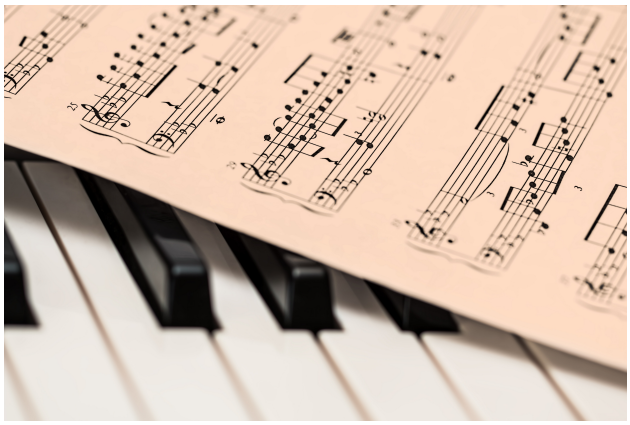
matsubara
Case-based Recommendation for Music Playlists

Alexander Stradnic



Prof. Derek G. Bridge
Prof. Rosane Minghim

Final Year Project, April 2023
BSc. Computer Science
University College Cork

Abstract

In this Final Year Project I present the use of Case-based Recommendation in creating music playlists, along with providing an implementation of this CBR system and examples of output. Playlists are sequences of songs arranged in a particular order. Using this information, context can be gained from previous playlists in order to build a new playlist from a given seed song. Patterns in previous playlists can be analysed and used to create an enjoyable and coherent listening experience.

Declaration of Originality

In signing this declaration, you are conforming, in writing, that the submitted work is entirely your own original work, except where clearly attributed otherwise, and that it has not been submitted partly or wholly for any other educational award. I hereby declare that:

- this is all my own work, unless clearly indicated otherwise, with full and proper accreditation;
- with respect to my own work: none of it has been submitted at any educational institution contributing in any way to an educational award;
- with respect to another's work: all text, diagrams, code, or ideas, whether verbatim, paraphrased or otherwise modified or adapted, have been duly attributed to the source in a scholarly manner, whether from books, papers, lecture notes or any other student's work, whether published or unpublished, electronically or in print.

Signed: Alexander Stradnic
Date: 12th April 2023

Contents

Chapter 1

Introduction

1.1 Music Listening in the Pre-Industrial Era

Throughout most of history, the only way for a person to listen to a piece of music was to either attend a live performance, or perform themselves. While the written recording of music as notation has existed for hundreds of years, the majority of the population would not have the education and time to be able to read and reproduce this music, aside from the nobility who would have court musicians writing and performing music for them. W.A. Mozart, one of the most well-known composers today, had to travel the courts of Europe after leaving Salzburg due to low pay and the desire to compose more freely; he battled with financial insecurity throughout his life. Turlough O'Carolan was a famous blind harper who travelled across Ireland composing planxties for patrons.

This also generated a rift between the organised, written music of the elites with professional performers, compared to more accessible folk/traditional music of the populace which could be performed solo or by a group, and was passed down by ear.

1.2 Early Sound Reproduction

Attempts to reproduce the sound of music have been attempted and improved as technology has advanced, beginning with spring-wound machines in which a metal pin strikes a series of raised pitches on a rotating cylinder or disc. Music boxes are the most well-known, and reproduce a single melody line, though more complex contraptions automatically playing flutes, violins and keyboard instruments also existed, and player-pianos continue to exist, though less common and usually digitally operated.

The issue with these machines is that they would play exactly as inscribed, and with simplistic musicality. They tended to be large, expensive and limited to few tracks.

1.3 19-20$^{\text{th}}$ Century Innovations

With the development of early cylinder-based phonographs, followed by disc-based gramophones, allowed the widespread recording of music to occur. Gramophone recording would inscribe the sound vibrations onto a master record, from which copies would be printed onto shellac-based, and later polyvinyl-based discs. These discs were analog recordings of actual artist performances and improved in clarity with advances in materials and recording techniques. Acoustic recording gave way to electronic with microphones allowing more control and flexibility. Later on, magnetic tape recording was invented, and while initially only used in studio recording, eventually allowed for a small, portable and rugged music format to be developed in the form of the cassette, which started being produced from the 1960s and became more popular into the '70s.

Towards the end of the 20$^{\text{th}}$ Century, digital solutions to storing and playing music were explored, with the Compact Disc exploding onto the scene in 1982, effectively replacing any mar-

ket share records had remaining, and competing with compact cassettes, replacing them for audio playback in the 2000s.

1.4 Music Ownership, Albums and The Computer Age

Until the rise of computers and the Internet, the way in which people listened to recorded music was starkly different. One would listen typically hear a song on the radio or live, and, if they wished to listen to it repeatedly, would visit the record store and purchase a copy of the song on vinyl, cassette, or CD. Cassettes also allowed for radio streams to be recorded.

They could also purchase a collection, typically an album on the medium of their choice, which would tend to be better value. In many styles of music such as pop and rock, collections of pieces or songs would be released together by an artist in albums. Music albums, initially inspired by photo albums, typically comprise a collection of tracks which tend to cover a theme or idea, and are intended by the artist to be listened sequentially. Even in other genres, music would be grouped and sold, length depending on the format and album. This allowed for longer uninterrupted listening before the listener would have to change the disc or cassette manually or with a suitable player.

The practice of "owning" your own music would continue to remain popular with many people until the advancement of Internet infrastructure and services. Apple pioneered the digital sale of music online, allowing songs to be purchased individually, to be played on MP3 players such as the iPod. This broke with the previous paradigm of selling and playing by the album. People now had the ability to create their own custom collections of songs called playlists. At this point, most people still consciously which songs to purchase, and gathered their own personal collec-

tions on their own. However, with the rise of Napster and later Spotify, this model of music ownership became less popular, as people wished to listen to more music without having to purchase every single song they wanted to listen to. Apple iTunes also started facilitating streaming, leading us to today, where most stream music from a platform such as YouTube, iTunes, or Spotify.

