POLITECNICO DI TORINO

Master's Degree in Computer Engeneering

Master's Degree Thesis

A recommender system for user matching in networked music interactions

Supervisors

Prof. Andrea BIANCO

Prof. Cristina ROTTONDI

Prof. Massimiliano ZANONI

Candidate

Luigi PIRISI

A.Y. 2020/2021

Abstract

Due to the social distancing countermeasures recently enforced during the COVID19 pandemic, a lot of effort is being dedicated to provide online support for the daily activities of musicians, composers, music teachers and students.

Several web-based applications are being developed with the aim of supporting remote musical interactions in an easy, fast and intuitive way for musicians of all levels. In such applications, the integration of recommendation and match-making functionalities are of pivotal importance to allow users to find and get in touch with other musicians and to collaborate with them for musical content creation online. The aim of this thesis project is the development of a recommendation system capable of providing suggestions on similarities between musicians based on their musical production. The framework is based on using features extracted from raw audio exploiting algorithm commonly known in the MIR context. After a preprocessing phase which involves also t-SNE, a method for dimensionality reduction, the artists are compared against each other through custom-made distance metrics over the t-SNE space.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Context and motivation

This work comes in response to the event that marked the last year: the COVID19 pandemic. The countermeasures adopted caused isolation and, in some cases, the total inability to practice a job. This happened for musicians, composers, music teachers and students. In such a scenario, the potential of the *Networked Musical Performance* (NMP) becomes more and more crucial.
Assuming the existence of an online platform that allows musicians to meet each other and play remotely, there would be the need to bring together musicians with common interests. In other words, artists should be enabled to connect with someone they would like to play with. This thesis focuses on finding similarities between artists in order to address this problem.

1.2 Background

Recommendation systems (RSs) are well known in the literature to solve this kind of tasks. After a dissertation about this topic's state of the art, by analyzing the pros and cons of different methodological approaces, we will focus on a recommendation approach called *content-based* filtering. This method differs from the *collaborative* one because it takes as input features extracted from the audio samples associated to the artists to be compared, instead of the history of the listeners' activities.
Later we will talk about the choice of *Million Song Dataset* as our input and the criteria adopted to make this choice. We will consider the audio features related to each song. These features are directly retrieved from the digital waveform through particular signal processing algorithms. Those algorithms are developed under a branch of a science called *Music Information Retrieval* (MIR).

1.3 Proposed Framework

After understanding which features are most suitable for our task, we will proceed by identifying what will be our target. The data set we rely on provides different kinds metadata that are attached to each artist. Given an artist, a set of *terms* is provided. A term is a string that semantically links that artist to belong to a musical, temporal or social context. By starting from numerical features, we want to compare artists to one another to estimate the level of similarity. After that, we will intersect term lists to verify if a low level of similarity (e.g. a high level of distance) corresponds to have few terms in common. Also, Each artist present in the data set is characterized by a set of songs and has an ordered list of similar artists associated. Thus we will define our second goal as to find our ranking of similar artists based on the features extracted from their songs in such a way that these two rankings are as similar as possible.
Before this, however, we will need to study our feature space in order to understand how the preprocessing phase could be useful to increase readability and performance. After usual preprocessing operations like normalization and outlier detection, we will be addressing the dimensionality problem using a method called *t-distributed stochastic neighbor embedding* (t-SNE) in order to work with a 2-dimensional dataset. At this point, we will also be able to display the songs in a 2-dimensional space. Firstly, for each artist, we will organize its songs into a fixed dimension matrix where each cell represents a region of the t-SNE space. We will refer to this matrix as a heatmap. Secondly we will proceed to compute, for each artist, our ranking of similar artists by comparing their respective heatmaps. In this phase, we will have to deal with comparing two ordered lists. After a review of algorithms and metrics about the topic, we will compare the similarity between rankings using a simple intersection metric and *Rank-biased Overlap* (RBO), a similarity measure for indefinite rankings.

1.4 Results

After generating our rankings of similarity, we will evaluate our result in two ways: one based on metadata attached to each artist, in particular on tag-lists; the other based on rankings already provided by the dataset. The evaluation based on tag-lists will be successful as it will show that, given an artist, the number of common tags decreases as the position within the ranking increases. The comparison between rankings will articulate in two approaches, one more strict than the other as it consider also the order of rankings. The results will be less satisfactory but still lead to a partially positive confirmation. We will show that the rankings we will produce will always perform better than random ones.

Chapter 2

Literature Review

2.1 Music Information Retrieval

Music Information Retrieval (MIR) is a field of research that aims to fetch music data in order to obtain useful information. It includes several domains such as musicology, psychoacoustics, psychology, academic music study, signal processing, information technology, machine learning, optical music recognition, computational intelligence [1]. With the increase of computing perfomance, MIR has become more effective and popular as a knowledge retrieval tool. Firstly the target was to work with digital music (such as MIDI), then MIR focused on dealing directly with audio signals. The development of effective music compression algorithms helped to make this process feasible especially on a large scale.
The information related to music perception is classified as shown in figure 2.1. *Music content based* approaches are strictly related to the audio signal while *music context* features identifies those data associated with the music but not directly derived by the audio signal. *User context* and *user properties* identify those features associated with those users who enjoy the contents (the listeners) . The difference is that the former has a dynamic behavior while the latter are constant or slowly changing.

2.1.1 Applications

As described by Schedl et al. in [3], the typical usages of MIR approaches are shown below:

Music retrieval

These applications accomplish the task of helping users to find music in a large collection. Most of MIR tasks differs according to properties like *specificity* (a low

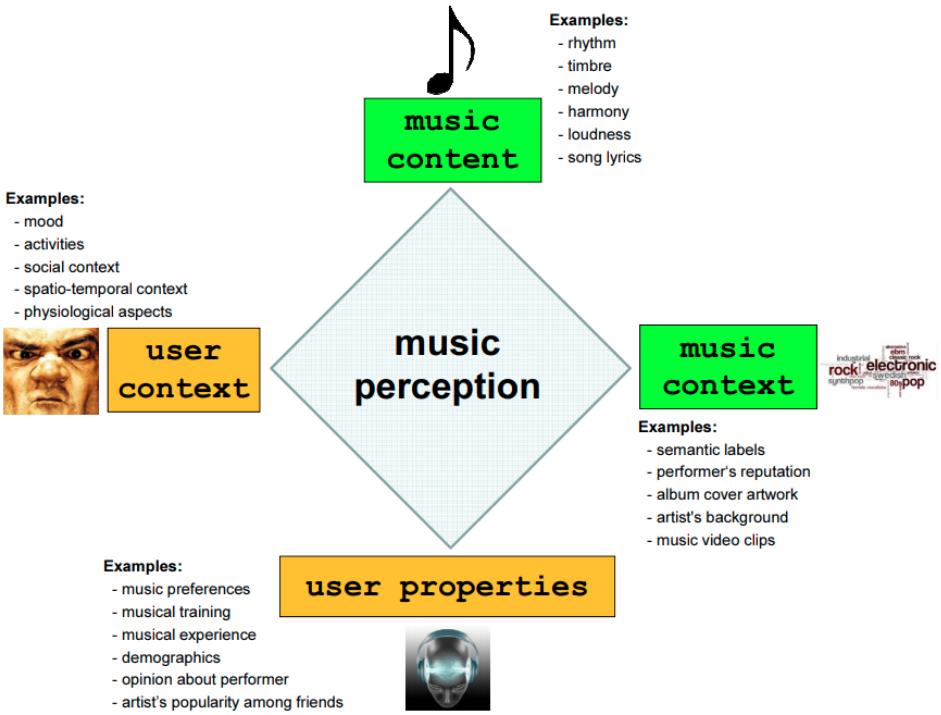


Figure 2.1: Categorization of perceptual music descriptors proposed in [2] [3]

value means to get similar music tracks, a high value means to retrieve and identify the audio track) and *granularity* (small to identify specific time portions of the file, large to retrieve a whole music piece). We have some example below.
Audio identification or *fingerprinting* has high specificity and low granularity. These kind of task aims to find the exact portion of a given music recording overcoming issues such as recording noise. The approach shown in [4] is used by the popular system *Shazam*[1].
Audio ailgnment represents a more complex task than the previous one. The objective is to identify and link time positions of two audio signals.

In *Cover song identification* scenario the specificity level is lower. Thus the model is able to retrieve different versions of the same song. As described in [5], to accomplish such a task it is useful to work with harmony or melody of audio signals.
In *Query by tapping* and *query by humming* scenarios, the user provide to the system a recording where they try to reproduce melodically or rithmically a song.

[1]https://www.shazam.com

The objective is to retrieve that song. Services such as *SoundHound*[2] made these implementation commercially available.
All the approaches listed above belong to the same class in which a query is made by *an example*. There are other approaches where the query is textual, they are called *Semantic/tag-based retrieval* systems.

Music recommendation

Music recommendation systems create a model for each user in order to propose a list of songs they may like. The main requirements for this kind of tasks, as stated in [6] and [7], are:

- *Accuracy*: the proposed songs should match users' interests.
- *Transparency*: the user should be aware about why a certain song is recommended.
- *Diversity*: the recommended songs shouldn't be too similar to each other.
- *Serendipity*: the recommender system should surprise the user with its recommendations.

Music playlist generation

The goal of this kind of applications is to build an ordered homogeneous list of songs. This process is also known as *Automatic DJing* and differs from a recommendation system because it creates a playlist without knowing the users preferences or past activity but only by arranging already known material. Consecutive songs in a playlist must show a certain trade off between similarity/diversity (we could see the one as the inverse of the other). The user may feel bored if the level of similarity is too high. In order to build an effective playlist, [8] and [9] suggest other features such as *familiarity/popularity*, *hotness/trendiness*, *recentness* and *novelty*.

2.1.2 Music Feature Extraction

We represent a recording using a time or frequency representation of its audio signal. We refer to f as the number of cycles per second in Hertz (Hz) while $T = \frac{1}{f}$ is the time taken by each cycle. In time domain, an analog signal $x(t)$ is sampled each T_s seconds to obtain a discrete digital representation $x[n]$, with $n = i \cdot T_s, i = 0,1,2, ...$

[2] https://www.soundhound.com

In order to avoid the *aliasing* effect, Nyquist-Shannon theorem states that the minimum sampling frequency is equal to the double of the maximum frequency present in the audio signal.

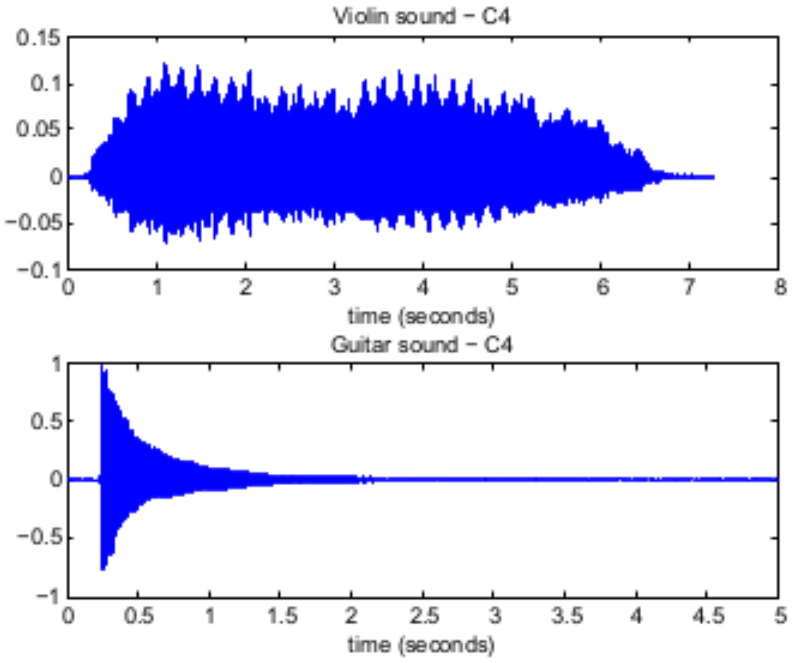


Figure 2.2: Time-Domain representation of two signals corresponding to a C4 [3]

The transition to the frequency domain is done via the Fourier Transform (FT). Since we are dealing with discrete signals and we are interested in seeing changes over time in terms of frequencies, we use Discrete version of FT on little segments of the signal called *frames* (Short Time Fourier Transform - STFT). The discrete signal $x[n]$ is multiplied by a window function $w[n]$ which has a bell-shape form and is zero-valued outside.

