UM MÉTODO GERAL PARA GERAÇÃO AUTOMÁTICA DE PLAYLISTS DE MÚSICA

MARCOS ALVES DE ALMEIDA

UM MÉTODO GERAL PARA GERAÇÃO AUTOMÁTICA DE PLAYLISTS DE MÚSICA

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Renato Martins Assunção
Coorientador: Pedro O. S. Vaz de Melo

Belo Horizonte
Abril de 2019

MARCOS ALVES DE ALMEIDA

A GENERAL METHOD TO AUTOMATICALLY GENERATE MUSIC PLAYLISTS

Dissertation presented to the Graduate Program in Computer Science of the Federal University of Minas Gerais in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: Renato Martins Assunção
Co-Advisor: Pedro O. S. Vaz de Melo

Belo Horizonte
April 2019

Ficha catalográfica elaborada pela Biblioteca do ICEx - UFMG

Almeida, Marcos Alves de.

A447g A General method to automatically generate music playlists / Marcos Alves de Almeida — Belo Horizonte, 2019.
xxiv, 77 f. il.; 29 cm.

Dissertação (mestrado) - Universidade Federal de Minas Gerais – Departamento de Ciência da Computação.

Orientador: Renato Martins Assunção
Coorientador: Pedro Olmo Stancioli Vaz de Melo

1. Computação – Teses. 2. Recuperação da Informação. 3. Sistemas de informação musical. 4. Lista de reprodução - música. I. Orientador. II. Coorientador. III. Título.

CDU 519.6*73(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

A General Method to Automatically Generate Music Playlists

MARCOS ALVES DE ALMEIDA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

[illegible]
Prof. Renato Martins Assunção - Orientador
Departamento de Ciência da Computação - UFMG

[illegible]
Prof. Pedro Olmo Stancioli Vaz de Melo - Coorientador
Departamento de Ciência da Computação - UFMG

[illegible]
Profa. Olga Nikolaevna Goussevskaia
Departamento de Ciência da Computação - UFMG

[illegible]
Prof. Flavio Vinicius Diniz de Figueiredo
Departamento de Ciência da Computação - UFMG

[illegible]
Prof. Nazareno Ferreira de Andrade
Departamento de Sistemas e Computação - UFCG

Belo Horizonte, 22 de Abril de 2019.

To my dad, my mom, my sister and my Godparents.

Acknowledgments

I'd like to thanks my dad, Antônio Eustáquio, for all the support and help to achieve my dreams. Without him, I wouldn't be completing this work. I also thank my mom, Sônia Maria, for teaching me the importance of the study, and my sister, Mariana, for being an example of dedication to studies. I feel grateful for having them as my family.

A special dedication is given to my Godparents, Marcio Chula and Maria Aparecida Chula, for all the love they have for me. A love that I always fell it's impossible to reciprocate. I will always love and be grateful for all the affection I receive from them.

I also want to thank all the friends I made during this journey. Friends I made in my hometown Divinópolis, friends from the Computational Mathematics course, friends from other courses of the university, friends I made during the Master degree, friends I made practicing Kung Fu. Friends that were and I know will always be there when I need. Friends that gave me the strength to keep fighting for my dreams. Friends that, when I was far from home, became my family. Friends I will take with me for the rest of my life. It's impossible to cite all the names here since all of them deserve a special thanks. But I would like all of them to feel grateful.

Last, but not least, I want to thanks my advisor, professor Renato Martins Assunção, and my co-advisor, professor Pedro Olmo Stancioli Vaz de Melo, for helping and guiding me during this work. I learned a lot with them, and will always be grateful for the knowledge they have transmitted to me. A knowledge that I will take with me for the rest of my life.

A lot of people went through my life, helping and supporting me to achieve my goals. Words are not enough to express my gratitude for all of them. Therefore, I leave here my simple thanks for all of them.

Thank you.

“If I have seen further it is by standing on the shoulders of Giants.”
(Isaac Newton)

Resumo

Música é uma das formas de entretenimento mais utilizadas por pessoas do mundo todo. Diferente de outros tipos de entreterimento como filmes e teatro, música é consumida por meio de playlists, isto é, várias músicas são agrupadas antes que sejam escutadas. Organizar as músicas em uma sequência é uma tarefa que demanda tempo, e pode requerer conhecimentos específicos de quem está criando as playlists. O objetivo deste trabalho é propor um método geral para gerar automaticamente playlists de música satisfazendo objetivos conflitantes. Inicialmente, nós iremos analisar playlists de música de usuários com o objetivo de entender suas características e gêneros musicais. Em seguida, iremos propor formas de calcular a similaridade entre músicas utilizando características acústicas e metadados. As funções de similaridade propostas serão utilizadas para mapear as músicas em um espaço de músicas onde músicas similares estão próximas uma das outras. Então iremos propor um método geral para gerar automaticamente uma playlist aleatória de música conectando duas músicas definidas pelo usuário. Baseado no método geral, iremos construir dois algoritmos para gerar playlists de música, chamados de ROPE e STRAW, e aplicá-los nos espaços de música construídos. Com os experimentos realizados, nós mostramos que os algoritmos propostos conseguem gerar playlists aleatórias de músicas heterogêneas com transições suaves entre as músicas. Finalmente, um protótipo online é desenvolvido para permitir usuários testarem o método proposto.

Palavras-chave: Recuperação de Informações Musicais, Mapeamento de Músicas, Gerador de Playlists.

Abstract

Music is one of the most used forms of entertainment, being consumed by people all over the world. Different from other types of entertainment such as movies and plays, music is consumed in playlists, that is, several tracks are grouped together before the users listen to them. Arranging the songs in a sequence is a time-consuming task, and may require specific knowledge from the playlist creator. The objective of this work is to propose a general method to automatically generate music playlists satisfying conflicting goals. First, we will analyze users' playlists in order to understand their characteristics and music genres. Next, we will propose methods to calculate the similarity between songs using acoustic characteristics and metadata. The proposed similarity functions will be used to embed the songs in a music space, where similar songs are close to each other. Then, we will propose a general method to automatically generate a random playlist of songs connecting two anchor songs defined by the user. Based on the general method, we will construct two algorithms to generate music playlists, named ROPE and STRAW, and apply them to the constructed music spaces. With the experiments carried out, we showed the proposed algorithms are able to generate random heterogeneous music playlists with smooth transitions between songs. Finally, an online prototype is developed to allow users to test the proposed method.

Keywords: Music Information Retrieval, Music Embedding, Playlist Generator.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Music has been present in our lives for a long time, found in every known civilization. It's difficult to determine the origin of music, but it probably began at the stone age as a form of culture expression. Although there are some divergence about the oldest music instrument[Diedrich, 2015], flutes made from bird bones dated around 40000 years ago are believed to be produced by *Homo sapiens*[1]. Music has been evolving through the time, going through the Medieval, Renaissance, Baroque, Classical and Romantic periods, being considered a piece of art together with words, as in songs, and with physical movements, as in dance.

The way we listen to music has also changed over ages. If in the past it was necessary to go to orchestras or concerts, with the advance of technology we started listening to music through other methods. At the beginning of the twentieth century, Thomas Edson developed the phonograph cylinder, an equipment able to record and reproduce sounds. Through time, other technologies have arisen, like the gramophone, vinyl LP, cassette tape, compact disc, and MP3 players. Today, with the growing of music streaming services such as Spotify or Pandora, we can listen to music on our phones, computer, tablet or smart TVs anywhere and at any moment.

Music is not only art but also an entertainment and has many other benefits, such as educational and therapeutic (as in music therapy, which uses music to prevent and support mental problems). It is present in almost every moment of our lives, as we listen to music at bars, restaurants, while driving, working out, or even resting at home. Music can also impact our psychosocial development [Miranda, 2013] and has the ability to influence our emotions and bring memories back. Just listening to a song we used to listen to when we were young brings back to our mind event and feeling of that time.

[1]https://www.bbc.com/news/science-environment-18196349

1.1 Motivation

With the arrival of digital music and advance of music stream services like Spotify and Groove Music, millions of songs are available to anyone, and any user can listen to their favorite almost instantly. Spotify, for example, provides more than 30 million songs to its users[2]. This digitalization and ease of access to a huge collection of songs come with many other problems, such as music organization, automatic music genre classification and music recommendation, and solving them has been demanded to a better iteration with such media. For instance, automatically classifying songs by its genre makes it easy to organize and retrieve similar songs, and recommending songs to users helps them discover new songs they may enjoy[Kamehkhosh et al., 2018].

Another problem we face with this large amount of available songs is creating an enjoyable playlist. Manually selecting the songs to add to the playlist demands time, and may require advanced knowledge from the creator. Jannach et al. [2014], for example, showed that, on users' created playlists, the tracks of the second halves seem to be slightly less carefully selected than those of the first halves. Because of that, methods to automatically generate music playlists have become essential to aid users in the task of selecting the appropriate songs to add to the playlist.

The problem of music recommendation is tied to automatic playlist generation, but they should be treated separately. When recommending songs to a user, it is done in form of playlists, as the consumption of music is faster than other media like movies. But the playlists generated should follow specific criteria. While some users have a well-defined music taste, others have an eclectic one and enjoy songs of several different genres. Recommending playlists to them is a challenge since users generally don't enjoy abrupt transitions between songs on a playlist. We should not recommend a classic song right after a rock one, for instance. Users also like to listen to songs by different artists and genres, so they can discover new artists they may enjoy. In this case, recommending a heterogeneous playlist is ideal.

According to Shao et al. [2009], the key to a successful music recommendation is to develop a good measurement of how similar two pieces of songs are and use it as an effective recommendation method. In other words, if we have an accurate similarity function between songs, we can use it to retrieve songs similar to the ones a user enjoys, as described by Ben-Elazar et al. [2017]. Another importance of a good similarity function between songs is discussed by Pohle et al. [2005], who proposed to reorder the songs in a playlist in an attempt to group similar songs and avoid abrupt transitions between tracks.

[2]https://www.digitaltrends.com/music/apple-music-vs-spotify/

Collaborative-filtering and context-based recommendations are two approaches also widely used for music recommendation tasks [Shao et al., 2009]. The idea behind these approaches is that if two items are consumed together, they may be similar to each other. Although achieving good results in practice [Barrington et al., 2009], such techniques face some problems. One of them is that they can overestimate the similarity of popular songs since they have a higher probability of appearing in playlists [Goussevskaia et al., 2008]. Another problem faced by these methods is the cold start problem, that is, when we have a new item (or user), we need to wait for it to be consumed by (or consume) other items before performing a recommendation.

1.2 Objectives

The objective of this work is to propose a general method to automatically generate music playlists comprising different songs. First, we will study users' playlists with the aim to understand the different music styles they have. With this analysis, we want to determine if users have an eclectic taste, enjoying songs of various genres, or if they are focused on few genres. We will represent a playlist by the genres it contains, and study the distribution of such representation. Then we will identify the music genres that are present in almost all playlist, and the rare ones, present in only a few playlists.

After the users' playlists analysis, we will study forms to calculate the similarity between songs: one using acoustic characteristics of the songs such as timbre and harmony, and another using metadata song characteristics like the artists' genres and their co-occurrence in users' playlists. These similarity functions will be used to construct a music space with a well-defined distance between them. This music space should be coherent, that is, close points on it should represent songs that are similar.

We will then propose a general method to automatically generate music playlists by walking through a music space. The proposed method should generate random playlists, so the user obtains a different one every time he/she uses it with the same input. Also, it should be able to generate playlists with a smooth transition between songs, at the same time the playlist may contain songs with diverse genres, satisfying the users' taste. Based on the general method we will propose two algorithms, named ROPE and STRAW, that connect seed songs defined by a user. These playlists generators should be fast, independently of the size of the playlist and the database.

1.3 Contributions

The main contributions of this dissertation are:

1. An analysis of user's music playlists, and the characterization of their listening profiles. This characterization allows us to better understand how different users listen to music, and how to better generate and recommend personalized music playlists to them.

2. The proposal of methods to measure how similar two songs are. This similarity function is important to cluster similar songs, recommend songs to users based on other songs they enjoy, and to generate playlists with smooth transitions.

3. The proposal of a general method to generate music playlists to users, whether their tastes are homogeneous or heterogeneous, and two algorithms based on the general method. These algorithms are important to satisfy all kind of users, including those that enjoy distinct music genres, and to satisfy a group of users with diverse tastes that share the same ambient at a given moment.

With the results obtained during this research, the following papers were published:

The Fast and Winding Roads that Lead to The Doors: Generating Heterogeneous Music Playlists, published in ACM Proceedings of the 23rd Brazillian Symposium on Multimedia and the Web in 2017 by the authors Marcos A. de Almeida, Carolina C. Vieira, Pedro O.S. Vaz de Melo, Renato M. Assunção.

Random Playlists Smoothly Commuting Between Styles, published in Transactions on Multimedia Computing Communications and Applications in 2019 by the authors Marcos A. de Almeida, Carolina C. Vieira, Pedro O.S. Vaz de Melo, Renato M. Assunção.

We also created an online prototype of one of our proposed algorithms to allow users to test and generate music playlists. The prototype receives as input from the user two anchor songs and the desired number of songs in the playlist, and allow the user to save and listen to the generated playlist on Spotify system.

1.4 Organization

The rest of this work is organized as followed: in chapter 2 we describe some of the song characteristics most used for calculate music similarity and generate playlist, and review some of the most relevant related works. In chapter 3 we are going to present the datasets we will use through this work. The study and characterization of the users will be presented in chapter 4. In chapter 5 we will describe ways to calculate the similarity between songs. These similarity functions will be used on the playlist generation algorithms that will be described in chapter 6. In chapter 7 we will present the experiments performed with the proposed algorithms and present the prototype implemented, followed by the conclusions and future works in chapter 8.

Chapter 2

Literature Review

We can find in the literature many works about music information retrieval, such as how to extract characteristics from audio and automatically classify songs according to the genre. In this chapter, we are going to review works related to the problem of music similarity and automatic playlist generation. Before diving into the related works, we will outline some qualities a playlist generator algorithm should satisfy. Then we are going to review the approaches most used to compute the similarity between songs based on the audio characteristics and metadata which is important to generate coherent playlists, and satisfying the users' taste. Finally, the most relevant proposed algorithms to generate music playlists will be explored.

2.1 Characterizing Songs

To quantify and characterize a sound is not a simple task. Analyzing the digital representation of a song and extracting representative features of the audio requires specific algorithms and possibly music theory knowledge. In physics, sound is defined as a vibration that typically propagates as an audible wave of pressure. In psychology, the term sound has a different meaning, referring to how the brain perceives it. Based on those definitions, there are some measures used to describe and analyze a sound, such as frequency, amplitude, pitch, and loudness. These characteristics are classified as content-based features. Besides acoustic characteristics, songs can be characterized by context-based features such as the genre of the song and the year and country it was composed. Some authors use the term metadata to refer to context-based features.

2.1.1 Content-based Characteristics

In the music theory field, there are characteristics used to describe the sound and how we perceive it. These characteristics are used to inform the differences and similarities between pieces of songs. Although they are mostly qualitative, there are algorithms that try to quantify them by analyzing the music audios' wave.

According to Knees and Schedl [2013], the idea behind content-based approaches is to extract information directly from the audio signal of the music using signal processing techniques. When it comes to content-based characteristics, the most used in the literature is the Mel Frequency Cepstral Coefficients (MFCCs)[1], since, as described by Muda et al. [2010], are based on how human ear perceives the sound. MFCCs are spectral-domain audio features for the description of timbre (a characteristic that allows us to differentiate music instruments) and are routinely used in speech recognition and Music Information Retrieval (MIR) tasks [Mauch et al., 2015].

Another form of representing the acoustic characteristics of a song is using the chroma features. Chroma features is a representation for music audio in which the entire spectrum is projected onto 12 bins representing the 12 distinct semitones (pitch classes) of the musical octave, and can characterize music harmony [2].

2.1.2 Context-based Characteristics

Besides acoustic characteristics, songs can be characterized by metadata features. Different from the content-based characteristics, these features describe general information about the audio, such as the rhythm and music instruments present on it. Some of the context-based characteristics presented in literature are:

- **Genre of the song:** this feature can describe acoustic characteristics such as the beats, the pace of the song, and the instruments that may or may not be present. For example, a Hip Hop song may imply fast beats, while a blues song may have the absence of drums.

- **Year of release:** This feature informs the freshness of the song. It can indicate some characteristics of the audio since even specific genres (e.g. rock) changes over time. For example, when listening to a rock song from The Beatles we observe some differences when compared to a rock song from Guns N' Roses.

[1]https://medium.com/prathena/the-dummys-guide-to-mfcc-aceab2450fd

[2]https://labrosa.ee.columbia.edu/matlab/chroma-ansyn/

- **Country of origin:** Since music is sometimes described as cultural expression, different cultures produce songs with different characteristics. Knowing the country of origin can indicate some audio characteristics of the sound. Sometimes, these characteristics are described in the genre of the song, such as Brazilian rock.

- **Text description:** we can describe a songs by its lyric or using a text description. For example, we can use the biography of an artist, or his/her information on a web page such as Wikipedia to describe the pieces of music he/she produces. This description may cover information described by the other content-based features, such as the music genre or the country of origin.

2.2 Playlist Generator Qualities

When creating playlists, different users desire different characteristics of their playlists depending on the situation they are going to listen to it. For example, some users may enjoy a surprising playlist, containing songs he doesn't know, while others may wish a heterogeneous playlist containing a wide variety of different music genres. Several authors have proposed characteristics a playlist generator algorithm should have. One of the characteristics pointed by authors is how users interact with the playlist generator. There are two main forms of interaction found in the literature. One is automatic, where the system creates the entire playlist based on some desired characteristics specified by the user. The other form is assisted, where the user continually interacts with the system selecting the songs to be added to the playlist given a list of suggested ones. Although some users may enjoy manually creating a playlist [Kamalzadeh et al., 2012], Kamehkhosh et al. [2018] showed that recommending songs to be added to a playlist may cause the task to be more difficult, besides requiring from the user constant interaction with the system, which can be time consuming.

The transitions and coherence of the playlists is also an important characteristic to be considered. Jannach et al. [2014] studied the adjacent songs of a playlist and showed that some users tend to group tracks from the same artist. One explanation for this behavior is that users attempt to keep smooth transitions between the songs. Even though it is important to generate playlists where adjacent songs are similar to each other, creating a playlist with heterogeneous songs may be required by users. As argued by Dias et al. [2017], a balance between homogeneity and diversity is important to satisfy listeners.

Another characteristic considered by authors is the popularity of the songs added to the playlist. Although some authors indicate users prefer to listen to familiar songs

[Dias et al., 2017], Kamalzadeh et al. [2012] showed in some situations serendipity may be an important factor. Lastly, the proposed method should be fast, independently of the size of the playlists and the database. Although one may argue the playlist can be generated while the user listens to the first song, it is desirable to know beforehand the songs that will be played, so the user can change some of them or create another playlist in the case he doesn't enjoy the generated one.

Based on the desired characteristics outlined by previews authors, we indicate five qualities a playlist generator method should have in order to satisfy some of the users need. These qualities are:

- *interaction*: it should be simple and easy to the user interact with the system and define what playlist he wants to listen to. In other words, the generator should require only a few parameters as input.
- *smooth transition*: the generator should be able to create playlists where consecutive songs are similar to each other, in order to keep smooth transition between tracks.
- *heterogeneity*: the generated playlist should be as heterogeneous as the user desires, going through diverse genres.
- *novelty*: the generator should be random, being able to generate different playlists given the same input.
- *scalability*: the generator should be fast and scale with the size of the playlist and the database.

Proposing an automatic playlist generator that satisfy these five quality constraints is a challenging task since some of the objectives are conflicting. For example, promoting the smooth transition quality may generate homogeneous playlists.

2.3 Song Similarity

To compute the similarity between songs is not a simple task, and many authors have proposed several approaches to solve this problem using both content-based and on context-based characteristics. As described by Knees and Schedl [2013], on one hand, computing the similarity between songs using content-based information extracted directly from the audio gives a more precise similarity measure, but it requires the audio file of the song and also music theory knowledge to correctly understand the features.

On another hand, context-based features don't require the audio file but may contain some redundant or incorrect information. For instance. when an artist produces a song with a rhythm different from his/her style (in a duet, for example), assigning to the song the tags of the artist may cause wrong information.

When it comes to content-based characteristics, authors have proposed several acoustic features to describe a piece of music [Casey et al., 2008]. Even though this variety of features exists, most of the authors use the MFCC since it has shown good performance in practice. Logan and Salomon [2001], for example, proposed to use K-means clustering on histograms of MFCC to characterize songs, and compared the characterization using Earth Mover's Distance. Aucouturier and Pachet [2002a] decided to model the MFCCs using Gaussian Mixture Model (GMM) instead of K-means clustering and pointed out possible applications of this method, including playlist generation. Pampalk et al. [2005b] proposed to combine MFCC features with three other information based on Fluctuation Patterns, showing the additional information improves the performance on the problem of genre classification.