1.5 Modern Problems Require Modern Solutions

This rise in music streaming, along with the ability to purchase individual songs for a cheap price, has meant that listening to albums from start to finish drastically reduced in popularity. Many people listen to playlists or just the individual songs themselves as shown in these surveys carried out by Deezer[D320] and MusicBiz[LM16].

A major convenience offered by music streaming providers nowadays is the automatic generation of playlists, making it easier for users to listen to a variety of music without having to create their own playlists manually. This is now done using many methods ranging from popularity counts (seen in many services as the "Hot" or "Trending" pages) to personalised user-user and user-item algorithms, which take into account previous user listening activity and the preferences of "similar" users. Thus the problem: *Which methods generate the most interesting playlists for users?* In this paper, I explore various ways of generating music playlists, focusing on case-based recommendation.

Chapter 2

General Implementation Details

2.1 Data

The data was acquired from Spotify's Million Playlist Dataset. Other options explored were MusicBrainz and the Million Song Datasets, which have various metadata tags available for songs and artists, but were dropped due to lack of source playlists.

Processing was done on the dataset running in SQLite. The data was downloaded from Spotify's servers, structured as a set of JSON files. This was then converted into an SQLite database to improve lookup performance. Alternatively, this set of files could have been sequentially processed into Pandas and converted into CSV or Pickle files. Additional data was queried from Spotify's API.

2.2 Tables

- `tracks` – specific track info, song name, album name, etc.
- `track_features` – generic track features as measured by Spotify such as 'acousticness', 'danceability', 'speechiness', etc.

- `artist_genres` – artists and their associated genres
- `playlists` – playlist information
- `playlist_tracks` – link between tracks and playlists
- `seq2_simple` – all sequences of $\langle song1, song2 \rangle$, usually aggregated in processing
- `similarities` – artist similarity rating based on their genres

2.3 Processing

The implementations are written in *Python*. This is due to the large amount of useful libraries available for the language such as *Numpy*, as well as the quick writing and readability.

2.4 Output

All algorithms described in this paper take in an input seed track URI, and output a generated list of track URIs.

`(spotify:track:1, spotify:track:2,..., spotify:track:`λ`)`

2.5 Limitations: Song Output

Due to the nature of Case-based Recommendation, it can only generate playlists containing songs that are already present in the seed dataset, thus limited to songs up to 2017. This is because it ranks songs rather than a generic set of features, and even an output comprising a list of feature sets would be limited to the dataset due to the inability of querying Spotify by features.

The similarity-based approach likewise only works for pre-computed similarities in the dataset.

2.6 Limitations: Personalisation

Personalising songs and genres based on a Spotify User Profile is technically possible by preferring user playlists and song features over generic playlists, it was beyond the scope of this project, requiring user authentication, processing and further development of the demonstration web-app[10.5]. User-user recommendation and weighting is also theoretically possible, but would require manual input of other users as Spotify does not have a public API for getting "similar users".

Chapter 3

Similarity-based Solutions

3.1 Background

A logical conclusion one would come to when asked how to generate a playlist given an input song would be to create a playlist of similar songs. In fact, there are many websites that generate playlists based off a song such as Chosic, Spotalike and SimilarSongFinder. These tend to be used by users to discover new artists and songs.

3.2 Simple Similarity

One method to create playlists is to simply choose the most similar song to the given song. The next most similar song to that song, aside from any songs already in the playlist, is then added. This process can be continued until the playlist of desired length is generated.

3.3 GLSF

An interesting solution utilising similarity is the Global-Local Similarity Function[Che+22]. In this approach, a playlist is generated in which the transition between tracks is constrained by

the difference in track features between the songs in order to prevent a hard transition between two songs. The transitions are measured by scoring functions and songs with scores in the desired range are chosen.

3.4 Conclusion

Similarity rating is a simple but powerful concept. It has many applications and can be used as a building block in more advanced algorithms or be the core of an algorithm as with GLSF. In this project I chose to implement a simplistic version of similarity, mainly as a reference against the various Case-based configurations[6].

Chapter 4

Similarity-based Development

4.1 Design

Based on the data available, I decided to design the algorithm to output a playlist with songs chosen randomly from an artist not already in the list. While I wanted to work based on the songs themselves, the artists actually contained the features, so the algorithm works using artist similarity. The seed song's artist is looked up and the most similar artist is chosen. A random song by that artist is chosen. The most similar artist to the previously chosen artist not already in the list is then chosen and the process is repeated until a playlist of desired length is achieved.

Jaccard Similarity

$$Sim(\boldsymbol{s_1}, \boldsymbol{s_2}) = \frac{|\boldsymbol{s_1} \cap \boldsymbol{s_2}|}{|\boldsymbol{s_1} \cup \boldsymbol{s_2}|}$$

4.2 Implementation

The implementation queries a table which contains the similarity score between all artists' genres and chooses the highest ranked

artist to the initial song's artist. It then continues as described in Design, outputting a list of tracks.

Another implementation utilising similarity between artist tags along with genres was written for the Million Song Dataset, but does not output Spotify URIs.

Chapter 5

Similarity-based Testing

5.1 Examples

A standard playlist of length 10.

1	**TOTO**	**Hold the Line**
2	Dary Hall & John Oates	You've Lost That Lovin' Feeling
3	Fleetwood Mac	Got to Move
4	Meta Loaf	Good Girls Go To Heaven
5	Dire Straits	Telegraph Road
6	Foreigner	Growing up the Hardway
7	Journey	Lights
8	Boston	You Gave up on Love (2.0)
9	Styx	Light Up
10	Kansas	The Pinnacle

Figure 5.1: `https://open.spotify.com/playlist/4qShHvmJIaFat9wV2yTgUK`

A playlist of length 20, 11–19 are skipped for brevity.

