Smart Shuffle: A Modified Shuffle Algorithm to Improve Genre Flow on a Spotify Playlist

Emily Wilbourn

Monday 2nd May, 2022

1 Abstract

User experience design is used "to make the experience of using a particular product as easy, smooth, and enjoyable as possible for the users" and is critical for securing a customer's loyalty to a business [4]. User experience is especially relevant when a user considers what music streaming service he would like to use. One aspect of the Spotify music streaming platform that could be improved is the Spotify playlist shuffle feature. Currently, Spotify shuffles playlists according to artist name and prevents clusters of songs by the same artist on the playlist, however the documentation of Spotify's algorithm does not account for the flow of song mood and thus can inadvertently create harsh transitions between songs [11]. To improve the user experience of the Spotify shuffle feature, I developed a "Smart Shuffle" algorithm to better sort songs on a playlist. To improve the genre flow of a shuffled playlist, the "Smart Shuffle" creates clusters of songs that have similar moods. I am determining if two songs have similar moods by comparing by the closeness of the "danceability" attribute of each song to reduce the number of harsh transitions between music genres. The danceability attribute is one of many attributes that Spotify collects on each song in its database; these songs attributes define characteristics such as mood by producing scores for the danceability, energy, and tempo of a song, in addition to producing scores for other properties such as loudness, speechiness, and instrumentalness [8]. The codebase I developed to implement the "Smart Shuffle" algorithm takes a playlist and prints out the contents of the playlist in the new order produced from applying the shuffle [18]. The results of applying a "Smart Shuffle" algorithm to a small Spotify playlist were that the new order of songs on the playlist formed clusters of songs that have a similar mood and thus created a smoother transition between song styles. Future work would involve considering additional audio attributes in the algorithm to further improve the quality of transitions between songs, in addition to exploring how this algorithm behaves on a larger playlist.

2 Overview

The purpose of my project was to find a way to improve the flow of genre from song-to-song on a Spotify playlist. A couple boundary conditions for this project include time constraints and machine learning knowledge (for holistically determining genre). One simplification I did was use one audio attribute of a song, I chose "danceability" to be this attribute, to drive the smart shuffle. The number representing the "danceability" of a song would be used to determine if two songs are of a similar genre or not. Ideally in the future, I would like to improve my algorithm to have it consider the overall genre of each song by assigning a numeric value, ideally based on all of its audio aspects, to produce a more accurate shuffle with regards to genre flow.

3 History and Previous Research

Spotify is one of the largest music streaming platforms in the world [2]. As of March 2022, Spotify has 82 million songs on its platform, has 406 million monthly listeners, and has around 8 millions artists and creators on its platform [5]. One of the features that Spotify offers to users is the ability to shuffle the contents of a playlist. Shuffle play is defined to be a mode of music playback that "prevents repeated tracks, which makes it distinct from random playback, in which the next track is chosen at random after the last track has ended" [17]. Originally, Spotify used the Fisher-Yates algorithm to randomize the order of songs when a user clicks the "Shuffle" button for a playlist [11]. The Fisher-Yates shuffle is "an algorithm to generate random permutations" [19]. The Fisher-Yates algorithm shuffles items by taking "time proportional to the total number of items being shuffled and shuffles them in place. The algorithm swaps the element at each iteration at random among all remaining unvisited indices, including the element itself... The algorithm should produce an unbiased permutation, i.e. every permutation is equally likely" [19]. Typically, generating a random permutation of elements with equally likely permutations is used in shuffling a deck of cards, however Spotify used this algorithm as a way to shuffle the order of song playback for a playlist [7]. However due to user feedback, developers at Spotify made the algorithm less random so that songs written by the same artist would not play repeatedly in a short period of time [11]. Fisher-Yates is a completely random algorithm, and since Fisher-Yates could produce a playback order of songs where the same artist could potentially be played 5 or 6 times in a row, users complained that the algorithm was not random enough based on a concept called the "Gambler's Fallacy" [11]. It is false to believe that this lack of variety in artists that were playing back to back was occurring due to a lack of randomness [11]. The Gambler's Fallacy "occurs when an individual erroneously believes that a certain random event is less likely or more likely to happen based on the outcome of a previous event or series of events. This line of thinking is incorrect, since past events do not change the probability that certain events will occur in the future" [20]. Thus, to improve the user experience for the shuffle playback feature, engineers at

Spotify created a shuffle algorithm that prevents songs that are performed by the same artist from repeatedly being played right next to each other. The new Spotify shuffle algorithm forces more variety in the order of music playback from a shuffled playlist. However, this modified shuffle algorithm does not take into account the flow of genre between shuffled songs, so I propose a smart shuffle algorithm. Since genre is tricky to define for a computer, my "Smart Shuffle" algorithm will smoothen out transitions between songs based on a song attribute called "danceability".