Using context-based characteristics to calculate the similarity between songs is also widely used by authors since there is no need to have access to the audio files to extract these features. Knees and Schedl [2013] separated this set of characteristics in three groups: text-based, co-occurrence and user rating. Text-based approaches concern with genre tags, web pages, and song lyrics. One of the proposed methods using this group of features is the work of Platt et al. [2002], who used songs' tags such as genre, style, and mood to compute the similarity between songs using a Gaussian Process regression. Other approaches use web-text data, as proposed by Knees et al. [2004], where they used a search engine to retrieve web pages about the artists and represent them by a variant of the TF-IDF. Based on this representation, the similarity between artists was computed using the cosine similarity measure. Similarly, Pampalk et al. [2005a] proposed to use the content of web pages ranked by Google to hierarchically organize artists. Another approach to represent song and artists using text is the work by Logan et al. [2004], who proposed to calculate the similarity between artists using the lyrics of the songs, showing it can be used to discover genre clusters, but concluded it is less useful than using acoustic information.

Measuring the similarity between songs based on the number of times they somehow co-occur together is another method found in the literature. Co-occurrence similarity is based on an idea where if two items appear very often in the same context, that's evidence they are similar to each other [Knees and Schedl, 2013]. In the context of music similarity, when two songs co-occur in many users' playlists, lists of favorite songs, or appear several times in sequence on music radio stations, we can probably recommend them together. For example, in the work of Pachet et al. [2001], the authors used the sequence of songs played in a French radio station to represent songs by a vector of its co-occurrence with other songs and calculated the similarity between songs using the correlation between the vectors. Pontello et al. [2017] proposed to calculate the similarity between songs using their co-occurrence on top-most listened songs of Last.fm[3] users. The similarity between songs were calculated using the cosine similarity, that is, $cos(i, j) = coocc(i, j)/\sqrt{occ(i)occ(j)}$, where $coocc(i, j)$ is the number of times two songs co-occurred on users top songs, and $occ(i)$ is the number of times item i appeared individually.

An interesting method used to calculate the similarity between songs is utilizing word2vec on users playlists. Word2vec is a model from natural language processing that uses a neural network to create a vector space word embedding from a text corpus where similar words (having the same context) are close to one another. The idea of using word2vec on a recommendation scenario was proposed by Grbovic et al. [2015], calling the approach as prod2vec. When applying this model to MIR field, each playlist is treated as a document, and each song is a word of the document. Then word2vec is used to create a music embedding on a vector space where similar songs are close to each other. Wang et al. [2016] used a similar approach to recommend music to users by modeling their preferences based on their playing records. Vasile et al. [2016] proposed to enhance the prod2vec model using metadata at training time, showing improvements when applying to 30Music dataset. Latter, Caselles-Dupré et al. [2018] studied the importance of word2vec hyperparameters in recommendation tasks by testing it on several different datasets, concluding the best values differ depending on the dataset.

2.4 Playlist Generation

The qualities a playlist should have to be considered enjoyable is widely discussed by authors when proposing playlist generator algorithms. Dias et al. [2017] reviewed some methods to generate playlists, and pointed out the most desired characteristics a playlist should satisfy. Some of them are the balance between homogeneity with song diversity and coherent transitions. These characteristics should be taken into consideration when proposing new algorithms to generate music playlists.

Authors have substantially studied how to automatically generate music playlists, proposing diverse methods to solve the problem. Some approaches iteratively select the songs to add to the playlist, while others use a set of predefined songs and define the

[3]https://www.last.fm

order they will be played. Bonnin and Jannach [2015] have done an extensive review of the methods most used in the literature. When talking about creating playlists, a discussion of its definition raises. Some authors define a playlist as an ordered sequence of tracks, while others consider it only as a set of songs, ignoring the order the user will listen to them. Since in this work we are worried about the transition between songs, we will take into consideration the order of the tracks in the playlist and use the playlist generation definition proposed by Bonnin and Jannach [2015].

Definition. Playlist Generation: Given (1) a pool of tracks, (2) a background knowledge database, and (3) some target characteristics of the playlist, create a sequence of tracks fulfilling the target characteristics in the best possible way.

In this definition, the pool of tracks are all the available tracks that can be included in the playlist. The background knowledge database consists of the information of the songs of the database, such as its acoustic characteristics. And finally, the target characteristics are the properties the playlists must satisfy to be considered enjoyable by a specific user or group of users. Depending on these items, some proposed algorithm may become unfeasible. For example, if we have a large pool of tracks, or define complicated target characteristics, selecting the appropriate track to add the playlist can be computationally expensive.

One algorithm that attempts to create music playlists maximizing the similarity between consecutive tracks is proposed by Pohle et al. [2005], where he used the Traveling Salesman Algorithm to define the order the tracks will be played. One disadvantage of this method is that the set of tracks of the playlist must be previously defined, requiring the user to select them before running the algorithm, decreasing the interaction. Another problem this method faces is the time consumed to compute the optimal sequence since TSP is known to be NP-Hard. Aucouturier and Pachet [2002b] proposed to generate an initial random playlist and iteratively change the songs of the playlist in order to satisfy a list of constraints. He showed his algorithm is linear with the size of the database, which can cause it to be unfeasible, since most of today music datasets contain millions of available songs. Similarly, Pauws et al. [2008] proposed to use a simulated annealing algorithm to iteratively perform modifications in the playlist and avoid local optimum. Both these methods require the playlists' constraints to be optimized, leaving to the users the effort to define them.

Other approaches generate a playlist with songs similar to a set of seed songs. Maillet et al. [2009] proposed a method that, given a seed song and a tag cloud created by the user, iteratively selects the next song to be added to the playlist. To use this algorithm the user needs to define the tag cloud, and it always produces the same playlist given the same input, not satisfying the novelty criteria. A method to generate music playlists based on radio playlists is proposed by Ragno et al. [2005]. He constructed a Markov Chain based on the transition between songs on radio streams and proposed to, given a seed song, generate a playlist by performing a random walk based on the estimated probabilities. Ben-Elazar et al. [2017] proposed a Bayesian Network based to model artists' genre and sub-genre and used it to recommend songs to be added to a playlist, also given a seed song.

There are two approaches to generate music playlists we want to give special attention. The first is proposed by Flexer et al. [2008], where a playlist is generated given the desired number of songs in the playlist and two anchor songs: the start and end song of the playlist. After defining these parameters, the algorithm creates a playlist connecting the anchor songs. The method to complete the playlist consist of selecting equal-spaced songs from a music space, where the distance is computed using the Kullback-Leiber divergence of the Gaussian models using MFCCs of the songs. The interesting thing about this algorithm is that by selecting two anchor songs, the user can define different music styles he/she would like to listen to in the same playlists, creating a heterogeneous playlist. And by defining the number of songs in the playlist, it is possible to implicitly define the desired size of the steps given on the music space, where a large number of songs implies small steps. One disadvantage of this method is that it is deterministic, always generating the same playlist given the anchor songs and the desired number of songs in the playlist. Another interesting approach is the work of Pontello et al. [2017], where the similarity between songs is computed based on co-occurrence on users' favourite songs lists, and a directional navigation is performed on the music space. Given a seed song, the method retrieves a set of K songs as candidates to be the next song. By retrieving this set, the method can ensure a smooth transition between songs of the playlist. The next song is randomly selected from this set, and the user can skip songs he doesn't like, guiding the navigation to songs he would enjoy listening to. The method also allows the user to define a target destination where the navigation should try to go, as proposed by Flexer et al. [2008]. Those works receive particular attention because they are similar to the methods that will be proposed in this work.

Chapter 3

Datasets

In this chapter, we will describe and present some basic statistics of the datasets used in this work. We are going to use datasets from three different sources: Billboard, Spotify and The Art of the Mix. The Billboard dataset contains mainly content-based characteristics extracted from the songs' audios. The other two datasets are composed of playlists manually created by users on public platforms and artists' music genres. These datasets will be used to propose how we can calculate the similarity between songs, create a music space, and generate music playlists.

3.1 Billboard Dataset

Billboard is an American magazine that weekly publishes charts with the songs and albums popularity. The metrics used to evaluate the popularity includes album sales, track downloads and streaming on Youtube and Spotify, for example. Mauch et al. [2015] studied the songs that appeared on the Billboard Hot 100 from 1960 to 2009 and analyzed the evolution of popular music in the USA. The authors published the dataset used in their studies, which consists of acoustic features and metadata of 17,094 songs. The characteristics were extracted from 30-seconds audio samples of each song. After extracting pitch and MFCC features from the audios, the authors used a Latent Dirichlet Allocation (LDA) to encode the songs as two probability distributions. The first probability distribution is over eight timbral topics that capture particular timbres (e.g. drums, aggressive, female voice), and the other is over eight harmonic topics that capture classes of chord changes (e.g. 'dominant-seventh chord changes'). Besides the timbre and harmony topics, each song is also associated with its year of release and with a subset of 140 genre tags, such as rock, pop, and soul. The value of the tag is 1 if the song is of that genre, and 0 otherwise, and a song may have more than one tag.

Since we already have the year of release as a feature, and the country may over discriminate similar songs, we removed 19 genre tags associated with the decade and the country of the music (e.g. 80s and UK). After removing these tags, all songs that were not associated with any genre tag were excluded from the dataset, resulting in 15,763 remaining songs, with 121 different tags. In order to put all features on the same scale, we transformed each group into a probability distribution, which is already the case for timbral and harmony features. For the genre group, we divided the value of each tag $g_i^j, 1 \leq j \leq 121$ by the number of tags associated with the song s_i, that is $\sum_{j=1}^{121} g_i^j$. For the *freshness* characteristics, since we know the range of the years are between 1960 and 2009, we performed a unity-based normalization, that is, $y_i' = \frac{y_i - 1960}{2009 - 1960}$. Since we want each set of features to be a probability distribution, we added another feature, which is $y_i'' = 1 - y_i'$, making the year to be represented by two real numbers, that is, $\hat{y}_i = (y_i', y_i'')$. After this processing, each song is represented by eight timbral topics, eight harmonic topics, 121 tags, and two *freshness* characteristics, totaling 139 features.

To analyze the distribution of the songs' years in the Billboard dataset, we plotted a CDF of its values, which can be seen on Figure 3.1. We can observe approximately 60% of the songs have release date before 1985. This shows how old the songs of this dataset are, which may be a problem when computing the similarity between songs based on acoustic features, as some old track files contain noise that may pollute the characteristics extracted from the audio.

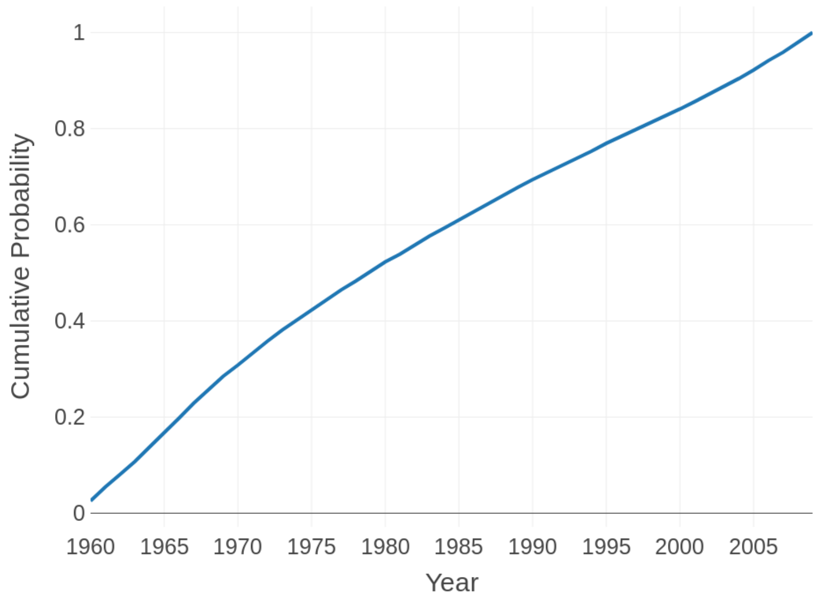


Figure 3.1: CDF of the songs' release year on Billboard dataset

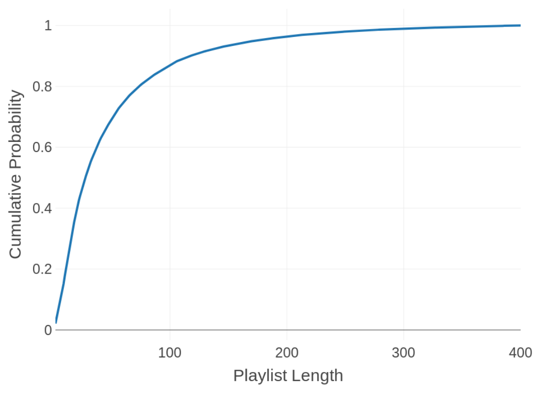
3.2 Spotify Dataset

Spotify is a music stream service created in 2008, containing today more than 30 million songs available for its users to listen to and create music playlists. Spotify also has an API that allows us to get some data from their system, which was used to get songs characteristics, artists information and users' public playlists. In this work we used only users' playlists and artists' information. To get the data, we used the API to search for playlists using some keywords that are listed in appendix A. For each playlist returned we saved the ID of the user who created it, totaling more than 25,565 different users. Then, for each user ID, we used the API to save all his/her public playlists, obtaining a total of 330,931 playlists with more than 3 million different tracks and more than 300 thousand different artists. For each artist, we also saved the set of tags it is associated with on Spotify system, which describes its genres such as *axe*, *rock* and *progressive metal*. Some artists are not associated with any tag, and only approximately 100 thousand of them are associated with at least one tag. We called this set of data as Spotify Dataset.

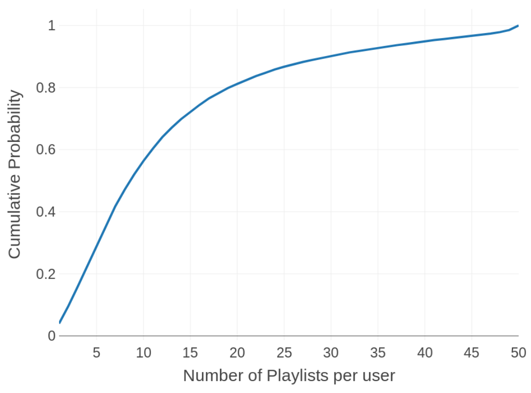
When analyzing the tracks of the playlists, we noted some of them have the same artist ID and track name, but different track IDs. This probably happens due to songs released in more than one album, such as singles or collections of the most popular songs of an artist. When this happens, the same song is assigned to different IDs for each of the albums it appears. To contour this problem, we created sets of tracks that have the same artist ID and track name. Then, on the users' playlists, we substituted each track ID by the most popular ID of its set, where popularity here is measured by the number of occurrence on users' playlists. This decreased the number of different track IDs in the playlists to approximately 2.4 million songs.

In Figure 3.2(a) we can see a CDF of the playlists' size. Since there are a few playlists size larger than 500 (some with thousands of songs), to better visualize the graph, we filtered the playlists with length higher than 400, which correspond to less than 1% of them. Analyzing the figure we can observe most of the playlists have length lower than 100, which indicates the ideal size of a playlist. Figure 3.2(b) shows the CDF of the number of playlists per user. Since Spotify limits each user to have up to 50 playlists, there is a small peak at number 50. We can observe most of the users have less than 15 playlists. We are going to leave an analysis of the users' playlists for the next chapter.

Regarding the tags we obtained, since we want to use the data to analyze and compare songs, we assigned to each track the tags of its artists. By analyzing the tags, we observed some of them contain redundant and unnecessary information. Most are



(a) CDF of Playlists' length



(b) CDF of Number of Playlists per user

Figure 3.2: CDFs of Playlists' length and Number of Playlists per User on Spotify Dataset

composed of more than one word and could be broken into several tags or replaced by only one word. Examples are *dance pop* (it could be divided into *dance* and *pop*), and *tropical house* (it sounds almost the same as *house*). To correct this problem, we replaced each tag by its last word (for example, *dance pop* became *pop* and *tropical house* became *house*). Because this process could create strange tags or loss of information (transforming *hip hop* to *hop* and *modern rock* to *rock*), we listened to songs with some specific tags, correcting the transformation performed when necessary. In the end, the changes performed reduced to 1,295 the number of unique tags. As we still had a large number of tags (more than a thousand tags), we decided to select only a subset of them to describe the artists. To determine a good number of tags to describe an artist, we computed the popularity of each tag (number of songs on the users' playlists containing it) and plotted the percentage of songs on the playlists covered given the k most popular tags (on logarithm scale), which can be seen on Figure 3.3. We can observe that 100 tags are enough to cover approximately 95% of the playlists' songs. Therefore, we decided to use only the 100 most popular tags.

In order to use the Spotify dataset to evaluate a music space created using the Billboard dataset, we made an intersection of the Spotify playlists with the Billboard dataset. For each track of the Spotify dataset we searched for a song in the Billboard dataset with the same artist name and track name, mapping a total of 11,078 Spotify track IDs to an IDs of the Billboard dataset. We then substituted in Spotify playlists the track IDs by the IDs of the Billboard, discarding from the playlists the tracks not found. At the end of the process, the playlists with less then 5 tracks were also discarded. The CDF of the logarithm of playlists length of the intersection can be seen

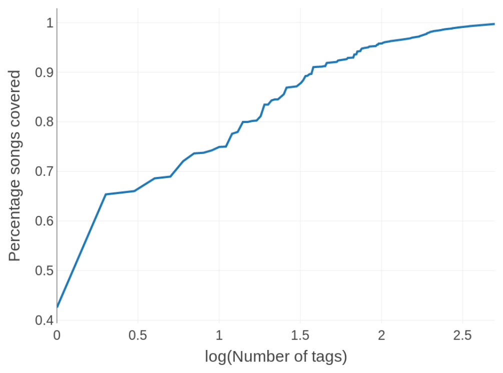


Figure 3.3: Number of songs covered given top k tags (log base 10)

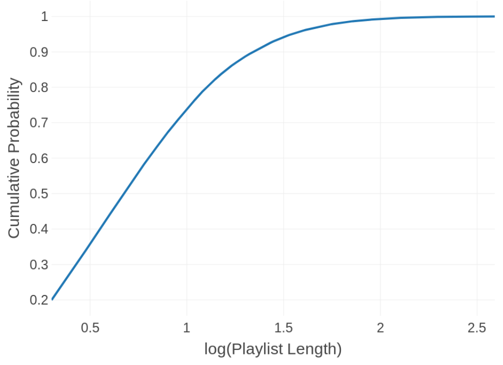


Figure 3.4: CDF of Playlists' length after intersection of Spotify and Billboard datasets (log base 10)

on Figure 3.4. We can observe there was a reduction in the number of tracks in the playlists, as most of the tracks in Spotify dataset are not in the Billboard dataset.

We present in Table 3.1 some summary statistics of the Billboard dataset, Spotify dataset, and their intersection. Since the Billboard dataset is only composed by tracks and their acoustic characteristics, it doesn't have playlists and users information. We can observe after the intersection there was a drastic reduction in the total number of playlists, but not in the number of users, indicating that most users create at least one playlist with popular songs. There were also a reduction on the average playlist length, since we are considering only the tracks that are on the Billboard dataset. We can also observe that in the Spotify dataset the average number of artist per playlist is less then half the average playlist length, which means that users tend to add to the playlist more than one song of the same artist.

	Billboard	Spotify	Spotify $\cap$ Billboard
Playlists	-	330931	46471
Users	-	25565	17957
Tracks	15765	2476180	11078
Average playlist length	-	59.04	16.8
Artists	4846	306418	3073
Average N° of artist per playlists	-	27.31	12.39
Average N° of playlists per user	-	12.95	2.59

Table 3.1: Summary statistics of the Billboard dataset, Spotify dataset and their intersection

3.3 The Art of the Mix Dataset

The Art of the Mix [1] (AOTM) is a warehouse for playlists integrated with iTunes, where users can create and post their playlists. The playlists of this website were used by McFee and Lanckriet [2012] who proposed an algorithm to automatically generate music playlists by performing a random walk on a hypergraph, making the dataset publicly available [2]. This dataset contains a total of 101,343 manually created playlists of 16,197 different users, having a total of more than 700 thousand different tracks.

This dataset will only be used in this work to asses the quality of the music spaces that will be created in Chapter 5. Therefore we made an intersection of the AOTM dataset with the Billboard dataset and Spotify dataset using the same process used when intersecting the Spotify dataset with the Billboard dataset. In Table 3.2 we present some summary statistics of the datasets intersection. We can observe that, since Spotify dataset is bigger than Billboard dataset, the intersection of AOTM dataset with Spotify dataset have more playlists and tracks. Also, observe that when intersecting with the Billboard dataset, most of the users were discarded for having playlists with less than 5 songs.