1	**George Michael**	**Careless Whisper**
2	Cher	Fires of Eden
3	Mandy Moore	Want You Back
4	Hannah Montana	Don't Wanna Be Torn
5	Zac Efron	Ladies' Choice ("Hairspray")
6	Kesha	Blow – Deconstructed Mix
7	Demi Lovato	Confident Outro/For You Intro
8	Christina Aguilera	When You Put Your Hands on Me
9	Liam Payne, Quavo, Nevada	Strip That Down – Nevada Remix
10	Hailee Steinfeld, Grey, Zedd, KillaGraham	Starving – Killagraham Remix
...	...	...
20	Ashley Tisdale	Acting Out

Figure 5.2: `https://open.spotify.com/playlist/1ocr27MzrILNfgpHznMVKn`

5.2 Compute Time

Playlists of length 10 were created, with the time to return a result for each captured below. The time to create a playlist based on similarity took on average 18 seconds.

TOTO	Hold the Line	18.002
George Michael	Careless Whisper	17.947
Jonathan Parecki	Miiro (From "KanColle: Kantai Collection")	18.006
Britney Spears	Toxic	17.971
PSY	Gangnam Style	17.959

Figure 5.3: List of seed songs and times to compute playlists

5.3 Conclusion

Similarity allows for a quick way to generate playlists from a seed song but may be inconsistent in the quality due to the random choice of song from each artist. Artists tend to produce more than one type of music, but unfortunately many songs were missing genre descriptors.

Chapter 6

Case-based Solutions

6.1 Background

Case-based solutions extrapolate sequences and patterns from input user data. The outcome of this analysis is then used when it is queried to generate a new output. There are many applications of case-based recommendation, from advising select products based on price and features to recommending restaurants and services[Smy07].

6.2 Application to Playlist Generation

As demonstrated in the paper by Baccigalupo and Plaza[BP06], it is possible to adapt case-based reasoning to generate playlists in various forms. In this project I design a case-based algorithm using user-generated playlists from Spotify, and expand on work done prior by testing various settings of parameters.

Chapter 7

Case-based Design

7.1 Case Base

The set of playlists may be filtered to remove meaningless and/or noisy playlists, forming the Case Base. They are then analysed, using their song order as well as the songs themselves, in order to come up with new playlists to generate from an initial seed song.

7.2 Retrieval of Playlists

From this Case Base of playlists $\mathcal{C}$, two main values are computed for each playlist:

- The Attribute Variety: How varied and diverse the playlist is, based on song attributes
- The Coherence of the playlist: How relevant the sequences are to the seed song, as well as how many

After computing the Variety and Coherence for each playlist $p \in \mathcal{C}$, a rating function combines both and allows the playlists to be ordered.

$$\rho(p, s) = Var(p) \cdot Coh(p, s), \forall p \in \mathcal{C}$$

7.3 Reuse

After the k playlists have been ranked, they have to be then converted in some way to form a single playlist of a particular length λ. This is to be done using a process called Constructive Adaptation[PA02], which comprises of two parts:

1. Hypothesis Generation(HG): How partial solutions are extended from the currently generated state of the playlist (either the seed song at the beginning or the highest ranked generated sequence chosen by HO afterwards)
2. Hypothesis Ordering(HO): Uses the ranking of sequences from $\mathcal{C}$ to rank the list of newly generated playlists after each iteration of HG

An ideal solution is generated by appending or prepending a song at a time to the seed song, decided by creating potential successor solutions using Hypothesis Generation and ranking using Hypothesis Ordering. This is continued recursively down the most ideal node. However, if no successor solution is generated at a particular level, then it is discarded from the list of ranked generated sequences, and the process continues from that point, until a playlist of length λ is generated.

7.4 Formulae Used[BP06, pp. 5–10]

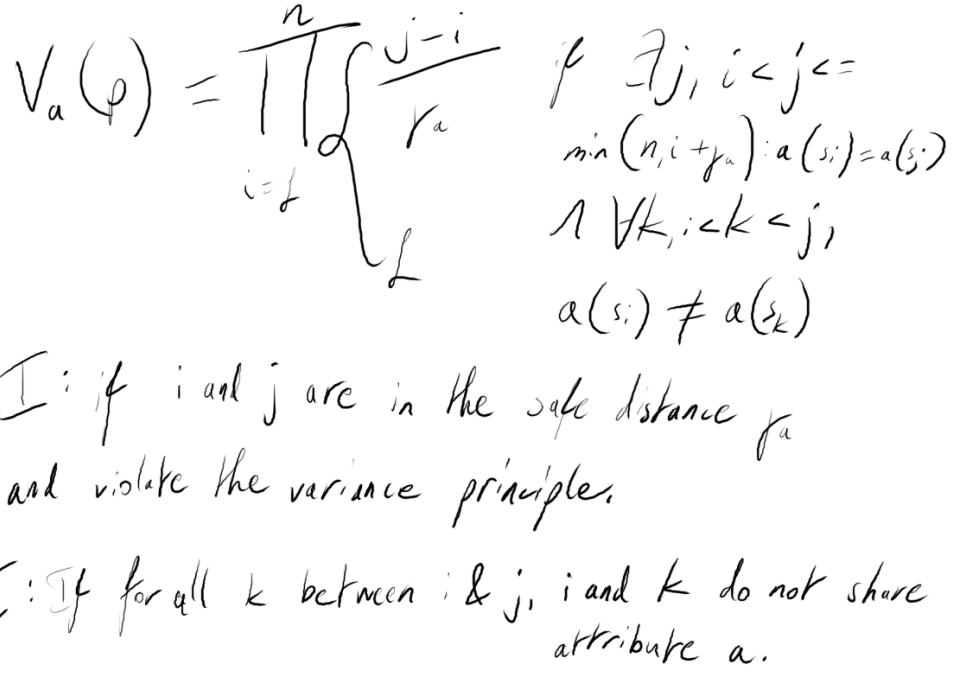
Variety The variety of each playlist in $\mathcal{C}$ is calculated based on the repetition of song features within a set distance. This is first calculated per attribute and then each attribute's variety

is combined together:

$$V_a(p) = \prod_{i=1}^{n} \begin{cases} \frac{j-i}{\gamma_a} & \text{if } \exists j, i < j <= \min(n, i+\gamma_a) : a(s_i) = a(s_j) \\ & \wedge \forall k, i < k < j, a(s_i) \neq a(s_k) \\ 1 & \text{otherwise} \end{cases} \tag{7.1}$$