4 Methods

The main interest I had was in building an algorithm that successfully shuffled songs based on genre flow between individual songs, which meant that I needed to have a concrete way to define a genre for a song. My initial impression was that each song on a playlist would be assigned an integer that detonated what genre it represented. For instance, if there were three different genres, maybe classical, rap, and pop, contained on a playlist, each genre would be assigned an integer based on how it compares to the other two genres contained on the playlist. Classical and rap music are the obvious polar ends of the scale, and pop music would fall in between classical and rap. Thus, we could assign a 1 to Classical music, a 2 to Pop music, and a 3 to Rap music to demonstrate the relationships between the genres of music contained on this playlist. Once these integers representing genre were assigned to each song, an algorithm that I'm referring to as a "Smart Shuffle" is applied to the list of songs. The Smart Shuffle Algorithm is described in more detail below.

4.1 The Smart Shuffle Algorithm

The Smart Shuffle Algorithm is outlined below. We begin by taking the first and second song records in a data structure that stores the content of a playlist. Then, we define a threshold for determining if two numbers have "danceability" values that are "close". I set this threshold to be the (max danceability factor in playlist)-(min danceability factor in playlist)/2. This value was chosen to ensure that the maximum difference in genre attributes of two songs is not too big or too small. Max danceability factor - min danceability factor was used to ensure that the closeness threshold was not larger than the range of danceability values in the playlist. Find the difference between the danceability values for the first and second songs in our data structure. If the difference is less than our closeness value, add the first song to a new data structure that will hold the contents of the playlist in its order defined by the smart shuffle. However if the difference between the danceability factors is greater than our closeness threshold, meaning that the moods of the first and second songs is too different, then we apply our smart shuffle as follows:

1. Create a deep copy of the container holding playlist elements and remove the first two elements so that we are searching for a new song through the rest of the

contents of the playlist.

2. Define a lower and upper range of values to search for a song with a danceability factor that falls in the closeness threshold with our current song. Meaning, we take the danceability factor of the first song in the data structure, which can also be referred to as the current song we are looking at, and base our range off of this value. We define our lower range value to be the current song's danceability factor - closeness threshold, as long as this value is greater than the minimum danceability factor contained in the data structure. If this value is less than the minimum danceability in the dataframe, then set the lower range value to the minimum danceability. We define the upper range value in a similar fashion.

3. Then, randomly select a song in the playlist that has a danceability factor in the range we previously defined. If there is no song that falls in this range, randomly select a song to play next.

4. Finally, we remove the song we selected to play after the first song from its current position in the playlist and insert it between the first and second song in the playlist. Remove the first song in the playlist from the playlist and add it to another data structure that stores the final contents of the playlist after the smart shuffle has been completed.

4.2 Spotify Shuffle Algorithm

Spotify's algorithm was based on an article called "The art of shuffling music" by Martin Fiedler, which explains a shuffle algorithm that provides a more uniform distribution of elements contained in the container [1,6]. The concept of Fiedler's algorithm is to "split the music collection into multiple logical groups" and then shuffle the tracks within each group and merge all the records back in a way that provides a more balanced shuffle [13]. Spotify's developers say that their algorithm is also similar to the methods used in dithering [11]. If one wanted to simplify a picture with black, white, and gray pixels by using dithering, one could simplify the picture by using only black and white pixels and then use random sampling to randomly decide if each pixel should be black or white based on the shade of gray [11]. Then, one would apply an algorithm that spreads out the black and white spots more evenly to avoid clusters of similar colored pixels [11]. I based my implementation of this algorithm off of pseudocode written at CodeGolf [3]. The algorithm is as follows:

1. Iterate through all the unique artists in a playlist and iterate through each song for a given user.

2. Create a dictionary that maps the track id (the unique identifier for a song) to a random offset, which will ultimately end up being the new index for the song in a shuffle playlist.