	AOTM $\cap$ Billboard	AOTM $\cap$ Spotify
Playlists	6480	95149
Users	2678	16172
Tracks	6556	294539
Average playlist length	6.83	15.54
Artists	2402	21302
Average N^o of artist per playlists	6.20	10.39
Average N^o of playlists per user	2.42	6.17

Table 3.2: Summary statistics of the AOTM dataset after intersection with the Billboard dataset and Spotify dataset

[1] http://www.artofthemix.org
[2] https://bmcfee.github.io/data/aotm2011.html

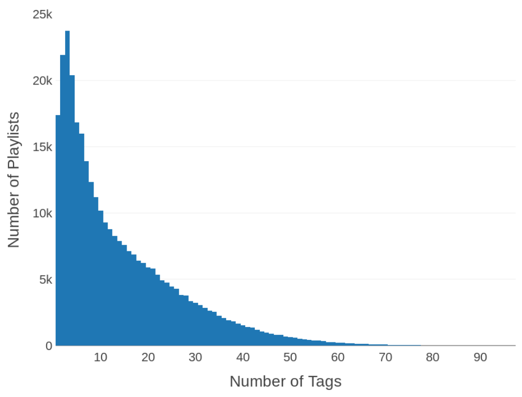
Chapter 4

Users Characterization

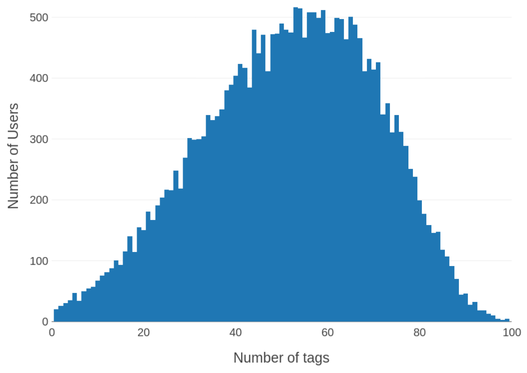
In this chapter, we will analyze the playlists of the Spotify dataset and characterize the users by their music taste. First, we will use the Shannon entropy to measure how heterogeneous are the playlists and users. Then, we will use the Theil index to determine which genres are equally present in the playlists, and which are present in only a few playlists. To perform this analysis, we will represent playlists and users using only the tags of the artists we have available. Here, we will use the notation v^i to indicate the i^{th} element of the vector v.

4.1 Playlists and Users Representation

As described in Chapter 3, the Spotify dataset contains users' public playlists composed of a list of tracks where each track is associated with its artists' genre. Since we want to describe a playlist by the music genres it contains, for each playlist we created a vector of size 100, where each position of the vector stores the number of times each of the 100 music genres appeared in that playlist. As not all tracks are associated with a music genre, we obtained only 248,216 playlists (75% of them) with at least one tag. We represented the users in the same way as the playlists, that is, by a vector with the counts of the genres he/she added to all his/her playlists. In other words, the vector representation of a user will be equal to the sum of all his/her playlists' representation. In Figure 4.1 we show the histogram of the total number of different tags each playlist and user have, that is, the number of elements of their vectors representation that are greater than 0. We can observe most of the playlists have a total number of tags lower than 30, but most of the users have a total number of tags around 50, indicating users tend to have an eclectic taste.



(a) Histogram of Playlists' number of tags



(b) Histogram of Users' number of tags

Figure 4.1: CDFs of Playlists and Users' total number of tags

4.2 Playlists and Users Heterogeneity

Since we want to determine if the users tend to be heterogeneous, we want to analyze how diverse are the playlists they create. In order to put all vectors representation on the same scale, we performed a normalization by dividing each vector by their sum. Therefore, we can now consider each user and playlist is represented by a probability distribution over the 100 music genres. That is, given a playlist p_i, its vector representation will be $p_i = (p_i^1, p_i^2, \ldots, p_i^{100})$, where p_i^j is the proportion of genre g_j in playlist p_i, $\sum_{j=1}^{100} p_i^j = 1$ and $p_i^j \geq 0\ \forall j$. The same is valid for all users u_i.

To perform an initial analysis of the users and playlists diversity, we will measure their heterogeneity using the Shannon entropy, an uncertainty measure widely used in information theory field. Given a discrete random variable X with possible values $\{x_1, x_2, \ldots, x_n\}$, we can measure its entropy $H(X)$ as:

$$H(X) = \sum_{i=1}^{n} -p(x_i) \log_2(p(x_i))$$

It can be proven that if X is a uniform distribution, that is, all possible values x_i are equally likely, the entropy of X assumes its maximum value $\log_2(n)$. On the other hand, when one value x_i has probability 1 and all $x_j \neq x_i$ has probability zero, the entropy of X is equal to 0. If we calculate the entropy of a user or playlist vector representation, we will have a measure of its heterogeneity. If the entropy is equal to 0, it will mean the playlist or user is composed of only one music genre (representing a restrict user), while if the entropy is equal to $\log_2(100)$, all genres are equally likely (representing an eclectic user). Since the entropy will be a value between 0 and

$\log_2(100)$, we normalized the values dividing them by its maximum value $\log_2(100)$. We will define then the heterogeneity of a playlist or user as its normalized entropy value. In Figure 4.2 we show a curve of what would be the heterogeneity of a playlist if it were equally distributed over only k different music genres. We can observe the heterogeneity increases very fast, where for a value of $k = 40$, the heterogeneity would be greater than 0.8.

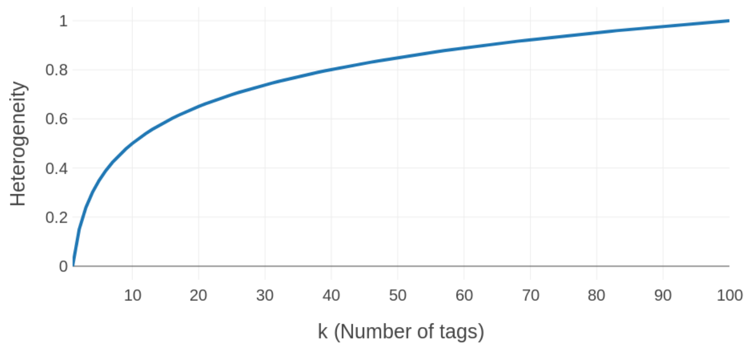
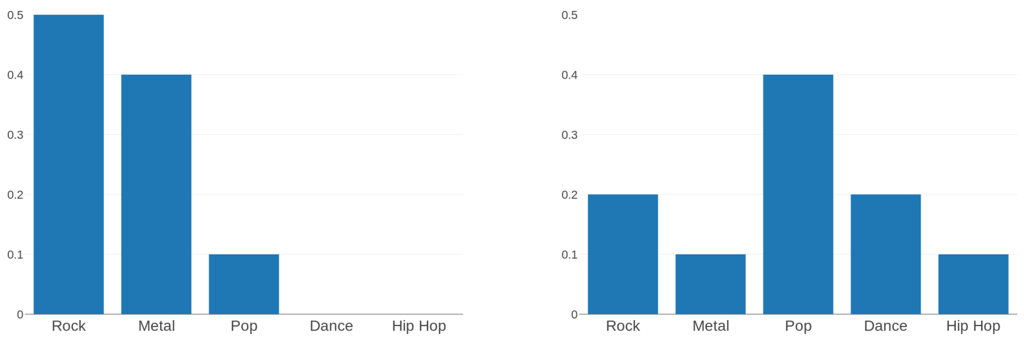


Figure 4.2: Heterogeneity of a playlist equally distributed over k tags

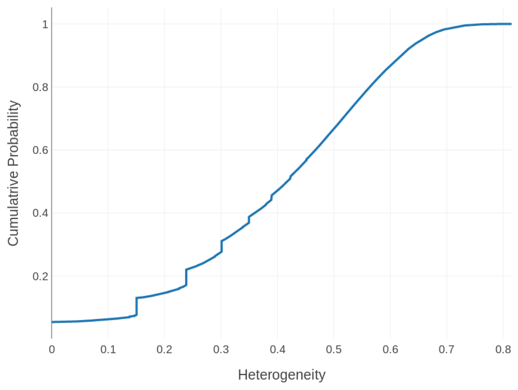
To illustrate the heterogeneity value, let's suppose we have two different users with a vector representation as in Figure 4.3, with genre proportion equal to 0 for all the other tags. Figure 4.3(a) represents a user that enjoys *rock* and *metal* songs and a few *pop* songs. Figure 4.3(b) represents a more eclectic user, who enjoys *pop* songs, but also listens to some *rock*, *metal*, *dance* and *hip hop* songs. The entropy of the first user is equal to approximately 0.2, while the entropy of the second user is equal to 0.32, a value higher than the first one.



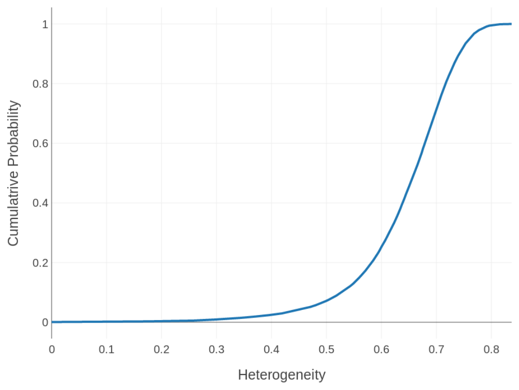
(a) Genre distribution of user 1 (b) genre distribution of user 2

Figure 4.3: Genre distribution of two users

We ploted the CDF of all playlists and users heterogeneity of the Spotify dataset, which can be seen in Figure 4.4. We can observe that while only approximately 40% of the playlists have heterogeneity greater than 0.5 (which means equally distributed over 10 music genres), more than 90% of the users have heterogeneity greater than this value. This shows that although the playlists usually have only a few music genres, users enjoy songs of several different styles.



(a) CDF of Playlists Heterogeneity



(b) CDF of Users Heterogeneity

Figure 4.4: CDFs of Playlists and Users' Heterogeneity

4.3 Genres Distribution Inequality

By calculating the entropy of playlists and users we can analyze how heterogeneous they are, but we are not able to identify which music genres are less equally distributed over the playlists than others. In this section, we are going to analyze each genre separately, and compare them to determine which are more equally present in the playlists, and which are rare to find in a randomly selected playlist. To perform this analysis we are going to use the *Theil index*.

The Theil index is a statistics mainly used to measure the inequality in economic scenarios [Theil, 1967]. It uses the same principle of entropy in information theory to measure the inequality of the income distribution over a population. To exemplify it, let us suppose a population of N individuals where each individual earns a non-negative fraction y_i of the total income of the population, and $\sum_{i=1}^{N} y_i = 1$. If we calculate the entropy of the income distribution, we will have

$$H(y) = \sum_{i=1}^{N} y_i \log\left(\frac{1}{y_i}\right)$$

where $H(y) = 0$ if a single individual earns all the income, and will be $\log(N)$ if every individual earns the same amount of income. Based on these results, we can say $H(y)$ measures the income equality. Since $H(y)$ is a value between 0 and $\log(N)$, we can measure the income *inequality* by calculating

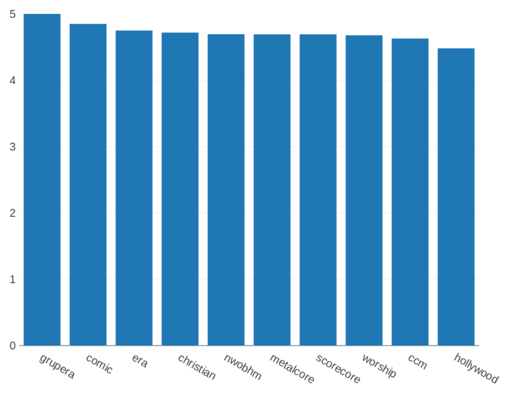
$$\log(N) - H(y) = \sum_{i=1}^{N} y_i \log\left(\frac{y_i}{1/N}\right)$$

We can observe the income inequality assumes values between 0 and $\log(N)$, where the value of 0 means everyone receives the same share. Also, observe the income inequality is the same as the Kullback-Leibler divergence between the income distribution of y and a distribution where all individuals earn the same fraction of income $\frac{1}{N}$.

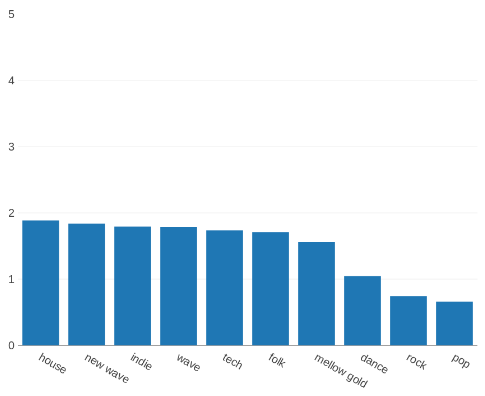
In this work, we are going to use the Theil index to measure the genre distribution inequality over the playlists. To illustrate how it will be used, let's consider only one music genre g_j. Given a playlist p_i, p_i^j is the proportion of the playlists' songs that have genre g_j. In this scenario, we are going to say the total income of all playlists will be $T_j = \sum_{k=1}^{N} p_k^j$, where N is the number of playlists in our dataset. Given the total income value, the income share of playlist p_i with respect to g_j is equal to $\frac{p_i^j}{T_j}$. Using these definitions, we can calculate the Theil index of the genres distribution over the playlists to determine which of them are equally present in all playlists, and which are present in some few playlists while absent on the others.

In Figure 4.5(a) we can see the Theil index of the 10 genres with biggest inequality, while Figure 4.5(b) shows the 10 genres with smallest inequality. We can observe genres like *pop*, *rock* and *mellow gold* are present almost in every playlists (have a small inequality) while genres like *grupera* (Mexican songs), *christian* and *worship* (religious songs) are present in only few playlists. This result reflects the users behaviour, as religious songs satisfy only a selective group of users. Based on this result, we can conclude music genres such as *christian* and *grupera* should not be recommended to all users (as only a few ones may enjoy it), while genres as *pop* and *mellow gold* can be recommended to most users, as they are likely to enjoy them.

We could also use Theil index to compute the inequality of the tags distribution among the users. Instead of that, we will use the fact that the total Theil index can be split in two parts, where one of them measures the inequality between groups of individuals, and the other measures the inequality within the individuals of the same group. Consider again a population with N individuals now divided in G groups, where each individual belongs to exactly one group, and S_g are the individuals of group g



(a) Theil index of tags with biggest inequality



(b) Theil index of tags with smallest inequality

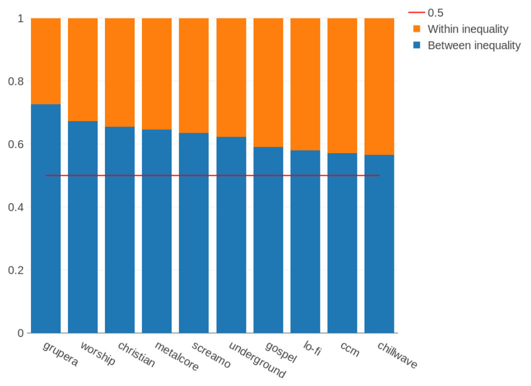
Figure 4.5: Theil index of tags

with N_g individuals ($\sum_{g=1}^{G} N_g = N$). Also, let Y_g the total share group g receives ($\sum_{g=1}^{G} Y_g = 1$).It can be shown that the Theil index of the population is equal to:

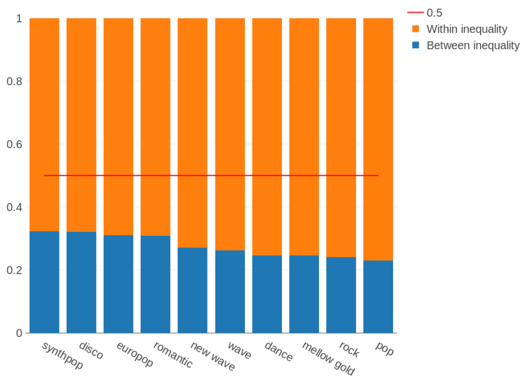
$$\sum_{i=1}^{N} y_i \log \frac{y_i}{1/N} = \sum_{g=1}^{G} Y_g \log \frac{Y_g}{N_g/N} + \sum_{g=1}^{G} Y_g \left[\sum_{i \in S_g} \frac{y_i}{Y_g} \log \frac{y_i/Y_g}{1/N_g} \right] \tag{4.1}$$

We can observe that the first term on the right-hand of the equation is equal to the KL divergence between the income distribution over the groups and the distribution where each group receives a share equal to $\frac{N_g}{N}$, which is the proportion of its population N_g compared with the total population N. This term is called *Between inequality*, as it measures the inequality of the share distribution among the groups. The second part of the right-hand is the *Within inequality*, which measures the inequality within each group, that is, how the total income of each group is distributed over its individuals.

In our problem, we treated each playlist as an individual, and each user as a group of individuals, where the playlists of each user are the individuals of the group. We then calculated, for each tag, the percentage of its total Theil index that is the *Between inequality* (some users create playlists containing specific music genres, while other users don't) and the percentage that is the *Within inequality* (given a specific user, some of his playlists contain only songs of a specific genre, while the other playlists don't have that genre). The proportion of the tags with biggest and smallest *Between inequality* can be seen in Figure 4.6. We can observe only a few users listen to *grupera* or *christian* songs, and almost all their playlists contain these genres. On the other hand, almost all users enjoy *rock* and *dance* songs, but they tend to create specific playlists for these music genres. This reinforces the conclusions of figure 4.5.



(a) Proportion of Theil index for tags with biggest Between inequality



(b) Proportion of Theil index for tags with smallest Between inequality

Figure 4.6: Proportion of Theil index of tags with biggest and smallest Between inequality

4.4 User Clustering

With the previews analysis, although we could observe some music genres such as rock are enjoyed by almost all users, someone may not enjoy such genres and prefer to listen to jazz or classic songs. In order to group users by their music taste (where users that enjoy similar songs are in the same group), we are going to cluster them using their representation. Since representing each user by a probability distribution over 100 tags may cause redundancy (as some music genres may be related to each other), we performed a dimensionality reduction using Singular-Value Decomposition (SVD). SVD is a method that allows us to represent a matrix by the multiplication of three other matrices and ease the elimination of less important parts of the original matrix Leskovec et al. [2014]. Given a $m \times n$ matrix M with rank r, we can write M as:

$$M = U\Sigma V^T$$

where U is a $m \times r$ matrix, V a $n \times r$ matrix and Σ is a $r \times r$ diagonal matrix, holding the singular values of M in the diagonal. By analyzing the matrix Σ, we can hold the largest k singular values, zeroing the $(r - k)$ smallest ones. This allows us to represent each song by the first k columns of U. The number of selected columns k is chosen to represent at least 95% of the original matrix M, that is, the sum of squares of the retained singular values should be at least 95% of the sum of squares of all the singular values, as in the formula

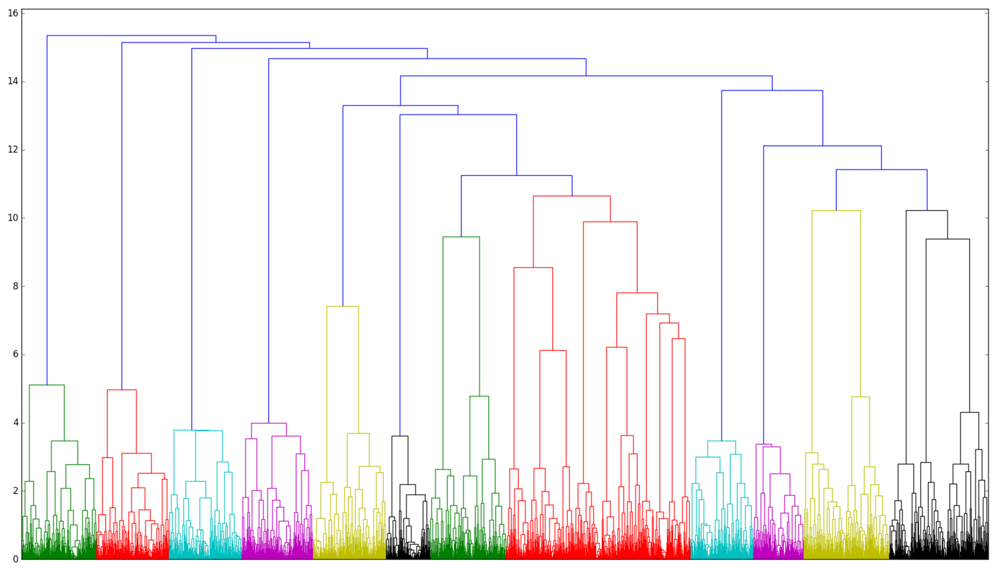


Figure 4.7: Users Clustering Dendrogram

$$\frac{\sum_{i=1}^{k}\sigma_i^2}{\sum_{i=1}^{r}\sigma_i^2} \geq 0.95$$

where σ_i is the ith singular value of M. Here, we used 5000 randomly selected users and constructed a matrix M where each row represents a user and each column represents one of the 100 tags. In other words, each row of matrix M is a user's probability distribution over the tags. In our experiments, the value $k = 26$ was enough to represent 95% of the original matrix M. Therefore, we represented each user by the first 26 columns of the matrix U. Using the new users' representation and the cosine distance between the users, we used a hierarchical clustering algorithm using Ward's method that, at each step, merges the pair of clusters that minimizes the within-cluster variance. The dendrogram of the hierarchical clustering can be seen in Figure 4.7, in which we decided to create 12 users' clusters.