If no value of a is repeated within the safe distance γ_a, then $V_a(p)$ is 1. Otherwise it is some value between 0 and 1. Combined variety for all attributes $a \in \mathcal{A}$ for playlist p:

$$Var(p) = \prod_{a \in \mathcal{A}} V_a(p)$$



Coherence The coherence of each playlist in $\mathcal{C}$ is calculated based on how related it is to the input song. In order to calcu-

late the Coherence for each playlist, the Relevance is first computed. The Relevance is determined by the amount of times the sequence q appears in the playlists that make up $\mathcal{C}$, adjusted for the biases of length and song popularity, and returns the degree in which q is a relevant pattern for song t.

$$Rel(q,t) = \phi(q) \cdot \frac{\alpha^{\theta - \Lambda(q)}}{\psi^{\beta}(q,t)}$$

The Coherence is then calculated from the sum of all these relevant patterns in a particular playlist.

$$Coh(p,s) = \sum_{q \in \Omega(s,p)} Rel(q,s)$$

Hypothesis Generation The nodes generated at each stage are sequences of songs from the k playlists, and consist of $T = (t_1, t_2, ..., t_n)$ songs. T' is a successor playlist, thus it contains all $t \in T$, along with an additional song u. This song is either prepended or appended to the list of songs. Thus $T' = \langle u + T \rangle$ or $\langle T + u \rangle$.

Considering the first case, $\langle u, t_1 \rangle$ should appear at least once in the set of k retrieved playlists, in order to be coherent. Similar is done in the second case, where $\langle t_n, u \rangle$ should appear at least once in k. Thus the set of all successors would be all combinations of these two rules done recursively.

Hypothesis Ordering The sequences are ordered by computing their Relevance and Variance, similar to how the k playlists were chosen.

$$H(T') = Rel(T', u) \cdot Var(T')$$

Where T' is the successor list and u is the added song. However, in order to determine whether future evolutions of T' will also be varied and coherent, a look-ahead parameter is added.

Look-ahead Heuristic (L)

- L = 0: The above formula is applied as is
- L = 1: H is performed on all the successors of T', and the maximum value found is returned
- L = 2: H is performed on *all the successors* of all the successors of T' (depth 2), and the maximum value is returned

As with many such parameters, a trade-off between precision and performance has to be chosen, as computing cost increases exponentially with L.

Chapter 8

Case-based Implementation

8.1 Retrieving Playlists

I assume all playlists have already been curated of bad or noisy data, and take the full dataset as the Case Base $\mathcal{C}$.

To begin, all playlists featuring the seed song are selected, with the others discarded. Only the playlists containing the seed would have a Coherence rating, since only they would have relevant patterns to the seed. As only sequences of length 2 only are analysed, the Relevance Function is simplified.

The sequences' ϕ values are calculated by using a `COUNT()` aggregate method, summing up the occurrences of each sequences in `seq2_simple`. These are balanced for popularity depending on the assignment of β, and the relevances are returned. The playlists' variances are then calculated based on the Δ and γ values given. Weighting variance is done by calculating $1 - Var(p)$ for a balancing variable *balance*. This difference is then multiplied by a value in $[0, 1]$, and added to variance. A parameter, `var_weight` is supplied, which is an inverse of the figure multiplied by *balance*, thus a high weight $\rightarrow$ less altered or unaltered variance.

After all ratings are calculated, the top k playlists are chosen.

8.2 Generating with Reuse

A `while` loop is opened, and remains open until the playlist reaches the requisite length, or the program fails.

Hypothesis Generation is implemented by creating a list of all possible successors of the current playlist, either prepending or appending songs to T from the k lists. This is done for each potential song u by checking whether the sequence of $\langle u, t_1 \rangle$ or $\langle t_n, u \rangle$ exists in the set of k playlists.

Hypothesis Ordering is done by ranking these partial playlists with ratings calculated using the same parameters as in the above section. If no successors were generated in Hypothesis Generation, then the current playlist is discarded and the highest ranked successor from the previous state is chosen, and HG is executed on that list.

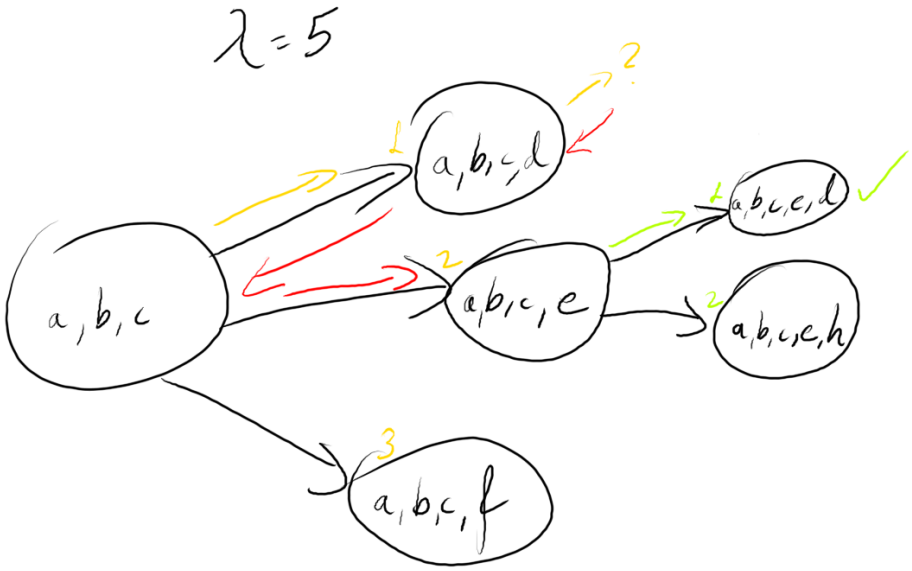


Figure 8.1: The highest rated successor to $\langle a, b, c\rangle$ is $\langle a, b, c, d\rangle$. However as it has no successors, the algorithm continues from $\langle a, b, c, e\rangle$, returning a playlist of length λ.