Low-level descriptors and timbre

Low-level descriptors are essential to build high-level analyses. We refer to the *color* of the sound as a mix of loudness and timbre. The timbre depends on three main features of music signals: Temporal evolution of energy (see figure 2.2), spectral envelope shape (relative strength of different frequency components, figure 2.3), and time variation of the spectrum. Low-level descriptors depend on these features.

A common instantaneous (frame-based) temporal descriptor is the short-time *Zero Crossing Rate*, that measures the number of times a signal crosses the zero axis per second and is an indicator of high frequency content and noisiness. Another

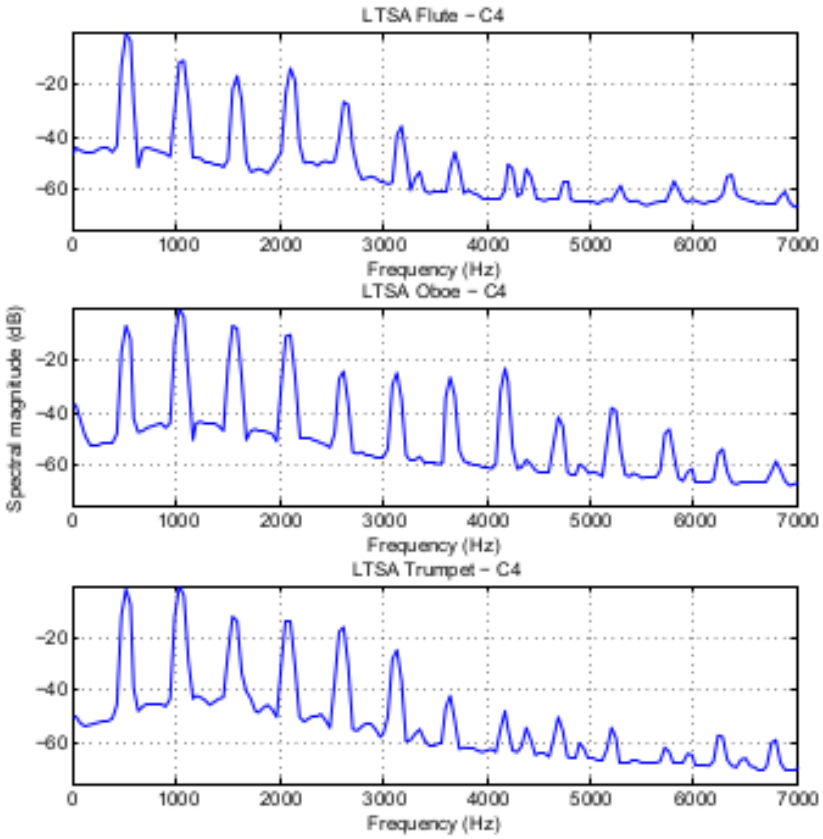


Figure 2.3: Frequency-Domain representation. Given a foundamental frequency f_0, an instrument produces also a set of harmonics. Each harmonic is a multiple of f_0, $i \cdot f_0$ where $i = 1,2,...$ [3]

instantaneous temporal descriptor is the energy, given by the root mean square (RMS) value of $x[n]$; it is a measure of loudness. Other used global temporal descriptors are the *log attack time* and *temporal centroid*. Log attack time is the duration of the note onset while temporal centroid measures the location of the signal energy and helps to separate sustained from non-sustained sounds.
One of the most common descriptors is *Mel-Frequency Cepstrum Coefficients* (MFCCs). It turned to be an efficient and compact way to represent the spectrum of a signal. The block diagram used to compute the MFCCs is shown in figure 2.5. The spectrum is filtered with a set of triangular filters that follow a Mel-frequency scale. The log of each output passes trough a Discrete Cosine Transform operator in order to obtain a set of coefficients. There are other descriptors such as the spectral moments (*spectral centroid*, *skewness*, *spread* and *kurtosis*).
Low-level descriptors are often used to define timbre. Higher-level analyses start

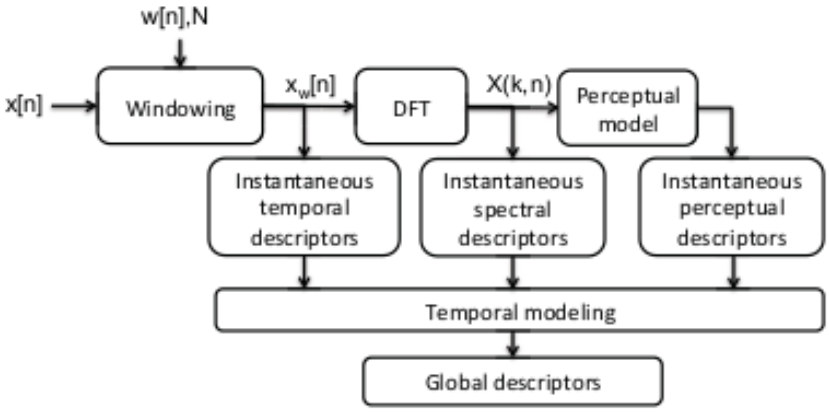


Figure 2.4: Diagram for low-level feature extraction [3]

from these features to discriminate over instrument, genre or rhythm. Since they have a compact but exhaustive form, they are used for audio fingerprinting.

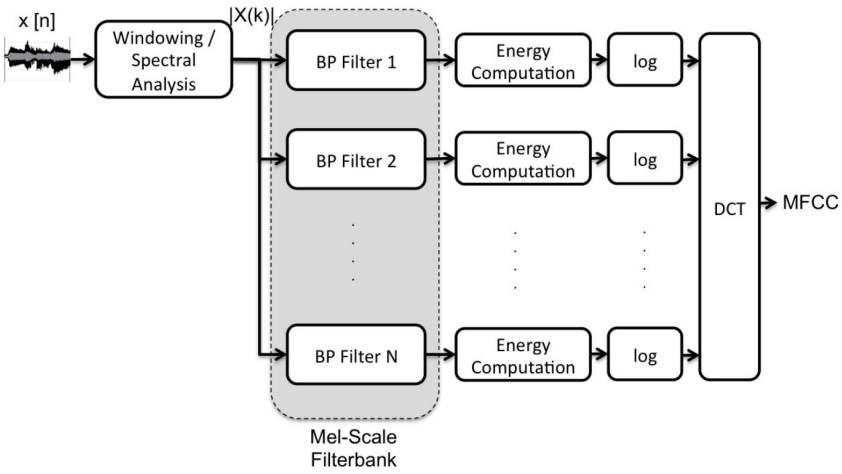


Figure 2.5: Diagram for MFCCs computation [3]

Pitch content descriptors

In frequency domain, as said before, each sound is composed by its fundamental frequency plus a set of harmonics whose frequency is a multiple of the fundamental one f_0. In a perceptual context the fundamental frequency is the pitch, which is a subjective quality often described as highness or lowness. The pitch scale is logarithmic, the intervals are measured in *cents* (1 semitone = 100 cents). In

western music the most common tuning system is the equal temperament. This systems divides the octave (interval between f_0 and $2 \cdot f_0$) into twelve equally spaced 100 cents semitones. The set of pitches that are a multiple of an octave distant are called *chroma* or *pitch class*. For instance, the pitch class C consists of all the C's in all octaves.
Thus, a pitch content descriptor gives information about melody, harmony, and tonality. Retrieving pitch descriptors is not an easy task because all f_0's need to be detected through time and spectrum among a whole set of non-fundamental frequencies. In addition to that, The more polyphonic the audio fragment is, the harder the task will be.
Common challenges in this field are:

- the computation of chroma features, all multiple f_0 are jointly analyzed
- the extraction of the f_0 envelope related to the most important "voice" inside a complex polyphony (melody extraction).
- the estimation of all f_0 in simple polyphonies.

Predominant melody extraction algorithms are an extension of melody extraction on monophonic music fragments. The assumption is that there is a predominant source in the spectrum. The objective is to detect it. There are two main approaches in literature: the first one is *salience-based*: it estimates the salience of each possible f_0. The second one is based on *source-separation*: it tries to isolate the predominant source and then it performs a monophonic melody extraction to retreive f_0. A block diagram of a salience-based algorithm by Salamon and Gòomez is shown in figure 2.7 [10].
Multi-pitch estimation algorithms try to detect all the pitches within a fragment. As for melody extraction, the approaches are the same: one is salience-based, the other is based on source-separation.
Chroma feature extraction methods represent the intensity of each of the 12 pitch classes of an equal-tempered chromatic scale. The process starts from the frequency spectrum. Since these methods depend only on pitched sounds, they should be robust to noise and timbre. Figure 2.6 shows the most common procedures to realize this kind of task.

Rhythm

Rhythm in an audio fragment depends on periodicity and temporal organization of musical events. Thus, rhythm descriptors are related to four different characteristics: timing (when events occur), tempo (how often events occur), meter (what structure best describes the event occurrences) and grouping (how events are structured in motives or phrases). The methods used are based on low-level descriptors such as

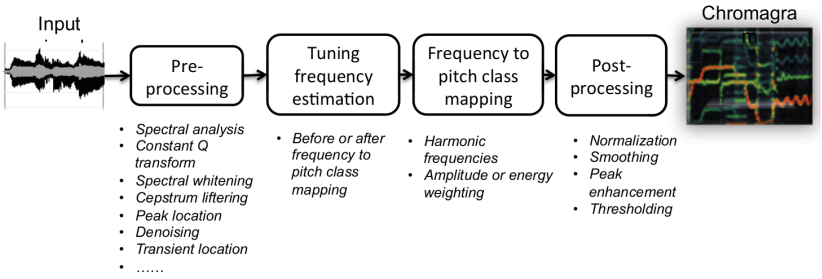


Figure 2.6: Block diagram with common procedures for computing a chromagram [3]

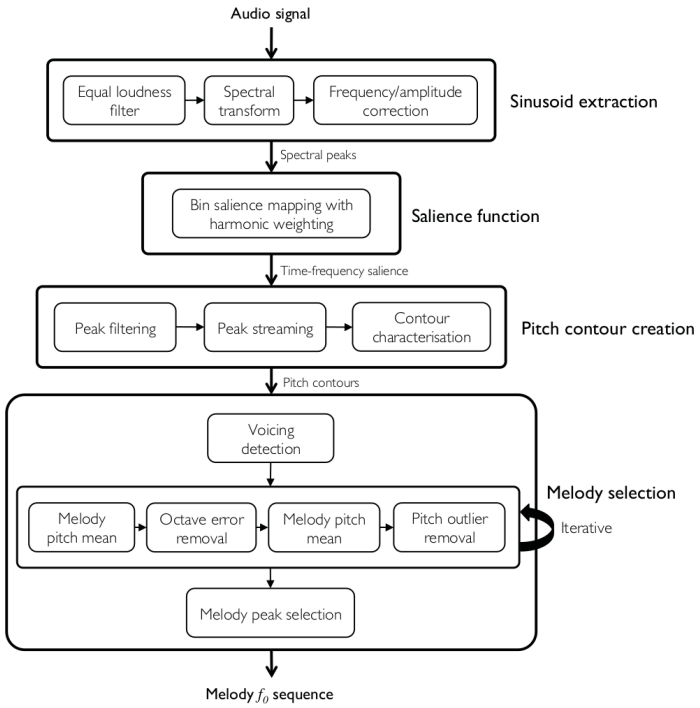


Figure 2.7: Block diagram for melody extraction designed by Salamon and Gòmez [10] [3]

the energy and spectral descriptors. For instance, in figure 2.8, one can notice the clear relationship between relative peaks and the downbeats. Estimating the beat becomes a challenge when the audio fragments don't contain percussive material (for instance an ensemble of strings or a choir).