To better visualize the genres each group of users enjoys, we plotted a bar plot for each cluster showing how likely a genre will be enjoyed by a user of that group. First, we calculated the mean of the users' probability distribution, that is:

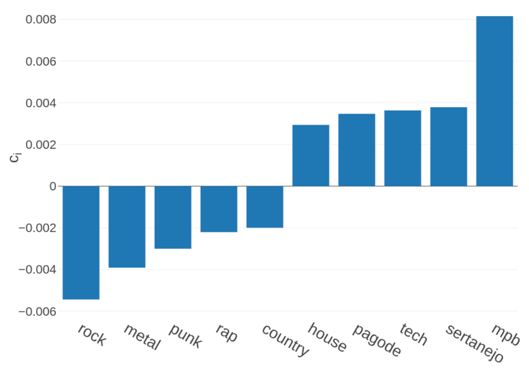
$$\bar{u} = \frac{1}{|U|}\sum_{u_i \in U} u_i$$

where U is the set of all users. $\bar{u}$ represents the probability distribution over the 100 tags for all users. In other words, $\bar{u}^i$ is the probability a user enjoys music genre g_i. Then, for each cluster C_i, we calculated the mean of all its user's probability distribution, that is, its probability distribution will be:

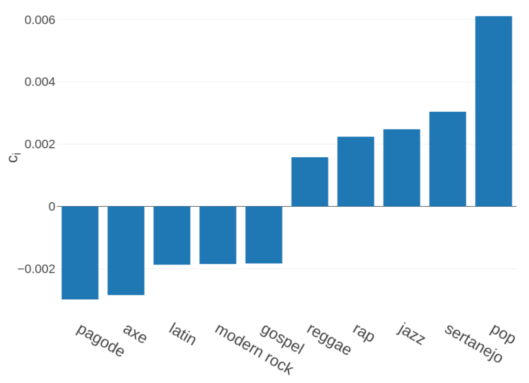
$$\bar{c_i} = \frac{1}{|C_i|} \sum_{u_i \in C_i} u_i$$

For each cluster C_i we calculated the difference $c_i = \bar{c_i} - \bar{u}$. This difference represents how the clusters enjoy each genre compared with the average user. If the value c_i^j is positive, it means the cluster C_i enjoys genre g_j more than the average user, and if c_i^j is negative, cluster C_i rejects genre g_j.

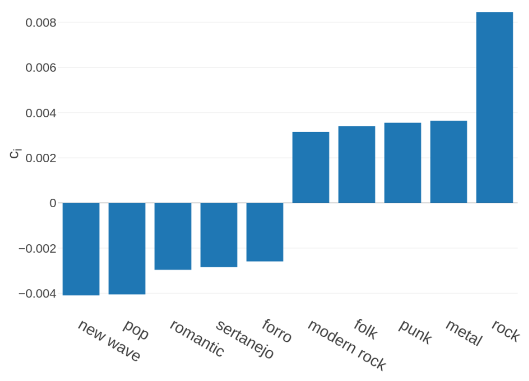
Figure 4.8 and 4.9 show the bar plots of the clusters we found. We plotted only 10 genres for each cluster, where the five bars on the left are the most rejected genres, and the five bars on the right are the most appreciated genres for that cluster. We can observe that the clusters represents groups of users with specific tastes. For example, Figure 4.9(b) represents a group of users that enjoys *mpb*, *pagode* and *bossa nova* songs. Figure 4.8(a) represents a group that also enjoys *mpb* songs, but prefers *sertanejo* than *pagode*, and rejects *rock* songs. Cluster represented on Figure 4.8(c), on the other side, enjoys *rock*, *metal* and *punk* songs, and rejects *sertanejo*. Figure 4.9(f) represents a group that enjoys *pop*, *dance* and *house* songs, but doesn't like *mpb* and *pagode*. We can also observe some clusters enjoy similar music genres, but differ on the genres they reject. For example, both clusters represented in figures 4.8(a) and 4.9(a) enjoy *mpb* songs. But while the first one enjoys *sertanejo* songs, the second one dislikes it. The same thing happens on clusters represented on figures 4.8(c) and 4.9(c). Both enjoys *rock* songs, but the first one doesn't like *pop* songs while the second one does. This results can help us understand users behaviour when listening to music, and also gives us an idea of which song we should recommend to a user. For example, if we know a user belongs to cluster represented in figure 4.8(a), we know we shouldn't recommend rock or rap songs to him /her.



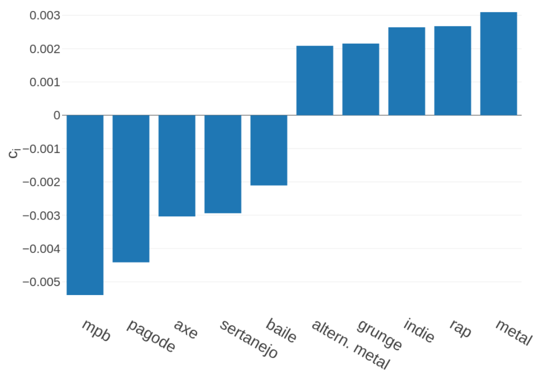
(a) Bar Plot of Cluster 1



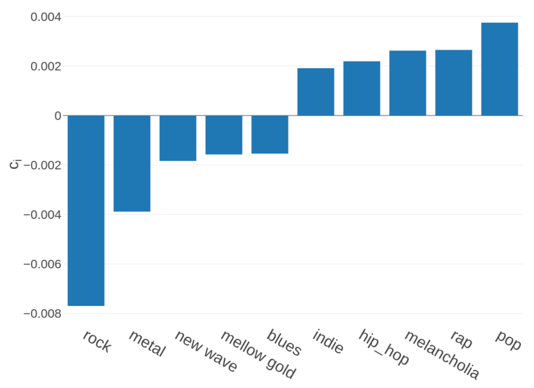
(b) Bar Plot of Cluster 2



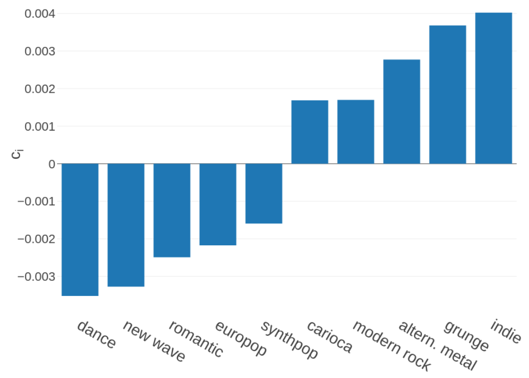
(c) Bar Plot of Cluster 3



(d) Bar Plot of Cluster 4

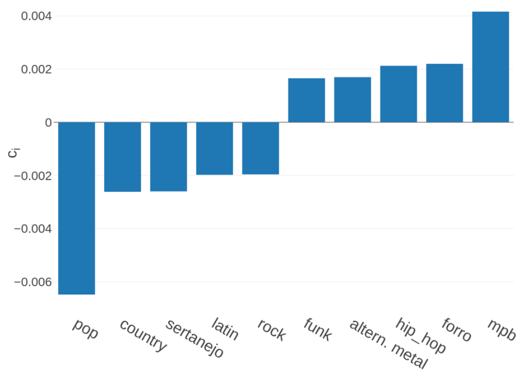


(e) Bar Plot of Cluster 5

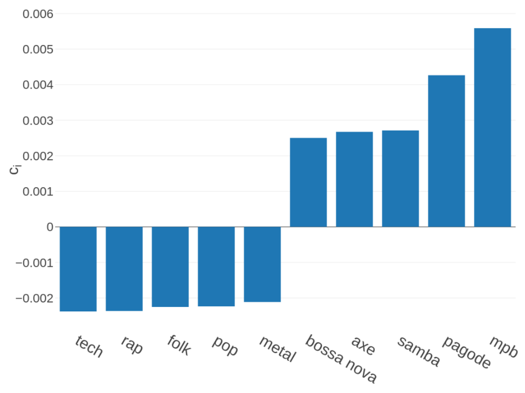


(f) Bar Plot of Cluster 6

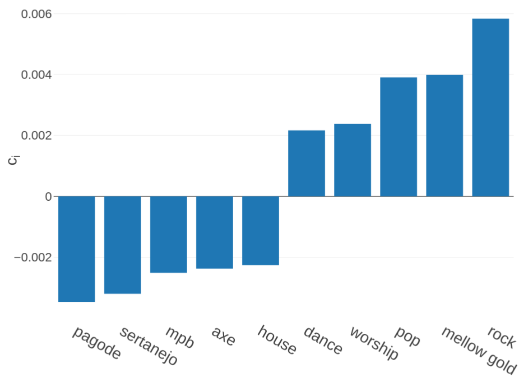
Figure 4.8: Bar Plot of clusters genres



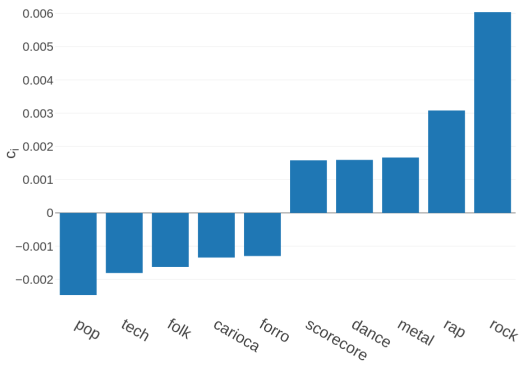
(a) Bar Plot of Cluster 7



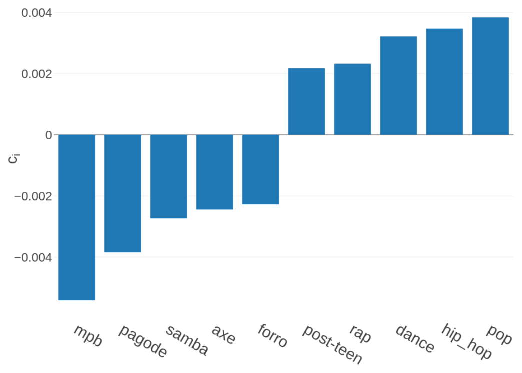
(b) Bar Plot of Cluster 8



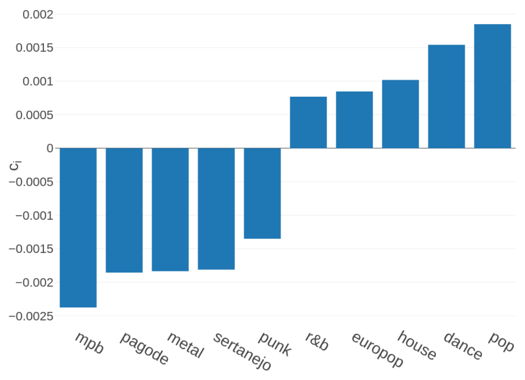
(c) Bar Plot of Cluster 9



(d) Bar Plot of Cluster 10



(e) Bar Plot of Cluster 11



(f) Bar Plot of Cluster 12

Figure 4.9: Bar Plot of clusters genres

Chapter 5

Song Similarity and Music Spaces

As described before, many algorithms have been proposed to calculate the similarity between songs. Some of them use content-based features, while others use context-based characteristics. In this chapter, we are going to describe how we used the data presented in chapter 3 to calculate the similarity (or distance) between songs. The first proposed similarity function will use the content-based characteristics of the Billboard dataset. The second will calculate the similarity between songs based on the co-occurrence on users' playlists, and a third similarity function will use the artists' genres. The proposed functions will be used to construct a music space. Here, we define a music space as a pair (S, d), where S is a set of N songs $S = \{s_i, i = 1, \ldots, N\}$ and $d : S \times S \to \mathbb{R}^+$ is a distance function between two songs. The music spaces constructed will be used to create music playlists by walking through it.

To simplify the description of the similarity functions, we will first establish some notations. Let us define S as the set of songs of the dataset and s_i as a specific song. Each song s_i will be characterized by a feature vector x_i. The feature vector of the songs will represent different characteristics depending on the dataset used. For the Spotify dataset, P will be the set of playlists, with $p_i \in P$ a playlist composed of a list of songs.

5.1 Similarity Using Content-based Characteristics

The similarity between songs using content-based characteristics will be calculated using the features of the Billboard dataset. As described in chapter 3, the Billboard dataset was used to analyze the evolution of songs in the USA from 1960 to 2009. After a preprocessing, the dataset is composed of 15,763 songs, where each song s_i is represented by four probability distributions $x_i = (h_i, t_i, \hat{g}_i, \hat{y}_i)$, where h_i is eight harmonic topics, t_i is eight timbral topics, $\hat{g}_i$ represents the genres of the songs, and $\hat{y}_i$ is the year of release represented by two numbers. Each set of features has sum 1. Given this song representation, we could use any distance function between the feature vectors, such as the Euclidean distance, the cosine distance or the KL divergence distance. In order to avoid the curse of dimensionality, we reduced the dimensionality of the data using an embedding algorithm. There are several embedding algorithms proposed in the literature, such as PCA, MDS, and Isomap. For this task, we decided to use t-SNE algorithm, the same used by Ben-Elazar et al. [2017] to visualize some artists' parameters.

T-SNE [Maaten and Hinton, 2008] is a dimensionality reduction algorithm with the objective to better preserve the distance between the data in the original space. After reducing the dimensionality of the data (in this work, we used PCA as the starting embedding) t-SNE uses a gradient descent algorithm to minimize the differences in the distances between songs in the original space and on the reduced space. Formally, given two songs s_i and s_j, let d_{ij} be the distance between the songs in the original music space, and $\hat{d}_{ij}$ the distance between the songs in the reduced music space, t-SNE tries to minimize the difference $|d_{ij} - \hat{d}_{ij}|$. When using t-SNE to reduce the dimensionality of the data to a 2D Euclidean space, songs that are close in this new space are also similar in the original space, i.e. they should be acoustically similar.

Figure 5.1 shows the music space generated by using t-SNE on the feature vectors x_i of the Billboard dataset. Each point represents a song and its color denote an associated genre on the Billboard dataset. The associated genre was selected as the less popular genre among those that have non-zero g_i and are among the 8 most popular genres in database (which covered approximately 85% of the songs). The reason to select the less popular rather than the most popular one is that, if we selected the latter one, the map in Figure 5.1 would have practically a single color. As we can observe, songs were coherently grouped according to their genres and similar genres are located near each other. For instance, a path from rock to hip-hop shall pass through pop and dance songs. However, songs from the same genre do not have to form a single cluster. In these cases, timbral and harmony features play an important role in selecting the appropriate location. Isolated islands were created with songs that are similar to each other but different from the remaining songs. For example, the two islands located on the West and South borders are composed mainly of oldies and soul songs, respectively, while the island located on the East are mainly Country songs. The horizontal axis is highly correlated with the freshness of the song. Older songs are located more on the left-hand side of the space, while new songs, like those from hip-hop, are more on the right-hand side. Based on these observations, we define the

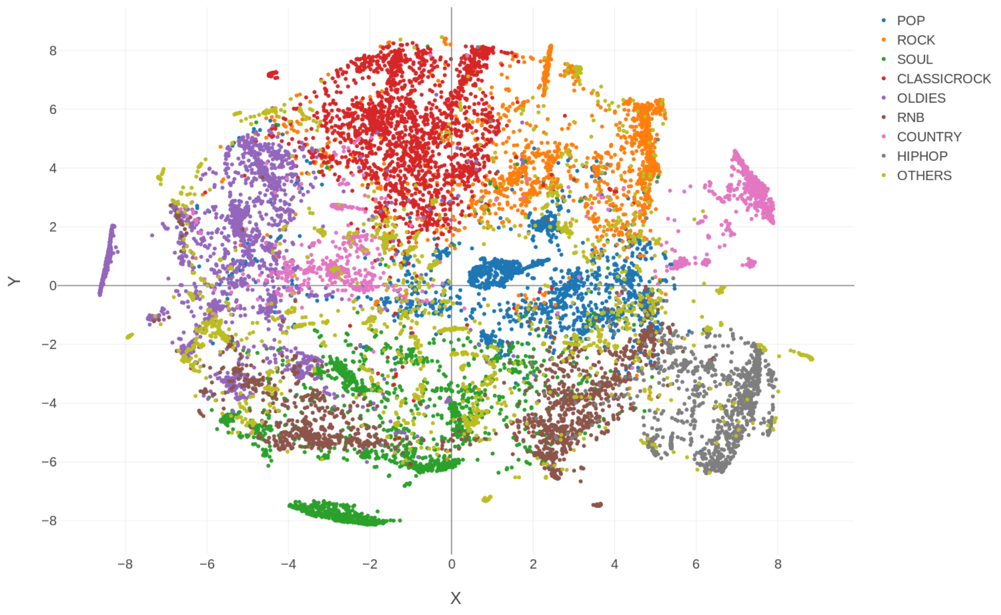


Figure 5.1: Billboard Music Space

Billboard music space as the set of songs of the Billboard dataset and the distance between songs as the Euclidean distance in the reduced music space.

To assess the quality of the generated music space, we compared the distance between songs in our space with their co-occurrences in the playlists of the intersection of Billboard dataset with AOTM dataset and the Spotify dataset. For each validation dataset we calculated the number of times each pair of songs co-occured in the same playlist and their Euclidean distance in our music space. In figures 5.2 and 5.3 we show the boxplots of the distances grouped by the number of co-occurrences for each dataset. Observe that the median distance decreases as the number of co-occurrences increases. This suggests that our music space is able to group similar songs together, as songs close in the music space tend to co-occur on users playlists.

To test if the model is coherent, we performed a qualitative evaluation by selecting three seed songs and searching for the most similar songs according to the model. The seed songs are *Sweet Child O' Mine* by *Guns N' Roses*, *How Deep Is Your Love* by *Bee Gees* and *Makes Me Wonder* by *Maroon 5*. The returned songs can be seen below.

Songs similar to *Sweet Child O' Mine* by *Guns N' Roses*:
Be Good To Yourself by *Journey*
Stand Up by *David Lee Roth*
Angel by *Aerosmith*
Sing Me Away by *Night Ranger*
Reason To Live by *Kiss*

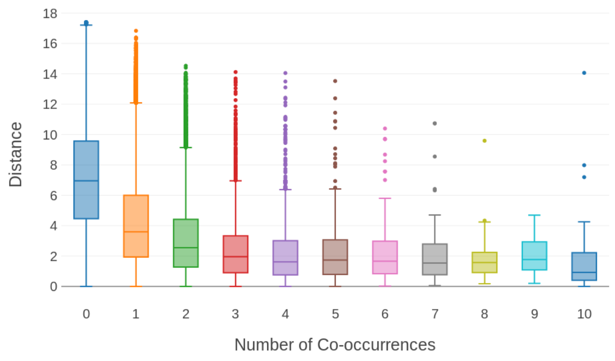


Figure 5.2: Boxplot of distance between songs on Billboard music space given number of cooccurrence in AotM dataset

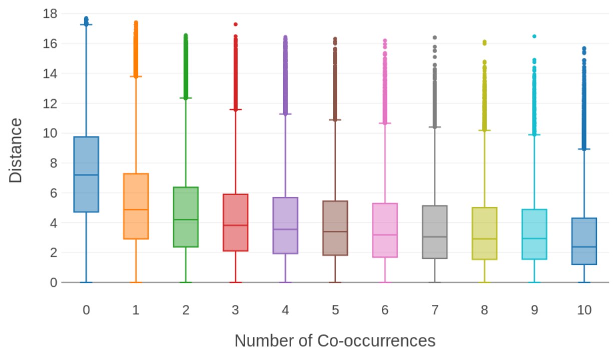


Figure 5.3: Boxplot of distance between songs on Billboard music space given number of cooccurrence in Spotify dataset

Songs similar to *How Deep Is Your Love* by *Bee Gees*:
A Fifth Of Beethoven by *Walter Murphy*
Lo Que Son Las Cosas by *Anais*
Missing You by *Brandy Tamia Gladys Knight & Chaka Khan*
Sitting Home by *Total*
Turning To You by *Charlie*

Songs similar to *Makes Me Wonder* by *Maroon 5*:
If I Never See Your Face Again by *Maroon 5*
Thunder by *Boys Like Girls*
Devil's Haircut by *Beck*
Better Than Me by *Hinder*
How Far We've Come by *Matchbox Twenty*

Observing the lists, we can see that the retrieved songs are similar to the seed songs. For example, the song *Sweet Child O' Mine* by *Guns N' Roses* retrieved songs by *Journey* and *Aerosmith*, which are rock songs, while a *Maroon 5* song retrieved other songs from *Maroon 5* and a song by *Boys Like Girls*, which is a pop artist.

It is important to point out that we could have reduced the dimensionality of the data to a larger dimensional space. The data were reduced to a 2D Euclidean space to make it visually appealing and ease the explanation of the proposed algorithms.

5.2 Similarity Using Context-based Characteristics

As described before, the Spotify dataset contains users' manually created playlists, and for each artist, we have a set of tags describing his music genres. These features were used to propose two methods to measure the similarity between songs based on context-based characteristics. The first method will compute the similarity between songs based on the co-occurrence of tracks on users' playlist, and the second method will be based on their tags.

Since the music spaces created here will be used to generate playlists, and we will use some users' playlists in our experiments, we decided to divide the set of playlists in two parts. The test set will contain playlists of 1000 users, leaving 304023 playlists to be the training set. When necessary, we will separate from the training set the playlists of 1000 users to be a validation set.