8.3 Optimisations

In order for the algorithm to run in a reasonable amount of time, a series of optimisations were added.

In retrieving the playlists from which ratings were calculated, songs directly before the seed were grouped into lists up to the SQLite limit of 1,000 (I use groups of 900), with songs directly after the seed grouped into another list of lists. These lists were then run in parallel using Python's *multiprocessing* library, using a Pool and asynchronous maps to fetch all tracks concurrently.

```
130     containing = cur.execute(f"SELECT pid, pindex FROM playlist_tracks WHERE track_uri = '{seed}';").fetchall()
131     totals = []
132
133     prev_cases = [f"(pid = {c[0]} AND pindex-1 = {c[1]})" for c in containing]
134     prev_split = []
135     for x in range(0, len(prev_cases), 900):
136         prev_split.append("OR".join(prev_cases[x:x+900]))
137
138     after_cases = [f"(pid = {c[0]} AND pindex+1 = {c[1]})" for c in containing]
139     after_split = []
140     for x in range(0, len(after_cases), 900):
141         after_split.append("OR".join(after_cases[x:x+900]))
142
143
144     # print("fetching tracks...")
145     fetched_prev_tracks = []
146     fetched_after_tracks = []
147     with Pool(32) as p:
148         fetched_prev_tracks = p.map_async(fetch, prev_split)
149         fetched_after_tracks = p.map_async(fetch, after_split)
150         p.close()
151         p.join()
```

Figure 8.2: The beginning of `main()`. Playlists containing the seed are retrieved, the adjacent songs are aggregated in SQL queries, and fetched in parallel with `Pool()` and `map_async()`.

These concepts of batch processing and asynchronous function calls from the main function are used throughout for calculating relevancies, variances, successors, successor ratings and more.

Chapter 9

Case-based Testing

I kept the seed song constant with *TOTO – Hold the Line* being selected. It is a popular song, with 4,707 hits in the playlist database, which allowed for a large variety of outputs when testing popularity balancing.

9.1 Properties

- k input playlists from which the final playlist is constructed, default of 20
- β popularity weighting – ranges between 0 (no popularity balancing) and 1 (full balancing), default of 1
- v variety weighting – ranges between 0 (no variety) and 1 (full variety rating is multiplied with coherence)
- Δ difference between two songs' features being compared in the variety function, usually a multiple of the feature's standard deviation. This was calculated on the dataset in advance for each feature used
- γ interval in which songs are compared for variety, set to 3 for all tests

9.2 Examples: No Variety

When popularity and variety were disabled, playlists would form a characteristic group of songs from the same artist surrounding the original seed song, eventually changing to another artist, with that artist featuring a few songs in a row. This is due to the large amount of sequences in the dataset playlists where consecutive songs from the same artist would feature one after another. Thus this configuration creates a set of popular songs from a few similar artists.

$$\{k = 20, \beta = 0, v = 0\}$$

1	The Edgar Winter Group	Free Ride
2	**TOTO**	**Hold the Line**
3	TOTO	Rosanna
4	TOTO	Africa
5	Journey	Any Way You Want It
6	Journey	Faithfully
7	Journey	Separate Ways (Worlds Apart)
8	Kansas	Carry on Wayward Son
9	Kansas	Dust in the Wind
10	Rick Springfield	Jessie's Girl

Figure 9.1: `https://open.spotify.com/playlist/31o7NXO6d63JjJjCZLbqzQ`

With only variety disabled but popularity enabled, a playlist of similar songs is created, but less popular songs are featured more.

$$\{k = 20, \beta = 0.5, v = 0\}$$

1	Toby Keith	Courtesy of the Red, White and Blue (The Angry American)
2	Brooks & Dunn	She's Not The Cheatin' Kind
3	Brooks & Dunn	Hillbilly Deluxe
4	Mark Chesnutt	It Sure Is Monday
5	Mark Chesnutt	Bubba Shot The Jukebox
6	OMFG	Hello
7	Walker Hayes	You Broke Up With Me
8	**TOTO**	**Hold the Line**
9	Aerosmith	Dream On - Live Version
10	Wild Cherry	Play That Funky Music

Figure 9.2: `https://open.spotify.com/playlist/1xp5G3XH4TIEK1V5giaXDl`

9.3 Examples: Variety Tuning

From this early test it was clear that I had made the delta for penalising similarity too large. The strength of the variety function shifted the playlist from the initial TOTO rock song, to dance-pop with Pitbull, to finally Scottish Pipes. It then continued to recommend Scottish Pipes due to their relatively low popularity, but also being the only tracks in sequence to each other.

$$\{k = 20, \beta = 0.5, v = 1, \Delta = 2\mu\}$$

1	I Corvi	Silvia's Mother
2	**TOTO**	**Hold the Line**
3	Queen	Bohemian Rhapsody
4	Dave Edmunds	I Hear You Knocking
5	Usher, Lil Jon, Ludacris	Yeah!
6	Pitbull, Ne-Yo	Time of Our Lives
7	Golden Earring	Radar Love
8	The Scottish Bagpipes Highland Pipes	Amazing Grace
9	The Scottish Bagpipes Highland Pipes	Amazing Grace Bagpipe Solo
10	The Scottish Bagpipes Highland Pipes	Scotland the Brave

