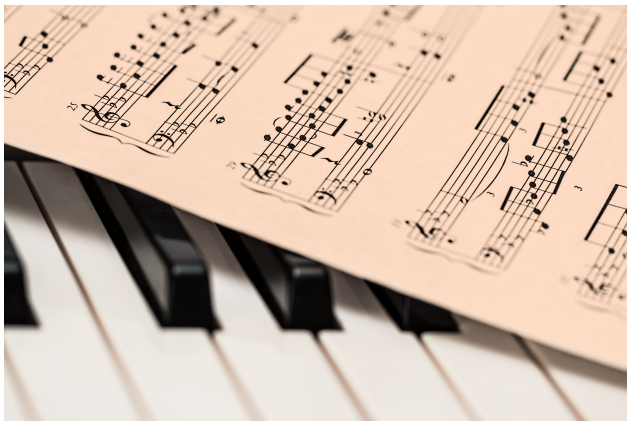
matsubara
(EXTENDED ABSTRACT)

Alexander Stradnic



Final Year Project on the Use of Sequential Ordering for Creating Playlists

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 or starting list. Patterns in previous playlists can be analysed and used to create an enjoyable and coherent listening experience. This paper mainly makes reference to the paper below[1]. Other research and ideas were drawn from [2] and [3].

Chapter 1

Summary

My project's goal is to work on recommender systems in order to create music playlists, focusing on Case-based Recommendations.

Case-based Recommendation (CBR): The approach in which a playlist is generated from a seed song/shorter playlist using sequential patterns learned from a dataset of existing playlists. CBR is a method of recommendation which can be used in contexts in which a meaningful order or sequence to objects is desired or useful. There can also be a large variety of possible values (such as songs in this case).

1.1 Case Base

A set of playlists may then be filtered to remove meaningless and/or noisy playlists, forming the Case Base. The playlists 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.

1.2 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.

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

1.3 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(references [2] in [1]), 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 but 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.

Chapter 2

Datasets

The data was acquired from Spotify's Million Playlist Dataset. 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 performance and maintain consistency with earlier developed similarity-based functions which used a shortened collection from MusicBrainz called the Million Song Dataset.

Tables

1. `tracks` – specific track info, song name, album name, etc.
2. `track_features` – generic track features and information such as genre, danceability, etc.
3. `playlists` – playlist information
4. `playlist_tracks` – link between tracks and playlists

Chapter 3

Implementation Details

3.1 Processing

The implementation is written in *Python*. This is due to a large amount of useful libraries available for the language such as *Pandas* and *Numpy*, as well as for quick writing and readability. The playlists are queried from the tables above and are then passed through the CBR, similarity-based and pseudo-random algorithms, returning a few lists of Spotify `track_uri` strings, which can then be sent in a request to Spotify in order to create playlists for an authenticated User.

`(spotify:track:1, spotify:track:2, spotify:track:3, spotify:track:4,..., spotify:track:`λ`)`

3.2 Formulae Used

[1, pp. 5–10]

Jaccard Similarity This was used to precompute the similarity between different songs based on their attributes. This can then be used to create a playlist based purely on track similarity.

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

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:

$$V_a(p) = \prod_{i=1}^{n} \begin{cases} \frac{j-i}{\gamma_a} & \text{if } \exists j, i < j <= \min(n, i+\gamma_a) : a(s_i) = a(s_j) \\ & \wedge \forall k, i < k < j, a(s_i) \neq a(s_k) \\ 1 & \text{otherwise} \end{cases} \tag{3.1}$$

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 calculate 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, $u + t_1$ 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 $t_n + u$ 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.

- 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 4

Testing

A variety of CBR systems will be tested with varying weightings of Coherence and Variance, along with the simpler pseudorandom and similarity-based algorithms. These will then be ranked out of 10 by survey, with top ranked lists passed back into the dataset.

Bibliography

[1] 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.

[2] 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.

[3] 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.