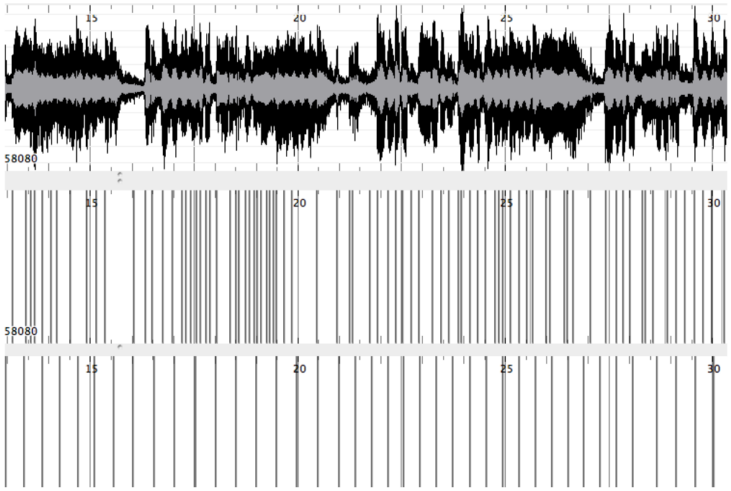


Figure 2.8: An Example of beat estimation [3]

2.1.3 Music similarity

Many machine learning tasks have to deal with quantifying similarity. In MIR context, the term similarity may be refer to two concepts. *Self-similarity analysis* corresponds to locate similar fragments of the same musical track. Global similarity is the distance between two musical pieces. Defining an effective way to measure such a distance is an open problem still today. Firstly, we need to select the descriptors involved as input. Secondly, we have to set a proper abstraction level in order to let some variations be accepted without increasing the false positive rate too much. Finally, we need to test our predictions. This phase often requires a human factor that introduces a subjective component not easy to model.
In general we could refer to a music track as a point in a n-dimensional space. The similarity can be seen as the inverse of the distance between two points in an hyperspace. Given two points $P(P_0, P_1, ..., P_n)$ and $Q(Q_0, Q_1, ..., Q_n)$, we report some measures of distance collected from [11]. Formula 2.1 correspond to *City Block* L_1 from Minkowsky family L_p, 2.2 is the *Soergel* distance and 2.3 is a measure of *non-intersection*. The euclidean distance is reported in formula 2.4

$$d_{CB} = \sum_{i=1}^{n} |P_i - Q_i| \tag{2.1}$$

$$d_{sg} = \frac{\sum_{i=1}^{n} |P_i - Q_i|}{\sum_{i=1}^{n} max(P_i, Q_i)} \tag{2.2}$$

$$d_{sg} = 1 - min(P_i, Q_i) \tag{2.3}$$

$$d_{eu} = \sqrt{\sum_{i=1}^{n} (P_i - Q_i)} \tag{2.4}$$

Chapter 3

Background

3.1 t-SNE

T-distributed stochastic neighbor embedding (t-SNE) is a statistical solution for dimensionality reduction. It was developed by van der Maaten and Hinton in [12] in 2008 as a non-linear technique that visualizes high dimensional datapoints in a two or three-dimensional space. It proved to be more effective than traditional methods such as such as Principal Components Analysis (PCA; Hotelling, 1933) and classical multidimensional scaling (MDS; Torgerson, 1952). It represents an improvement over Stochastic Neighbor Embedding (SNE). Unlike PCA, that tries to preserve the global shape of the data, tSNE takes into account the local structure (presence of clusters).
Given a datapoint $x_i = [x_{i1}, x_{i2}, ..., x_{id}]$ in a d-dimensional space, we refer to its *neighborhood* $N(x_i)$ as the set of other datapoints $x_j = [x_{j1}, x_{j2}, ..., x_{jd}]$ such that x_i and x_j are geometrically close. The term *embedding* in our case is the topological representation of a d-dimensional set of points in a different dimensional space (generally lower) in such a way that the original structure is preserved.

3.1.1 SNE

Stochastic Neighbor Embedding (SNE) starts with defining the similarity between two points not in terms of Euclidean distance but in terms of conditional probability. The similarity of datapoint x_j to datapoint x_i is the conditional probability, $p_{j|i}$, that x_i would choose x_j as its neighbor if neighbors were picked in proportion to their probability density under a Gaussian centered at x_i. Thus, the closer will be

the points, the higher $p_{j|i}$ will be.

$$p_{i|j} = \frac{\exp\left(\frac{-\|x_i - x_j\|^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(\frac{-\|x_i - x_k\|^2}{2\sigma_i^2}\right)} \tag{3.1}$$

σ_i is the variance of the Gaussian centered on x_i, while value for $p_{i|i}$ is set to zero by definition. Let be y_i and y_j the low-dimensional counterparts for datapoints x_i and x_j. We compute the similar conditional probability, this time we set the variance of the Gaussian to $\frac{1}{\sqrt{2}}$.

$$q_{i|j} = \frac{\exp\left(-\|y_i - y_j\|^2\right)}{\sum_{k \neq i} \exp\left(-\|y_i - y_k\|^2\right)} \tag{3.2}$$

Even in this case $q_{i|i} = 0$.
The goal of SNE is to find a low-dimensional representation that minimizes the difference between $p_{j|i}$ and $q_{j|i}$. An appropriate metric to estimate this difference is the Kullback-Leibler divergence. SNE minimizes the sum of Kullback-Leibler divergences over all datapoints using a gradient descent method. The cost function C is given by

$$C = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}, \tag{3.3}$$

where P_i and Q_i are the conditional probability over all datapoints x_i and y_i respectively. The Kullback-Leibler divergence is not symmetric. This means that a mismatch given by two points distant from each other in the original space but closer in the new one will be barely penalized. A mismatch of two points from the same neighborhood placed far from each other in the new space, in contrast, will be heavily penalized. This behavior explains why SNE preserves locality of datapoints. An important parameter to set is the value of σ_i of the Gaussian centered over each datapoint x_i in the original space. Usually, the optimal value for this parameter depends on the density of the data. In dense regions, a smaller value of σ_i is usually more appropriate than in sparser regions. As the value of σ_i increases, the entropy of the probability distribution P_i increases too. SNE performs a binary search for the optimal value of σ_i based on a *perplexity* that is a tunable input parameter. The perplexity is given by

$$Perp(P_i) = 2^{H(P_i)}, \tag{3.4}$$

where $H(P_i)$ is the Shannon entropy of P_i, expressed in bits

$$H(P_i) = -\sum_j p_{j|i} \log_2 p_{j|i}. \tag{3.5}$$

Typical values for perplexity are between 5 and 50. It can be seen as a measure of the effective number of neighbors. The minimization of the cost function 3.3 is done through the gradient descent method

$$\frac{\delta C}{\delta y_i} = 2\sum_j (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(y_i - y_j). \tag{3.6}$$

Firstly, the gradient descent is initialized by sampling map points randomly from an isotropic Gaussian with small variance that is centered around the origin. At each time step, the gradient is updated following the formula

$$\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta\frac{\delta C}{\delta \mathcal{Y}} + \alpha(t)\left(\mathcal{Y}^{(t-1)} + \mathcal{Y}^{(t-2)}\right). \tag{3.7}$$

In 3.7, $\mathcal{Y}^{(t)}$ represents the solution at iteration t while η is the learning rate multiplied by the gradient of the cost function. In addition to that, a momentum term $\alpha(t)$ is added to the gradient in order to speed the optimization up and avoid poor local minima.

3.1.2 t-Distributed Stochastic Neighbor Embedding

t-Distributed Stochastic Neighbor Embedding or *t-SNE* comes to alleviate two issues of SNE: the optimization difficulty and the so-called *crowding problem*. The cost function of t-SNE differs from the one used in SNE because it is symmetric and it uses a Student-t distribution rather than a Gaussian to compute similarity between two points in the *low-dimensional space*. Student-t distribution has longer tails with respect to a Gaussian distribution and this helps to keep clusters farther from each other.

3.1.3 Symmetric SNE

A first change is to compute a single Kullback-Liebler divergence between joints probabilities P and Q instead of computing a sum of multiple Kullback-Liebler divergences between conditional probabilities $p_{j|i}$ and $q_{j|i}$. The cost function becomes

$$C = \sum_i KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}, \tag{3.8}$$

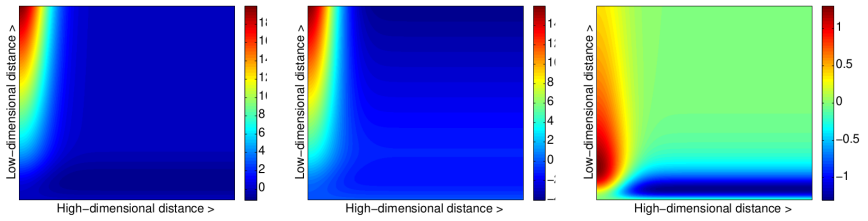
where p_{ii} and q_{ii} are set to zero. The cost function is symmetric because $p_{ij} = p_{ji}$ and $q_{ij} = q_{ji}$. The symmetry results in a simpler form of the gradient, which is faster to compute. It is given by

$$\frac{\delta C}{\delta y_i} = 4\sum_j (p_{ij} - q_{ij})(y_i - y_j). \tag{3.9}$$

3.1.4 The crowding problem

Sometimes it is not possible to preserve distances in all the neighborhoods. For instance, suppose that we want to represent a 2-dimensional square in a 1-dimensional space. Each vertex will have the two adjacent vertexes in its neighborhood and the opposite one left out. There is no low dimensional representation that could satisfy the constraints on all of the neighborhoods. Generally, this phenomenon happens when dealing with mutually equidistant data points and it is known as *crowding problem*. SNE reacts to this problem crushing together the points in the center of the map, which prevents from forming gaps between the natural clusters. According to Cook et al.(2007), adding a slight repulsion can address this problem. Using a uniform backgorund model with a small mixing proportion, ρ, with n representing the number of points, helps q_{ij} never fall below $\frac{2\rho}{n(n1)}$. In this technique, called UNI-SNE, q_{ij} will be larger than p_{ij} even for the far-apart datapoints.
SNE in general uses probability distributions to convert distances into probabilities. In order to alleviate the crowding problem it could be effective to use a Gaussian distribution for the high dimensional points and to use another distribution with heavier tails in the low dimensional space. This allows a moderate distance in the high-dimensional space to be faithfully modeled by a much larger distance in the map and, as a result, it eliminates the unwanted attractive forces between map points that represent moderately dissimilar datapoints [12]. t-SNE uses a Student t-distribution with one degree of freedom. The joint probabilities q_{ij} are defined as

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq l} (1 + \|y_k - y_l\|^2)^{-1}}. \tag{3.10}$$



(a) Gradient of SNE. (b) Gradient of UNI-SNE. (c) Gradient of t-SNE.

Figure 3.1: Gradients of three types of SNE as a function of high dimensional distance and the corresponding low dimensional distance [12]

At this point, the gradient of the Kullback-Leibler divergence between P and Q (Student-t based) is given by

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j) \left(1 + \|y_i - y_j\|^2\right)^{-1}. \tag{3.11}$$

3.2 Probability Distributions

3.2.1 Gaussian distribution

The **Gaussian distribution** (or **normal**) is by far the best known in literature. We will use such a distribution later in this work, for instance in the normalization phase. It is used to describe a real-variable random variable and its general form is:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \tag{3.12}$$

where μ is the mean and σ is the standard deviation.

3.2.2 *t*-distribution

Student's *t*-distribution is a generalized version of the normal distribution that aims to deal with situation where the sample size is small and standard deviation is unknown. The density function has the following form:

$$f(t) = \frac{\Gamma(\frac{\nu+1}{2})}{\sqrt{\nu\pi}\Gamma(\frac{\nu}{2})} \left(1 + \frac{t^2}{\nu}\right)^{-\frac{\nu+1}{2}} \tag{3.13}$$

where ν is the number of degrees of freedom and $\Gamma(n) = (n-1)!$ is the gamma funcion.

3.3 Rank Similarity Measures

A ranking is an ordered list of items. There are several ways in literature to estimate the similarity of two rankings. The key concepts behind a comparison between ranked lists are:

- The top of each ranking is more important than the bottom. This property is called top-weightedness.
- Most of the time we need estimate the similarity of two rankings that don't contain the same elements. In these circumstances we define the two rankings as *non-conjoint*.