Figure 9.3: `https://open.spotify.com/playlist/4AE8JviFihfX7U9BN5800h`

With an adjustment of the variety delta to 1μ, the range of variety was lowered to a more reasonable level. The popularity balancing allowed for a more diverse cast of artists to feature while not completely discarding similar and popular songs.

$$\{k = 20, \beta = 0.5, v = 1, \Delta = 1\mu\}$$

1	Simon & Garfunkel	The Sound of Silence – Acoustic Version
2	Timeflies	Worth It
3	TOTO	Africa
4	**TOTO**	**Hold the Line**
5	Arjun Kaul	Can't Fight This Feeling
6	Boston	More Than a Feeling
7	Bryan Adams	Summer Of '69
8	Kansas	Dust in the Wind
9	Queen	Bohemian Rhapsody
10	Samuel Barber, Leonard Bernstein, New York Philharmonic	Adagio for String, Op. 11

Figure 9.4: `https://open.spotify.com/playlist/1uToCokTDqze4UBbloNy9d`

1.5μ offers perhaps a more exciting playlist than 1μ, if not straying a bit too far from the seed track.

$$\{k = 20, \beta = 0.5, v = 1, \Delta = 1.5\mu\}$$

1	Don McLean	American Pie
2	I Corvi	Silvia's Mother
3	**TOTO**	**Hold the Line**
4	TOTO	Africa
5	Simon & Garfunkel	The Sound of Silence – Acoustic Version
6	Bee Gees	Stayin' Alive
7	Simon & Garfunkel	Cecilia
8	John Travolta, Olivia Newton-John	You're The One That I Want – From "Grease"
9	Simon & Garfunkel	Mrs. Robinson – From "The Graduate"
10	Boney M.	Rasputin

Figure 9.5: `https://open.spotify.com/playlist/1usMuMwjrNruBFoLeqx0zr`

9.4 Examples: Changing k

Increasing the amount of k input playlists to 50 returns a playlist with more popular tracks, as these songs which previously did not make the list with only the top 20 playlists now have the opportunity to appear, even if their playlists are less relevant to the seed.

$$\{k = 50, \beta = 0.5, v = 1, \Delta = 1\mu\}$$

1	Edwin Starr	War
2	Commodores	Brick House
3	Thin Lizzy	The Boys Are Back In Town
4	Kansas	Carry on Wayward Son
5	Kansas	Dust in the Wind
6	Journey	Any Way You Want It
7	TOTO	Africa
8	**TOTO**	**Hold the Line**
9	TOTO	Rosanna
10	Orquestra Discotheque, Beto Montes	Tú

Figure 9.6: `https://open.spotify.com/playlist/1UKuGg9ZADwF9zGAyAupsC`

Altering β to 1 reduces the power of popular songs yet again, but perhaps reduces it too much...

$$\{k = 50, \beta = 1, v = 1, \Delta = 1\mu\}$$

1	I Corvi	Silvia's Mother
2	**TOTO**	**Hold the Line**
3	Arjun Kaul	Can't Fight This Feeling
4	Boston	More Than A Feeling
5	Harry Nilsson	Without You
6	Scorpions	The Future Never Dies
7	Scorpions	House of Cards
8	Scorpions	Humanity
9	REO Speedwagon	Take It On The Run
10	Iron Maiden	Fear of the Dark

Figure 9.7: `https://open.spotify.com/playlist/4SWteaUeuATt9kPKS7KoUf`

Fine-tuning β further yields the following list:

$$\{k = 50, \beta = 0.75, v = 1, \Delta = 1\mu\}$$

1	I Corvi	Silvia's Mother
2	**TOTO**	**Hold the Line**
3	Arjun Kaul	Can't Fight This Feeling
4	Boston	More Than A Feeling
5	Bryan Adams	Summer Of '69
6	Kansas	Dust in the Wind
7	Queen	Bohemian Rhapsody
8	Elton John	Rocket Man (I Think It's Going To Be A Long Long Time)
9	Earth, Wind & Fire	September
10	Marvin Gaye	Let's Get It On

Figure 9.8: `https://open.spotify.com/playlist/6922qeZ91GFbkAzycgHhPG`

9.5 Examples: Other Seeds

Generating a playlist from a less common seed song, $v = 0.5, \beta = 0.5$ returns a nice list of cover songs. What is more interesting, however, is the fact that an identical list was generated from setting $v = 1$, $\beta = 1$, or both parameters to 1. Even changing the amount of input playlists to 50 did not change the output. It seems that there is not much choice in candidate songs and thus the same songs are chosen, even with different parameters for popularity and variance.

$$\{k = 20/50, \beta = 0.5/1, v = 0.5/1, \Delta = 1\mu\}$$

1	Jonathan Parecki	Hacking to the Gate (from "Steins;Gate")
2	**PianoDreams**	**Gates Of Steiner - Steins;Gate OST**
3	Eddie van der Meer	Believe Me [Steins;Gate]
4	RMaster	Resuscitated Hope (from "Gosick")
5	Kenzie Smith Piano	This Game (from "No Game No Life")
6	Kenzie Smith Piano	Aqua Terrarium (from "Nagi no Asukara")
7	Kenzie Smith Piano	Ebb and Flow (from "Nagi no Asukara")
8	Eddie van der Meer	My Dearest [Guilty Crown]
9	Eddie van der Meer	Hacking To The Gate (OP from 'Steins;Gate')
10	Eddie van der Meer	Main Theme Slow [Log Horizon]

Figure 9.9: `https://open.spotify.com/playlist/2LxN8uMTPnLc7iPFTs4rNW`

Generating a playlist from a Brazilian song results in a fairly diverse and localised playlist, however some English songs are able to make it into the playlist. Then it returns back to Portuguese, which implies that English songs are present and common in the foreign-language input playlists.