3. Create an initial random offset for all songs by a specific artist in the range of (0, 1/number of songs by that artist).

Then, for each song by an artist, get another random value in the range of (-1/10*number of songs by that artist), (1/10*number of songs by that artist). This produces random offsets that are related to the number of songs an artist has performed. Take the contents of the dictionary and add all the songs into a playlist and index them in the order of the random offsets. The values in the dictionary, when sorted from least to greatest, define the order for the songs in the newly shuffled playlist.

4.3 Progression of Development of the Smart Shuffle

My first step in tackling the goal of achieving a smoother transition between shuffled songs on a playlist was creating a small program that applied my smart shuffle algorithm to a list of integers. First, I found the maximum integer value in my "genre levels" list, which stored integers in the range 0-n for the n different genres being stored on the playlist. Then, a threshold for determining if two integers are "close" or not was defined. While the length of the genre levels list was greater than 0, the "current song" genre was defined to be the 0th index in the list of integers. If there was an integer in index 1 of the list, that integer was defined as the genre of the "next song". The current song represents the current song that is being played, and the next song is the song to be played after the current song finishes. Then, the absolute value of the difference between the current and next song genres was taken. If the difference was less than the closeness threshold, the integer was printed to the screen and then removed from the genre levels list. If the difference between the current and next song integers was greater than the closeness threshold, a temporary copy of the genre levels list was created and the first two indices were removed so that the current song and next song were not included in the temporary list. Lower and upper limits were then defined to determine a range in which to find the next song genre value. If an integer in that range can be found, insert this integer into index 1 of the genre levels list so that it becomes the next song. Finally, print out the current song index and remove the current song index from the list and repeat the process until the list is empty.

From here, I decided that I needed to work with some data that was more complicated than a list of integers. I decided to apply the smart shuffle to a list with sublists that had artist names and integers representing genres at each index. At this stage, I implemented the Spotify Shuffle Algorithm and applied both the Spotify Shuffle and then the Smart Shuffle to my data and compared the results. The dataset I used with this codebase only held 11 records, so it was trivial to observe that the difference between every pair of 2 songs was no larger than the closeness threshold that I defined.

At this point, I decided that I needed to investigate how I would go about assigning an integer value to a song that represents the song's genre. I began by researching how genre is defined. When attempting to pinpoint a definition for each specific genre, we run into Moravec's Paradox. Moravec's Paradox states that "activities like abstract

thinking and reasoning or skills classified as "hard" — engineering, maths or art are way easier to handle by machine than sensory or motor based unconscious activities)" [14]. Madars Biss, an electronic music producer, provides tips for learning how to identify music genres: "The best way to get really good at genre identifying is to expand your listening experience. Choose a track, Google what genre it's from, listen to it and make notes about what's distinctive. Repeat this step as much as you can, most preferably across different genres" [12]. Adam Lefaivre and John Z. Zhang outlined how their proposed genre classification could be accomplished through comparing "the differences of the characteristics of two genres in a symmetric manner and using them to classify music genres" [2]. Other researchers online described how they used convolutional neural networks to attempt to classify music genres [15]. However, due to time constraints, neither of these approaches were practical for me. My next thought was to try to use the concept of a word vector to define genres using numbers that define the different attributes of a song [9]. That was then that I discovered that Spotify collects a set of at least 15 attributes of each song contained in its database [8]. These attributes include numbers that define the mood of a song by assigning float values between 0 and 1 to properties that represent the mood of a song like danceability, valence, energy, and tempo [8]. Spotify also stores information about other song attributes, including loudness, speechiness, instrumentalness, and acousticness [8].