3.3.1 Ranked Biased Overlap

Rank Biased Overlap (**RBO**) is a similarity measure for indefinite rankings. This is an overlap-based metric [13].
Let S and T be two infinite rankings, and let S_i be the element at rank i in list

S. Denote the set of the elements from position c to position d in list S, that is $\{S_i : c \leq i \leq d\}$, as $S_{c:d}$. Let $S_{:d}$ be equivalent to $S_{1:d}$, and $S_{c:}$ be equivalent to $S_{c:\infty}$. At each depth d, the *intersection* of lists S and T to depth d is:

$$I_{S,T,d} = S_{:d} \cap T_{:d} \tag{3.14}$$

The size of this intersection is the *overlap* of lists S and T to dept d,

$$X_{S,T,d} = |I_{S,T,d}| \tag{3.15}$$

and the proportion of S and T that are overlapped at depth d is their *agreement*,

$$A_{S,T,d} = \frac{X_{S,T,d}}{d}. \tag{3.16}$$

From now on we will refer to I_d X_d A_d for the sake of brevity. The average overlap is defined as:

$$AO(S,T,k) = \frac{1}{k}\sum_{d=1}^{k} A_d \tag{3.17}$$

where k is the evaluation depth. The overlap-based rank similarity measures have the form

$$SIM(S,T,w) = \sum_{d=1}^{\infty} w_d \cdot A_d \tag{3.18}$$

where w is a vector of weights. w_d is the weight at position d. Then $0 \leq SIM \leq \sum_1 w_d$, and if w is convergent, each A_d has a fixed contribution $w_d / \sum_1 w_d$. An example of convergent series is the geometric progression, where the d term has the value p^{d-1}, for $0 < p < 1$, and the infinite sum is:

$$\sum_{d=1}^{\infty} p^{d-1} = \frac{1}{1-p}. \tag{3.19}$$

Setting w_d to $(1-p)\dot{p}^{d-1}$, so that $\sum_d w_d = 1$, derives rank-biased overlap:

$$RBO(S,T,p) = (1-p)\sum_{d=1}^{\infty} p^{d-1} \cdot A_d. \tag{3.20}$$

RBO assume values in the range [0,1], where 0 means disjoint, and 1 means identical. The parameter p is called *persistence* and determines the top-weightedness aspect. The smaller p is, the more importance the top positions have in relations to the latest ones. When $p = 0$ only the first element is considered and RBO can assume value 0 or 1. On the other hand, when p goes to 1 the weights become more flat and all positions assume the same importance.

3.4 Cross-correlation

In signal processing, cross-correlation is a measure of similarity of two series as a function of the displacement of one relative to the other. This is also known as a sliding dot product or sliding inner product. As an example, consider two real-valued functions f and g differing only by an unknown shift along the x-axis. One can use the cross-correlation to find how much g must be shifted along the x-axis to make it identical to f. The formula essentially slides the g function along the x-axis, calculating the integral of their product at each position. When the functions match, the value of $(f \star g)$ is maximized. For continuous function f and g the cross correlation is given by

$$(f \star g)(\tau) \triangleq \int_{-\infty}^{+\infty} \overline{f(t)} g(t+\tau) dt \tag{3.21}$$

where $\overline{f(t)}$ represents the complex conjugate of $f(t)$ while τ is the displacement or *lag*. [14]

3.4.1 Cross-correlation between two 2-dimensional arrays

To compute the cross-correlation of two matrices, compute and sum the element-by-element products for every offset of the second matrix relative to the first. This can be used to calculate the offset required to get 2 matrices of related values to overlap [15]. The 2-D cross-correlation of an M-by-N matrix X, and a P-by-Q matrix, H, is a matrix C, of size $(M+P)-1$ by $(N+Q)-1$. Its elements are given by

$$\mathbf{C}(k,l) = \sum_{M=0}^{M-1} \sum_{n=0}^{N-1} \mathbf{X}(m,n) \bar{\mathbf{H}}(m-k, n-l) \tag{3.22}$$

where

$$-(P-1) < k < M-1$$

$$-(Q-1) < l < N-1$$

and $\bar{\mathbf{H}}$ is the complex conjugate of $\mathbf{H}$. From a practical point of view, the output matrix, $\mathbf{C}(k,l)$ has negative and positive row and column indices:

- A negative row index corresponds to an upward shift of the rows of $\mathbf{H}$.
- A negative column index corresponds to a leftward shift of the columns of $\mathbf{H}$.
- A positive row index corresponds to a downward shift of the rows of $\mathbf{H}$.
- A positive column index corresponds to a rightward shift of the columns of $\mathbf{H}$.

3.5 Hungarian Algorithm

The Hungarian Matching Algorithm is an optimization method to solve the assignment problem in polynomial time. The method was developed by Harold Kuhn in 1955 who gave the name "Hungarian method" because the algorithm was largely based on the earlier works of two Hungarian mathematicians: Dénes Kőnig and Jenő Egerváry.

3.5.1 The assignment problem

An allocation problem is defined on a bipartite graph e.g. a graph $G = (N, A)$ where $N = N_1 \cup N_2$ and $A \subseteq N_1 \times N_2$. Each arc $(i, j) \in A$ has a cost $c_{i,j}$. The goal is to find the lowest-cost matching set $B \subseteq A$ such that:

$$B = \{C(A') : A' \subset A, \forall i \in N_1 : \exists j \in N_2 : (i, j) \in A' \forall j \in N_2 : \exists i \in N_1 : (i, j)\}, \tag{3.23}$$

and

$$C(A') = \sum_{(i,j)\in A'} c_{i,j}. \tag{3.24}$$

3.5.2 The Algorithm

In its original form (the one presented in 1955), given n the number of nodes of each part, the algorithm has time complexity $\mathcal{O}(n^4)$ while another more recent implementation has a time complexity of $\mathcal{O}(n^3)$. The implementation shown below is the 1955 one. Let be C the $(n \times n)$ cost matrix. Let be C the $(n \times n)$ cost matrix.

- Step 0 (initialization). Subtract the least element of each row from that row of C. Then, do likewise for each column. The resulting matrix, C^0 has a zero in every row and column. (Further, a least-cost assignment for C^0 is also a least-cost assignment for C) Redefine $C = C^0$.
- Step 1 (cover zeros). Draw the minimum number of lines through the rows and columns to cover all zeros in C. If that minimum is n, you can assign i to j such that $C_{ij} = 0$; then you can remove row i and column j, repeating the process to obtain an optimal assignment. Otherwise, if that minimum is greater than n, continue.
- Step 2 (revise C). Select the minimum uncovered element. Subtract this from each uncovered element and add it to each twice-covered element (i.e., to those with both horizontal and vertical intersecting). Return to step 1 [16].

Chapter 4

Proposed Framework

4.1 Million Song Dataset

Million Song Dataset (**MSD**) is a collection of one million contemporary popular music tracks. It is presented by Bertin-Mahieux et al in [17]. It is by far the largest dataset available. Instead of including raw audio of tracks, MSD contains audio features for each song in an array format for legal reasons. In addition to that, MSD includes also metadata of each song and artists. The dataset was built by exploiting the API of *The Echo Nest* [18], a music intelligence platform acquired by Spotify in 2014 [19]. From a practical point of view, the whole dataset was obtained using a python wrapper called *pyechonest* [20].

4.1.1 Content

The MSD contains audio features and metadata for a million contemporary popular music tracks. It contains:

- 280GB of data
- 1,000,000 songs/files
- 44,745 unique artists
- 7,643 unique terms (Echo Nest tags)
- 2,321 unique musicbrainz[1] tags
- 43,943 artists with at least one term (i.e. a terms list associated with at least one tag)

[1] `https://musicbrainz.org/`

- 2,201,916 asymmetric similarity relationships
- 515,576 dated tracks starting from 1922

The data are stored using the HDF5 format[2], which is one of the most suitable file formats for storing efficiently heterogeneous data. Each .hdfs file represents a song and contains multiple tree-organized binary arrays. The feature content of each song is listed in table 4.1. The main acoustic features are *pitches*, *timbre* and *loudness*. Pitches and timbre are a set of 12 values for each *segment*, a segment is a portion of the track, the segments are usually delimited by note onsets, or other discontinuities in the signal. Figure 4.1 shows a representation of these three main features.

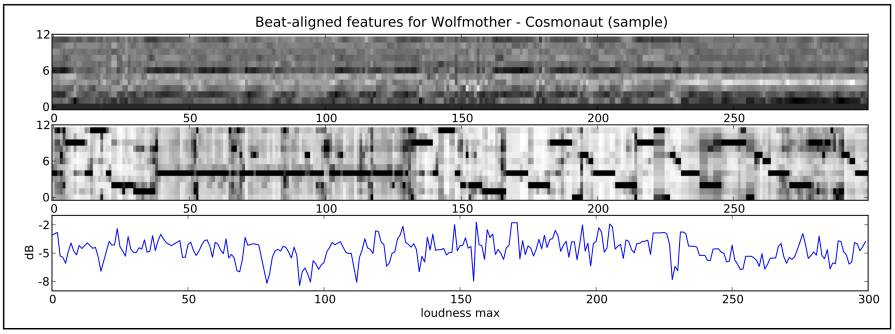


Figure 4.1: Example of audio features (timbre, pitches and loudness max) for one song from the MSD dataset

4.1.2 Usage

The dataset is useful for many purposes:

- Metadata analysis.
- Artist recognition.
- Automatic music tagging, each artist has a variable number of tags called *terms*.
- Recommendation, the one we are interested in.
- Cover song recognition (see section 2.1.1).

[2] `http://www.hdfgroup.org/HDF5/`

- Lyrics analysis using the associated dataset musiXmatch. This type of analysis is useful, for instance, for mood prediction.

4.1.3 Terms overview

Table 4.2 shows the most common terms in descending order of occurrence, as they appear in the whole dataset.

4.1.4 Retrieval

Due to its considerable size, the whole dataset is only available through as an *Amazon Public Dataset snapshot* which can easily be attached to an Amazon EC2 virtual machine[3]. There is also a reduced version of the MSD called *MillionSong-Subset* which contains 1000 songs (1% of the whole dataset). The general rule of thumb is to develop code on the subset, then port it to the full dataset.

4.2 Pipeline Structure

In this section we provide a general overview of the approach chosen to retrieve ranking lists starting from the raw data set. Figure 4.2 shows the main phases of the process. The principal reference is a *.pkl* file that is initially built during the Data Acquisition phase. Then it is gradually updated. The framework needs to be executed on a machine equipped with at least 350 GB of RAM in order to work on the whole dataset. On the other hand, while working on the subset, 16 GB of RAM will be enough. A public repository is available on *GitHub*[4]. The following sections give a more detailed view of the main phases.

4.2.1 Data Acquisition

As mentioned in 4.1.4, MSD is available as an *Amazon Public Dataset snapshot*. To get the whole data, it is necessary to instantiate an EC2 machine with the MSD snapshot[5] attached. At this point, if you have enough computational power, you might prefer to download the data. This can be done by relying on SSH File Transfer Protocol. Once downloaded, 10^6 *.hdf5* files, one per song, are organized in a tree structure.

[3] `http://millionsongdataset.com/pages/getting-dataset/`

[4] `https://github.com/gigpir/MSD_Environment.git`

[5] `https://aws.amazon.com/it/datasets/million-song-dataset/`

Field name	Type	Description
analysis sample rate	float	sample rate of the audio used
artist 7digitalid	int	ID from 7digital.com or -1
artist familiarity	float	algorithmic estimation
artist hotttnesss	float	algorithmic estimation
artist id	string	Echo Nest ID
artist latitude	float	latitude
artist location	string	location name
artist longitude	float	longitude
artist mbid	string	ID from musicbrainz.org
artist mbtags	array string	tags from musicbrainz.org
artist mbtags count	array int	tag counts for musicbrainz tags
artist name	string	artist name
artist playmeid	int	ID from playme.com, or -1
artist terms	array string	Echo Nest tags
artist terms freq	array float	Echo Nest tags freqs
artist terms weight	array float	Echo Nest tags weight
audio md5	string	audio hash code
bars confidence	array float	confidence measure
bars start	array float	beginning of bars, usually on a beat
beats confidence	array float	confidence measure
beats start	array float	result of beat tracking
danceability	float	algorithmic estimation
duration	float	in seconds
end of fade in	float	seconds at the beginning of the song
energy	float	energy from listener point of view
key	int	key the song is in
key confidence	float	confidence measure
loudness	float	overall loudness in dB
mode	int	major or minor
mode confidence	float	confidence measure
release	string	album name
release 7digitalid	int	ID from 7digital.com or -1
sections confidence	array float	confidence measure
sections start	array float	largest grouping in a song, e.g. verse
segments confidence	array float	confidence measure
segments loudness max	array float	max dB value
segments loudness max time	array float	time of max dB value, i.e. end of attack
segments loudness max start	array float	dB value at onset
segments pitches	2D array float	chroma feature, one value per note
segments start	array float	musical events, ∼note onsets
segments timbre	2D array float	texture features (MFCC+PCA-like)
similar artists	array string	Echo Nest artist IDs (sim. algo. unpublished)
song hotttnesss	float	algorithmic estimation
song id	string	Echo Nest song ID
start of fade out	float	time in sec
tatums confidence	array float	confidence measure
tatums start	array float	smallest rythmic element
tempo	float	estimated tempo in BPM
time signature	int	estimate of number of beats per bar, e.g. 4
time signature confidence	float	confidence measure
title	string	song title
track id	string	Echo Nest track ID
track 7digitalid	int	ID from 7digital.com or -1
year	int	song release year from MusicBrainz or 0