$$\{k = 20, \beta = 0.5, v = 1, \Delta = 1\mu\}$$

1	**Zeca Baleiro**	**Bandeira**
2	Detonautas Roque Clube	Olhos Certos
3	Charlie Brown Jr.	Zóio De Lula
4	Leoni	Garotos II - O Outro Lado
5	Vanessa Da Mata	Amado
6	Pitty	Equalize
7	Legião Urbana	Ainda É Cedo
8	Caetano Veloso, Maria Gadú	Nosso Estranho Amor
9	The Chainsmokers, ROZES	Roses (feat. ROZES)
10	Paula Mattos, Fernando Paloni	Que Sorte A Nossa

Figure 9.10: `https://open.spotify.com/playlist/5iuJd0P5MCO3IToDCXcQ1x`

Recommendations for K-Pop, seems to have favours popular artists, thus a more stringent β may improve the diversity.

$$\{k = 20, \beta = 0.5, v = 1, \Delta = 1\mu\}$$

1	Red Velvet	Dumb Dumb
2	BLACKPINK	PLAYING WITH FIRE – KR Ver.
3	BLACKPINK	BOOMBAYAH – KR Ver.
4	BLACKPINK	WHISTLE – KR Ver.
5	BTS	I Need U
6	BTS	Dope
7	**BTS**	**Burning Up (Fire)**
8	BTS	Epilogue: Young Forever
9	EXO	Ko Ko Bop
10	EXO	Power

Figure 9.11: `https://open.spotify.com/playlist/0qdvzm5HzE66hSGy28VZ45`

9.6 More Playlists

- PSY — Gangnam Style – $\{k = 20, \beta = 0.75, v = 1, \Delta = 1\mu\}$, **Very slow execution thanks to constant node failures**
- PSY — Gentleman – $\{k = 20, \beta = 0.5, v = 1, \Delta = 1\mu\}$
- Stuttgart Chamber Orchestra/Antonio Vivaldi — Four Seasons – $\{k = 20, \beta = 0.5, v = 1, \Delta = 1\mu\}$

9.7 Compute Time

The time taken to construct playlists using the case-based algorithm varied ranging from just a few minutes with playlists such as *Gate of Steiner*[9.9] up to 20 minutes or more with *Hold the Line*[9.6], especially with $k = 50$. The compute time is proportional to amount of songs gathered from the retrieved playlists, and takes longer when backtracking from nodes that fail to produce full lists.

Jupyter Notebooks

Writing the implementation for Case-based learning was mostly done in a Jupyter Notebook, which ran on *Windows Subsystem for Linux*. It was observed that playlists took far longer to generate in the Jupyter Notebook under WSL, even for cases with no node discarding and $k = 20$, compared to a version rewritten as a Python file with the same optimisations[8.3].

9.8 Conclusion

Case-based recommendation is a highly configurable approach to generating playlists, allowing for a user to choose whether they

want a list of songs that are mainly similar, popular, diverse, or a combination of all three, by tweaking the available parameters.

Chapter 10

Future Work

10.1 Further Testing and Development

Continued testing and work on areas such as:

- Implementing the Look-ahead factor described in Case-based Design[7]
- More intelligent node handling for edge cases where constant node traversal slows down the algorithm immensely, an example of which was running *PSY – Gangnam Style*[9.6]. Could be done with caching node length results, and potentially improve performance with implementations of $L > 0$
- Adding artist genres as a feature in the Variety Function
- Beginning from a list of seed songs and retrieving the combined top ranked playlists for Reuse
- Adding an element of randomness when choosing the next song to be added to the list (weighted on ranking or rating of each prospective track)
- Adding length 3 sequences
- Enhancing similarity-based generation with track feature analysis

10.2 Surveys

In this project I used my own intuition and logical reasoning when determining the strengths and weaknesses of the various playlist generation parameters. A way to improve this further would be to open the algorithms to the public and conduct surveys to get more opinions. Ultimately, people's tastes in music are subjective, and some may prefer any of the given options rather than simply the most popular. The variety of options gives users a freedom of choice in determining the style of playlist they desire, in contrast to many services which tend to be a simple button.

10.3 Additional Algorithms

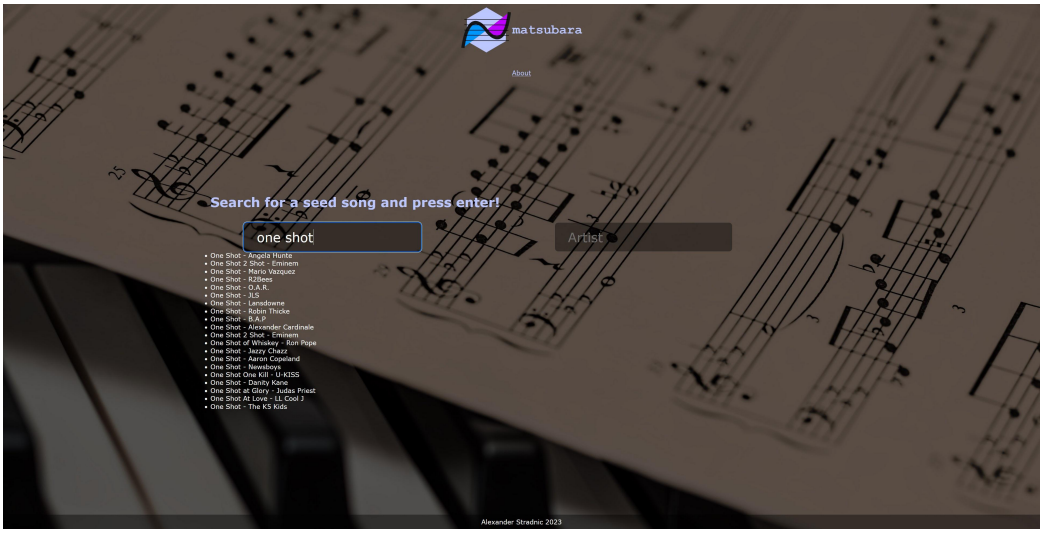
Other styles of algorithms such as machine learning approaches could be explored and compared to the case-based configurations, taking the input playlists as the training set. These would most likely benefit from using the Pandas approach to data processing rather than SQLite.