As a first step towards discovering how to define a comprehensiveness genre for songs stored in the Spotify database, I decided to apply a smart shuffle to a Spotify playlist and shuffle with regards to the danceability feature of each song. To test this algorithm, I decided to create a small playlist of 15 songs with 4 classical piano songs, 4 electronic dance music (edm) songs, 4 pop songs, and 2 rhythm and blues (r&b) songs). I then used the Spotify for Developers platform to create a Spotify integration to connect to the small playlist that I created [16]. From here, I read in the playlist from spotify using Spotipy. Spotipy is a "lightweight Python library for the Spotify Web API" that provides "full access to all of the music data provided by the Spotify platform" [18]. After extracting the playlist via Spotipy, I separately applied the smart shuffle and the spotify shuffle algorithms to the playlist. However, one difference here was that I now implemented the Fisher-Yates algorithm to randomize the list of songs before I applied the shuffle algorithms. The core algorithm was the same, except at this point I was storing the playlist data in a dataframe, which is contained in the Python Pandas library, to ensure that the artist name, album name, track name, track id, and danceability characteristics of a song stayed together.

5 Results

5.1 Smart Shuffling a List of Integers

This application was successful in providing smaller gaps in each pair of adjacent numbers, as seen in Figures 1-2 [6]. Here, it was defined that each pair of adjacent

integers should be two steps away from each other.

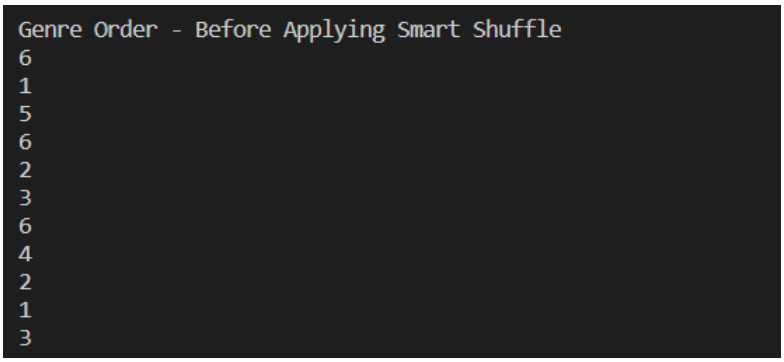

Figure 1: List contents before completing the Smart Shuffle.

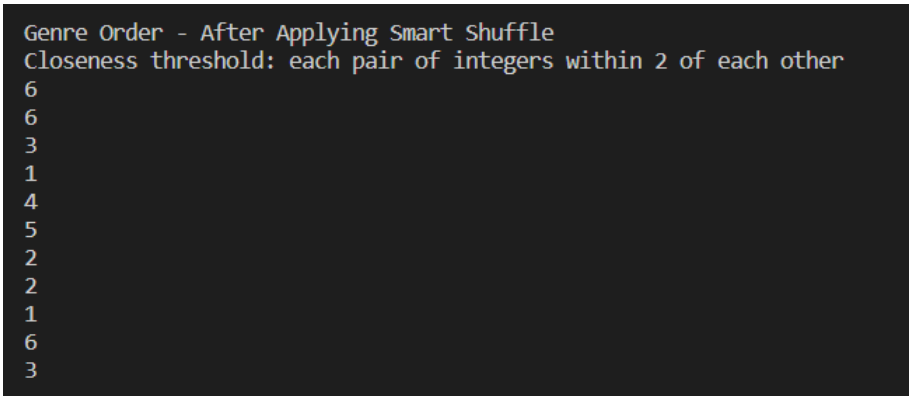

Figure 2: List contents after completing the Smart Shuffle.

5.2 Smart Shuffling a List of Lists

The output from my final code product was a printed list of the original playlist contents, a printed list of the playlist contents after the Fisher-Yates Shuffle, Spotify Shuffle, and Smart Shuffle. By looking at each adjacent pair of songs in the playlist, the Smart Shuffle produced a playlist where the difference in the danceability values of each adjacent pair of songs is mostly not larger than the "closeness" threshold that we defined. When observing the flow of genre in the playlist, I looked at the order of songs and compared them to the genres that I knew them to be. From noting the order of genres on the playlist, as seen in Figures 3-5, it seemed that this Smart Shuffle algorithm resulted in clusters of similar genre songs on the playlist, which results in a playlist with music that has a more logical genre flow than the Spotify Shuffle algorithm [6]. The Smart Shuffle algorithm was successful in improving the genre flow between songs on a playlist.