Table 4.1: List of the 55 fields provided in each per-song HDF5 file

One one side, having multiple small files guarantees a certain degree of flexibility, since it allows to load and release just the needed ones without having to allocate

Term	Occ.	Term	Occ.	Term	Occ.	Term	Occ.
rock	27272	deep house	4677	acid jazz	2544	dancehall	1527
electronic	24072	80s	4623	power pop	2469	doom metal	1506
pop	19635	french	4588	hard house	2458	gothic rock	1504
alternative rock	15372	ballad	4517	oldies	2440	beats	1482
hip hop	13836	industrial	4298	dj	2388	melodic	1481
united states	13512	trip hop	4274	canada	2368	english	1466
house	11835	progressive house	4249	lo-fi	2307	spanish	1446
jazz	11671	male vocalist	4185	german	2299	gothic	1436
alternative	11089	minimal	4180	70s	2238	happy hardcore	1433
indie	11043	heavy metal	4126	new york	2225	funky	1433
electro	10972	90s	4082	bass	2180	uk garage	1418
experimental	10009	psychedelic	4076	60s	2179	england	1406
indie rock	9987	easy listening	3991	remix	2166	world music	1395
pop rock	9731	funk soul	3965	spain	2082	male	1394
punk	9692	soft rock	3771	chanson	2081	tribal	1388
techno	9599	00s	3771	thrash metal	2062	heavy	1388
folk	9101	progressive rock	3758	fusion	2057	underground	1346
downtempo	8952	indie pop	3719	singer	1996	swedish	1345
ambient	8285	r&b	3673	party music	1945	group	1329
soul	8181	progressive trance	3639	british pop	1944	ebm	1324
germany	8074	noise	3537	acid	1939	canadian	1318
electronica	7494	drum and bass	3463	dark	1893	groove	1312
trance	7445	emo	3438	mellow	1884	latin jazz	1306
disco	7427	california	3437	urban	1880	world reggae	1304
american	7282	british	3411	original	1880	metalcore	1299
synthpop	7138	abstract	3347	art rock	1879	grindcore	1286
world	7080	intelligent dance music	3312	big beat	1868	progressive metal	1261
funk	6908	beautiful	3252	black	1862	glitch	1260
blues	6824	classic	3206	grunge	1846	contemporary jazz	1250
hardcore	6670	club	3125	smooth jazz	1831	mix	1244
country	6509	progressive	3019	italian disco	1815	japanese	1240
dance	6502	hard trance	2973	alternative pop rock	1812	relax	1235
acoustic	6497	ska	2947	japan	1807	german pop	1219
breakbeat	6122	death metal	2933	cover	1770	pop punk	1217
reggae	6105	blues-rock	2919	sexy	1747	underground hip hop	1210
female vocalist	6012	post rock	2876	rock 'n roll	1695	stoner rock	1207
classic rock	5920	psychedelic rock	2873	sweden	1691	alternative country	1192
metal	5895	europop	2863	jazz funk	1678	political	1189
vocal	5756	pop rap	2857	black metal	1669	comedy	1187
latin	5633	female	2816	rockabilly	1649	hardcore punk	1182
singer-songwriter	5630	classical	2797	jungle music	1647	old school	1181
folk rock	5623	country rock	2706	avant-garde	1613	deep	1174
hard rock	5614	lounge	2679	belgium	1600	female vocals	1169
instrumental	5581	future jazz	2677	soul jazz	1598	gospel	1161
rap	5538	los angeles	2676	americana	1579	garage	1155
chill-out	5452	euro-house	2674	shoegaze	1578	london	1154
dub	5449	italy	2660	beat	1574	free jazz	1145
guitar	5428	nederland	2650	swing	1573	drums	1132
new wave	5163	european	2642	tribal house	1544	nu metal	1126
tech house	5057	piano	2588	christian	1539	roots reggae	1124
soundtrack	4938	garage rock	2554	broken beat	1528	modern classical	1121

Table 4.2: List of the most common terms in the dataset

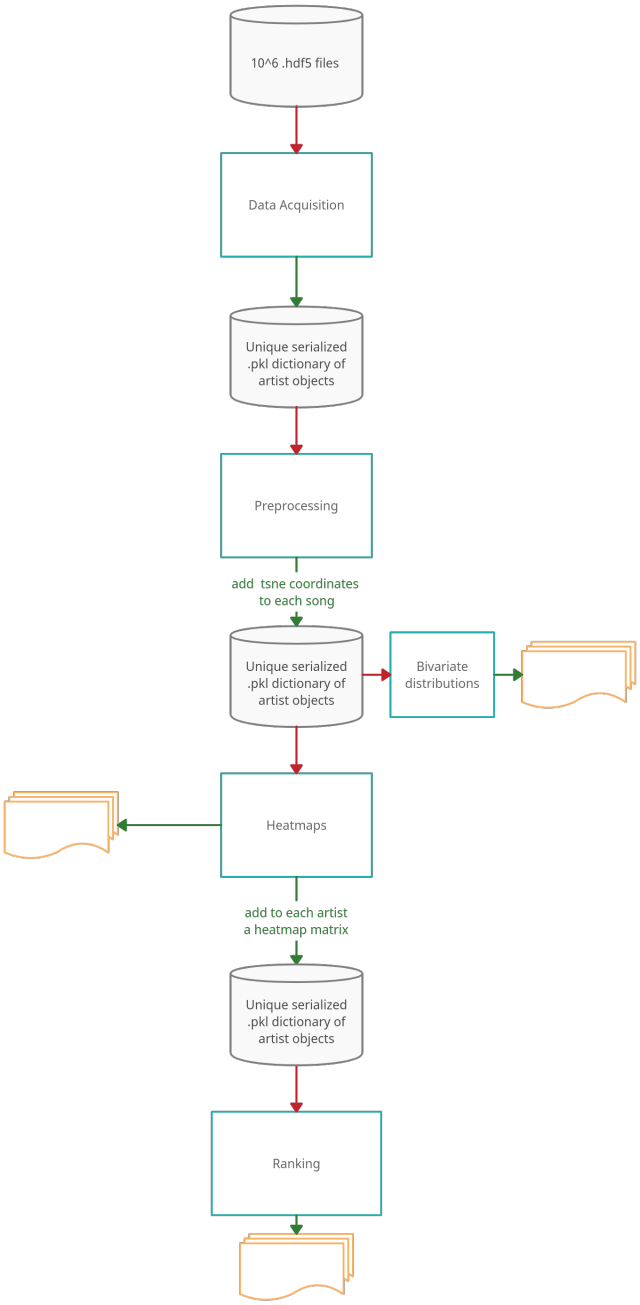


Figure 4.2: Main pipeline

a considerable amount of memory. On the other side, I/O operations are known to be slow. Every time you need data for a certain song you will perform an I/O task. Also, it is likely that you will need the same resource in different moments in time.

This means that you will perform multiple I/O operations on same resources and this is potentially inefficient.
A solution, resources permitting, is to have data coming from all files condensed in one. Loading and saving is easier and faster. Furthermore, having all data saved in RAM means that you can take advantage of the efficient *search by key* feature provided by the python associative arrays[6] better known as *dictionaries*.
Each element of the dictionary is a custom object of type *Artist*. Each artist contains some fields plus another dictionary of custom objects of type *Song*. Tables 4.4 and 4.5 show information about fields and types of these two classes.
Every file is saved through serialization, therefore it is a valid alternative with respect to .hdf file format in terms of time. The tables represented in figures 4.4 and 4.5 show the fields we kept to build our version of the dataset. The downside of this approach is that a considerable amount of RAM is essential.

Table 4.3: Description of custom classes fields

Table 4.4: class Artist

Fields	Type
name	str
id	str
terms	list<str>
similar_artists	list<str>
song_list	dict<Song>
tsne_heatmap	matrix(20, 20)
my_similar_artists	list<str>

Table 4.5: class Song

Fields	Type
id	str
name	str
loudness	float
segments_pitches	matrix(n, 12)
segments_timbre	matrix(n, 12)

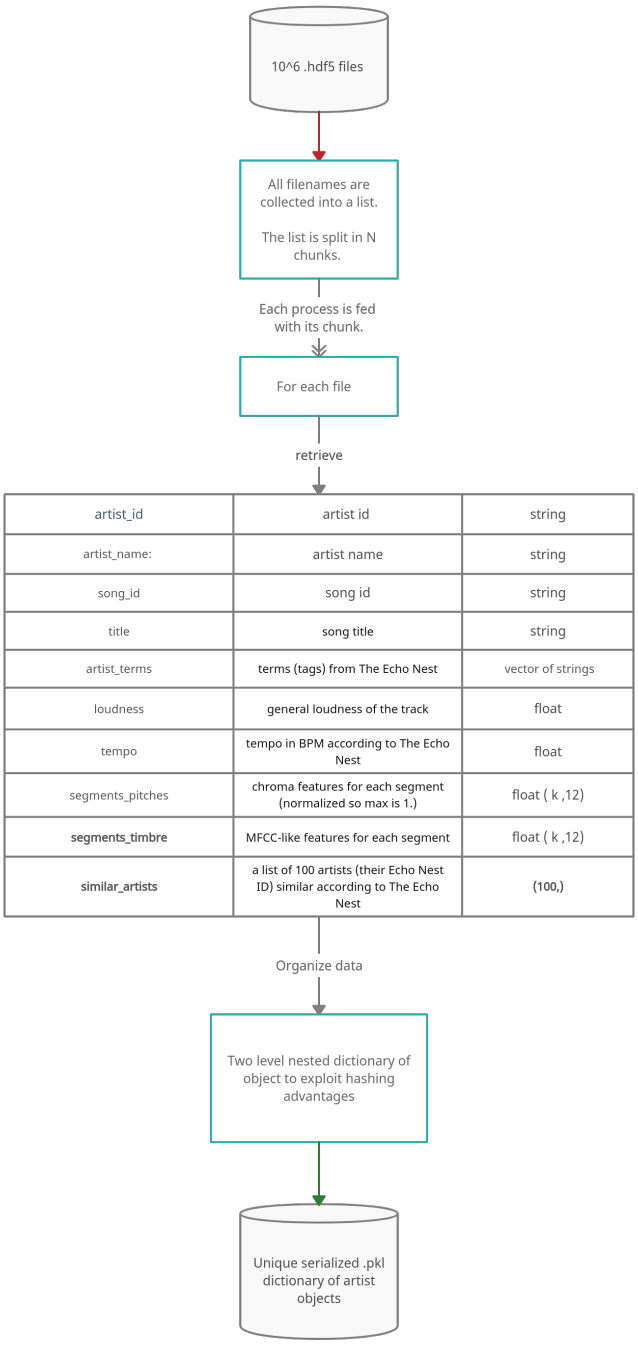
4.2.2 Preprocessing

The main goal of this phase is to transform the input (timbres, pitches, loudness and tempo) in order to obtain an output which is better suited to large-scale data processing. As shown in figure 4.4, this stage articulates as follows:

1. For each song, transform the initial input format into an n-dimensional feature vector.

2. Arrange data as a matrix where each row represents a song and each coloumn represents a feature in the n-dimensional space.

[6] `https://docs.python.org/3/tutorial/datastructures.html`



artist_id	artist id	string
artist_name:	artist name	string
song_id	song id	string
title	song title	string
artist_terms	terms (tags) from The Echo Nest	vector of strings
loudness	general loudness of the track	float
tempo	tempo in BPM according to The Echo Nest	float
segments_pitches	chroma features for each segment (normalized so max is 1.)	float (k ,12)
segments_timbre	MFCC-like features for each segment	float (k ,12)
similar_artists	a list of 100 artists (their Echo Nest ID) similar according to The Echo Nest	(100,)

Figure 4.3: Data acquisition pipeline

3. Perform outlier remotion.

4. Standardize data.

5. Apply *t-SNE* to map the n-dimensional space to a reduced 2-dimensional feature space.

Transforming initial input format

As shown before, the selected features are Loudness (dB), Tempo (BPM), segments_pitches and segments_timbre. The latters two are matrices where each row refers to a particular time segment of the track. This means that the number of rows is variable while the number of columns is fixed to 12 for both matrices.
In order to create a proper space for the representation. The songs in the dataset must have a fixed number of features. The 1-dimensional features (loudness and tempo) are directly ported into the new space while the two matrices need to be transformed (e.g. reduced) without losing too much information. From a practical point of view, each column of the matrices indicates the changes of a certain indicator over time segments. For instance, if we just take the average of each column we would neglect all information about the evolution of that indicator over time. On the other side, adding certain personalized features might damage the clarity of the dataset.