5.2.1 Similarity Between Songs Based on Co-occurrence

There are many proposed algorithms to compute the similarity between items based on co-occurrence. An example is the work of Goussevskaia et al. [2008], where the similarity between two songs are calculated based on the number of times they co-appear in users most favourite songs. Another approach is collaborative filtering [Sarwar et al., 2001], an algorithm widely used in recommendation systems [Linden et al., 2003]. The premise of collaborative filtering is that items that are mutually consumed by users are similar and can be recommended together. In this work, the similarity between songs based on their co-occurrence on users' playlists will be calculated using the word2vec model. Word2vec is a neural network model proposed by Mikolov et al. [2013] to produce a word embedding. The model takes as input a corpus of texts and, for each word of the corpus, creates a representation for it in a large dimension Euclidean vector space. After the embedding, words that are similar in context (that is, they co-occur in the same sentence many times) are mapped to similar vectors, where similarity is computed using cosine similarity. There are two approaches used to create the word embedding: Continuous bag of words (CBOW) and Skip-gram. The difference between the approaches is that while CBOW attempts to predict a word given the context, Skip-gram tries to predict the words of the context given a specific word.

Using word2vec to calculate the similarity between songs has already been studied by other authors, such as [Wang et al., 2016] and Caselles-Dupré et al. [2018]. To adapt our problem to the word2vec model, we treated each playlist as a text of the corpus, and each song as a word. This way, songs having a high co-occurrence on users' playlists will be considered to be in the same "context", and therefore will be mapped to close vectors on the embedding space. As discussed by Caselles-Dupré et al. [2018], word2vec has several hyperparameters, and most of them are already tuned to perform well in NLP tasks. When using it to other scenarios, the optimal hyperparameters can assume different values. To test what are the best parameters, we need a metric to asses the quality of a model. To calculate this metric, we selected from the training set 1000 users, and separated their playlists, constructing the validation set V. Given a playlist p_i of the validation set, we can calculate its consistency on a model m as

$$C_m(p_i) = \frac{1}{\binom{|p_i|}{2}} \sum_{s_i \in p_i} \sum_{\substack{s_j \in p_i \\ s_j \neq s_i}} sim_m(s_i, s_j)$$

where $sim(s_i, s_j)$ is the cosine similarity between songs s_i and s_j and $\binom{|p_i|}{2}$ is the binomial coefficient (number of different pairs of songs in the playlist p_i). If the constructed model is able to correctly identify similar songs, the playlists' consistency should be close to 1, that is, the songs that co-occur on a users' playlist should be considered similar to each other. We can evaluate a model m_i by calculating the mean of the playlists consistency

$$C(m_i) = \frac{1}{|V|} \sum_{p_i \in V} C_{m_i}(p_i) \tag{5.1}$$

and the higher this metric is, the better is the constructed model.

In our problem, we will use the Skip-gram approach, since we want to predict what songs are in the same "context" of a seed song, that is, what songs are similar to a specific song. When training the models, we used a *min count* parameter equal to 50. therefore a song must have appeared at least 50 times on the playlist to be considered part of the model. By setting this value, we discarded the less popular songs and created a music embedding for the 46606 most popular ones. When computing the consistency, we removed from the playlists the songs that were not embedded by the model. Using this value of *min count* we run experiments with the objective of finding the best parameters of word2vec to our dataset. Using as reference the experiments of Caselles-Dupré et al. [2018], we created models varying the following hyperparameters: number of negative samples NS (5 and 20), window size W (100 and 300), negative sampling distribution parameter α (-1.0 and 1.0), *learning rate* (0.0025 and 0.25) and embedding size S (100 and 300). For each configuration, we computed the consistency of the constructed model m_i using the formula 5.1. We choose only two levels for each parameter so we could perform a 2^k factorial experiment that informs which factors have a higher impact on the response variable, that in our case is $C(m_i)$. With the factorial experiment we found the *learning rate* and α are the variables most significant for the model (explaining approximately 82% and 2% of the response variable, respectively). Since those parameters are the most important in our application, we performed another experiment, fixing NS in 20, window size W in 100, and embedding size S in 100, and varying α (from -1.5 to 1.25 with a step of 0.25) and learning rate (with values 0.001, 0.0025, 0.025 and 0.25). Based on the experiment results, the best parameters found were $\alpha = -1.5$ and learning rate 0.001, creating a model m_i with $C(m_i) \approx 0.86$. Using this model, we define the **Word2vec music space** as the set of songs of the word2vec model, where each song is represented by a vector in a 100-dimensional Euclidean space and the distances between songs are computed using the cosine distance between their vectors representation, which is the same distance used to compute the similarity between words in a word2vec embedding.

We also compared the distance between songs in the Word2vec music space with their co-occurrences in users' playlists. We used here the playlists of the AOTM dataset and the playlists in the test set of the Spotify dataset. We also filtered the playlists that have less than five songs of the Word2vec music space, as those playlists would not be considered relevant for the analysis. The boxplots are in figures 5.4 and 5.5. Again, we can observe that the median distance decreases as the number of co-occurrences increases, suggesting the Word2vec music space is also able to group similar songs together.

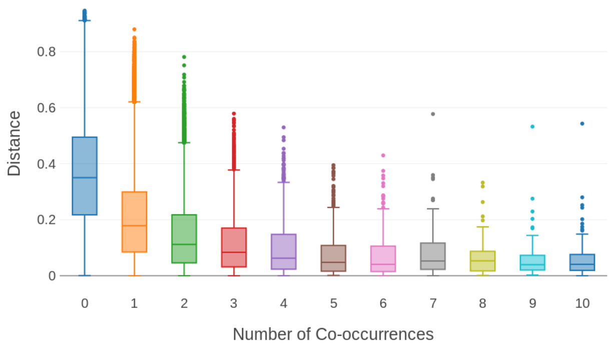


Figure 5.4: Boxplot of distance between songs on Word2vec music space given number of cooccurrence in AotM dataset

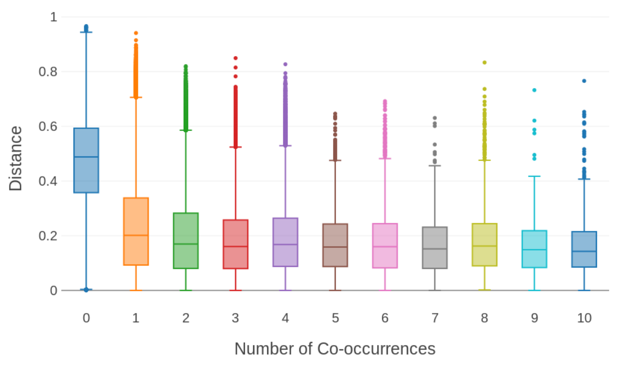


Figure 5.5: Boxplot of distance between songs on Word2vec music space given number of cooccurrence in Spotify dataset

Performing a qualitative evaluation on the Word2vec music space using the same seed songs used to evaluate the Billboard music space, the songs returned by each seed song can be seen below.

Songs similar to *Sweet Child O' Mine* by *Guns N' Roses*:
Livin' On A Prayer by *Bon Jovi*
Paradise City by *Guns N' Roses*
Welcome To The Jungle by *Guns N' Roses*
You Give Love A Bad Name by *Bon Jovi*
Knockin' On Heaven's Door by *Guns N' Roses*

Songs similar to *How Deep Is Your Love* by *Bee Gees*:
How Deep Is Your Love (2007 Remastered Saturday Night Fever) by *Bee Gees*
Mandy by *Barry Manilow*
Skyline Pigeon - Piano Version by *Elton John*
I Started A Joke by *Bee Gees*
Three Times A Lady by *Commodores*

Songs similar to *Makes Me Wonder* by *Maroon 5*:
Wake Up Call by *Maroon 5*
Sober by *P!nk*
Please Don't Leave Me by *P!nk*
Funhouse by *P!nk*
Harder To Breathe by *Maroon 5*

Again, we can observe the model is coherent and tend to retrieve songs similar to the seed songs. A song by *Guns N' Roses* retrieved rock songs, while a song by *Maroon 5* retrieved other songs of the same artist.

5.2.2 Similarity between songs based on artists' tags

Here we are going to describe how we used the artists' tags to compute the similarity between songs. Since we are using only the 100 most popular tags, each song s_i is represented by a vector $s_i = (s_i^1, s_i^2, ..., s_i^{100})$ where $s_i^j = 1$ if song s_i has the jth tag. Initially, each song is described with the tags of its artist. Since the artists are represented by a small set of tags, creating a sparse representation for the songs, we enriched the songs representation with the tags of the songs it co-occurs on users' playlists. Given the songs' vectors s_i, the new representation of song s_i will be:

$$x_i = \sum_{p_i \in P : s_i \in p_i} \sum_{\substack{s_j \in p_i \\ s_j \neq s_i}} s_j$$

In other words, for each song s_i, we will represent it as the sum of the songs' vectors it co-occurred with. Therefore, each song will be represented by a vector containing the counts of how many times each tag appeared in the playlists it occurred. In order to represent each song by a probability distribution over the tags, we divided each vector x_i by its sum. We also discarded the songs that didn't receive any tag, since for those songs we don't have any information about it. This process resulted in a set of approximately 2.2 million songs represented by a probability distribution over 100 tags.

Again, we could use any distance function between the tag vectors to compute the similarity between songs. Since representing the songs by 100 genres may cause redundancy, we used SVD to reduce the dimensionality of the vectors using the same procedure we used to reduce the dimensionality of the user's representation on chapter 4. Here, we constructed a matrix M where each row represents a song of our dataset, and each column represents one of the 100 tags. By analyzing the matrix's singular values, we found $k = 5$ was enough to represent 95% of the constructed matrix. Therefore, we represented each song by the first 5 columns of the matrix U. Here we define the **SVD music space** as the set of songs represented by the SVD, where each song is represented by a vector in a 5-dimensional Euclidean space and the distance between songs are also computed using the cosine distance.

Comparing the distance between songs in the SVD music space with their co-occurrence in users playlists using the same playlists used to asses the quality of the Word2vec music space, we obtained the boxplots in figures 5.6 and 5.7. Observing the boxplots we can take the same conclusions as before, that is, when the co-occurrence of the songs on the playlists increases, their distance on the music space decreases.

We also performed a qualitative evaluation using the same three seed songs used before and searched for most similar songs. Before performing the evaluation we removed from the SVD music space the songs that did not appeared at least 50 times on users' playlists (the same *min count* used on word2vec model). This resulted in a set of 46594 songs. The 5 most similar songs returned can be seen below.

Songs similar to *Sweet Child O' Mine* by *Guns N' Roses*:
November Rain by *Guns N' Roses*
Janie's Got A Gun - Single Version by *Aerosmith*
Pour Some Sugar On Me (2012) by *Def Leppard*
Patience by *Guns N' Roses*
Lie To Me by *Bon Jovi*

Songs similar to *How Deep Is Your Love* by *Bee Gees*:
You Should Be Dancing - Edit by *Bee Gees*
The Air That I Breathe by *Simply Red*
More Than A Woman (2007 Remastered Saturday Night Fever) by *Bee Gees*
How Deep Is Your Love (2007 Remastered Saturday Night Fever) by *Bee Gees*
Stuck On You by *Lionel Richie*

Songs similar to *Makes Me Wonder* by *Maroon 5*:
This Time Around by *Hanson*
All The Things She Said by *t.A.T.u.*
Big City Life by *Mattafix*
Complicated by *Avril Lavigne*
Good Girl by *Carrie Underwood*

Analyzing the list of similar songs, we can observe the model is coherent and tend to retrieve songs similar to the seed song. For example, the songs similar to *Guns N' Roses* are rock songs, and a song by *Bee Gees* returned songs that was popular on 70'.

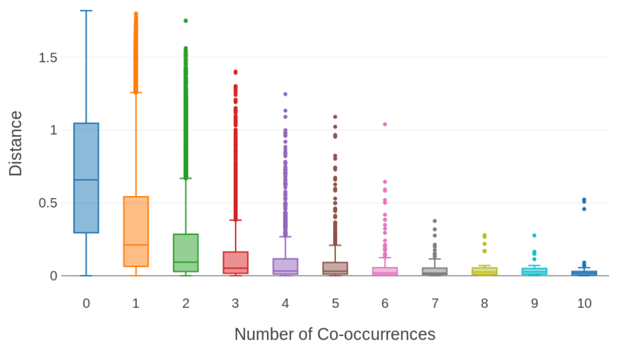


Figure 5.6: Boxplot of distance between songs on SVD music space given number of cooccurrence in AotM dataset

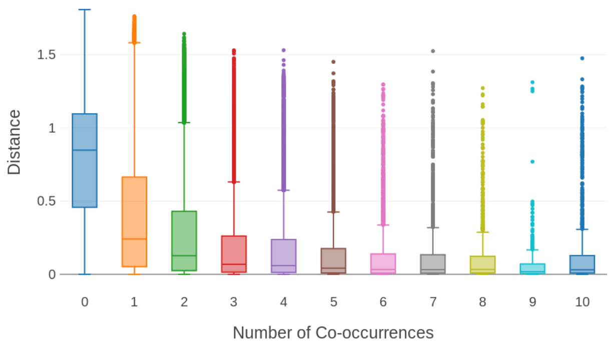


Figure 5.7: Boxplot of distance between songs on SVD music space given number of cooccurrence in Spotify dataset

For each music space we plotted the boxplot of the distance between pairs of songs given their co-occurrence on users playlists and observed, in all music spaces, the distance between the songs decreases as the number of co-occurrence increases, indicating all music spaces are able to group similar songs together. In order to compare the music spaces and determine which better model users behaviour (that is, group songs that users tend to be listened to in the same playlist) we computed the Pearson correlation between the number of co-occurrence and the log of the distance between the songs, since we observed an exponential decay (which is expected, since the distance between songs can not be negative). We can observe, in table 5.1, that the Billboard Music Space better grout songs that co-occur on the Art of the Mix playlists while the Word2vec Music space better groups songs co-occurring in the Spotify Playlists.

To better compare the SVD music space and the Word2vec music space, we removed from the Word2vec music space the songs that was not represented by the SVD model, that is, the songs that didn't receive any tag (which are only 12 songs). Therefore, both models will have 46594 songs.

We end this chapter with the definition of three music spaces: the Billboard music space, the SVD music space and the Word2vec music space. Using these music spaces we will propose a general method to automatically generate music playlists that satisfy the quality constraints described in chapter 2.

	AotM Playlists	Spotify Playlists
Billboard Music Space	-0.3587	-0.1595
Word2vec Music Space	-0.1733	-0.2840
SVD Music Space	-0.2556	-0.2257

Table 5.1: Correlation between songs co-occurrence and log of distance on music spaces.

Chapter 6

Playlists Generators

In this chapter, we are going to propose a general method to automatically generate music playlists. First, we will describe the inputs of the method and how it generates playlists satisfying the qualities presented in Chapter 2. Then, based on the general method, we will propose two algorithms, named ROPE and STRAW, and describe how to use them to generate playlists on the music spaces defined on Chapter 5.

6.1 General Method to Generate Playlists

In this work, we aim to propose a general method to automatically generate music playlists satisfying users' taste. As discussed before, we want to construct an algorithm to generate music playlists satisfying the following five quality constraints: interaction, smooth transition, heterogeneity, novelty, and scalability. The objective of a general method satisfying these qualities is that it can be used as a model to propose different algorithms that can be applied in any music space, where the algorithms derived from it will also satisfy the five qualities constraints.

Here, we will create the general method in steps, where at each step we will point out how the method satisfies one of the desired qualities. In order to satisfy the interaction criteria, we would like to require the minimum input from the user as possible. Therefore, the method will receive as input only the region of the music space the user enjoys, and the time he/she wants to spend listening to the playlist. Since the user may enjoy several different styles (desiring a heterogeneous playlist), he/she should be able to define more than one region of the music space. We will first describe a method that receives from the user only two regions, defined as anchor songs. Since a user may want to define more than two anchor songs, at the end of this chapter, we will show how we can extend the algorithms to receive an arbitrary number of regions, making it possible to create playlists containing several music styles. To control the time the user will spend listening to the playlist we will require the number of songs he/she wants to listen, which implicitly controls the time duration of the playlist. Given these inputs, the method should be able to generate a playlist connecting the anchor songs selected by the user using the specified number of songs. From now on, we will define S as the set of available songs, s_0 as the first anchor song (or seed song), s_d as the second anchor song (or the destination song), and k as the desired length of the playlist. It is important to point out the anchor songs represents regions of the music space and can be replaced by other parameters, such as an artist or a music genre. In case the user specifies a music genre, we can retrieve all songs of that genre and represent it as the mean of the songs vector representation, for example.

Even though a user may enjoy music genres that are, at first, very different, it is annoying to switch suddenly from one genre to another, violating the smooth transition criteria. Thus, the process of selecting songs from s_0 to s_d will be done recursively, where a new song s_{i+1} is added to the playlist if it is similar to the current song s_i. This ensures adjacent songs in the playlist will be similar, avoiding abrupt transitions. Since we want the method to be stochastic (in order to satisfy novelty criteria), the new song to be added to the playlist must be randomly selected from a given set of songs. To satisfy these specifications, the general method will be based on two functions:

1. $F(s_i)$: This function receives as a parameter a seed song s_i and returns a set of eligible songs considered similar to s_i. The size of the set can be fixed and defined beforehand, or it can be defined at the algorithm execution and limited by a constant. Let 2^S be the image of F, that is, all possible sets that can be returned by this function.

2. $c(F(s_i), s_i, s_d)$: This function randomly selects a song from the set $F(s_i) \in 2^S$ satisfying $d(s_i, s_d) > d(s_{i+1}, s_t)$, i.e. the selected song must be closer to the destination song s_d than song s_i.

Using these two functions, the method to generate a music playlist can be constructed as described in Algorithm 1. At each step, the algorithm uses $F(.)$ to retrieve a set of eligible songs $\mathcal{P}$, and randomly selects the next song to be added to the playlist using function $c(.)$. By analyzing the algorithm, the complexity of the proposed method is $k \times (O(F) + O(c))$. If we limit $F(.)$ to retrieve a constant number of eligible nodes, then the complexity of $c(.)$ is constant, or $O(1)$. Therefore, if we have an efficient algorithm to retrieve songs similar to a seed song, the complexity of the algorithm will be small, satisfying the scalability property.

Algorithm 1 Generates a music playlist

```
1: procedure GENERATEPLAYLIST($\mathcal{S}, s_0, s_d, k$)        ▷ a heterogeneous playlist with $k$ songs
2:     $playlist \leftarrow array(k)$                            ▷ The playlist will be an array with $k$ songs
3:     $playlist[0] \leftarrow s_0$                              ▷ First song of the playlist
4:     $i \leftarrow 0$
5:     while $i < k - 1$ do
6:         $s_i \leftarrow playlist[i]$                          ▷ Last song added to the playlist
7:         $\mathcal{P} \leftarrow F(s_i)$                       ▷ set of eligible songs
8:         $playlist[i+1] \leftarrow c(\mathcal{P}, s_i, s_d)$   ▷ add a new song to the playlist
9:         $i \leftarrow i + 1$
       return $playlist$
```

6.2 Algorithms to Generate Music Playlists

In this section, we are going to describe two algorithms to generate music playlists constructed based on the general method described in Algorithm 1. The first algorithm, named ROPE, generates a Brownian path that connects the two anchor songs defined by the user. The second one, named STRAW, performs a steered random walk on a music similarity graph constructed using a music space.

6.2.1 ROPE

The first proposed algorithm to generate a music playlist based on the general method is called Brownian Path Generator (ROPE). Given an initial song s_0, a final song s_d, and the desired number of songs in the playlist k, ROPE generates a random path in the music space from s_0 to s_d with $k-2$ intermediate points connected by a sequence of $k-1$ line segments. Then, for each of the path's intermediate points, ROPE selects the closest song of the music space and allocates it to that position, creating a playlist with exactly k songs. To generate the playlist, ROPE uses a d-dimensional Euclidean space where each song is represented by a point in this space.

To generate the random path, ROPE uses a standard Brownian motion process (also called Wiener process) with discrete steps [Durrett, 2010]. The Wiener process is a stochastic process $W(t) : t \in \mathbb{R}^+$ such that all its realizations are continuous functions, with $W(0) = 0$ with probability 1, and with independent increments ($W(t+u) - W(t)$ is independent of $W(s)$ for $0 < s < t$) which are Gaussian ($W(t+u) - W(t) \sim \mathcal{N}(0, \sigma)$). Here, we create a discrete Wiener process in a (d-1)-dimensional Euclidean space with $k-1$ steps, where $W(0) = (0, 0, \ldots, 0)_{(d-1)}$, and $W(t+1) - W(t) \sim \mathcal{N}(\mu, \Sigma)$, where $\mu = (0, 0, ..., 0)_{d-1}$ is the mean and Σ is the covariance matrix of the multidimensional normal distribution. Although the songs are in a d-dimensional Euclidean space, we generate the Wiener process in a (d-1)-dimensional space because we use the discrete time t as the first coordinate of the points in the d-dimensional space, forcing the path to go to a specific direction. This process results in a sequence of k points $p_0, p_1, \ldots, p_{k-1}$

representing a random path in a d-dimensional Euclidean space connecting the points p_0 and p_{k-1}. After generating the path, we just need to transform it in order to make it connect the songs s_0 and s_d.