Additionally, if one were to go through and compare the differences between each pair of songs on the playlist, almost every pair of songs should have a difference in danceability values that is less than that closeness threshold that was previously defined.

```
Playlist Order - Before Shuffle
['A', 5]
['AA', 2]
['AAA', 7]
['B', 9]
['B', 6]
['C', 3]
['CC', 7]
['CC', 1]
['D', 5]
['E', 2]
['EE', 7]
['EEE', 1]
['EEEE', 2]
```

Figure 3: List contents before shuffling.

```
Playlist Order - Spotify Shuffle
['E', 2]
['C', 3]
['AA', 2]
['A', 5]
['EE', 7]
['EEE', 1]
['D', 5]
['AAA', 7]
['CC', 1]
['EEEE', 2]
['B', 6]
```

Figure 4: List contents after completing the Spotify Shuffle.

```
Playlist Order - Spotify Shuffle and then Smart Shuffle
Closeness threshold: current and next song danceability integers within 4 of each other
['A', 5]
['C', 3]
['AA', 2]
['E', 2]
['B', 6]
['EE', 7]
['AAA', 7]
['D', 5]
['CC', 1]
['EEE', 1]
['EEEE', 2]
```

Figure 5: List contents after completing the Spotify Shuffle and the Smart Shuffle.

5.3 Smart Shuffle a Playlist

The output from my final code product was a printed list of the original playlist contents, a printed list of the playlist contents after the Fisher-Yates Shuffle, Spotify Shuffle, and Smart Shuffle. By looking at each adjacent pair of songs in the playlist, the Smart Shuffle produced a playlist where the difference in the danceability values of each adjacent pair of songs is mostly not larger than the "closeness" threshold that we defined.

When observing the flow of genre in the playlist, I looked at the order of songs and compared them to the genres that I knew them to be. From noting the order of genres on the playlist, as seen in Figures 6-9, it seemed that this Smart Shuffle algorithm resulted in clusters of similar genre songs on the playlist, which results in a playlist with music that has a more logical genre flow than the Spotify Shuffle algorithm. The Smart Shuffle algorithm was successful in improving the genre flow between songs on a playlist [6]. Additionally, if one were to go through and compare the differences between each pair of songs on the playlist, almost every pair of songs should have a difference in danceability values that is less than that closeness threshold that was previously defined.

Playlist Before Shuffling:

	danceability	artist	track_name	album
0	0.63	Meghan Trainor	Like I'm Gonna Lose You (feat. John Legend)	Title (Deluxe)
1	0.779	Meghan Trainor	Title	Title (Deluxe)
2	0.257	Paul Ji	Liszt: Concert Paraphrase on Rigoletto, S. 434	Piano
3	0.73	Jonas Blue	Perfect Strangers	Blue
4	0.739	Martin Garrix	Reboot	Sentio
5	0.64	Boyz II Men	End Of The Road	Cooleyhighharmony (Bonus Tracks Version)
6	0.685	Bobby Brown	Rock Wit'cha	Don't Be Cruel
7	0.795	Jonas Brothers	Only Human	Happiness Begins
8	0.498	Bruno Mars	Talking to the Moon	Doo-Wops & Hooligans
9	0.818	Ed Sheeran	Sing	x (Deluxe Edition)
10	0.273	Claude Debussy	2 Arabesques: Arabesque No. 1	Debussy: Clair de lune et d'autres favoris
11	0.411	Claude Debussy	2 Arabesques: Arabesque No. 2	Debussy: Clair de lune et d'autres favoris
12	0.354	Frédéric Chopin	Nocturne in E-Flat Major, Op. 9, No. 2 "Andante"	Longing: Music of Frédéric Chopin
13	0.864	AronChupa	Trombone	Trombone
14	0.672	Jonas Blue	Fast Car	Blue

Figure 6: Playlist contents before shuffling.