In this regard, we propose four different configurations:

- Mode 0 (**m0**). This configuration includes 24 mean values, loudness and tempo (26 features).

- Mode 1 (**m1**). This configuration includes 24 mean values, 24 variance values, loudness and tempo (50 features).

- Mode 2 (**m2**). This configuration includes 24 mean values, 24 variance values, 24 first derivatives, loudness and tempo (74 features).

- Mode 3 (**m3**). This configuration includes 24 mean values, 24 variance values, 24 first derivative, 24 second derivatives, loudness and tempo (98 features).

Where:

- The mean values are the average terms. Each one is computed over its respective column.

- The variance values represent the variance of each column.

- The first derivative terms are computed by averaging the gradient vector of each column. The gradient vector is computed using central differences in the

interior and first differences at the boundaries. This means that, given a n sized vector x, the interior values of the gradient vector g will be given by:

$$g[i] = \frac{x[i+1] - x[i-1]}{2},$$

while the elements at the boundaries are given by:

$$g[0] = x[1] - x[0];$$

$$g[n-1] = x[n-1] - x[n-2].$$

- The second derivatives values are retrieved by computing the gradient two times and then by averaging it.

Arrange data as a matrix

Once selected a configuration, the matrix X is generated such that each row represents a song as a n-dimensional point.

Outlier remotion

At this point, since the features might have different scales, the outlier remotion is performed on each feature separately. The chosen criterion to define if an element is an outlier is quite common. Given a distribution X, its mean μ and its standard deviation σ, $x \in X$ is considered an outlier if $x \notin [\mu - \tau\sigma, \mu + \tau\sigma]$, where τ is an arbitrarily threshold (in this framework it is set to 3.5). A song is classified as outlier if at least one of its feature is an outlier.

Normalization

With the aim of reducing the dimensionality, the features must be on a common scale. The method used for normalization is called *robust scaler* [7]. It is similar to a standardization where, given an element of a distribution $x \in X$, it is converted to a standard score z following the formula

$$z = \frac{x - \mu_X}{\sigma_X}. \tag{4.1}$$

The standard distribution Z has $\mu_Z = 0$ and $\sigma_Z = 1$. The robust scaler process differs because μ_X and σ_X are computed only by taking into account those values that are included into an arbitrarily chosen *quantile range*. We tune this interval to $[15, 85]$ as an optimal trade-off between stability to anomalies and consistency.

[7] `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html`

Dimensionality reduction with t-SNE

At this stage the framework relies on implementation of t-SNE algorithm (see section 3.1) provided by *scikit-learn*[8]. An effective configuration of the paramenter is the following:

```
sklearn.manifold.TSNE(n_components=2,learning_rate=1000,n_jobs=-1,
    perplexity=30)

```

It is essential to tune properly these parameters, especially the *learning rate*. Indeed, a certain value that fits on a dataset with a specific number of points might perform poorly on a larger dataset. In these cases, you may notice that all data points are heavily condensed in a small region of the t-SNE space therefore they look "like a ball".

By applying *t-SNE* algorithm on the four configurations mentioned in section 4.2.2, we get four different versions of *t-SNE* coordinates. We adopt this approach to understand if a configuration performs better than the others in the further steps. Now we attach *t-SNE* coordinates to each song and we update and save four versions of the dataset.

4.2.3 Heatmap creation

Figure 4.6 shows the main phases of the heatmap creation process. The main goal is to find a way to compare artists without involving a variable amount of songs (points in tSNE space). The idea is to create a map for each artist and to populate it with its songs. The process starts by retrieving the range values over each of the two t-SNE coordinates i.e. the minimum and the maximum values. Then, for each artist, a matrix of zeros is created. The code below shows how the matrix is filled.

```
for a_id in artists_ids:
    if len(artists[a_id].song_list.values()) != 0:
        # the artist a_id has actually >0 songs associated

        n_outlier = 0
        # initialize heatmap to all zeros
        result[a_id] = np.zeros((dimension, dimension))
        for song in artists[a_id].song_list.values():
            # check if song is an outlier
            if song.tsne != None

                # assign to coordinates tsne_0 and tsne_1
                # a row and a column index respectively
                # max and min are the boundaries
```

[8] https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

```
                # of the t-SNE space
                row_idx = int(((song.tsne[0] + abs(min[0]))
                    / (max[0] + abs(min[0]))) * dimension)
                col_idx = int(((song.tsne[1] + abs(min[1]))
                    / (max[1] + abs(min[1]))) * dimension)

                # add a song to that region
                result[a_id][row_idx, col_idx] += 1
            else:
                #song is an outlier
                n_outlier += 1

        # check if the the artist a_id has
        # at least one inliner song
        if len(artists[a_id].song_list) - n_outliers != 0:

            # normalize by number of artists inliner songs
            result[a_id] /= len(artists[id].song_list)
                - n_outliers
        else:
            # The artist is considered as an outlier
            # -> set its heatmap to null value
            result[a_id] = None

    else:
        # The artist is considered as an outlier
        # -> set its heatmap to null value
        result[a_id] = None
return result

```

Listing 4.1: Heatmap generation function

Figure 4.5 shows some sample heatmaps.

4.2.4 Ranking generation

At this point, the framework aims to compare heatmaps (which represent artists) to each other to state if they are similar or not. Figure 4.7 shows the main phases. In terms of metrics, if the *distance* between two heatmaps is *sufficiently small*, then those two associated artists will be *similar* to each other. After an extensive research about metrics of distance between matrices, we propose two custom metrics:

- **cc_peak_1**. It is a metric that exploits cross-correlation. Identifies the coordinates of the peak of the resulting matrix and computes a version of the euclidean distance between this point and another reference point.

- **biv_hun**. It is a metric that builds an adjacency matrix of the complete bipartite graph where the nodes are non-zero cells of each heatmap. The matrix is then filled with the *weighted* version of the euclidean distance between positions of the nodes. Then it solves a combinatorial optimization algorithm using the *Hungarian algorithm* and retrieves the minimal cost associated and returns it as a distance between matrices.

cc_peak_1 metric

Given two 20×20 matrices (or heatmaps), the cross-correlation matrix is retrieved (see section 3.4). The cross-correlation matrix has dimension 39×39. At this point, the algorithm retrieves the position in which the cross-correlation matrix reaches the maximum value. Practically, the position of this value reveals the entity of the shift to applying to one of the heatmaps to make it as similar as possible to the other one. At this stage, if the peak value is larger than the mean of all other values of the cross-correlation matrix the entity of the shift is taken as distance. Otherwise, the distance is set to the largest possible value for this metric. Here is the code:

```
1
2  # define the coordinates of the point
3  # corresponding to feeding the algorithm
4  # with identical matrices
5  shft_0 = np.array([19, 19])
6
7  # compute cross correlation matrix
8  X = signal.correlate2d(h1, h2)
9
10 # find peak in matrix
11 peak = find_peaks(data=X, threshold=0, box_size=1, npeaks=1)
12     try:
13         # retrieve the value of the peak
14         peak_value = p['peak_value']
15
16         # compute the mean of cross-correlation
17         # matrix excluding the peak
18         X[p['x_peak'], p['y_peak']] = 0
19         mean = np.sum(X) / (39 * 39 - 1)
20
21         if peak_value > mean:
22             # compute the entity of the shift
23             # as an euclidean distance
24             dist = np.linalg.norm(np.array([p['x_peak'], p['y_peak
    ']]) - shft_0)
25         else:
26             # otherwise set to maximum shift possible
27             dist = np.sqrt((19 ^ 2) + (19 ^ 2))
```

```
    except:
        print("NO PEAK FOUND")
        dist = np.sqrt((19 ^ 2) + (19 ^ 2))

return dist


```

Listing 4.2: cc_peak_1 metric function

biv_hun metric

Given two 20×20 matrices (or heatmaps), firstly, for each heatmap, the number of non zero values are retrieved. Then an adjacency matrix is instantiated and filled with a weighed version of euclidean distances between non-zero nodes. The Hungarian algorithm (see section 3.5) is fed with the adjacency matrix. The total distance is given by the sum of the cost plus a penalty for non-matching nodes. Here is the code:

```


# find non-empty bins number for each heatmap
non_zero_h1 = np.count_nonzero(h1)
non_zero_h2 = np.count_nonzero(h2)

# instantiate adjancency matrix
# of complete bipartite graph
distances = []

count1 = 0
# fill in adjacency matrix with link lengths
for i in range(dim):
    for j in range(dim):
        if h1[i, j] > 0:
            count2 = 0
            row = []
            for k in range(dim):
                for w in range(dim):
                    if h2[k, w] > 0:
                        # compute euclidean distance considering
                        # also the diffenence between
                        # 2 heatmap values * w
                            row.append( np.sqrt(
                                (k - i) ** 2 + (w - j) ** 2 +
                                (w *(h2[k, w] - h1[i, j])) ** 2))
                            count2 += 1
            count1 += 1
            distances.append(row)
```

```
distances = np.array(distances)

if non_zero_h1 > non_zero_h2:
    distances = distances.transpose()

#compute hugarian algorithm
row_ind, col_ind = linear_sum_assignment(distances)

# retreive non matched columns
non_matched_distances = np.delete(distances, col_ind, 1)
cost = distances[row_ind,col_ind].sum()
penalty = 0
# there is a penalty only if the adjacency
# matrix is not squared
if non_zero_h1 != non_zero_h2:
    penalty = np.amin(non_matched_distances).sum()
total = cost + penalty
return total
```

Listing 4.3: biv_hun metric function

4.2.5 Evaluation

The evaluation phase is based on two different approaches.

1. By comparing our ranking with the one provided by the MSD accessible through the field *similar_artists* attached to each artist object.

2. By comparing the tags lists associated with each artist accessible through the field *terms*. For instance, one could see if, given an artist, the other artists at the top of our ranking have more tags in common than those ones at the bottom.

comparing rankings

This paragraph addresses point 1 of the list above. Given an artist A with its ranking of similar artists GT and the ranking computed using this framework R, we propose to compare two rankings are:

- Rank-biased Overlap (RBO) (see section 3.3.1), is a method for comparing undefined ordered lists giving more weight to top-ranked items.

- Computing the size of the intersection between GT and the first i elements of R and dividing the result by the size of GT, by increasing i, one can see how many positions are required to reach the maximum intersection. This approach doesn't consider the order of rankings of list GT.

comparing tags

The tags list is unordered, thus the framework propose:

- To compute the size of the intersection between the tags of an artist A and the tags of the artist $A_{r(i)}$ picked from position i of the ranking of A (not_norm).
- Same as before but, in addition, the final result is divided by the number of tags of artist A ($norm\_std$).
- Same as first point but, in addition, the final is divided by the number of tags of artist $A_{r(i)}$ ($norm\_other$).