The transformation of the path $p_0, p_1, \ldots, p_{n-1}$ in a path $\hat{p}_0, \hat{p}_1, \ldots, \hat{p}_{k-1}$, where $\hat{p}_0 = s_0$ and $\hat{p}_{k-1} = s_d$, will be done in three steps: a sequence of rotations, followed by a scale and translation operation. To better explain how ROPE generates the playlists, first we will explain how it works in a 2-dimensional Euclidean space, that is, each point p_i and each song s_i is represented by two coordinates (x, y). Let's define a point p_i of the path as $p_i = (p_i^x, p_i^y)$ and a song s_i of the music space as $s_i = (s_i^x, s_i^y)$. The first step of the transformation, which are rotations, will be done to make $(p_{k-1} - p_0) = \alpha(s_d - s_0)$, that is, the vector connecting p_0 to p_{k-1} will have the same direction of the vector connecting s_0 and s_d. Since we are on a 2-dimensional Euclidean space, we will only need to perform one rotation, applying to each point the transformation matrix

$$M = \begin{bmatrix} \cos(\theta) & \text{-}\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

where θ is the angle of rotation

$$\theta = \arctan\left(\frac{s_d^y - s_0^y}{s_d^x - s_0^x}\right) - \arctan\left(\frac{p_{k-1}^y - p_0^y}{p_{k-1}^x - p_0^x}\right)$$

After the rotation, we perform a scale operation (in order to make $(p_{k-1} - p_0) = (s_d - s_0)$) followed by a translation operation, creating a path $\hat{p}_0, \hat{p}_1, \ldots, \hat{p}_{k-1}$, where $\hat{p}_0 = s_0$ and $\hat{p}_{k-1} = s_d$. Figure 6.1 illustrates how ROPE performs these operations.

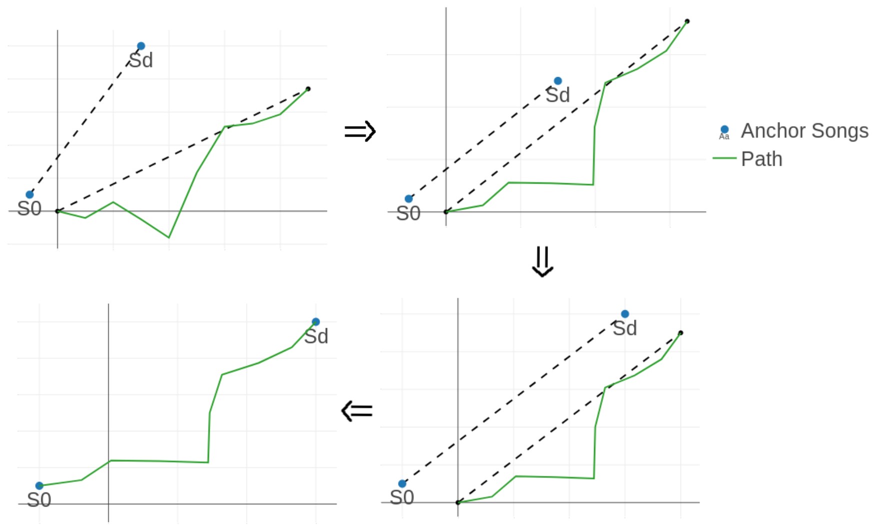


Figure 6.1: Illustration of ROPE algorithm

In order to adapt ROPE to generate a playlist in a d-dimensional Euclidean space, after generating the random path $p_0, p_1, \ldots, p_{k-1}$, we need to perform $d-1$ rotations to make $(p_{k-1} - p_0) = \alpha(s_d - s_0)$. Let's suppose $p_i = (p_i^0, p_i^1, \ldots, p_i^{d-1})$ and $s_i = (s_i^0, s_i^1, \ldots, s_i^{d-1})$. If $(p_{k-1} - p_0) = \alpha(s_d - s_0)$, then $(p_{k-1}^j - p_0^j) = \alpha(s_d^j - s_0^j)\ \forall\ j$. If we fix the coordinates s_0^0, s_d^0, p_0^0 and p_{k-1}^0, than we must have $\alpha = (p_{k-1}^0 - p_0^0)/(s_d^0 - s_0^0)$. Therefore, we will make $d-1$ rotations where the jth rotation will make

$$\frac{s_d^j - s_0^j}{s_d^0 - s_0^0} = \frac{p_{k-1}^j - p_0^j}{p_{k-1}^0 - p_0^0}$$

The jth rotation will be performed by applying the the points p_i the rotation matrix M_j where

$$M_j = \begin{array}{c} \\ \left[\begin{array}{cccccc} 0 & 1 & \ldots & j & \ldots & d-1 \\ \cos(\theta_j) & 0 & \ldots & -\sin(\theta_j) & \ldots & 0 \\ 0 & 1 & \ldots & 0 & \ldots & 0 \\ 0 & 0 & \ldots & 0 & \ldots & 0 \\ \sin(\theta_j) & 0 & \ldots & \cos(\theta_j) & \ldots & 0 \\ 0 & 0 & \ldots & 0 & \ldots & 0 \\ 0 & 0 & \ldots & 0 & \ldots & 1 \end{array}\right] \end{array} \begin{array}{c} \\ 0 \\ 1 \\ \vdots \\ j \\ \vdots \\ d-1 \end{array}$$

where θ_j is the angle of rotation

$$\theta_j = \arctan\left(\frac{s_d^j - s_0^j}{s_d^0 - s_0^0}\right) - \arctan\left(\frac{p_{k-1}^j - p_0^j}{p_{k-1}^0 - p_0^0}\right)$$

There are two important observations to make about the rotation operations. The first is that since the first point p_0 is the origin of the Euclidean space ($p_0 = (0, 0, \ldots, 0)$), the rotations never change it. The second observation is that after the jth rotation, the value of p_{k-1}^0 will be modified. Let $(p_i)_j$ be the value of p_j after the jth rotation. We have that

$$(p_{k-1}^0)_j = \cos(\theta_j) \cdot (p_{k-1}^0)_{j-1} - \sin(\theta_j) \cdot (p_{k-1}^j)_{j-1}$$

Since we modified the value of p_{k-1}^0, the equality $(s_d^l - s_0^l)/(s_d^0 - s_0^0) = (p_{k-1}^l - p_0^l)/(p_{k-1}^0 - p_0^0)$ for $l < j$ will not be satisfied anymore. To bypass this problem, after the jth rotation, we calculate the ratio

$$r_j = \frac{(p_{k-1}^0)_j}{(p_{k-1}^0)_{j-1}}$$

and multiply p_i^l for all i and $l = 1, \dots, j-1$ by r_j, ensuring the equalities will be satisfied. In other words, we transform the points using the following matrix:

$$S_j = \begin{array}{c} \begin{array}{cccccccc} 0 & 1 & 2 & \dots & j-1 & j & \dots & d-1 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & r_j & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & 0 & r_j & \dots & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \ddots & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & r_j & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 & \dots & 1 \end{bmatrix} \end{array} \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ \vdots \\ j-1 \\ j \\ \vdots \\ d-1 \end{array}$$

After these $d-1$ rotations, we only need to perform a scale transformation followed by a translation, as we did in the 2-dimensional Euclidean space, generating a sequence of k points $\hat{p}_0, \hat{p}_1, \dots, \hat{p}_{k-1}$ where $\hat{p}_0 = s_0$ and $\hat{p}_{k-1} = s_d$.

With a random path connecting songs s_0 and s_d, for each of the $k-2$ intermediate points we just need to search in the music space for the song closest to point p_i and add it to the playlist at that position. After replacing each intermediate point by a song of the music space, we will have a playlist connecting songs s_0 and s_d.

ROPE works as described in algorithm 1, with some slight differences. We first randomly generate $k-2$ intermediate points, and then use $c(.)$ to select the songs closest to them. However, the point p_{i+1} depends on the point p_i, since $p_{i+1} = p_i + \mathcal{N}(0, \Sigma)$. Function $c(.)$ can simply go through all the songs of the music space to select the song closest to the specified point. However, since the dataset may contain a large number of songs, this may cause the method to be slow. To solve this problem, we can use a k-d tree to store the songs and perform nearest neighbor searches on it, which has an average complexity of $O(\log(N))$.

Note the randomness of the algorithm is controlled by the covariance matrix Σ. If we increase the variance of the coordinates, the method will be able to generate more diverse playlists. If we set the variances to 0, the method will be deterministic, becoming similar to the algorithm proposed by Flexer et al. [2008], which selects equally spaced songs to be added to the playlist.

6.2.2 STRAW

The second proposed algorithm to generate music playlists is the Steered Random Walker (STRAW). Although ROPE is able to generate directed random paths from s_0 to s_d with exactly $k-1$ steps, it is not topology-aware, that is, it doesn't know the structure of the music space. Because of that, ROPE may generate a path crossing a sparse or even empty region between s_0 and s_d in the music space (S, d), possibly populating the playlist with harsh transitions. This is the motivation behind STRAW, a topology-aware heterogeneous playlist generator based on similarity graphs.

STRAW is an algorithm that uses a music similarity graph, where the nodes represent songs in our dataset and the edges connect songs with a certain degree of similarity. There are several forms to construct this graph. For example, one can construct a graph where songs are connected if they co-appeared on users' playlists a minimum number of times, or connect songs with a distance lower than a threshold in a music space. Given the similarity graph, the seed song s_0, the destination song s_d and the desired number of songs k, STRAW performs a directed random walk in the graph from s_0 toward s_d. Starting from song s_0, STRAW randomly selects the next song of the playlist from the neighbors of the current song. We call this walk directed because at each step STRAW gives higher probabilities to visit songs closer to the destination.

Differently from ROPE, STRAW cannot guarantee the song s_d will be reached in exactly k steps. Nevertheless, it guarantees that the playlist will have exactly k songs and that s_d is on it. If song s_d is reached before k steps, then a random walk on the graph from node s_d is initiated until the playlist length equals k. The random walk simply selects a random song from the neighbors of the current song to be the next song of the playlist. If s_d is not reached with k steps from s_0, the directed random walk proceeds until s_d is reached, generating a playlist p' with $k' > k$ songs. After that, we remove from p' $k' - k$ songs, creating a playlist with the desired number of songs. Selecting the songs to be removed from the playlist can be defined by the implementer and may depend on the desired qualities of the playlist.

Given the similarity graph $G(V, E)$, STRAW works as described in Algorithm 1. Since the navigation structure is a graph, retrieving the set of songs that could be added after song s_i using $F(s_i)$ is straightforward. $F(s_i)$ simply outputs the set P of songs that are connected to the correspondent node s_i in G. If we limit the degree of the nodes by an upper bound, the complexity of $F(.)$ is $O(1)$. The function $c(.)$ needs to randomly select a song from the set of songs retrieved by $F(.)$, giving higher probability to songs closer to the destination song s_d. The function $c(.)$ can be defined by the implementer and may depend on the music space the algorithm is being applied.

6.3 Application of the Algorithms

Here we will detail the implementation of the proposed algorithms to generate music playlists on the music spaces constructed in Chapter 5. First, we will explain the application of ROPE and detail how it was used to generate playlists on the music spaces where the distance between songs are calculated using the cosine similarity. Then, we will present how we constructed a music similarity graph using the music spaces ans used STRAW to generate a music playlist.

6.3.1 Application of ROPE

When applying ROPE on a music space, if the distance function between songs is computed using the Euclidean distance, the application of the algorithm is straightforward, since we only need to generate a Wiener process and transform it to connect the songs s_0 and s_t defined by a user. This is the case for the Billboard music space, where the songs are embedded in a 2D Euclidean space. In other words, for this music space, we only need to generate a Wiener process and perform a rotation followed by a scalar and translation transformation as described before.

For the Word2vec and SVD music spaces, ROPE may not be suitable, since the distances between songs are computed using the cosine distance instead of the Euclidean distance. If we have two vectors $\vec{a}$ and $\vec{b}$ where $\vec{a} = \alpha\vec{b}$ for $\alpha \neq 1$, although the cosine distance is equal to 0 (which means the songs are similar), their Euclidean distance may inform they are very different. One thing we can do to bypass this problem is to embed the songs in a Euclidean space using an embedding algorithm like MDS or t-SNE. Instead of that, we will normalize the songs vector representation and show that, if two vectors have the same norm, the square of the Euclidean distance between them will be proportional to the cosine distance.

First, note that given two vectors $\vec{a}$ and $\vec{b}$, if we multiply one of them by a scalar α, the cosine of the angle between them will not change, that is:

$$\frac{< \vec{a}, \alpha\vec{b} >}{||\vec{a}||||\alpha\vec{b}||} = \frac{\alpha < \vec{a}, \vec{b} >}{\alpha||\vec{a}||||\vec{b}||} = \frac{< \vec{a}, \vec{b} >}{||\vec{a}||||\vec{b}||}$$

Now let's suppose we want to calculate the distance between two points represented by the vector $\vec{a}$ and $\vec{b}$. The square of the distance is equal to:

$$||\vec{b} - \vec{a}||^2 = ||\vec{a}||^2 + ||\vec{b}||^2 - 2||\vec{a}||||\vec{b}||cos\theta$$

where θ is the angle between the two vectors. If we scale both vectors to have norm equal to $\frac{1}{\sqrt{2}}$, we will have:

$$||\vec{b} - \vec{a}||^2 = \left(\frac{1}{\sqrt{2}}\right)^2 + \left(\frac{1}{\sqrt{2}}\right)^2 - 2\left(\frac{1}{\sqrt{2}}\right)\left(\frac{1}{\sqrt{2}}\right) cos\theta$$

$$||\vec{b} - \vec{a}||^2 = 1 - cos\theta$$

in other words, if we scale all the vectors to have a length equal to $\frac{1}{\sqrt{2}}$, the Euclidean distance between the songs will be equal to the cosine distance.

One important observation to make is that when we scale all songs to have norm equal to $\frac{1}{\sqrt{2}}$, they become points of a hypersphere on a Euclidean space with its center in the origin. In such music space, generating a playlist becomes the same as randomly walking in the spherical shell. Therefore, given a point p_i from the random path, before searching for its nearest song, we also scaled it to have norm equal to $\frac{1}{\sqrt{2}}$.

To complete the explanation of ROPE application, in our experiments, when generating the Wiener process, we set Σ as the identity matrix, that is, a multivariate normal distribution where the variables are not correlated and have variance 1.

6.3.2 Application of STRAW

As described before, STRAW generates a playlist by performing a random walk on a music similarity graph. Before explaining the application of STRAW, we will describe how we constructed a graph based on the music spaces constructed on Chapter 5. For all the three music spaces, we can calculate the distance between each pair of songs. On the Billboard music space, the distance can be computed using the Euclidean distance between their representation, while on the Word2vec and SVD music spaces, the distance can be computed using the cosine distance $(1 - \cos(\theta))$. We can then create a song similarity graph by connecting the most similar songs of the music space.

The process of constructing the similarity graph is as follow. We start with a complete weighted graph $G' = (V, E)$ where all pair of songs are connected and the weight of the edge connecting s_i and s_j is the distance $d(s_i, s_j)$ between them. Using this graph, we create a minimum spanning tree (MST) represented by a graph $MST_1(V, E_1)$. We then remove from the original complete graph G' all edges E_1 and generate a second MST $MST_2(V, E_2)$ with the remaining sub-graph $G' = (V, E \backslash E_1)$. We keep the process of removing the edges E_i from G' and creating MSTs until we have κ MSTs $MST_i(V, E_i)$, $1 \leq i \leq \kappa$. We then define the music similarity graph G as the union of the MSTs, that is, $G = (V, \bigcup_{i=1}^{\kappa} E_i)$. This graph has a guaranty that every song is adjacent to at least κ other songs. After this process, since many pairs of similar songs may be out of E, we select from E_κ the edge with the maximum weight τ. We then create a sorted list $l_e = (e_1, e_2, \ldots)$ containing all edges with weight lower than or equal to τ and iteratively, starting from e_1, add edge $e_i = (u, v)$ to the graph G if nodes u or v have degree smaller than L_d. After this process, every song is connected to at least L_d.

Given the graph G, we only need to define how the function $c(.)$ selects a song from the set $F(s_i)$. As described before, the function $c(.)$ needs to randomly select a song satisfying $d(s_i, s_d) > d(s_{i+1}, s_d)$. Here we propose two methods to satisfy this restriction. The first one uses the songs vector representation, and filter the set of songs returned by $F(s_i)$ selecting only the ones that are closer to song s_d than the current song s_i, creating a set of eligible songs $\mathcal{P}$. Thus, function $c(.)$ only needs to select a random song from $\mathcal{P}$. Since we want to generate a playlist with $k-1$ steps, we should give higher probabilities to nodes that do not advance too much or too little in the path. Therefore, we define the desired step size $\hat{\delta}$ in a playlist with $k-1$ steps from s_0 to s_d as $\hat{\delta} = d(s_0, s_d)/(k-1)$.

For each node $v \in \mathcal{P}$, we calculate the step size δ_v that will be given toward s_d if v is selected: $\delta_v = d(s_i, s_d) - d(v, s_d)$. We can give to each node v a likelihood Θ_v of being selected by $c(.)$, which is $\Theta_v = (1 + |\delta_v - \hat{\delta}|)^{-1}$. Θ_v is proportional to how close the step δ_v is to the desired step $\hat{\delta}$. With that, the probability of selecting node $v \in \mathcal{P}$ is given by:

$$P(v) = \frac{\Theta_v}{\sum_{u \in \mathcal{P}} \Theta_u}$$

Thus, function $c(.)$ selects the next song to be added to the playlist by drawing a random song from $\mathcal{P}$, where node $v \in \mathcal{P}$ has probability $P(v)$ of being selected. Since even defining these probabilities we can not guarantee to reach song s_d in exactly $k-1$ steps, we still need to perform a random walk or remove songs from the playlist as described before. We will simply call this method of selecting songs based on the step size as Straw.

Another form to satisfy the restriction $d(s_i, s_d) > d(s_{i+1}, s_d)$ is to not consider the vector representation of the songs, and use only the similarity graph. In this case, the distance between two songs is the length of the shortest path connecting them, considering all edges of the similarity graph having weight 1. When selecting the next song to be added to the playlist, we can filter the songs returned by $F(s_i)$ in which the distance to song s_d is lower than the distance from s_i to to song s_d. This ensures the set $\mathcal{P}$ will contain only songs closer to the destination song. Given this set of songs, we can assign equal probability to each of them, and randomly select a song to be the next song added to the playlist. To do not need to compute at each step the length of the shortest path from each adjacent song to song s_d, we can pre-compute the shortest path of all songs to s_d by simply running a Breadth First Search (BFS) from s_d. From now on we will call this method as StrawBFS.

In our experiments, to generate the similarity graph we used a value of $\kappa = 5$ and $L_d = 20$, that is, we first represented the graph as the union of 5 MSTs, and then added edges to the graph until all vertices have degree at least 20.

All the algorithms presented in this chapter receives as input from the user only two regions of the music space: the start song s_0 and the destination song s_d. As we discussed before, some users may want to specify more than two anchor songs to create a more heterogeneous playlist. For example, one may want to start listening classic songs, going towards pop, and finally listens to rock songs. A simple form to adapt our algorithms for this situation is, when receiving n anchor songs $s_0, \ldots, s_{n-1}$, create $n-1$ playlists, connecting s_0 to s_1, s_1 to s_2, $\ldots$, s_{n-2} to s_{n-1}. After generating the playlists we can concatenate them to form the final playlist.

We can construct another variant of the algorithm requiring only the seed songs s_0. This way, the user does not need to input the destination song s_d. In this variant, the function $c(.)$ only needs to return a random song from the set $F(s_i)$, not having a destination region of the music space. This approach still guarantee smooth transitions (since function $F(s_i)$ returns songs similar to s_i), but may head to a region of the music space not enjoyed by the user.

In this chapter we proposed a general method to automatically generate music playlists, designed two algorithms based on this method, and illustrated their application on the music spaces created on Chapter 5. On next chapter we will present the experiments performed to asses the qualities of the algorithms, and verify how well the algorithms satisfy the quality metrics presented in Chapter 2.

Chapter 7

Metrics and Experiments

In this chapter, we are going to describe the experiments performed to evaluate the proposed algorithms ROPE and STRAW. In Chapter 2 we presented five qualities a playlist generator algorithm should satisfy: interaction, smooth transition, heterogeneity, novelty, and scalability. To evaluate the algorithms, first, we will propose evaluation metrics to asses the qualities of a playlist regarding these desired characteristics. Then we are going to describe the baseline algorithms that were implemented to be compared with our methods, and finally, use the proposed metrics to compare the playlists generated by the implemented algorithms.

7.1 Evaluation Metrics

The evaluation of music similarity and playlists qualities are not well established in the literature, since there is no ground truth to compare, and a measure of how similar two songs are can be subjective and vary depending on the user. Some authors use a subjective evaluation by analyzing the retrieved songs given a seed song [Zadel and Fujinaga, 2004]. Logan and Salomon [2001] computed the average number of the top k songs that are from the same artist, album or genre of the seed song. When evaluating playlists, some authors implement an online system to let users evaluate the created playlists [Barrington et al., 2009; Cardoso et al., 2016].

In Chapter 6, when we described the application of the general method to generate playlists, we have shown our method requires only a few inputs from the user, and also scales with the size of the music dataset, satisfying scalability, and interaction criteria. In this section, we are going to propose four metrics to measure the effectiveness of the algorithms with respect to smooth transition, heterogeneity, and novelty.