Playlist After Applying Fisher-Yates:

	danceability	artist	track_name	album
0	0.818	Ed Sheeran	Sing	x (Deluxe Edition)
1	0.498	Bruno Mars	Talking to the Moon	Doo-Wops & Hooligans
2	0.779	Meghan Trainor	Title	Title (Deluxe)
3	0.411	Claude Debussy	2 Arabesques: Arabesque No. 2	Debussy: Clair de lune et d'autres favoris
4	0.73	Jonas Blue	Perfect Strangers	Blue
5	0.685	Bobby Brown	Rock Wit'cha	Don't Be Cruel
6	0.354	Frédéric Chopin	Nocturne in E-Flat Major, Op. 9, No. 2 "Andante"	Longing: Music of Frédéric Chopin
7	0.739	Martin Garrix	Reboot	Sentio
8	0.257	Paul Ji	Liszt: Concert Paraphrase on Rigoletto, S. 434	Piano
9	0.672	Jonas Blue	Fast Car	Blue
10	0.864	AronChupa	Trombone	Trombone
11	0.273	Claude Debussy	2 Arabesques: Arabesque No. 1	Debussy: Clair de lune et d'autres favoris
12	0.64	Boyz II Men	End Of The Road	Cooleyhighharmony (Bonus Tracks Version)
13	0.795	Jonas Brothers	Only Human	Happiness Begins
14	0.63	Meghan Trainor	Like I'm Gonna Lose You (feat. John Legend)	Title (Deluxe)

Figure 7: Playlist contents after completing the Fisher-Yates Shuffle.

Playlist After Applying Spotify Shuffle:

	danceability	artist	track_name	album
0	0.779	Meghan Trainor	Title	Title (Deluxe)
1	0.864	AronChupa	Trombone	Trombone
2	0.354	Frédéric Chopin	Nocturne in E-Flat Major, Op. 9, No. 2 "Andante"	Longing: Music of Frédéric Chopin
3	0.273	Claude Debussy	2 Arabesques: Arabesque No. 1	Debussy: Clair de lune et d'autres favoris
4	0.411	Claude Debussy	2 Arabesques: Arabesque No. 2	Debussy: Clair de lune et d'autres favoris
5	0.818	Ed Sheeran	Sing	x (Deluxe Edition)
6	0.498	Bruno Mars	Talking to the Moon	Doo-Wops & Hooligans
7	0.795	Jonas Brothers	Only Human	Happiness Begins
8	0.685	Bobby Brown	Rock Wit'cha	Don't Be Cruel
9	0.640	Boyz II Men	End Of The Road	Cooleyhighharmony (Bonus Tracks Version)
10	0.739	Martin Garrix	Reboot	Sentio
11	0.672	Jonas Blue	Fast Car	Blue
12	0.730	Jonas Blue	Perfect Strangers	Blue
13	0.257	Paul Ji	Liszt: Concert Paraphrase on Rigoletto, S. 434	Piano
14	0.630	Meghan Trainor	Like I'm Gonna Lose You (feat. John Legend)	Title (Deluxe)

Figure 8: Playlist contents after completing the Spotify Shuffle.

Playlist Order - Spotify Shuffle and then Smart Shuffle
Closeness threshold: current and next song danceability integers within 0.3035 of each other.

	danceability	artist	track_name	album
0	0.779	Meghan Trainor	Title	Title (Deluxe)
1	0.864	AronChupa	Trombone	Trombone
2	0.818	Ed Sheeran	Sing	x (Deluxe Edition)
3	0.739	Martin Garrix	Reboot	Sentio
4	0.640	Boyz II Men	End Of The Road	Cooleyhighharmony (Bonus Tracks Version)
5	0.354	Frédéric Chopin	Nocturne in E-Flat Major, Op. 9, No. 2 "Andante"	Longing: Music of Frédéric Chopin
6	0.273	Claude Debussy	Arabesques: Arabesque No. 1	Debussy: Clair de lune et d'autres favoris
7	0.411	Claude Debussy	Arabesques: Arabesque No. 2	Debussy: Clair de lune et d'autres favoris
8	0.498	Bruno Mars	Talking to the Moon	Doo-Wops & Hooligans
9	0.795	Jonas Brothers	Only Human	Happiness Begins
10	0.685	Bobby Brown	Rock Wit'cha	Don't Be Cruel
11	0.672	Jonas Blue	Fast Car	Blue
12	0.730	Jonas Blue	Perfect Strangers	Blue
13	0.257	Paul Ji	Liszt: Concert Paraphrase on Rigoletto, S. 434	Piano
14	0.630	Meghan Trainor	Like I'm Gonna Lose You (feat. John Legend)	Title (Deluxe)

Figure 9: Playlist contents after completing the Smart Shuffle.