10.4 Addressing Limitations

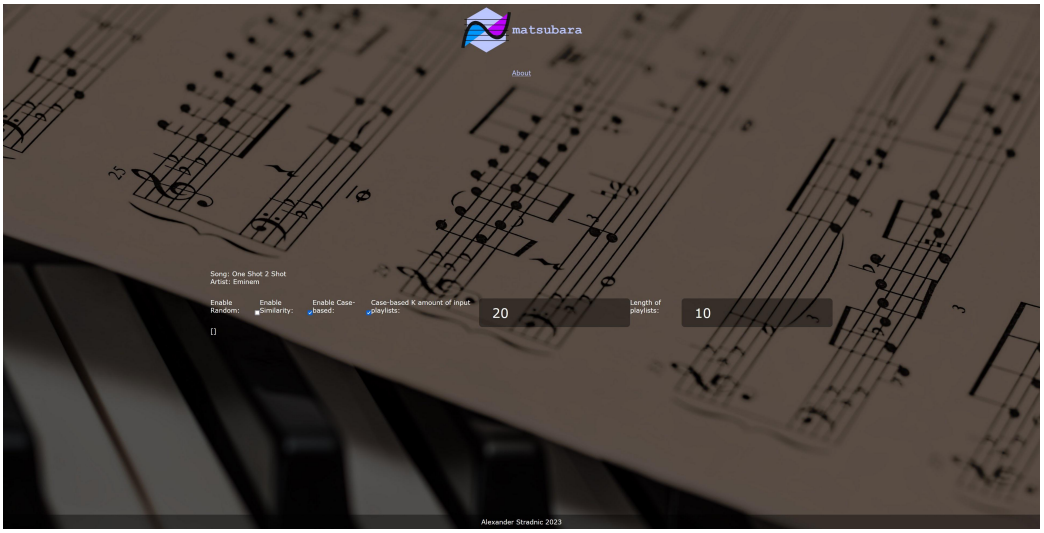
The song[2.5] and personalisation[2.6] limitations detailed earlier could be solved or mitigated with further development. By querying a different service which offers more recent playlists, the limitation in song output would be solved. Personalisation would require the analysis of at least the user's account for user-song recommendation, and more users for user-user recommendation, and could be implemented with Spotify or alternatives.

10.5 Web Application

Current Implementation



A webapp was developed for the open day and showcases a potential application of the project. A user chooses their desired song from the list returned after querying a title and/or artist name and is then taken to a screen where they can configure different settings and options, such as the length of the desired playlist and K amount of input playlists to be factored in. More settings could also be added here. Multiple algorithms could be configured with different settings.



Further Development

Unfortunately the main case-based algorithm was not able to output an answer in time due to connection timeout which I did not have the time to solve, but the premise still works and similarity-based playlists are able to be generated and returned to the client in the form of a list of playlists, each consisting of song URIs.

Song previews could be queried from Spotify on the website, and if a user likes any of the playlists, they could log into their Spotify account and add them to their profile.

10.6 Browser Extension

Instead of developing a separate website that users have to travel to separately from Spotify, an extension could be developed which a user would simply install, and would modify the Spotify web client, adding a tab, or simply as a popup like standard browser addons, allowing the user to do the same as above but be more convenient and even reading context from Spotify. The

dataset and program could be downloaded with the addon, allowing local generation instead of pinging a server.

Bibliography

[PA02] Enric Plaza and Josep-Lluís Arcos. "Constructive Adaptation". In: *Advances in Case-Based Reasoning*. Ed. by Susan Craw and Alun Preece. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 306–320. ISBN: 978-3-540-46119-7. DOI: 10.1007/3-540-46119-1.

[BP06] Claudio Baccigalupo and Enric Plaza. "Case-Based Sequential Ordering of Songs for Playlist Recommendation". In: *Advances in Case-Based Reasoning*. Ed. by Thomas R. Roth-Berghofer, Mehmet H. Göker, and H. Altay Güvenir. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 286–300. ISBN: 978-3-540-36846-5. DOI: 10.1007/11805816.

[Smy07] Barry Smyth. "Case-Based Recommendation". In: vol. 4321. Jan. 2007, pp. 342–376. ISBN: 978-3-540-72078-2. DOI: 10.1007/978-3-540-72079-9_11.

[LM16] LOOP and MusicBiz. *Playlists Overtake Albums in Listenership, Says LOOP Study*. 2016. URL: https://musicbiz.org/news/playlists-overtake-albums-listenership-says-loop-study/.

[D320] Deezer and 3Gem. *Over half admit to listening to less albums in last five years*. 2020. URL: https://www.deezer-blog.com/press/over-half-admit-to-listening-to-less-albums-in-last-five-years/.

[Che+22] Haonan Cheng et al. "Global-Local Similarity Function for Automatic Playlist Generation". In: *2022 IEEE International Conference on Multimedia and Expo (ICME)*. 2022, pp. 01–06. DOI: 10.1109/ICME52920.2022.9859699.

Reference Links

- Spotify Million Playlist Dataset – https://engineering.atspotify.com/2018/05/introducing-the-million-playlist-dataset-and-recsys-challenge-2018/
- Spotify Web API Docs – https://developer.spotify.com/documentation/web-api

- MusicBrainz – https://musicbrainz.org/
- Million Song Dataset – http://millionsongdataset.com/

- Python Documentation – https://docs.python.org/3.11/index.html
- SQLite Documentation – https://www.sqlite.org/doclist.html
- Numpy Documentation – https://numpy.org/doc/stable/index.html#numpy-docs-mainpage/
- Vue.js Documentation – https://vuejs.org/guide/introduction.html
- Golang Documentation – https://go.dev/doc/

- Sheet Music on Keys Image – https://www.pexels.com/photo/chords-sheet-on-piano-tiles-210764/