Kramer, “Scikit-Learn,” in Machine Learning for Evolution Strategies, O. Kramer, Ed., in Studies in Big Data. , Cham: Springer International Publishing, 2016, pp. 45–53. doi: 10.1007/978-3-319-33383-0_5.

[63] C. R. Harris et al., “Array programming with NumPy,” Nature, vol. 585, no. 7825, pp. 357–362, Sep. 2020, doi: 10.1038/s41586-020-2649-2.

[64] “numpy.average — NumPy v1.26 Manual.” Accessed: Nov. 16, 2023. [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.average.html

[65] J. D. Hunter, “Matplotlib: A 2D Graphics Environment,” Comput. Sci. Eng., vol. 9, no. 3, pp. 90–95, May 2007, doi: 10.1109/MCSE.2007.55.

[66] E. Winston and L. Saywood, “Beats to Relax/Study To: Contradiction and Paradox in Lofi Hip Hop,” IASPM J., vol. 9, no. 2, Art. no. 2, Dec. 2019.

[67] A. by J. M. Russell, D. D. from Barbara Illowsky and Susan Dean, and J. V. and D. Harrington, “5.2 The Normal Distribution,” Jan. 2021, Accessed: Nov. 21, 2023. [Online]. Available: https://ecampusontario.pressbooks.pub/significantstats/chapter/introduction-18/

[68] G. Loy, “Musicians Make a Standard: The MIDI Phenomenon,” Comput. Music J., vol. 9, no. 4, pp. 8–26, 1985, doi: 10.2307/3679619.

[69] L. Callender, C. Hawthorne, and J. Engel, “Improving Perceptual Quality of Drum Transcription with the Expanded Groove MIDI Dataset.” arXiv, Dec. 01, 2020. Accessed: Nov. 19, 2023. [Online]. Available: http://arxiv.org/abs/2004.00188

Appendix A

Beat Bin Test Code

```
import numpy as np
import librosa
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import librosa.display

def getBeatBins(signal):
    onsetTimes = librosa.onset.onset\_detect(y=signal, sr=sr, hop\_length=220, units='time'
        )
    tempo, beatTimes = librosa.beat.beat\_track(signal, sr=sr, hop\_length=220,units='time'
        )
    beatBins = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    secondsPerBeat = 60 / tempo
    bins = 12
    binDivder = secondsPerBeat / 12

    y\_beats = librosa.clicks(times=onsetTimes, length=len(signal), sr=sr)
    sf.write("testOnsets4.wav", signal + y\_beats, sr)

    for i in (range(len(beatTimes) - 1)):
        currentBeat = float(beatTimes[i])
        nextBeat = float(beatTimes[i+1])

        binSeconds = (nextBeat - currentBeat) / bins
        binBegin = currentBeat - (binSeconds/2)
        binEnd = currentBeat + (binSeconds/2)

        for j in range(bins):
            # Extract a subarray of onsets for each 16th note to remove duplicate notes
            currentOnsets = onsetTimes[(onsetTimes >= binBegin) & (onsetTimes < binEnd)]
            if np.size(currentOnsets) != 0:
                beatBins[j] += 1

            binBegin += binSeconds
            binEnd += binSeconds

    totalHits = 0
    offHits = 0
    for i in range(0, len(target)):
        totalHits += target[i]
        offHits += abs(target[i] − beatBins[i])

    accuracy = 100 ∗ (1 − (offHits / totalHits))
    beatBins.append(accuracy)
    return beatBins

import pandas as pd
import soundfile as sf

target = [20, 4, 4, 4, 8, 4, 12, 4, 8, 4, 4, 4]
df = pd.DataFrame(\{'target': [20, 4, 4, 4, 8, 4, 12, 4, 8, 4, 4, 4, 'accuracy (%)']})

files120 = ['data/justDrumsTest120.wav', 'data/justDrumsTest120Spleeter.mp3', 'data/
    songTest120.wav', 'data/songTest120Spleeter.mp3', ]
sr = 44100

for track in files120 :
    audio, sr = librosa.load(track, mono=True, sr=44100)
    harmonic, percussive = librosa.effects.hpss(audio)
    df[track + " raw audio"] = getBeatBins(audio)
    df[track + " HPSS percussive seperation"] = getBeatBins(percussive)

df.to\_csv('out.csv', encoding='utf−8', index=False)
```

Appendix B

Spleeter PSS Script

```
import wave
import subprocess
import os
import sys

genreList = os. listdir ('fma_small')

for genre in genreList:
    if genre != '.DS_Store' and genre != 'checksums' and genre != 'README.txt':
        path = "instrumentals/" + genre
        if os.path.exists(path) == False:
            songList = os. listdir ('fma_small/' + genre)
            for song in songList:
                if song != '.DS_Store':
                    print(song)
                    output = "instrumentals/" + genre
                    print(output)
                    inputFile = "fma_small/" + genre + "/" + song
                    print(inputFile)
                    command = "python −m spleeter separate −c mp3 −p spleeter:4stems −o " +
                         output + " " + inputFile
                    print(command)
                    os.system(command)
                outputList = os. listdir (output)
                for newFolder in outputList:
                    if newFolder != '.DS_Store':
                        newFolderList = os. listdir (output + "/" + newFolder)
                        for newFile in newFolderList:
                            if newFile != 'drums.mp3' and newFile != '.DS_Store':
                                os.remove(output + "/" + newFolder + "/" + newFile)
```

Appendix C

Get Onsets and Beats Code

C.1 Get Onsets and Beats Of Librosa PSS Audio

```
import numpy as np
import librosa
import pandas as pd
import os
import soundfile as sf

def getBeatInfo(fileName, signal):
    onsets = librosa.onset.onset_detect(y=signal, sr=sr, hop_length=220, units='time')
    tempo, beatTimes = librosa.beat.beat_track(signal, sr=sr, hop_length=220,units='time')
    filesArray.append(fileName)
    tempoArray.append(tempo)
    beatTimesArray.append(beatTimes)
    onsetsArray.append(onsets)


df = pd.DataFrame()
sr = 44100
genreList = os.listdir('fma_small')
filesArray = []
tempoArray = []
beatTimesArray = []
onsetsArray = []

for genre in genreList:
    if genre != '.DS_Store' and genre != 'checksums' and genre != 'README.txt':
        tracks = os.listdir('fma_small/' + genre)
        for track in tracks:
            if track != '.DS_Store' and track != 'checksums' and track != 'README.txt':
                audioFile = 'fma_small/' + genre + '/' + track
                audio, sr = librosa.load(audioFile, mono=True, sr=44100)
                harmonic, percussive = librosa.effects.hpss(audio)
                print(track)
                getBeatInfo(track, audio)

df['file'] = filesArray
df['tempo'] = tempoArray
```

```
df['beatTimes'] = beatTimesArray
df['onsets'] = onsetsArray
df.to_csv('hpssTrackInfo.csv', encoding='utf-8', index=False)
```