6 Future Work

Future work would consist of shuffling songs with regards to all aspects of the genre of a song instead of just one attribute. An article by Kaylin Pavlik depicted how classifying genres using Spotify audio aspects is possible [10]. While it may not be fully accurate yet, Pavlik's approach could prove to be valuable to my endeavors to classify music genres on Spotify playlists [10]. Future work would also include investigating how I could improve the runtime of my program. I would be curious to test my code on a large playlist with over 700 songs and compare the time to execute the program between a large playlist and the small 15 song playlist that I used for this project. If the runtime was significantly different, I would explore ways to improve the space and time complexity of my program.

7 Acknowledgements

I would like to thank Andy D. Digh, Ph.D. and Andrew J. Pounds, Ph.D. for their support and mentorship to me in this project.

8 References

[1] Adam Lefaivre and John Z. Zhang. 2018. Music Genre Classification: Genre-Specific Characterization and Pairwise Evaluation. In Proceedings of the Audio Mostly 2018 on Sound in Immersion and Emotion (AM'18). Association for Computing Machinery, New York, NY, USA, Article 22, 1–4. DOI:https://doi.org/10.1145/3243274.3243310

[2] Becca Caddy. 2022. The best music streaming services 2022: Spotify, Apple Music, Tidal and more. Retrieved from https://www.techradar.com/best/the-best-music-streaming-services-2021

[3] Bubbler. 2020. Spotify Shuffle (music playlist shuffle algorithm). Retrieved from https://codegolf.stackexchange.com/q/198094

[4] Chelsea Blake. What is UX and Why is it Important? Retrieved from https://www.workshopper.c
is-ux-and-why-is-it-important

[5] Daniel Ruby. 2022. Spotify Stats 2022 (Facts & Data Listed). Retrieved from https://www.demandsage.com/spotify-stats/

[6] Emily Wilbourn. 2022. Spotipy Danceability. https://github.com/ewilbourn/SmartShuffle.git. (2022).

[7] GeeksforGeeks. 2022. Shuffle a given array using Fisher–Yates shuffle Algorithm. Retrieved from https://www.geeksforgeeks.org/shuffle-a-given-array-using-fisher-yates-shuffle-algorithm/

[8] Hugh Rawlinson. 2018. Spotify Audio Analysis. Retrieved from https://developer.spotify.com/co
audio-analysis/

[9] Jayesh Bapu Ahire. 2018. Introduction to Word Vectors. Retrieved from https://dzone.com/articles/introduction-to-word-vectors#: :text=Word

[10] Kaylin Pavlik. 2019. Understanding + classifying genres using Spotify audio features. Retrieved from https://www.kaylinpavlik.com/classifying-songs-genres/

[11] Lukáš Poláček. 2014. How to Shuffle Songs? Retrieved from https://engineering.atspotify.com/2
to-shuffle-songs/

[12] Madars Biss. 2017. Rhythm Tips for Identifying Music Genres by Ear. Retrieved from https://www.musical-u.com/learn/rhythm-tips-for-identifying-music-genres-by-ear/.

[13] Martin Fiedler. 2007. The art of shuffling music. Retrieved from http://keyj.emphy.de/balanced-shuffle/

[14] Mirek Stanek. 2017. Moravec's paradox. Retrieved from https://medium.com/@froger_mcs/mor
paradox-c79bf638103f.

[15] Parul Pandey. 2018. Music Genre Classification with Python. Retrieved from https://towardsdatascience.com/music-genre-classification-with-python-c714d032f0d8

[16] Samantha Jones. 2021. Extracting Your Fav Playlist Info with Spotify's API. Retrieved from https://www.linkedin.com/pulse/extracting-your-fav-playlist-info-spotifys-api-samantha-jones/

[17] Shuffle Play. https://educalingo.com/en/dic-en/shuffle-play

[18] Spotipy. 2014. Spotipy Library Documentation. Retrieved from https://spotipy.readthedocs.io/e

[19] TechnieDelight. 2016. Shuffle an array using Fisher–Yates shuffle algorithm. Retrieved from https://www.techiedelight.com/shuffle-given-array-elements-fisher-yates-shuffle/).

[20] Will Kenton. 2021. Gambler's Fallacy. Retrieved from https://www.investopedia.com/terms/g/g