When generating a playlist, we desire consecutive songs to be similar to each other, satisfying the smooth transition criteria. To measure the transition between the songs of the playlist, we proposed two metrics. The first metric, called $ST1$ (smooth transition 1) measures the mean distance between consecutive songs. That is, given a playlist $p = \{s_0, s_1, \ldots, s_{k-1}\}$ of size k, $ST1$ is calculated as:

$$ST1 = \frac{\sum_{i=0}^{k-2} d(s_i, s_{i+1})}{k-1}$$

The second metric to measure the smooth transition, called $ST2$ (smooth transition 2) measures the most abrupt transition between songs, calculated as:

$$ST2 = \max_{0 \leq i \leq k-2} d(s_i, s_{i+1})$$

For both $ST1$ and $ST2$ metrics, the smaller the returned value the smooth is the transition between the songs of the playlist.

To measure the heterogeneity of a playlist, we proposed a metric called HC (heterogeneous coefficient) that measures the longest distance the playlist traveled in the music space and is calculated as the maximum distance between any two songs in the playlist, that is

$$HC = \max_{\substack{0 \leq i \leq k-2 \\ i < j \leq k-1}} d(s_i, s_j)$$

where the higher the value of HC, the more heterogeneous is the generated playlist.

And finally, to measure the novelty of a playlist generator method, we proposed a metric called RC (repetition coefficient) that compares two playlists generated using the same input parameters, that is, the same start song s_0, end song s_d, and the same playlist size k. To compare the playlists, we used the Jaccard similarity coefficient. Given two playlists $p = \{s_0, \ldots, s_{k-1}\}$ and $\hat{p} = \{\hat{s}_0, \ldots, \hat{s}_{k-1}\}$, we created a set with their songs $S = \{s_i : s_i \in p\}$ and $\hat{S} = \{s_i : s_i \in \hat{p}\}$, removed from the sets the songs s_0 and s_d (since they were used as input to the method and therefore are in both playlists), and calculated RC as

$$RC = \frac{S \cap \hat{S}}{S \cup \hat{S}}$$

Since we want a method that is able to generate different playlists for the same input, lower values of RC is desired.

Using the metrics $ST1$, $ST2$, HC and RC we are able asses the qualities of the playlists generated and compare the proposed methods between then and with the methods proposed on literature.

7.2 Baseline Algorithms

To compare the proposed algorithms with former methods, we implemented two algorithms found in the literature. The first is the one proposed by Flexer et al. [2008], that generates a music playlists given the start and end song, similar to the general method proposed in this work. The implementation of this method works as follows: first, we connected the first and end song of the playlist with a straight line in the music space. Then we selected $k-2$ equally spaced intermediate points of this line, and for each point searched for the nearest song in the music space using a k-d tree, creating a playlist of size k. We will call this method as *Flexer*.

The second algorithm implemented is the one proposed by Pontello et al. [2017], which is an algorithm to navigate large media collections, in our case a music space. In this method, starting at song s_0, at each step, a set K of the songs most similar to the current song s_i is retrieved. Then, given a target direction vector $\vec{V}$, a song is randomly selected from this set, where the probabilities of selecting a song s_j is inversely proportional to the angle between the vector $s_j - s_i$ and $\vec{V}$. After selecting a song, the direction vector $\vec{V}$ is updated before used to select the next song. In this method, we defined a destination song s_d which is also used to update the direction vector $\vec{V}$. In our implementation, given the input s_0, s_d and k, we used the same graph constructed for STRAW to retrieve the set of similar songs and used the destination song s_d to guide the navigation. We kept the process of selecting the next song, and after reaching song s_0, we randomly removed songs or performed a random walk so the generated playlist will have k songs. Therefore, the implementation of this algorithm is similar to STRAW, where the difference is only on the probabilities assigned to the songs in the set of similar songs K. It's important to point ou that in the original algorithm proposed by Pontello et. al. the user constantly interacting with the system and can give feedback about the songs being recommended and reject the ones he doesn't like. In our implementation, we are considering the user will accept all recommended songs. From now on, we will call this method as *Pontello*.

Besides these two methods, we implemented two other approaches to generate playlists. One is a random walk on the music similarity graph starting from song s_0, as proposed by Maillet et al. [2009] and Ragno et al. [2005]. The difference from the previews implementations is that this method does not require to reach the destination song s_d. Since it uses the similarity graph (and therefore keeps smooth transitions between songs), it will be able to show the effect of the large steps required by ROPE and STRAW to reach song s_d. We will call this method as *RWalk*. The other implemented method is a completely random playlist, where each of the $k-2$ songs between s_0 and s_d is uniformly selected from the set of available songs S. This method will be called *Random*, and will give us an idea of the step size that can be taken on the music space.

Finally, in all the algorithms implemented, we didn't allow a song that has already been added to the playlist to be selected again. Using this restriction we ensured all the songs of the playlist are different from one another.

7.3 Experiments

In this section, we are going to describe the experiments performed to compare the implemented algorithms. As described in Chapter 5, we constructed three music spaces: the Billboard music space, the Word2vec music space, and the SVD music space. Since each of them were constructed using different features, we are going to evaluate them separately. In all the experiments, we created playlists of size $k = \{25, 50, 75, 100\}$.

7.3.1 The Billboard Music Space

The first experiments we are going to describe uses the Billboard music space. As described, the Billboard music space was constructed using acoustic characteristics extracted from the audio files. To perform the experiments, we randomly selected 1000 pairs of songs to be used as input to the algorithms. Since our goal is to generate heterogeneous playlists, we required that the distance between the first and last song of the playlist to be high. Since the diameter (the distance between the two furthest songs) in the Billboard music space is approximately 18, in our experiments we forced that the distance between s_0 and s_d to be at least 6, which is one-third of the diameter. After generating all playlists, for each algorithm and each playlist size we calculated the mean of the metrics found for all playlists.

Figure 7.1 shows the $ST1$ of the playlists generated. As expected, Random obtained the maximum values of $ST1$, since the method randomly selects songs from the dataset, without worrying about the transition between songs. Since StrawBFS finds the "shortest" path to the destination song s_d and then performs a random walk to complete the playlist, its results are similar to RWalk. Our implementation of Pontello uses the similarity graph to generate the playlist and therefore has a result similar to

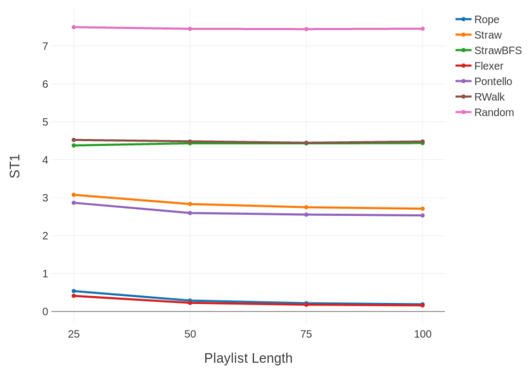


Figure 7.1: ST1 of playlists in the Billboard music space

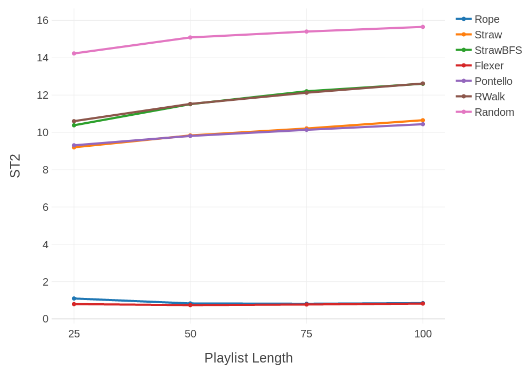


Figure 7.2: ST2 of playlists in the Billboard music space

STRAW. Flexer and Rope obtained the best results since both of then only give steps of size enough to arrive at the destination point. With respect to $ST2$, we can see on figure 7.2 Random also obtained the worst results, while Flexer and Rope showed to perform better.

In Figure 7.3 we can see the HC metric for the playlists. Since Random selects at each step a random song to be added to the playlist, it obtained the best values of HC. StrawBFS and RWalk also obtained similar results for HC, as they tend to randomly walk on the similarity graph and therefore goes to regions distant from the start song s_0. Since Rope and Flexer tend to select songs in the middle of the start and end songs, their HC tend to be the distance from the start and end song, obtaining equal values.

When calculating RC we can observe in Figure 7.4, as expected, that Flexer obtained a value of 1, since it is deterministic. To better visualize the results of the other algorithms, we plotted another graph without Flexer results, that can be seen in Figure 7.5. We can observe Random obtained the best results, as expected, and that RWalk also obtained results better than the proposed algorithms. Straw and Pontello tend to increase the RC of the playlists when increasing the size of the playlist, while StrawBFS decreases it. This probably happens because when StrawBFS reaches the destination song, it starts a random walk, creating a different playlist. Rope, on the other hand, increases fast when increasing the size of the playlist. When generating a random path connecting the start and end song with many intermediate points, most of the points fall in the same region of the music space, and therefore the algorithm selects the same songs selected in the previous iteration.

Summarizing the results obtained for the Billboard music space, Pontello performed similar to Straw, as it uses the same music similarity graph. Rope and Flexer

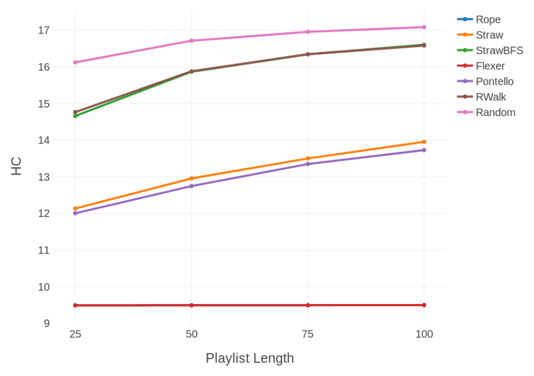


Figure 7.3: HC of playlists in the Billboard music space

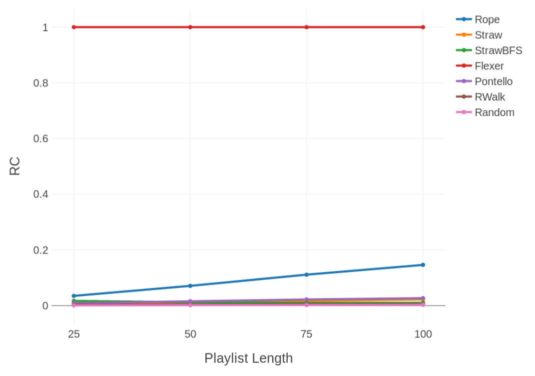


Figure 7.4: RC of playlists in the Billboard music space

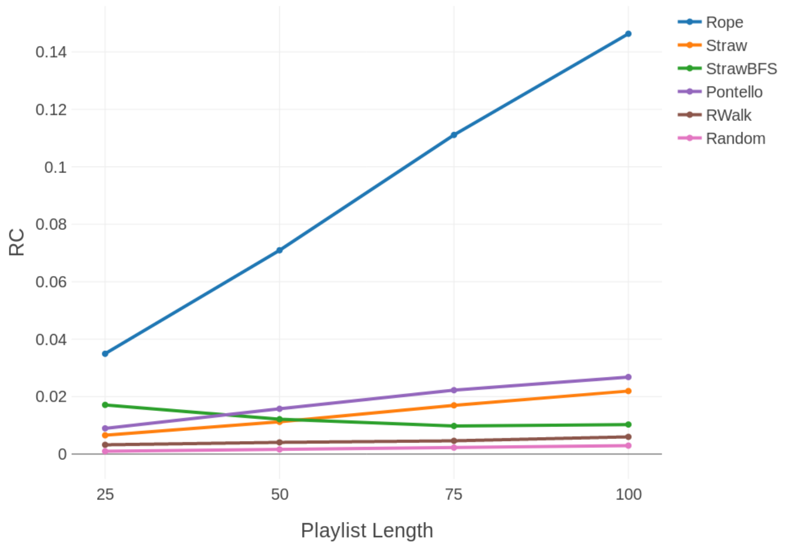


Figure 7.5: RC of playlists in the Billboard music space without Flexer

obtained the best values of $ST1$ and $ST2$, but a small value of HC. Besides that, Flexer is deterministic and obtained a value of RC equal to 1. Although StrawBFS and RWalk obtained similar results, StrawBFS receives as input a destination song, allowing the user to have a better control over the playlist generated. Random obtained the best values of HC and RC, but the worst values of $ST1$ and $ST2$. Based on these results, if the user prefers a playlist with smooth transition, it's recommended to use ROPE algorithm. But if he desires a more heterogeneous playlist, it's recommended to use Straw, StrawBFS, or Pontello.

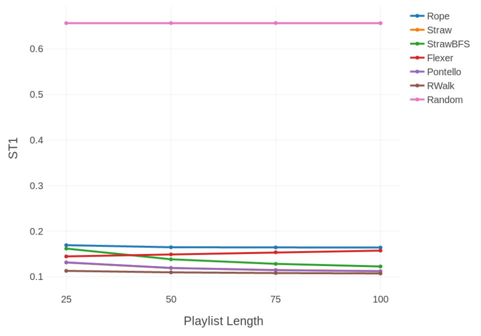


Figure 7.6: ST1 of playlists in the Word2vec music space

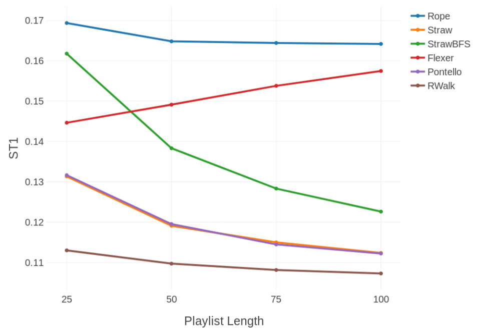


Figure 7.7: ST1 of playlists in the Word2vec music space without Random

7.3.2 The Word2vec Music Space

Here we are going to describe the experiments using the Word2vec Music Space. For this music space and the SVD music space, we used the users of the test set to run the experiments. For each user u, we created a set of songs S_u containing all songs from his/her playlists and selected the 500 users with the highest values of $|S_u|$. For each selected user u, we extracted 20 random songs from the set S_u and grouped the songs in pairs, where one song was used as s_0, and the other as s_d. Therefore, for each user u, we generated 10 pairs of songs to use as input to the algorithms, totaling 5000 pairs of songs. As we did for the Billboard dataset, for each algorithm and each playlist size we calculated the mean of the metrics found for all playlists.

In Figure 7.6 we can see the mean of $ST1$ metric of the playlists generated. Again, as expected, Random obtained the highest values. To better compare the other algorithms, we plotted a new graph without the results of Random algorithm, which can be seen on figure 7.7. As we can observe, RWalk, Pontello and Straw obtained the smallest values, Although Rope obtained $ST1$ values small compared with Random algorithm, it obtained the worst results among the proposed algorithms. We can observe that restricting the songs to be adjacent on the similarity graph forces the playlists to have smooth transition between songs, and selecting songs using a random path connecting the start and end song trying to can cause abrupt transitions.

In Figure 7.8 we can see the $ST2$ metric for the generated playlists. We can see that on the Word2vec music space Rope obtained values better than Flexer and, unlike the Billboard music space, it obtained $ST2$ values greater than Straw, StrawBFS, Pontello and RWalk. Again, Pontello obtained $ST2$ values similar to Straw. For the HC metric, we can see on figure 7.9 Rope was able to obtain results similar to Flexer, but not as good as Straw, StrawBFS and Pontello.

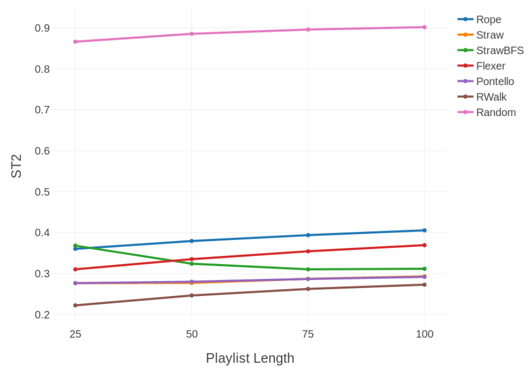


Figure 7.8: ST2 of playlists in the Word2vec music space

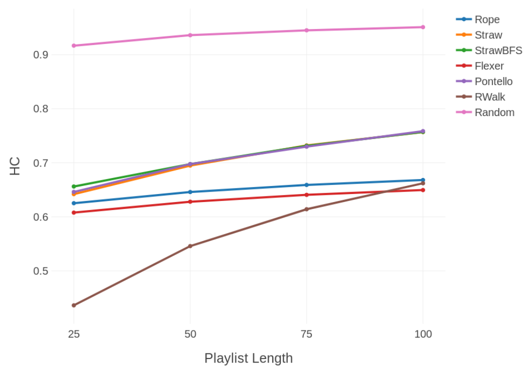


Figure 7.9: HC of playlists in the Word2vec music space

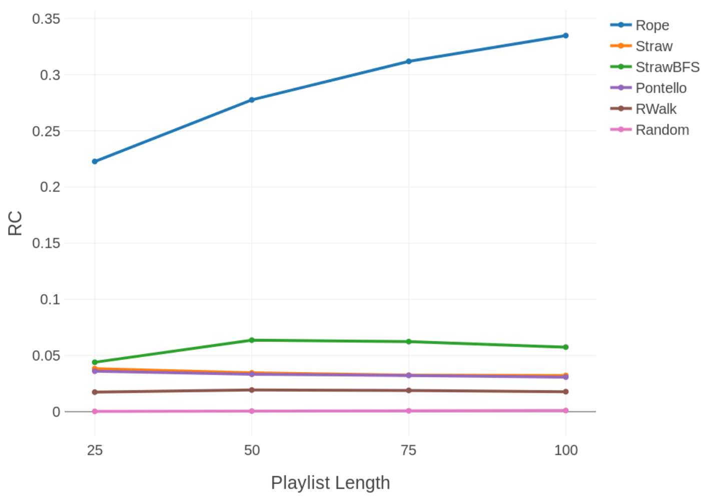


Figure 7.10: RC of playlists in the Word2vec music space

Figure 7.10 shows the mean of RC metric of the playlists. Since Flexer always obtain an RC value equal to 1, we didn't plot its results. As we can observe, ROPE obtained the worst values between the proposed algorithms, increasing RC value with the size of the playlist as in the Billboard music space. All the other algorithms obtained RC values small compared with ROPE and Flexer. Straw generated playlists with RC values close to Pontello, and StrawBFS obtained values greater than RWalk.

Summarizing the results of the Word2vec music space, among the proposed algorithms, Straw, StrawBFS and Pontello obtained the best results, with small values of $ST1$, $ST2$ and RC, and a high value of HC.

7.3.3 The SVD Music Space

The last music space used in our experiments were the SVD music space. To perform the experiments we used the same inputs used on the Word2vec music space, that is, we generated 10 playlists for each of the 500 different users selected from the test set. Analysing Figures 7.11, 7.12, 7.13 and 7.14 we can observe the results are very similar to those found on the Word2vec music space. This happens because in both music spaces, the songs are represented as vectors of size $\frac{1}{\sqrt{2}}$ and the distance between the songs are the as the cosine distance. We can therefore conclude again that among the proposed algorithms, Straw and StrawBFS are the best option to generate music playlists, as it is able to create playlists with high values of HC, but low values of $ST1$, $ST2$ and RC.

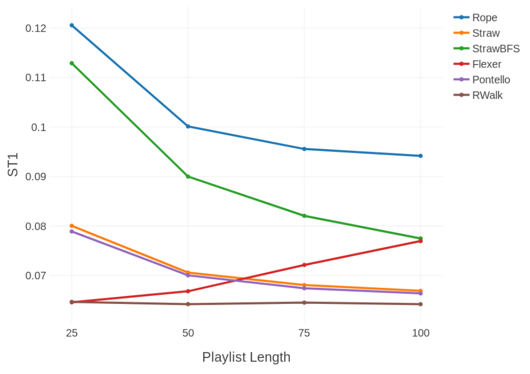


Figure 7.11: ST1 of playlists in the SVD music space

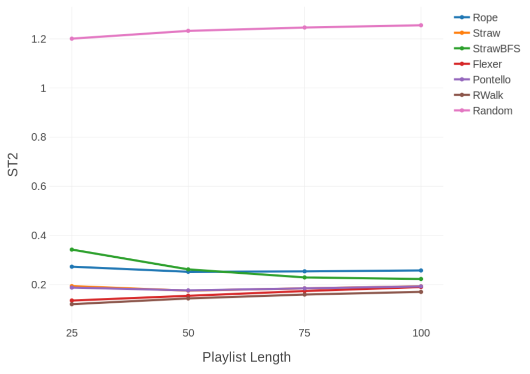


Figure 7.12: ST2 of playlists in the SVD music space

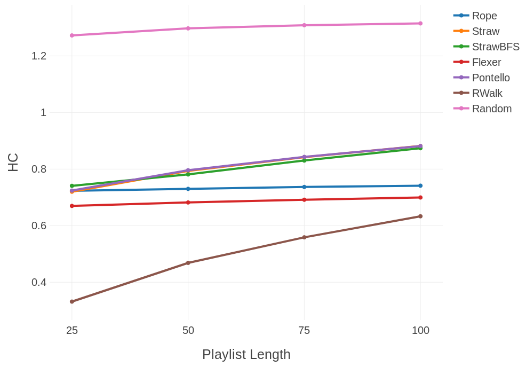


Figure 7.13: HC of playlists in the SVD music space

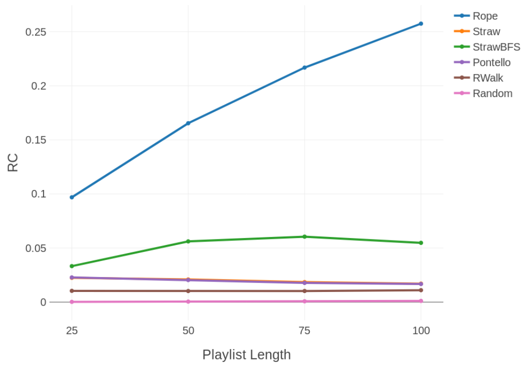


Figure 7.14: RC of playlists in the SVD music space

7.3.4 Comparing Word2vec and SVD music space

It is desirable that when generating a playlist for a user, it will be enjoyed by him/her. Since both Word2vec and SVD were constructed using the Spotify dataset, we will compare the music spaces by comparing the generated playlists with the playlists of the user from who we extract the anchor songs. As described before, we generated 10 playlists for each of the 500 selected users using as input to the algorithms songs extracted from the set S_u. To assess the quality of the playlists, for each playlist p_i generated for user u we calculated the proportion of songs in p_i (excluding the anchor songs s_0 and s_d) that are in the set S_u. If this proportion is high, it means the algorithm is able to generate playlists with songs the user would enjoy to listen to in his/her playlist. This metric is similar to precision in recommendation systems, which are the fraction of retrieved items that are relevant to the user. Therefore, we will call this metric as P (precision). Given a playlist $p = \{s_0, s_1, \ldots, s_{k-1}\}$ of size k generated for user u, we compute P as:

$$P = \frac{1}{k} \sum_{i=0}^{k-1} \mathbb{1}[s_i \in S_u]$$

We calculated the mean of P for each algorithm and playlist size, which can be seen in Table 7.1 and Table 7.2. To better visualize the values, we plotted the Figure 7.15. We can observe Word2vec music space generates playlists with high values of P and therefore is better suitable to generate playlists that would be enjoyable by users. This happens because Word2vec music space was constructed using the playlists from Spotify and therefore could better model users taste. We can also observe Flexer obtained the best values of P, thus it is a good algorithm to generate enjoyable playlists, although it is deterministic and may generate playlists with abrupt transitions, as shown before. When comparing Straw, StrawBFS and Pontello, we can observe StrawBFS obtained better values of P.