C.2 Get Onsets and Beats Of Spleeter PSS Audio

```
import numpy as np
import librosa
import pandas as pd
import os
import soundfile as sf

def getBeatInfo(fileName, signal):
    onsets = librosa.onset.onset_detect(y=signal, sr=sr, hop_length=220, units='time')
    tempo, beatTimes = librosa.beat.beat_track(signal, sr=sr, hop_length=220,units='time')
    filesArray.append(fileName)
    tempoArray.append(tempo)
    beatTimesArray.append(beatTimes)
    onsetsArray.append(onsets)


df = pd.DataFrame()
sr = 44100
genreList = os.listdir('isolatedDrumsSpleeter')
filesArray = []
tempoArray = []
beatTimesArray = []
onsetsArray = []

for genre in genreList:
    if genre != '.DS_Store' and genre != 'checksums' and genre != 'README.txt':
        tracks = os.listdir('isolatedDrumsSpleeter/' + genre)
        for track in tracks:
            if track != '.DS_Store' and track != 'checksums' and track != 'README.txt':
                audioFile = 'isolatedDrumsSpleeter/' + genre + '/' + track + '/drums.mp3'
                audio, sr = librosa.load(audioFile, mono=True, sr=44100)
                print(track)
                getBeatInfo(track, audio)

df['file'] = filesArray
df['tempo'] = tempoArray
df['beatTimes'] = beatTimesArray
df['onsets'] = onsetsArray
df.to_csv('spleeterTrackInfo.csv', encoding='utf-8', index=False)
```

Appendix D

Feature Extraction Code

```
import numpy as np
import math
import csv
import pandas as pd

def getWnbd(beatTimes, onsetTimes, bins):
    noteCount = 0
    wnbdTotal = 0

    numberOfBeatsSinceLastNote = 0
    tx = 0

    for i in (range(len(beatTimes) - 1)):
        currentBeat = float(beatTimes[i])
        nextBeat = float(beatTimes[i+1])

        binSeconds = (nextBeat - currentBeat) / bins
        binBegin = currentBeat - (binSeconds/2)
        binEnd = currentBeat + (binSeconds/2)

        for j in range(bins):
            # Extract a subarray of onsets for each 16th note to remove duplicate notes
            currentOnsets = onsetTimes[(onsetTimes >= binBegin) & (onsetTimes < binEnd)]
            if np.size(currentOnsets) != 0:
                # Calculate WNBD for previous note
                if tx != 0:
                    if numberOfBeatsSinceLastNote == 1:
                        wnbdTotal += 2 / tx
                    else:
                        wnbdTotal += 1 / tx

                numberOfBeatsSinceLastNote = 0
                noteCount += 1
                tx = min(j, bins - j) / bins

            binBegin += binSeconds
            binEnd += binSeconds
            numberOfBeatsSinceLastNote += 1

    if noteCount == 0:
        return 0
    else:
        return (wnbdTotal / noteCount)


def getRhythmValues(beatTimes, onsetTimes, bins):
    beatBins = np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])

    for i in (range(len(beatTimes) − 1)):
        currentBeat = float(beatTimes[i])
        nextBeat = float(beatTimes[i+1])

        binSeconds = (nextBeat − currentBeat) / bins
        binBegin = currentBeat − (binSeconds/2)
        binEnd = currentBeat + (binSeconds/2)

        for j in range(bins):
            # Extract a subarray of onsets for each 16th note to remove duplicate notes
            currentOnsets = onsetTimes[(onsetTimes >= binBegin) & (onsetTimes < binEnd)]
            if np.size(currentOnsets) != 0:
                beatBins[j] += 1

            binBegin += binSeconds
            binEnd += binSeconds

    totalHits = np.sum(beatBins)
    if totalHits != 0:
        pulse = beatBins[0] / totalHits
        eighthNote = beatBins[6] / totalHits
        sixteenthNote = (beatBins[3] + beatBins[9]) / totalHits
        triplet = (beatBins[4] + beatBins[8]) / totalHits
        sixteenthNoteTriplet = (beatBins[2] + beatBins[10]) / totalHits
        offBeat = (beatBins[1] + beatBins[5] + beatBins[7] + beatBins[11]) / totalHits

        return [pulse, eighthNote, sixteenthNote, triplet, sixteenthNoteTriplet, offBeat]

    else:
        return 0

# For Librosa HPSS
#with open("hpssTrackInfo.csv", 'r') as file:
# For Spleeter
with open("spleeterTrackInfo.csv", 'r') as file:
    bins = 12
    invalidFiles = 0

    filesArray = []
    featuresArray = []
```

```
df = pd.DataFrame()

csvreader = csv.reader(file)
header = next(csvreader)
for row in csvreader:
    fileName = row[0].replace('.mp3', '')
    beatTimes = row[2].split()
    onsetTimes = row[3].split()

    if beatTimes[0] == '[':
      beatTimes.pop(0)
    if onsetTimes[0] == '[':
      onsetTimes.pop(0)
    if beatTimes[len(beatTimes) - 1] == ']':
      beatTimes.pop(len(beatTimes) - 1)

    if onsetTimes[len(onsetTimes) - 1] == ']':
      onsetTimes.pop(len(onsetTimes) - 1)

    beatTimes[0] = beatTimes[0].replace('[', '')
    onsetTimes[0] = onsetTimes[0].replace('[', '')
    beatTimes[len(beatTimes) - 1] = beatTimes[len(beatTimes) - 1].replace(']', '')
    onsetTimes[len(onsetTimes) - 1] = onsetTimes[len(onsetTimes) - 1].replace(']', '')

    if(onsetTimes != [''] and beatTimes != ['']):
        dummy = np.array(beatTimes)
        beatTimes = dummy.astype(float)
        dummy = np.array(onsetTimes)
        onsetTimes = dummy.astype(float)

        wnbd = getWnbd(beatTimes, onsetTimes, bins)
        rhythmValues = getRhythmValues(beatTimes, onsetTimes, bins)

        if rhythmValues == 0:
          invalidFiles += 1
        else:
          features = [wnbd] + rhythmValues
          featuresArray.append(features)
          filesArray.append(fileName)
    else:
          invalidFiles += 1

df['file'] = filesArray
df['features'] = featuresArray
df.to_csv('spleeterFeatures.csv', encoding='utf-8', index=False)
print("invalid files = " + str(invalidFiles))
```