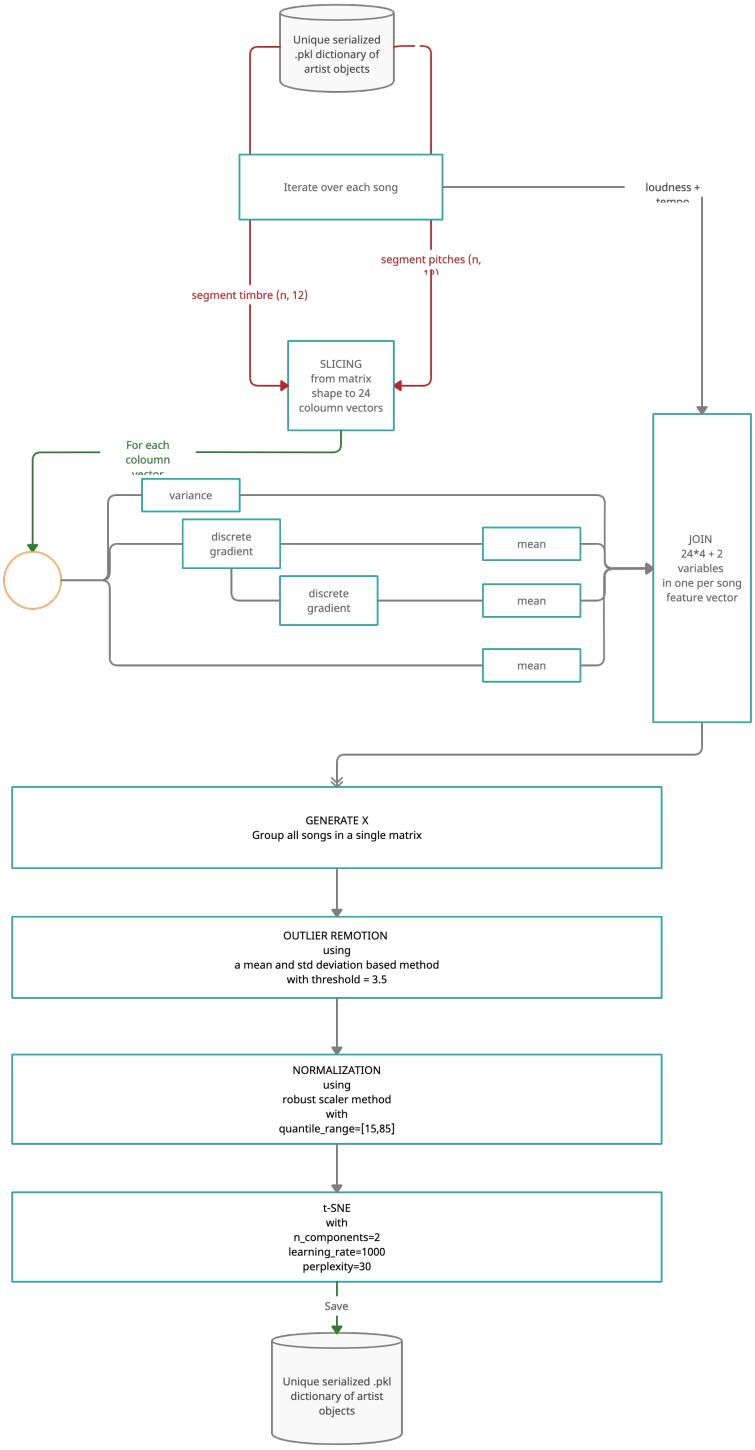


Figure 4.4: Preprocessing pipeline

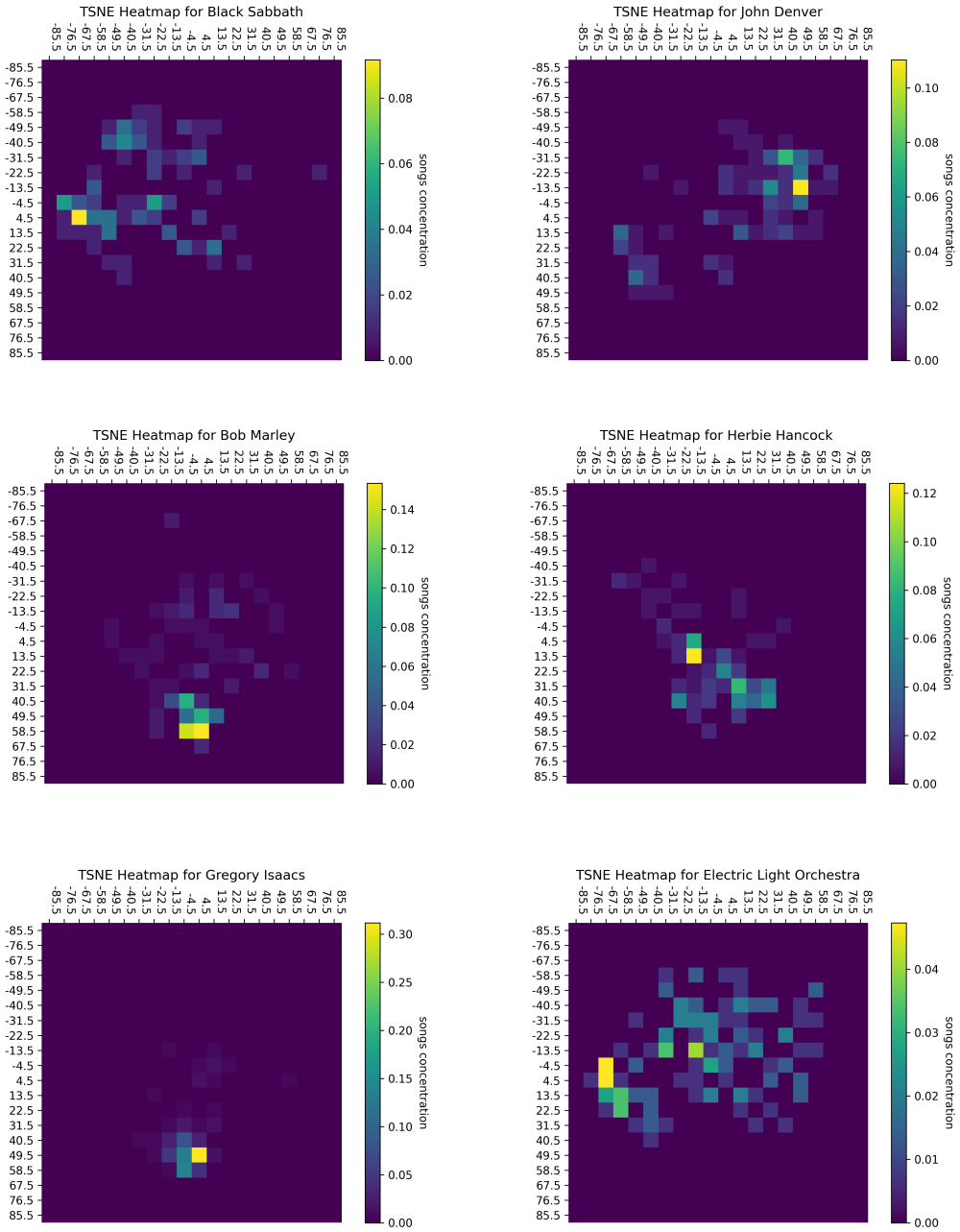


Figure 4.5: some examples of heatmaps

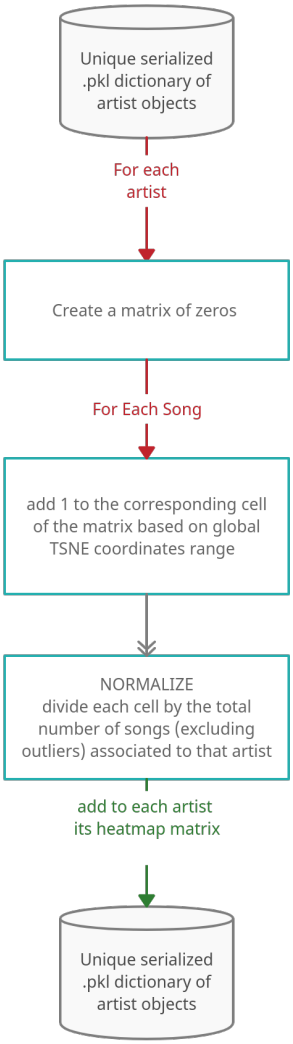


Figure 4.6: Heatmap generation pipeline

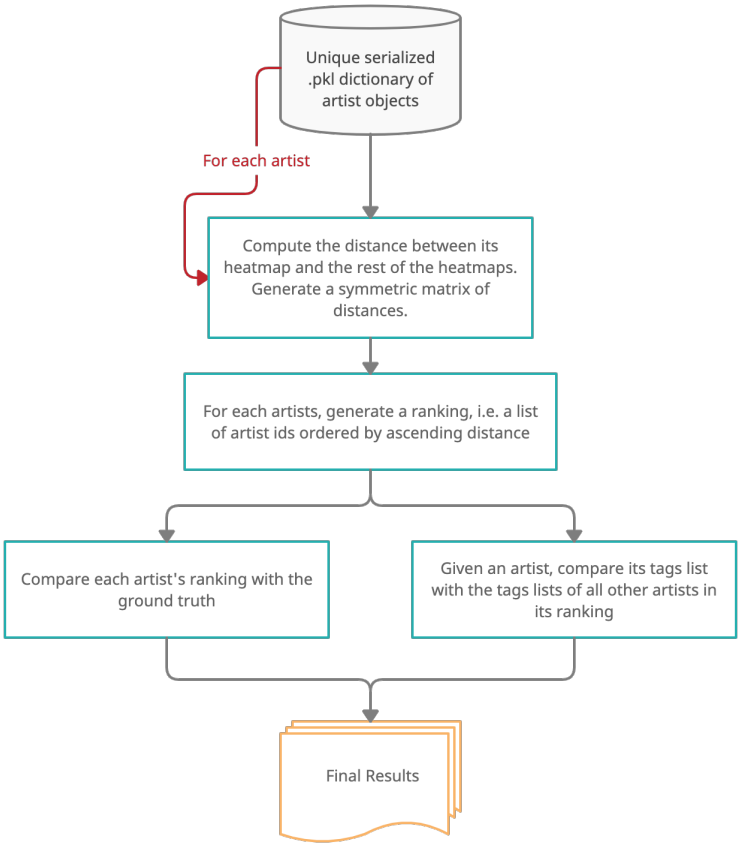


Figure 4.7: Ranking generation pipeline

Chapter 5

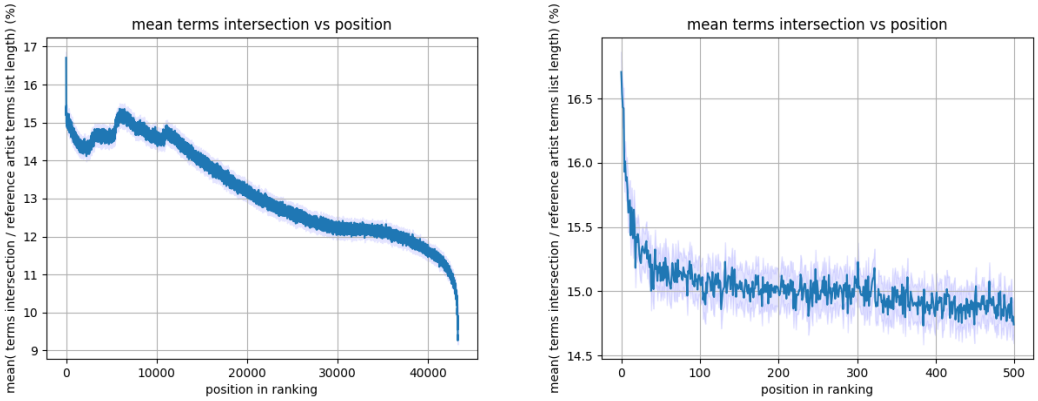
Numerical Assessments

5.1 Tag-lists comparison

Since we widely discussed the methodological approaches used in chapter 4, here we summarize the main steps to properly present our results. By starting from four different sets of t-SNE coordinates, four sets of heatmaps are produced. By comparing heatmaps with one another using two metrics, we got distances that were arranged as a symmetric matrix of distances. By sorting in ascending order each row of the matrix, a full ranking of similar artists is obtained, one for each artist. At this point, for each artist, we compare its terms list with those of the artists in its ranking and we plot the result as an average value over artists.

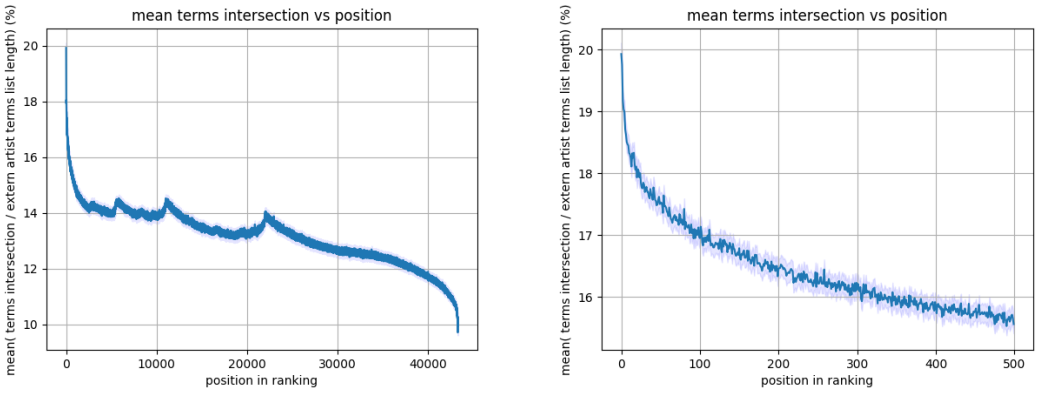
Briefly, we have:

- 4 sets of t-SNE coordinates: *m0*, *m1*, *m2* and *m3* (described in 4.2.2).
- 2 heatmap distance metrics: *biv_hun* and *cc_peak_1* (described in 4.2.4).
- 3 tag list comparison techniques: *norm_std*, *norm_other* and *not_norm* (described in 4.2.5).
- A filter to consider only the most common tags, i.e. those tags having a number of occurrences in the lists greater than a value n (see table 4.2 for an overview of the most common tags in the MSD).