	Algorithm						
Length	Rope	Straw	StrawBFS	Flexer	Pontello	RWalk	Random
25	0.066	0.071	0.072	0.097	0.070	0.057	0.013
50	0.069	0.061	0.069	0.095	0.059	0.047	0.013
75	0.071	0.053	0.064	0.092	0.052	0.042	0.013
100	0.07	0.048	0.059	0.089	0.047	0.038	0.013

Table 7.1: Precision values of playlists in the Word2vec music space

	Algorithm						
Length	Rope	Straw	StrawBFS	Flexer	Pontello	RWalk	Random
25	0.175	0.098	0.128	0.178	0.097	0.080	0.013
50	0.159	0.079	0.113	0.159	0.079	0.061	0.013
75	0.149	0.067	0.098	0.146	0.066	0.053	0.013
100	0.137	0.059	0.084	0.136	0.059	0.047	0.013

Table 7.2: Precision values of playlists in the SVD music space

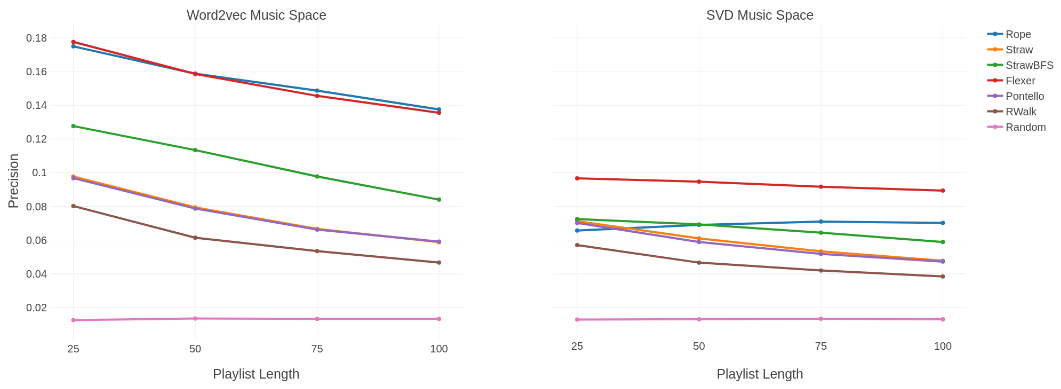


Figure 7.15: Precision of playlists in the Word2vec music space and SVD music space

7.4 Experimental Conclusions

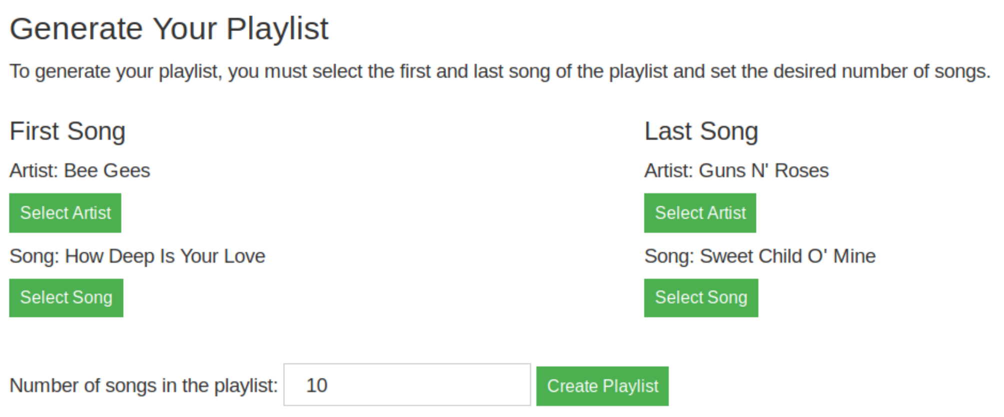
In this chapter, we performed experiments to compare the proposed algorithms with other baseline algorithms. From the experiments, we can conclude the approaches using a similarity graph were the ones that created playlists with smoother transitions between the songs, in special the algorithms Straw and Pontelo. This happens because the graph can guarantee the maximum step we can give. But different from the SVD and Word2vec music spaces, on the Billboard music space both Pontelo and Straw didn't obtain small values of ST1 compared with Rope and Flexer. This probably happened due to the construction of the graph, which was very dense. This is illustrated by the result of the RWalk on the same music space. The $ST1$ values of the RWalk on the Billboard music space (which we can consider as the average of the distance between connected songs) is higher than Pontello and Straw values.

With respect to heterogeneity, Straw, StrawBFS, and Pontello obtained the best results. Although these algorithms obtained the best values of HC, it is important to point out that, in our implementation, when the algorithm reaches song s_d, it starts a random walk in the similarity graph. Although this causes the playlist to have a higher value of heterogeneity (illustrated by the graphics, as the values of HC increases when increasing the size of the playlist), the algorithm may get distant from song s_d, going to a region of the music space not enjoyed by the user. But, analyzing figure 7.15, we can observe the precision values decreases as we increase the size of the playlist, indicating a random walk can generate playlists with songs the user enjoys.

Finally, except Rope and Flexer, all the implemented algorithms obtained small values of RC, showing they are able to generate random playlists and surprise the user.

7.5 Prototype

In order to make the algorithm publicly available to be used by users, we implemented an online prototype to generate music playlists, that can be accessed at `https://homepages.dcc.ufmg.br/~marcos.almeida/playlistgenerator`. In this prototype, we implemented StrawBFS as it obtained AD values higher than Straw, and used the Word2vec music space, since it obtained AD values higher than SVD music space. To use the system, the user only needs to input the start and end song of the playlist, and the desired number of songs. Figure 7.16 shows the interface of the system. After generating the playlist, the user can also connect to his Spotify account and save the generated playlist in his library. Figure 7.17 shows a playlist generated using as anchor songs *How Deep Is Your Love* by *Bee Gees* and *Sweet Child O' Mine* by *Guns N' Roses*.


Generate Your Playlist

To generate your playlist, you must select the first and last song of the playlist and set the desired number of songs.

First Song

Artist: Bee Gees

Song: How Deep Is Your Love

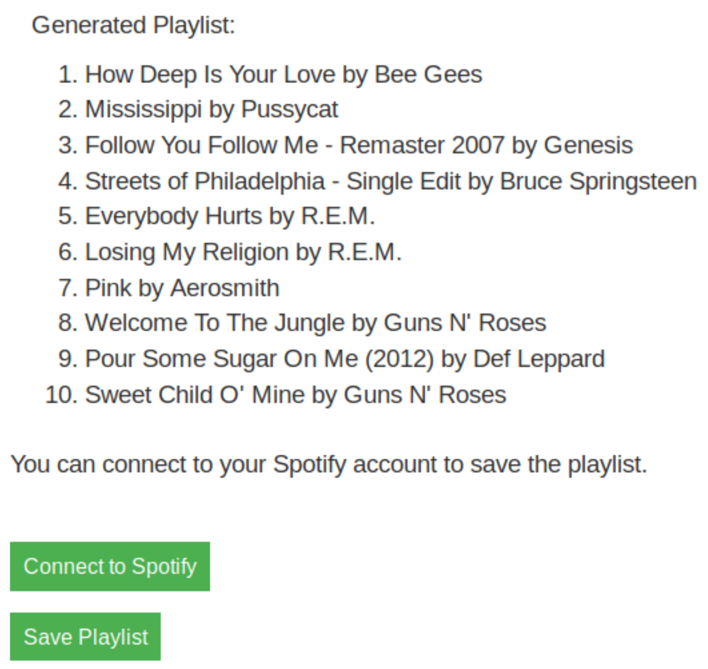
Last Song

Artist: Guns N' Roses

Song: Sweet Child O' Mine

Number of songs in the playlist: 10

Figure 7.16: Prototype interface



Generated Playlist:

1. How Deep Is Your Love by Bee Gees
2. Mississippi by Pussycat
3. Follow You Follow Me - Remaster 2007 by Genesis
4. Streets of Philadelphia - Single Edit by Bruce Springsteen
5. Everybody Hurts by R.E.M.
6. Losing My Religion by R.E.M.
7. Pink by Aerosmith
8. Welcome To The Jungle by Guns N' Roses
9. Pour Some Sugar On Me (2012) by Def Leppard
10. Sweet Child O' Mine by Guns N' Roses

You can connect to your Spotify account to save the playlist.

Figure 7.17: Playlist generated using the prototype

Chapter 8

Conclusions and Future Works

In this work, we studied the problem of automatic generation of music playlists. In our experiments, we used three different datasets. The first one contains acoustic characteristics of songs that appeared on the Billboard between 1960 and 2010. The second dataset is composed of users' playlists and artists tags extracted from Spotify. The third one contains users' playlists of the Art of the Mix website.

Analyzing the users' playlists from Spotify dataset we showed that, although most users create playlists with only a few different genres, several users enjoy listening to several different music genres, having an eclectic taste. Using the entropy to measure the heterogeneity of the users and playlists we could take the same conclusion. We also determined which music genres are present in almost all playlists, and which are present in only a few ones. Also, we clustered the users using as their representation the music genres they listen to and showed the songs most listened in each cluster, and the ones least listened.

After analyzing the users, we proposed three different forms to calculate the similarity between songs using the datasets. One using the songs acoustic characteristics, other using the co-occurrence of songs on users playlists and the third using artists tags. Using the similarity function between songs, we constructed three music spaces, where each music is mapped to a point in the Euclidean Space, and the distance between the songs can be computed using the Euclidean distance. We showed all music spaces are able to group tracks of the same genre and, given a seed song, retrieve similar songs.

Using the music spaces, we proposed a general method to automatically generate music playlists satisfying the following qualities: usability, smooth transition, heterogeneity, novelty, and scalability. Based on the general method, we proposed two algorithms to generate playlists. The first algorithm, named ROPE, is based on a Brownian motion on the music space. The second algorithm, named STRAW, performs a random walk on a music similarity graph. Both proposed algorithm requires the user only three inputs and scales with the size of the dataset, satisfying the usability and scalability criteria.

We performed experiments to evaluate the proposed algorithms with respect to smooth transitions, heterogeneity, and novelty. Based on our experiments, we concluded the proposed algorithms are able to satisfy the proposed qualities constraints. Among the proposed algorithms, we showed Straw obtained the best results in all the three constructed music spaces, outperforming other algorithms of the literature. We also showed Word2vec music space is best suitable to generate playlists that would be enjoyable by users since the playlists generated on this music space tend to contain songs similar to the anchor songs and that would be added to the playlist by the users. We also implemented the StrawBFS algorithm on an online system where users can use it to generate music playlists and listen to them on Spotify, available at `https://homepages.dcc.ufmg.br/~marcos.almeida/playlistgenerator`.

For future work, we would propose to use other songs characteristics to calculate the similarity between songs. Spotify API allows us to extract some acoustic characteristics of the tracks, such as danceability, energy, loudness, and valence. Other characteristics that can be retrieved using the Spotify API are the timbre and pitch values through the songs. Those characteristics can provide important information about the similarity between songs, improving the music spaces. We also propose to perform a user evaluation of the proposed algorithms, in order to receive direct feedback of the generated playlists are enjoyable.

Bibliography

Aucouturier, J.-J. and Pachet, F. (2002a). Finding songs that sound the same. In *Proc. of IEEE Benelux Workshop on Model based Processing and Coding of Audio*, pages 1--8.

Aucouturier, J.-J. and Pachet, F. (2002b). Scaling up music playlist generation. In *Multimedia and Expo, 2002. ICME'02. Proceedings. 2002 IEEE International Conference on*, volume 1, pages 105--108. IEEE.

Barrington, L., Oda, R., and Lanckriet, G. R. (2009). Smarter than genius? human evaluation of music recommender systems. In *ISMIR*, volume 9, pages 357--362. Citeseer.

Ben-Elazar, S., Lavee, G., Koenigstein, N., Barkan, O., Berezin, H., Paquet, U., and Zaccai, T. (2017). Groove radio: A bayesian hierarchical model for personalized playlist generation. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 445--453. ACM.

Bonnin, G. and Jannach, D. (2015). Automated generation of music playlists: Survey and experiments. *ACM Computing Surveys (CSUR)*, 47(2):26.

Cardoso, J. P. V., Pontello, L. F., Holanda, P. H., Guilherme, B., Goussevskaia, O., and da Silva, A. P. C. (2016). Mixtape: Direction-based navigation in large media collections. In *ISMIR*, pages 454--460.

Caselles-Dupré, H., Lesaint, F., and Royo-Letelier, J. (2018). Word2vec applied to recommendation: Hyperparameters matter. *arXiv preprint arXiv:1804.04212*.

Casey, M. A., Veltkamp, R., Goto, M., Leman, M., Rhodes, C., and Slaney, M. (2008). Content-based music information retrieval: Current directions and future challenges. *Proceedings of the IEEE*, 96(4):668--696.

Dias, R., Gonçalves, D., and Fonseca, M. J. (2017). From manual to assisted playlist creation: a survey. *Multimedia Tools and Applications*, 76(12):14375--14403.

Diedrich, C. G. (2015). 'neanderthal bone flutes': simply products of ice age spotted hyena scavenging activities on cave bear cubs in european cave bear dens. *Royal Society open science*, 2(4):140022.

Durrett, R. (2010). *Probability: theory and examples.* Cambridge university press.

Flexer, A., Schnitzer, D., Gasser, M., and Widmer, G. (2008). Playlist generation using start and end songs. In *ISMIR*, volume 8, pages 173--178.

Goussevskaia, O., Kuhn, M., Lorenzi, M., and Wattenhofer, R. (2008). From web to map: Exploring the world of music. In *Web Intelligence and Intelligent Agent Technology, 2008. WI-IAT'08. IEEE/WIC/ACM International Conference on*, volume 1, pages 242--248. IEEE.

Grbovic, M., Radosavljevic, V., Djuric, N., Bhamidipati, N., Savla, J., Bhagwan, V., and Sharp, D. (2015). E-commerce in your inbox: Product recommendations at scale. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1809--1818. ACM.

Jannach, D., Kamehkhosh, I., and Bonnin, G. (2014). Analyzing the characteristics of shared playlists for music recommendation. In *RSWeb@ RecSys.*

Kamalzadeh, M., Baur, D., and Möller, T. (2012). A survey on music listening and management behaviours.

Kamehkhosh, I., Jannach, D., and Bonnin, G. (2018). How automated recommendations affect the playlist creation behavior of users.

Knees, P., Pampalk, E., and Widmer, G. (2004). Artist classification with web-based data. In *ISMIR.*

Knees, P. and Schedl, M. (2013). A survey of music similarity and recommendation from music context data. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 10(1):2.

Leskovec, J., Rajaraman, A., and Ullman, J. D. (2014). *Mining of massive datasets.* Cambridge university press.

Linden, G., Smith, B., and York, J. (2003). Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, (1):76--80.

Logan, B., Kositsky, A., and Moreno, P. (2004). Semantic analysis of song lyrics. In *Multimedia and Expo, 2004. ICME'04. 2004 IEEE International Conference on*, volume 2, pages 827--830. IEEE.

Logan, B. and Salomon, A. (2001). A music similarity function based on signal analysis. In *null*, page 190. IEEE.

Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579--2605.

Maillet, F., Eck, D., Desjardins, G., Lamere, P., et al. (2009). Steerable playlist generation by learning song similarity from radio station playlists. In *ISMIR*, pages 345--350.

Mauch, M., MacCallum, R. M., Levy, M., and Leroi, A. M. (2015). The evolution of popular music: Usa 1960–2010. *Royal Society open science*, 2(5):150081.

McFee, B. and Lanckriet, G. R. (2012). Hypergraph models of playlist dialects. In *ISMIR*, volume 12, pages 343--348. Citeseer.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111--3119.

Miranda, D. (2013). The role of music in adolescent development: much more than the same old song. *International Journal of Adolescence and Youth*, 18(1):5--22.

Muda, L., Begam, M., and Elamvazuthi, I. (2010). Voice recognition algorithms using mel frequency cepstral coefficient (mfcc) and dynamic time warping (dtw) techniques. *arXiv preprint arXiv:1003.4083*.

Pachet, F., Westermann, G., and Laigre, D. (2001). Musical data mining for electronic music distribution. In *Web Delivering of Music, 2001. Proceedings. First International Conference on*, pages 101--106. IEEE.

Pampalk, E., Flexer, A., and Widmer, G. (2005a). Hierarchical organization and description of music collections at the artist level. In *International Conference on Theory and Practice of Digital Libraries*, pages 37--48. Springer.

Pampalk, E., Flexer, A., Widmer, G., et al. (2005b). Improvements of audio-based music similarity and genre classificaton. In *ISMIR*, volume 5, pages 634--637. London, UK.

Pauws, S., Verhaegh, W., and Vossen, M. (2008). Music playlist generation by adapted simulated annealing. *Information Sciences*, 178(3):647--662.

Platt, J. C., Burges, C. J., Swenson, S., Weare, C., and Zheng, A. (2002). Learning a gaussian process prior for automatically generating music playlists. In *Advances in neural information processing systems*, pages 1425--1432.

Pohle, T., Pampalk, E., and Widmer, G. (2005). Generating similarity-based playlists using traveling salesman algorithms. In *Proceedings of the 8th International Conference on Digital Audio Effects (DAFx-05)*, pages 220--225. Citeseer.

Pontello, L. F., Holanda, P. H., Guilherme, B., Cardoso, J. P. V., Goussevskaia, O., and Silva, A. P. C. D. (2017). Mixtape: Using real-time user feedback to navigate large media collections. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 13(4):50.

Ragno, R., Burges, C. J., and Herley, C. (2005). Inferring similarity between music objects with application to playlist generation. In *Proceedings of the 7th ACM SIGMM international workshop on Multimedia information retrieval*, pages 73--80. ACM.

Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285--295. ACM.

Shao, B., Wang, D., Li, T., and Ogihara, M. (2009). Music recommendation based on acoustic features and user access patterns. *IEEE Transactions on Audio, Speech, and Language Processing*, 17(8):1602--1611.

Theil, H. (1967). Economics and information theory. Technical report.

Vasile, F., Smirnova, E., and Conneau, A. (2016). Meta-prod2vec: Product embeddings using side-information for recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 225--232. ACM.

Wang, D., Deng, S., Zhang, X., and Xu, G. (2016). Learning music embedding with metadata for context aware recommendation. In *Proceedings of the 2016 ACM on International Conference on Multimedia Retrieval*, pages 249--253. ACM.

Zadel, M. and Fujinaga, I. (2004). Web services for music information retrieval. In *ISMIR*. Citeseer.

Appendix A

Keywords list

The list of keywords used to search for the playlists on Spotify system are:

"anos 60", "anos 70", "anos 80", "anos 90", "anos 2000", "anos 2010", "years 60", "years 70", "years 80", "years 90", "years 2000", "years 2010", "rock", "pop", "soul", "funk", "party", "festa", "country", "old musics", "antigas", "mpb", "favorites", "favoritas", "hiphop", "oldies", "rap", "rnb", "viagem", "travel", "wedding", "gym", "academia", "casamento", "samba", "folk", "forro", "jazz", "blues", "sertanejo", "bossa nova", "classic", "classica"