Appendix E

Cluster Formation Code

E.1 Form Clusters Using Groove Features Calculated From Librosa PSS Onsets and Beats

```
import numpy as np
import pandas as pd
import csv
import os

file1 = open("hpssFeatures.csv", 'r')
csvReader1 = csv.reader(file1)
header = next(csvReader1)
fileArray1 = np.array([])
featuresArray1 = []

rowCount = 0
for row in csvReader1:
    fileName = row[0]
    features = row[1].split()

    if features[0] == '[':
        features.pop(0)
    if features[len(features) - 1] == ']':
        features.pop(len(features) - 1)

    for i in range(len(features)):
        features[i] = features[i].replace(',', '')

    features[0] = features[0].replace('[', '')
    features[len(features) - 1] = features[len(features) - 1].replace(']', '')
    features = np.array(features)
    features = features.astype(float)
    featuresArray1.append(features)
    fileArray1 = np.append(fileArray1, fileName)
    rowCount += 1

featuresNpArray1 = np.zeros((rowCount,7))
for i in range(len(featuresArray1)):
    featuresNpArray1[i, :] = featuresArray1[i]
```

```
import matplotlib.pyplot as plt
import matplotlib.style as ms
import sklearn.cluster
import sklearn.preprocessing

scaler1 = sklearn.preprocessing.StandardScaler()
scaler1.fit(featuresNpArray1)
featuresNpArray1 = scaler1.transform(featuresNpArray1)

sse = [] #sum of squared errors
sil_coeff = [] #silhouette coefficients

range_n_clusters = range(2, 10)

for k in range_n_clusters:
    kmeans = sklearn.cluster.KMeans(n_clusters=k).fit(featuresNpArray1)

    sse.append(kmeans.inertia_)

    label = kmeans.labels_
    sil_coeff.append(sklearn.metrics.silhouette_score(featuresNpArray1, label, metric='
        euclidean'))

fig = plt.figure(figsize=(8,5))
fig.add_subplot(121)
plt.subplots_adjust(wspace=0.4)
plt.plot(range_n_clusters, sse,'b-',label='Sum of Squared Error')
plt.xlabel("Number of clusters")
plt.ylabel("Sum of Squared Error")
plt.legend()
fig.add_subplot(122)
plt.plot(range_n_clusters, sil_coeff,'b-',label='Silhouette Score')
plt.xlabel("Number of clusters")
plt.ylabel("Silhouette Score")
plt.legend()
plt.show()

clusters = 6
kmeans1 = sklearn.cluster.KMeans(n_clusters=clusters).fit(featuresNpArray1)

pred_classes1 = kmeans1.predict(featuresNpArray1)
print(len(fileArray1))

for cluster in range(clusters):
    print('cluster: ', cluster)

    indiciesWhere = np.where(pred_classes1 == 0)
    for index in indiciesWhere:
        subArray = fileArray1[np.where(pred_classes1 == cluster)]
        print(len(subArray))
```

```
from sklearn.metrics import pairwise_distances

distances = pairwise_distances(kmeans1.cluster_centers_, featuresNpArray1)
ind = [np.argpartition(i, 6)[:6] for i in distances]
closest = [fileArray1[indexes] for indexes in ind]
print(closest)
```

E.2 Form Clusters Using Groove Features Calculated From Spleeter PSS Onsets and Beats

```
import numpy as np
import pandas as pd
import csv
import os

file1 = open("spleeterFeatures.csv", 'r')
csvReader1 = csv.reader(file1)
header = next(csvReader1)
fileArray1 = np.array([])
featuresArray1 = []

rowCount = 0
for row in csvReader1:
    fileName = row[0]
    features = row[1].split()

    if features[0] == '[':
        features.pop(0)
    if features[len(features) - 1] == ']':
        features.pop(len(features) - 1)

    for i in range(len(features)):
        features[i] = features[i].replace(',', '')

    features[0] = features[0].replace('[', '')
    features[len(features) - 1] = features[len(features) - 1].replace(']', '')
    features = np.array(features)
    features = features.astype(float)
    featuresArray1.append(features)
    fileArray1 = np.append(fileArray1, fileName)
    rowCount += 1

featuresNpArray1 = np.zeros((rowCount,7))
for i in range(len(featuresArray1)):
    featuresNpArray1[i, :] = featuresArray1[i]

import matplotlib.pyplot as plt
import matplotlib.style as ms
import sklearn.cluster
import sklearn.preprocessing

scaler1 = sklearn.preprocessing.StandardScaler()
scaler1 . fit (featuresNpArray1)
featuresNpArray1 = scaler1.transform(featuresNpArray1)

sse = [] #sum of squared errors
sil_coeff = [] #silhouette coefficients

range_n_clusters = range(2, 10)

for k in range_n_clusters:
    kmeans = sklearn.cluster.KMeans(n_clusters=k).fit(featuresNpArray1)

    sse.append(kmeans.inertia_)

    label = kmeans.labels_
    sil_coeff .append(sklearn.metrics.silhouette_score(featuresNpArray1, label, metric='
        euclidean'))

fig = plt. figure ( figsize =(8,5))
fig .add_subplot(121)
plt .subplots_adjust(wspace=0.4)
plt . plot(range_n_clusters, sse,'b—',label='Sum of Squared Error')
plt . xlabel("Number of clusters")
plt . ylabel("Sum of Squared Errors")
plt .legend()
fig .add_subplot(122)
plt . plot(range_n_clusters, sil_coeff ,'b—',label='Silhouette Score')
plt . xlabel("Number of clusters")
plt . ylabel("Silhouette Score")
plt .legend()
plt .show()

clusters = 5
kmeans1 = sklearn.cluster.KMeans(n_clusters=clusters).fit(featuresNpArray1)

pred_classes1 = kmeans1.predict(featuresNpArray1)
print(len(fileArray1))

for cluster in range(clusters):
    print('cluster : ', cluster)

    indiciesWhere = np.where(pred_classes1 == 0)
    for index in indiciesWhere:
        subArray = fileArray1[np.where(pred_classes1 == cluster)]
        print(len(subArray))

from sklearn.metrics import pairwise_distances

distances = pairwise_distances(kmeans1.cluster_centers_, featuresNpArray1)
```

```
ind = [np.argpartition(i, 6)[:6] for i in distances]
closest = [fileArray1[indexes] for indexes in ind]
print(closest)
```

Appendix F

Evaluation Analysis Code

```
import numpy as np
import csv
import matplotlib.pyplot as plt
import math

def calculateSerendipityPerUser(serendipityPerUser, avgQualityPerUser,
      avgExpectednessPerUser,
    stdQualityPerUser, stdExpectednessPerUser, pssMethod, serendipityType):
    belowAverageQualityClustersWithSerendipity = 0
    aboveAverageQualityClustersWithSerendipity = 0
    belowAverageExpectednessClustersWithSerendipity = 0
    aboveAverageExpectednessClustersWithSerendipity = 0

    #Librosa start and end
    start = 0
    end = 6
    if (pssMethod == "Spleeter"):
        start = 6
        end = 11

    totalClusters = 0
    for i in range(0, len(serendipityPerUser)):
        for j in range(start, end):
            totalClusters += 1
            if (serendipityPerUser[i][j] > 0):
                userAvgQualityCluster = avgQualityPerUser[i][j]
                userAvgExpectednessCluster = avgExpectednessPerUser[i][j]
                userStdQualityCluster = stdQualityPerUser[i][j]
                userStdExpectednessCluster = stdExpectednessPerUser[i][j]

                userAvgQualityTotal = np.average(avgQualityPerUser[i][start:end])
                userAvgExpectednessTotal = np.average(avgExpectednessPerUser[i][start:end])
                userStdQualityTotal = np.average(stdQualityPerUser[i][start:end])
                userStdExpectednessTotal = np.average(stdExpectednessPerUser[i][start:end])

                if (userAvgQualityCluster <= userAvgQualityTotal):
                    belowAverageQualityClustersWithSerendipity += 1
                else:
```

```
                aboveAverageQualityClustersWithSerendipity += 1

            if (userAvgExpectednessCluster <= userAvgExpectednessTotal):
                belowAverageExpectednessClustersWithSerendipity += 1
            else:
                aboveAverageExpectednessClustersWithSerendipity += 1

    print('Total clusters analyzed = ' + str(totalClusters))
    totalClustersWithSerendipity = belowAverageQualityClustersWithSerendipity +
        aboveAverageQualityClustersWithSerendipity
    print('Number of ' + pssMethod + ' clusters with ' + serendipityType + ' = ' + str(
        totalClustersWithSerendipity))
    print('Number of ' + pssMethod + ' clusters with ' + serendipityType + ' with above
        average quality = ' + str(aboveAverageQualityClustersWithSerendipity))
    print('Number of ' + pssMethod + ' clusters with ' + serendipityType + ' with below
        average quality = ' + str(belowAverageQualityClustersWithSerendipity))
    print('Number of ' + pssMethod + ' clusters with ' + serendipityType + ' with above
        average expectedness = ' + str(aboveAverageExpectednessClustersWithSerendipity))
    print('Number of ' + pssMethod + ' lusters with ' + serendipityType + ' with below
        average expectedness = ' + str(belowAverageExpectednessClustersWithSerendipity))

    print('Percentage of ' + pssMethod + ' clusters with ' + serendipityType + ' = ' + str(
        totalClustersWithSerendipity/totalClusters))
    print('Percentage of ' + pssMethod + ' clusters with ' + serendipityType + ' with above
        average quality = ' + str(aboveAverageQualityClustersWithSerendipity/
        totalClustersWithSerendipity))
    print('Percentage of ' + pssMethod + ' clusters with ' + serendipityType + ' with below
        average quality = ' + str(belowAverageQualityClustersWithSerendipity/
        totalClustersWithSerendipity))
    print('Percentage of ' + pssMethod + ' clusters with ' + serendipityType + ' with above
        average expectedness = ' + str(aboveAverageExpectednessClustersWithSerendipity/
        totalClustersWithSerendipity))
    print('Percentage of ' + pssMethod + ' lusters with ' + serendipityType + ' with below
        average expectedness = ' + str(belowAverageExpectednessClustersWithSerendipity/
        totalClustersWithSerendipity))
    print('')
    print('')

stdQualityCluster = []
stdExpectedCluster = []
avgQualityCluster = []
avgExpectedCluster = []

stdQualityPerUser = []
stdExpectednessPerUser = []
avgQualityPerUser = []
avgExpectednessPerUser = []
serendipityPerUser = []
antiserendipityPerUser = []

serendipityPerCluster = np.zeros(11)
```

```
antiserendipityPerCluster = np.zeros(11)

trueNegativePerUser = []
truePositivePerUser = []
trueNegativePerCluster = np.zeros(11)
truePositivePerCluster = np.zeros(11)

favoriteTrackPerUser = []
leastFavoriteTrackPerUser = []

for i in range(0, 11):
    stdQualityCluster.append(np.array([]))
    stdExpectedCluster.append(np.array([]))
    avgQualityCluster.append(np.array([]))
    avgExpectedCluster.append(np.array([]))
    trueNegativePerUser.append(np.array([]))
    truePositivePerUser.append(np.array([]))

with open('evaluationResponses.csv', 'r') as csvFile:
    next(csvFile)
    reader = csv.reader(csvFile)
    userIndex = 0
    for row in reader:
        stdQualityPerUser.append(np.array([]))
        stdExpectednessPerUser.append(np.array([]))
        avgQualityPerUser.append(np.array([]))
        avgExpectednessPerUser.append(np.array([]))
        serendipityPerUser.append(np.array([]))
        antiserendipityPerUser.append(np.array([]))

        favoriteNumber = [int(i) for i in row[133].split() if i.isdigit()]
        leastFavoriteNumber = [int(i) for i in row[134].split() if i.isdigit()]
        if (len(favoriteNumber) == 0):
            favoriteTrackPerUser.append(-1)
        else:
            favoriteTrackPerUser.append(favoriteNumber[0])
        if(len(leastFavoriteNumber) == 0):
            leastFavoriteTrackPerUser.append(-1)
        else:
            leastFavoriteTrackPerUser.append(leastFavoriteNumber[0])
        for i in range(0, 11):
            # extract array of quality and expected values
            clusterArray = row[(i*12) + 1:(i*12) + 13]
            qualityArray = np.array(clusterArray[0::2]).astype(int)
            expectedArray = np.array(clusterArray[1::2]).astype(int)
            stdQualityPerUser[userIndex] = np.append(stdQualityPerUser[userIndex], np.std(
                qualityArray, ddof=1))
            stdExpectednessPerUser[userIndex] = np.append(stdExpectednessPerUser[userIndex
                ], np.std(expectedArray, ddof=1))
            avgQualityPerUser[userIndex] = np.append(avgQualityPerUser[userIndex], np.
                average(qualityArray))
            avgExpectednessPerUser[userIndex] = np.append(avgExpectednessPerUser[userIndex
                ], np.average(expectedArray))
            stdQualityCluster[i] = np.append(stdQualityCluster[i], np.std(qualityArray, ddof
                =1))
            stdExpectedCluster[i] = np.append(stdExpectedCluster[i], np.std(expectedArray,
                ddof=1))
            avgQualityCluster[i] = np.append(avgQualityCluster[i], np.average(qualityArray))
            avgExpectedCluster[i] = np.append(avgExpectedCluster[i], np.average(
                expectedArray))

            # get serendipity and antiserendipity
            serendipity = np.zeros(6)
            antiserendipity = np.zeros(6)
            trueNegative = np.zeros(6)
            truePositive = np.zeros(6)
            for j in range(0, 6):
                if (qualityArray[j] > 4 and expectedArray[j] < 4):
                    serendipity[j] += 1
                    serendipityPerCluster[i] += 1
                elif (qualityArray[j] < 4 and expectedArray[j] > 4):
                    antiserendipity[j] += 1
                    antiserendipityPerCluster[i] += 1
                elif (qualityArray[j] < 4 and expectedArray[j] < 4):
                    trueNegative[j] += 1
                    trueNegativePerCluster[i] += 1
                elif (qualityArray[j] > 4 and expectedArray[j] > 4):
                    truePositive[j] += 1
                    truePositivePerCluster[i] += 1

            serendipityPerUser[userIndex] = np.append(serendipityPerUser[userIndex], sum(
                serendipity))
            antiserendipityPerUser[userIndex] = np.append(antiserendipityPerUser[userIndex],
                sum(antiserendipity))
            truePositivePerUser[i] = np.append(truePositivePerUser[i], truePositive)
            trueNegativePerUser[i] = np.append(trueNegativePerUser[i], trueNegative)

        userIndex += 1

# Serendipity vs. average cluster values
calculateSerendipityPerUser(serendipityPerUser, avgQualityPerUser, avgExpectednessPerUser,
    stdQualityPerUser, stdExpectednessPerUser, 'Librosa', 'serendipity')
calculateSerendipityPerUser(serendipityPerUser, avgQualityPerUser, avgExpectednessPerUser,
    stdQualityPerUser, stdExpectednessPerUser, 'Spleeter', 'serendipity')
calculateSerendipityPerUser(antiserendipityPerUser, avgQualityPerUser,
     avgExpectednessPerUser,
    stdQualityPerUser, stdExpectednessPerUser, 'Librosa', 'anti-serendipity')
calculateSerendipityPerUser(antiserendipityPerUser, avgQualityPerUser,
     avgExpectednessPerUser,
    stdQualityPerUser, stdExpectednessPerUser, 'Spleeter', 'anti-serendipity')

# Favorite track cluster vs. average
```

```
favoriteTracksInBelowAverageQualityCluster = 0
favoriteTracksInAboveAverageQualityCluster = 0
favoriteTracksInBelowAverageExpectednessCluster = 0
favoriteTracksInAboveAverageExpectednessCluster = 0
favoriteTracksInLibrosaClusters = 0
favoriteTracksInSpleeterClusters = 0
totalValidFavoriteTracks = len(favoriteTrackPerUser)

for i in range(0, len(favoriteTrackPerUser)):
    if (favoriteTrackPerUser[i] == −1):
        totalValidFavoriteTracks −= 1
    else:
        favoriteTrackCluster = math.floor((favoriteTrackPerUser[i] − 1) / 6)
        if(favoriteTrackCluster <= 5):
            favoriteTracksInLibrosaClusters += 1
        else:
            favoriteTracksInSpleeterClusters += 1

        averageUserQuality = np.average(avgQualityPerUser[i])
        averageUserExpectedness = np.average(avgExpectednessPerUser[i])

        averageQualityFavoriteTrackCluster = avgQualityPerUser[i][favoriteTrackCluster]
        averageExpectednessFavoriteTrackCluster = avgExpectednessPerUser[i][
             favoriteTrackCluster]

        if (averageUserQuality > averageQualityFavoriteTrackCluster):
            favoriteTracksInBelowAverageQualityCluster += 1
        else:
            favoriteTracksInAboveAverageQualityCluster += 1
        if (averageUserExpectedness > averageExpectednessFavoriteTrackCluster):
            favoriteTracksInBelowAverageExpectednessCluster += 1
        else:
            favoriteTracksInAboveAverageExpectednessCluster += 1

print('Total participants with valid favorite tracks = ' + str(totalValidFavoriteTracks))
print('Number of favorite tracks in Librosa clusters = ' + str(
    favoriteTracksInLibrosaClusters))
print('Number of favorite tracks in Spleeter clusters = ' + str(
    favoriteTracksInSpleeterClusters))
print('Number of participants with favorite tracks in above average quality clusters = ' +
    str(favoriteTracksInAboveAverageQualityCluster))
print('Number of participants with favorite tracks in below average quality clusters = ' +
    str(favoriteTracksInBelowAverageQualityCluster))
print('Number of participants with favorite tracks in above average expectedness clusters =
     ' + str(favoriteTracksInAboveAverageExpectednessCluster))
print('Number of participants with favorite tracks in below average expectedness clusters =
     ' + str(favoriteTracksInBelowAverageExpectednessCluster))

print('Percentage of favorite tracks in Librosa clusters = ' + str(
    favoriteTracksInLibrosaClusters / totalValidFavoriteTracks))
```

```
print('Percentage of favorite tracks in Spleeter clusters = ' + str(
    favoriteTracksInSpleeterClusters / totalValidFavoriteTracks))
print('Percentage of participants with favorite tracks in above average quality clusters = '
     + str(favoriteTracksInAboveAverageQualityCluster / totalValidFavoriteTracks))
print('Percentage of participants with favorite tracks in below average quality clusters = '
     + str(favoriteTracksInBelowAverageQualityCluster / totalValidFavoriteTracks))
print('Percentage of participants with favorite tracks in above average expectedness
    clusters = ' + str(favoriteTracksInAboveAverageExpectednessCluster /
    totalValidFavoriteTracks))
print('Percentage of participants with favorite tracks in below average expectedness
    clusters = ' + str(favoriteTracksInBelowAverageExpectednessCluster /
    totalValidFavoriteTracks))
print('')
print('')

# Least favorite track cluster vs. average
leastFavoriteTracksInBelowAverageQualityCluster = 0
leastFavoriteTracksInAboveAverageQualityCluster = 0
leastFavoriteTracksInBelowAverageExpectednessCluster = 0
leastFavoriteTracksInAboveAverageExpectednessCluster = 0
leastFavoriteTracksInLibrosaClusters = 0
leastFavoriteTracksInSpleeterClusters = 0
totalValidLeastFavoriteTracks = len(leastFavoriteTrackPerUser)

for i in range(0, len(leastFavoriteTrackPerUser)):
   if (leastFavoriteTrackPerUser[i] == -1):
      totalValidLeastFavoriteTracks -= 1
   else:
      leastFavoriteTrackCluster = math.floor((leastFavoriteTrackPerUser[i] - 1) / 6)
      if(leastFavoriteTrackCluster <= 5):
         leastFavoriteTracksInLibrosaClusters += 1
      else:
         leastFavoriteTracksInSpleeterClusters += 1

      averageUserQuality = np.average(avgQualityPerUser[i])
      averageUserExpectedness = np.average(avgExpectednessPerUser[i])

      averageQualityLeastFavoriteTrackCluster = avgQualityPerUser[i][
          leastFavoriteTrackCluster]
      averageExpectednessLeastFavoriteTrackCluster = avgExpectednessPerUser[i][
          leastFavoriteTrackCluster]

      if (averageUserQuality > averageQualityLeastFavoriteTrackCluster):
         leastFavoriteTracksInBelowAverageQualityCluster += 1
      else:
         leastFavoriteTracksInAboveAverageQualityCluster += 1
      if (averageUserExpectedness > averageExpectednessLeastFavoriteTrackCluster):
         leastFavoriteTracksInBelowAverageExpectednessCluster += 1
      else:
         leastFavoriteTracksInAboveAverageExpectednessCluster += 1
```

```
print('Total participants with valid least favorite tracks = ' + str(
    totalValidLeastFavoriteTracks))
print('Number of least favorite tracks in Librosa clusters = ' + str(
    leastFavoriteTracksInLibrosaClusters))
print('Number of least favorite tracks in Spleeter clusters = ' + str(
    leastFavoriteTracksInSpleeterClusters))
print('Number of participants with least favorite tracks in above average quality clusters
    = ' + str(leastFavoriteTracksInAboveAverageQualityCluster))
print('Number of participants with least favorite tracks in below average quality clusters
    = ' + str(leastFavoriteTracksInBelowAverageQualityCluster))
print('Number of participants with least favorite tracks in above average expectedness
    clusters = ' + str(leastFavoriteTracksInAboveAverageExpectednessCluster))
print('Number of participants with least favorite tracks in below average expectedness
    clusters = ' + str(leastFavoriteTracksInBelowAverageExpectednessCluster))

print('Percentage of least favorite tracks in Librosa clusters = ' + str(
    leastFavoriteTracksInLibrosaClusters / totalValidLeastFavoriteTracks))
print('Percentage of least favorite tracks in Spleeter clusters = ' + str(
    leastFavoriteTracksInSpleeterClusters / totalValidLeastFavoriteTracks))
print('Percentage of participants with least favorite tracks in above average quality
    clusters = ' + str(leastFavoriteTracksInAboveAverageQualityCluster /
    totalValidLeastFavoriteTracks))
print('Percentage of participants with least favorite tracks in below average quality
    clusters = ' + str(leastFavoriteTracksInBelowAverageQualityCluster /
    totalValidLeastFavoriteTracks))
print('Percentage of participants with least favorite tracks in above average expectedness
    clusters = ' + str(leastFavoriteTracksInAboveAverageExpectednessCluster /
    totalValidLeastFavoriteTracks))
print('Percentage of participants with least favorite tracks in below average expectedness
    clusters = ' + str(leastFavoriteTracksInBelowAverageExpectednessCluster /
    totalValidLeastFavoriteTracks))
print('')
print('')

# Graph avg and std
stdQualityClusterLibrosa = stdQualityCluster[0:6]
stdExpectedClusterLibrosa = stdExpectedCluster[0:6]
stdQualityClusterSpleeter = stdQualityCluster[6:11]
stdExpectedClusterSpleeter = stdExpectedCluster[6:11]

avgQualityClusterLibrosa = avgQualityCluster[0:6]
avgExpectedClusterLibrosa = avgExpectedCluster[0:6]
avgQualityClusterSpleeter = avgQualityCluster[6:11]
avgExpectedClusterSpleeter = avgExpectedCluster[6:11]

numParticipants = len(stdQualityClusterLibrosa[0])
x = np.arange(numParticipants)

qualityLibrosaFig, qualityLibrosaAxs = plt.subplots(nrows=len(stdQualityClusterLibrosa),
    ncols=2, figsize=(18, 20))
```

```
qualityLibrosaFig.subplots_adjust(left=0.05, bottom=0.05, right=0.95, top=0.93, wspace
    =0.2, hspace=0.4)
qualityLibrosaFig.suptitle("Standard Deviation and Average of Quality Values Per
    Participant in Librosa PSS Clusters", fontsize=25)
for ax in qualityLibrosaAxs.flat:
    ax.set_xticks(x)
for ax in qualityLibrosaAxs[:,0]:
    ax.set(xlabel='Participant', ylabel='Standard Deviation')
for ax in qualityLibrosaAxs[:,1]:
    ax.set(xlabel='Participant', ylabel='Average')


expectedLibrosaFig, expectedLibrosaAxs = plt.subplots(nrows=len(
    stdExpectedClusterLibrosa), ncols=2, figsize=(18, 20))
expectedLibrosaFig.subplots_adjust(left=0.05, bottom=0.05, right=0.95, top=0.93, wspace
    =0.2, hspace=0.4)
expectedLibrosaFig.suptitle("Standard Deviation and Average of Expectedness Values Per
    Participant in Librosa PSS Clusters", fontsize=25)
for ax in expectedLibrosaAxs.flat:
    ax.set_xticks(x)
for ax in expectedLibrosaAxs[:,0]:
    ax.set(xlabel='Participant', ylabel='Standard Deviation')
for ax in expectedLibrosaAxs[:,1]:
    ax.set(xlabel='Participant', ylabel='Average')

qualitySpleeterFig, qualitySpleeterAxs = plt.subplots(nrows=len(stdQualityClusterSpleeter),
     ncols=2, figsize=(18, 20))
qualitySpleeterFig.subplots_adjust(left=0.05, bottom=0.05, right=0.95, top=0.93, wspace
    =0.2, hspace=0.4)
qualitySpleeterFig.suptitle("Standard Deviation and Average of Quality Values Per
    Participant in Spleeter PSS Clusters", fontsize=25)
for ax in qualitySpleeterAxs.flat:
    ax.set_xticks(x)
for ax in qualitySpleeterAxs[:,0]:
    ax.set(xlabel='Participant', ylabel='Standard Deviation')
for ax in qualitySpleeterAxs[:,1]:
    ax.set(xlabel='Participant', ylabel='Average')

expectedSpleeterFig, expectedSpleeterAxs = plt.subplots(nrows=len(
    stdExpectedClusterSpleeter), ncols=2, figsize=(18, 20))
expectedSpleeterFig.subplots_adjust(left=0.05, bottom=0.05, right=0.95, top=0.93, wspace
    =0.2, hspace=0.4)
expectedSpleeterFig.suptitle("Standard Deviation and Average of Expectedness Values Per
    Participant in Spleeter PSS Clusters", fontsize=25)
for ax in expectedSpleeterAxs.flat:
    ax.set_xticks(x)
for ax in expectedSpleeterAxs[:,0]:
    ax.set(xlabel='Participant', ylabel='Standard Deviation')
for ax in expectedSpleeterAxs[:,1]:
    ax.set(xlabel='Participant', ylabel='Average')
```

```
averageQualityPerClusterLibrosa = np.zeros(6)
averageStdQualityPerClusterLibrosa = np.zeros(6)
averageExpectedPerClusterLibrosa = np.zeros(6)
averageStdExpectedPerClusterLibrosa = np.zeros(6)

for i in range(0, len(stdQualityClusterLibrosa)):
    stdQuality = stdQualityClusterLibrosa[i]
    qualityLibrosaAxs[i, 0].set_title("Standard Deviation Cluster " + str(i), fontsize=15)
    qualityLibrosaAxs[i, 0].axhline(np.average(stdQuality), color='red', linestyle='--',
         linewidth=0.5, label='std = ' + str(round(np.average(stdQuality), 2)))
    qualityLibrosaAxs[i, 0].legend(loc='upper right')
    qualityLibrosaAxs[i, 0].plot(x, stdQuality)
    averageStdQualityPerClusterLibrosa[i] = np.average(stdQuality)
    avgQuality = avgQualityClusterLibrosa[i]
    qualityLibrosaAxs[i, 1].set_title("Average Cluster " + str(i), fontsize=15)
    qualityLibrosaAxs[i, 1].axhline(np.average(avgQuality), color='red', linestyle='--',
         linewidth=0.5, label='avg = ' + str(round(np.average(avgQuality), 2)))
    qualityLibrosaAxs[i, 1].legend(loc='upper right')
    qualityLibrosaAxs[i, 1].plot(x, avgQuality)
    averageQualityPerClusterLibrosa[i] = np.average(avgQuality)

    stdExpected = stdExpectedClusterLibrosa[i]
    expectedLibrosaAxs[i, 0].set_title("Cluster " + str(i), fontsize=15)
    expectedLibrosaAxs[i, 0].axhline(np.average(stdExpected), color='red', linestyle='--',
         linewidth=0.5, label='std = ' + str(round(np.average(stdExpected), 2)))
    expectedLibrosaAxs[i, 0].legend(loc='upper right')
    expectedLibrosaAxs[i, 0].plot(x, stdExpected)
    averageStdExpectedPerClusterLibrosa[i] = np.average(stdExpected)
    avgExpected = avgExpectedClusterLibrosa[i]
    expectedLibrosaAxs[i, 1].set_title("Average Cluster " + str(i), fontsize=15)
    expectedLibrosaAxs[i, 1].axhline(np.average(avgExpected), color='red', linestyle='--',
         linewidth=0.5, label='avg = ' + str(round(np.average(avgExpected), 2)))
    expectedLibrosaAxs[i, 1].legend(loc='upper right')
    expectedLibrosaAxs[i, 1].plot(x, avgExpected)
    averageExpectedPerClusterLibrosa[i] = np.average(avgExpected)

print('Average standard deviation quality value for all Librosa clusters: ' + str(np.
    average(averageStdQualityPerClusterLibrosa)))
print('Average quality value for all Librosa clusters: ' + str(np.average(
    averageQualityPerClusterLibrosa)))
print('Average standard deviation expectedness value for all Librosa clusters: ' + str(np.
    average(averageStdExpectedPerClusterLibrosa)))
print('Average expectedness value for all Librosa clusters: ' + str(np.average(
    averageExpectedPerClusterLibrosa)))
print('')

averageQualityPerClusterSpleeter = np.zeros(5)
averageStdQualityPerClusterSpleeter = np.zeros(5)
averageExpectedPerClusterSpleeter = np.zeros(5)
averageStdExpectedPerClusterSpleeter = np.zeros(5)
```

```
for i in range(0, len(stdQualityClusterSpleeter)):
    stdQuality = stdQualityClusterSpleeter[i]
    qualitySpleeterAxs[i, 0].set_title("Standard Deviation Cluster " + str(i), fontsize=15)
    qualitySpleeterAxs[i, 0].axhline(np.average(stdQuality), color='red', linestyle='--',
        linewidth=0.5, label='std = ' + str(round(np.average(stdQuality), 2)))
    qualitySpleeterAxs[i, 0].legend(loc='upper right')
    qualitySpleeterAxs[i, 0].plot(x, stdQuality)
    averageStdQualityPerClusterSpleeter[i] = np.average(stdQuality)
    avgQuality = avgQualityClusterSpleeter[i]
    qualitySpleeterAxs[i, 1].set_title("Average Cluster " + str(i), fontsize=15)
    qualitySpleeterAxs[i, 1].axhline(np.average(avgQuality), color='red', linestyle='--',
        linewidth=0.5, label='avg = ' + str(round(np.average(avgQuality), 2)))
    qualitySpleeterAxs[i, 1].legend(loc='upper right')
    qualitySpleeterAxs[i, 1].plot(x, avgQuality)
    averageQualityPerClusterSpleeter[i] = np.average(avgQuality)

    stdExpected = stdExpectedClusterSpleeter[i]
    expectedSpleeterAxs[i, 0].set_title("Standard Deviation Cluster " + str(i), fontsize=15)
    expectedSpleeterAxs[i, 0].axhline(np.average(stdExpected), color='red', linestyle='--',
        linewidth=0.5, label='std = ' + str(round(np.average(stdExpected), 2)))
    expectedSpleeterAxs[i, 0].legend(loc='upper right')
    expectedSpleeterAxs[i, 0].plot(x, stdExpected)
    averageStdExpectedPerClusterSpleeter[i] = np.average(stdExpected)
    avgExpected = avgExpectedClusterSpleeter[i]
    expectedSpleeterAxs[i, 1].set_title("Average Cluster " + str(i), fontsize=15)
    expectedSpleeterAxs[i, 1].axhline(np.average(avgExpected), color='red', linestyle='--',
        linewidth=0.5, label='avg = ' + str(round(np.average(avgExpected), 2)))
    expectedSpleeterAxs[i, 1].legend(loc='upper right')
    expectedSpleeterAxs[i, 1].plot(x, avgExpected)
    averageExpectedPerClusterSpleeter[i] = np.average(avgExpected)

print('Average standard deviation quality value for all Spleeter clusters: ' + str(np.
    average(averageStdQualityPerClusterSpleeter)))
print('Average quality value for all Spleeter clusters: ' + str(np.average(
    averageQualityPerClusterSpleeter)))
print('Average standard deviation expectedness value for all Spleeter clusters: ' + str(np.
    average(averageStdExpectedPerClusterSpleeter)))
print('Average expectedness value for all Spleeter clusters: ' + str(np.average(
    averageExpectedPerClusterSpleeter)))

qualityLibrosaFig.savefig('qualityLibrosaFig.png')
expectedLibrosaFig.savefig('expectedLibrosaFig.png')
qualitySpleeterFig.savefig('qualitySpleeterFig.png')
expectedSpleeterFig.savefig('expectedSpleeterFig.png')

# Serendipity
serendipityPerClusterLibrosa = serendipityPerCluster[0:6]
antiserendipityPerClusterLibrosa = antiserendipityPerCluster[0:6]
serendipityPerClusterSpleeter = serendipityPerCluster[6:11]
antiserendipityPerClusterSpleeter = antiserendipityPerCluster[6:11]
```

```
trueNegativePerUserLibrosa = trueNegativePerUser[0:6]
truePositivePerUserLibrosa = truePositivePerUser[0:6]
trueNegativePerUserSpleeter = trueNegativePerUser[6:11]
truePositivePerUserSpleeter = truePositivePerUser[6:11]

trueNegativePerClusterLibrosa = trueNegativePerCluster[0:6]
truePositivePerClusterLibrosa = truePositivePerCluster [0:6]
trueNegativePerClusterSpleeter = trueNegativePerCluster[6:11]
truePositivePerClusterSpleeter = truePositivePerCluster [6:11]

serendipityPerClusterLibrosaFig, serendipityPerClusterLibrosaAxs = plt.subplots( figsize
    =(10, 3))
serendipityPerClusterLibrosaFig.subplots_adjust(bottom=0.2)
serendipityPerClusterLibrosaFig. suptitle ("Total Serendipity Across All Participants Per
    Librosa PSS Cluster")
serendipityPerClusterLibrosaAxs.set(xlabel='Cluster', ylabel='Total Serendipity Found')
serendipityPerClusterLibrosaAxs.set_xticks(np.arange(6))
serendipityPerClusterLibrosaAxs.axhline(np.average(serendipityPerClusterLibrosa), color='
    red', linewidth=0.5, label='Serendipity per Cluster = ' + str(round(np.average(
    serendipityPerClusterLibrosa), 2)) + ', Total Serendipity = ' + str(np.sum(
    serendipityPerClusterLibrosa)))
serendipityPerClusterLibrosaAxs.legend(loc='upper right')
serendipityPerClusterLibrosaAxs.plot(np.arange(6), serendipityPerClusterLibrosa)
serendipityPerClusterLibrosaFig. savefig ( 'serendipityPerClusterLibrosaFig.png')

serendipityPerClusterSpleeterFig, serendipityPerClusterSpleeterAxs = plt.subplots( figsize
    =(10, 3))
serendipityPerClusterSpleeterFig.subplots_adjust(bottom=0.2)
serendipityPerClusterSpleeterFig. suptitle ("Total Serendipity Across All Participants Per
    Spleeter PSS Cluster")
serendipityPerClusterSpleeterAxs.set(xlabel='Cluster', ylabel='Total Serendipity Found')
serendipityPerClusterSpleeterAxs.set_xticks(np.arange(5))
serendipityPerClusterSpleeterAxs.axhline(np.average(serendipityPerClusterSpleeter), color='
    red', linewidth=0.5, label='Serendipity per Cluster = ' + str(round(np.average(
    serendipityPerClusterSpleeter), 2)) + ', Total Serendipity = ' + str(np.sum(
    serendipityPerClusterSpleeter)))
serendipityPerClusterSpleeterAxs.legend(loc='upper right')
serendipityPerClusterSpleeterAxs.plot(np.arange(5), serendipityPerClusterSpleeter)
serendipityPerClusterSpleeterFig. savefig ( 'serendipityPerClusterSpleeterFig.png')

antiserendipityPerClusterLibrosaFig, antiserendipityPerClusterLibrosaAxs = plt.subplots(
    figsize =(10, 3))
antiserendipityPerClusterLibrosaFig.subplots_adjust(bottom=0.2)
antiserendipityPerClusterLibrosaFig. suptitle ("Total Antiserendipity Across All Participants
    Per Librosa PSS Cluster")
antiserendipityPerClusterLibrosaAxs.set(xlabel='Cluster', ylabel='Total Anti−Serendipity
    Found')
antiserendipityPerClusterLibrosaAxs.set_xticks(np.arange(6))
antiserendipityPerClusterLibrosaAxs.axhline(np.average(antiserendipityPerClusterLibrosa),
    color='red', linewidth=0.5, label='Anti−serendipity per Cluster = ' + str(round(np.
    average(antiserendipityPerClusterLibrosa), 2)) + ', Total Anti−Serendipity = ' + str(np.
    sum(antiserendipityPerClusterLibrosa)))
antiserendipityPerClusterLibrosaAxs.legend(loc='upper right')
antiserendipityPerClusterLibrosaAxs.plot(np.arange(6), antiserendipityPerClusterLibrosa)
antiserendipityPerClusterLibrosaFig. savefig ( 'antiserendipityPerClusterLibrosaFig.png')

antiserendipityPerClusterSpleeterFig, antiserendipityPerClusterSpleeterAxs = plt.subplots(
    figsize =(10, 3))
antiserendipityPerClusterSpleeterFig.subplots_adjust(bottom=0.2)
antiserendipityPerClusterSpleeterFig. suptitle ("Total Antiserendipity Across All Participants
    Per Spleeter PSS Cluster")
antiserendipityPerClusterSpleeterAxs.axhline(np.average(antiserendipityPerClusterSpleeter),
    color='red', linewidth=0.5, label='Anti−serendipity per Cluster = ' + str(round(np.
    average(antiserendipityPerClusterSpleeter), 2)) + ', Total Anti−Serendipity = ' + str(np.
    sum(antiserendipityPerClusterSpleeter)))
antiserendipityPerClusterSpleeterAxs.set(xlabel='Cluster', ylabel='Total Anti−Serendipity
    Found')
antiserendipityPerClusterSpleeterAxs.set_xticks(np.arange(5))
antiserendipityPerClusterSpleeterAxs.legend(loc='upper right')
antiserendipityPerClusterSpleeterAxs.plot(np.arange(5), antiserendipityPerClusterSpleeter)
antiserendipityPerClusterSpleeterFig. savefig ( 'antiserendipityPerClusterSpleeterFig.png')

trueNegativePerClusterLibrosaFig, trueNegativePerClusterLibrosaAxs = plt.subplots(figsize
    =(10, 3))
trueNegativePerClusterLibrosaFig.subplots_adjust(bottom=0.2)
trueNegativePerClusterLibrosaFig.suptitle("Total True Negative Across All Participants Per
    Librosa PSS Cluster")
trueNegativePerClusterLibrosaAxs.set(xlabel='Cluster', ylabel='Total trueNegative Found')
trueNegativePerClusterLibrosaAxs.set_xticks(np.arange(6))
trueNegativePerClusterLibrosaAxs.axhline(np.average(trueNegativePerClusterLibrosa), color=
    'red', linewidth=0.5, label='True Negative per Cluster = ' + str(round(np.average(
    trueNegativePerClusterLibrosa), 2)) + ', Total True Negative = ' + str(np.sum(
    trueNegativePerClusterLibrosa)))
trueNegativePerClusterLibrosaAxs.legend(loc='upper right')
trueNegativePerClusterLibrosaAxs.plot(np.arange(6), trueNegativePerClusterLibrosa)
trueNegativePerClusterLibrosaFig.savefig('trueNegativePerClusterLibrosaFig.png')

trueNegativePerClusterSpleeterFig, trueNegativePerClusterSpleeterAxs = plt.subplots(figsize
    =(10, 3))
trueNegativePerClusterSpleeterFig.subplots_adjust(bottom=0.2)
trueNegativePerClusterSpleeterFig.suptitle("Total True Negative Across All Participants Per
    Spleeter PSS Cluster")
trueNegativePerClusterSpleeterAxs.set(xlabel='Cluster', ylabel='Total trueNegative Found')
trueNegativePerClusterSpleeterAxs.set_xticks(np.arange(5))
trueNegativePerClusterSpleeterAxs.axhline(np.average(trueNegativePerClusterSpleeter), color
    ='red', linewidth=0.5, label='True Negative per Cluster = ' + str(round(np.average(
    trueNegativePerClusterSpleeter), 2)) + ', Total True Negative = ' + str(np.sum(
    trueNegativePerClusterSpleeter)))
trueNegativePerClusterSpleeterAxs.legend(loc='upper right')
trueNegativePerClusterSpleeterAxs.plot(np.arange(5), trueNegativePerClusterSpleeter)
trueNegativePerClusterSpleeterFig.savefig('trueNegativePerClusterSpleeterFig.png')
```

```
truePositivePerClusterLibrosaFig, truePositivePerClusterLibrosaAxs = plt.subplots( figsize
    =(10, 3))
truePositivePerClusterLibrosaFig.subplots_adjust(bottom=0.2)
truePositivePerClusterLibrosaFig. suptitle ("Total True Positive Across All Participants Per
    Librosa PSS Cluster")
truePositivePerClusterLibrosaAxs.set(xlabel='Cluster', ylabel='Total Serendipity Found')
truePositivePerClusterLibrosaAxs.set_xticks(np.arange(6))
truePositivePerClusterLibrosaAxs.axhline(np.average(truePositivePerClusterLibrosa), color='
    red', linewidth=0.5, label='True Positive per Cluster = ' + str(round(np.average(
    truePositivePerClusterLibrosa), 2)) + ', Total True Positive = ' + str(np.sum(
    truePositivePerClusterLibrosa)))
truePositivePerClusterLibrosaAxs.legend(loc='upper right')
truePositivePerClusterLibrosaAxs.plot(np.arange(6), truePositivePerClusterLibrosa)
truePositivePerClusterLibrosaFig. savefig ( 'truePositivePerClusterLibrosaFig.png')

truePositivePerClusterSpleeterFig, truePositivePerClusterSpleeterAxs = plt.subplots( figsize
    =(10, 3))
truePositivePerClusterSpleeterFig.subplots_adjust(bottom=0.2)
truePositivePerClusterSpleeterFig. suptitle ("Total True Positive Across All Participants Per
    Spleeter PSS Cluster")
truePositivePerClusterSpleeterAxs.axhline(np.average(truePositivePerClusterSpleeter), color
    ='red', linewidth=0.5, label='True Positive per Cluster = ' + str(round(np.average(
    truePositivePerClusterSpleeter), 2)) + ', Total True Positive = ' + str(np.sum(
    truePositivePerClusterSpleeter)))
truePositivePerClusterSpleeterAxs.set(xlabel='Cluster', ylabel='Total Serendipity Found')
truePositivePerClusterSpleeterAxs.set_xticks(np.arange(5))
truePositivePerClusterSpleeterAxs.legend(loc='upper right')
truePositivePerClusterSpleeterAxs.plot(np.arange(5), truePositivePerClusterSpleeter)
truePositivePerClusterSpleeterFig. savefig ( 'truePositivePerClusterSpleeterFig.png')
```

Appendix G

Relevant Evaluation Session Additional Comments

- Please add a audible tag for each clip. For example a voice that tells you what track it is. Could not do A-B testing between tracks
- Track 6 was hard to hear/close to inaudible.
- It could've been interesting to record the timecode of when the rating happened (in the 30 sec windows), as it seems that I make up my mind pretty quickly. But a few times I changed my mind for better or for worse later in the excerpt.
- I would have appreciate to have the audio files embedded in the Google Form to make it easier, but apart from that good job!
- I think I could have made a decision on each section in 15 seconds.
- Track 40 stuck with me as the one I liked the least. The instrumental music was not that much to my taste but it was okay-ish, that's why I gave it a higher rating in likeliness to listen. The problem is that I understood the lyrics and really hated them.