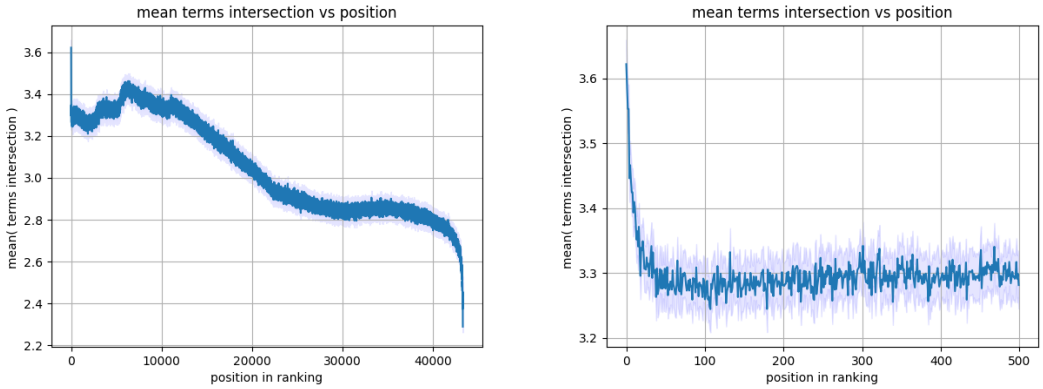
(a) Full ranking plot **(b)** First 500 ranking positions

Figure 5.1: intersection vs position mode: *m3*, metric: *biv_hun*, *norm-std*, terms with at least 100 occurrences



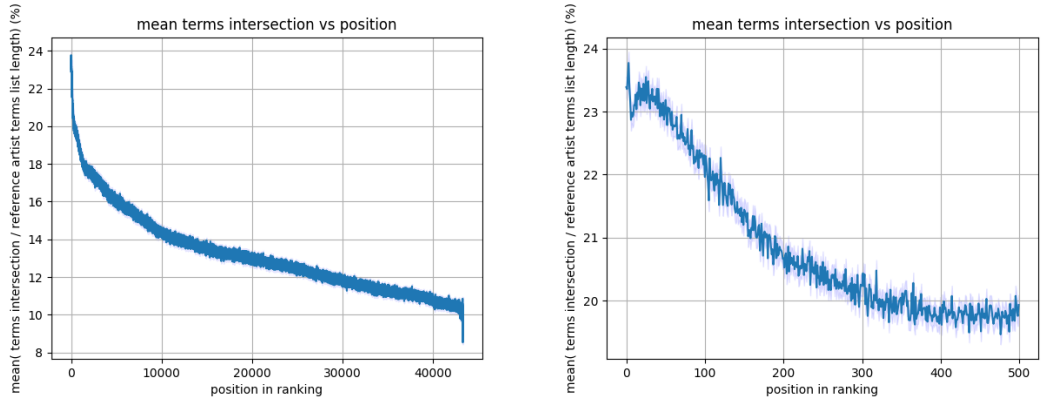
(a) Full ranking plot **(b)** First 500 ranking positions

Figure 5.2: intersection vs position mode: *m3*, metric: *biv_hun*, *norm-other*, terms with at least 100 occurrences



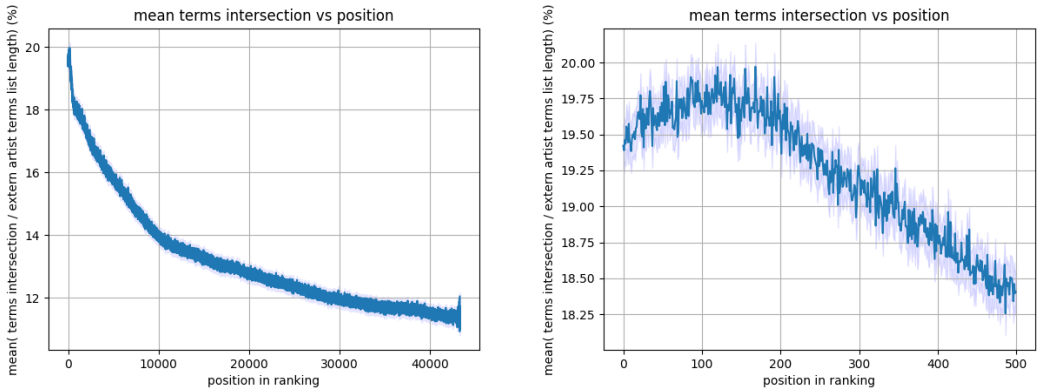
(a) Full ranking plot **(b)** First 500 ranking positions

Figure 5.3: intersection vs position mode: *m3*, metric: *biv_hun*, *not-norm*, terms with at least 100 occurrences



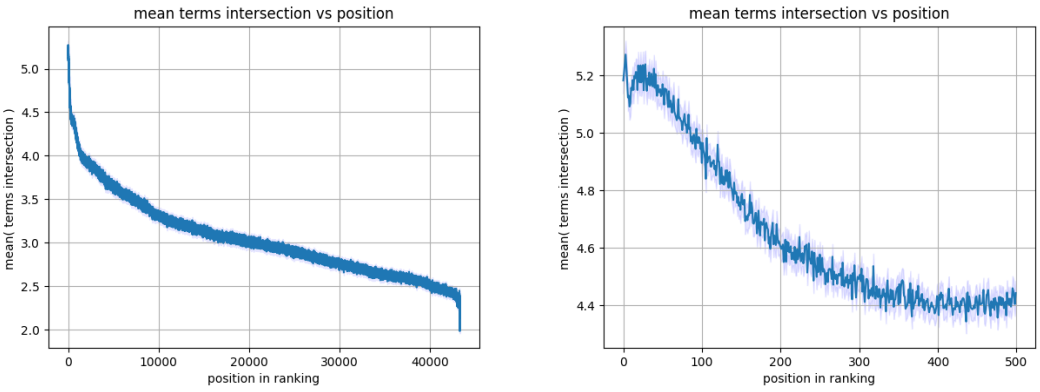
(a) Full ranking plot **(b)** First 500 ranking positions

Figure 5.4: intersection vs position mode: *m3*, metric: *cc_peak_1*, *norm-std*, terms with at least 100 occurrences



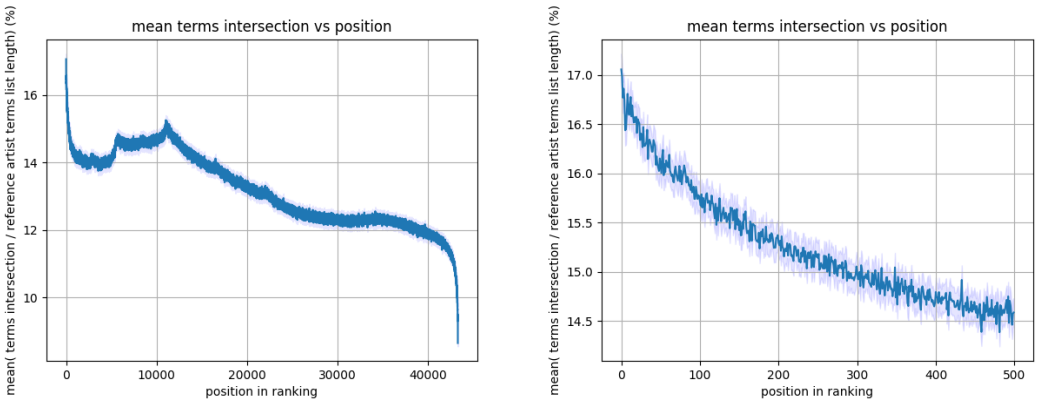
(a) Full ranking plot **(b)** First 500 ranking positions

Figure 5.5: intersection vs position mode: *m3*, metric: *cc_peak_1*, *norm-other*, terms with at least 100 occurrences



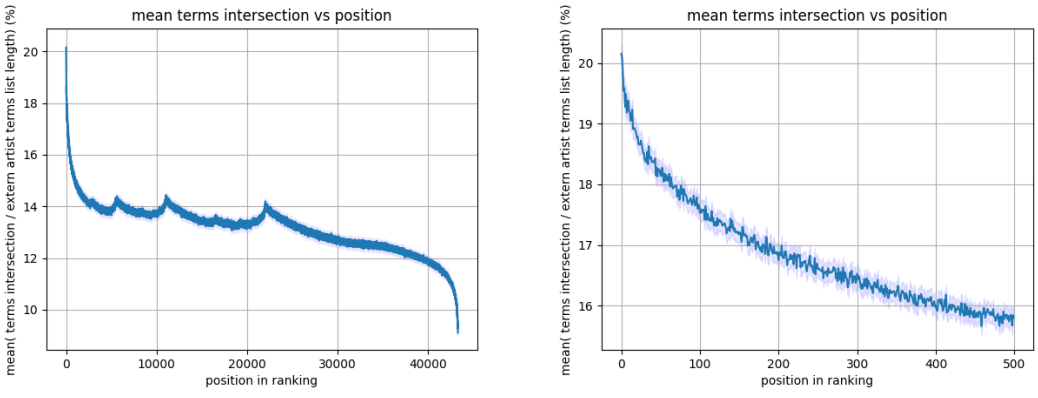
(a) Full ranking plot **(b)** First 500 ranking positions

Figure 5.6: intersection vs position mode: *m3*, metric: *cc_peak_1*, *not-norm*, terms with at least 100 occurrences



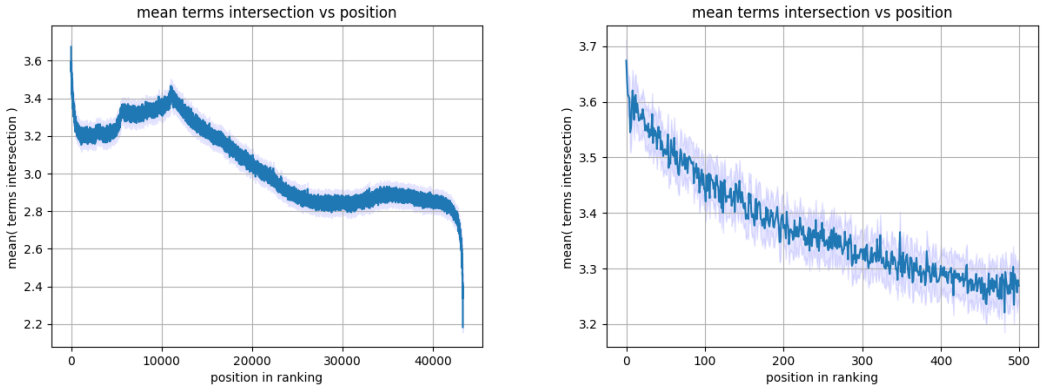
(a) Full ranking plot **(b)** First 500 ranking positions

Figure 5.7: intersection vs position mode: *m2*, metric: *biv_hun*, *norm-std*, terms with at least 100 occurrences



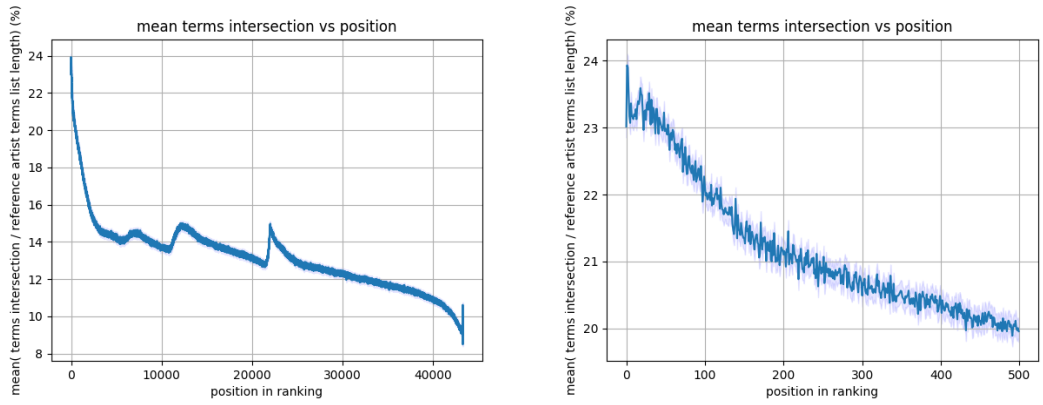
(a) Full ranking plot **(b)** First 500 ranking positions

Figure 5.8: intersection vs position mode: *m2*, metric: *biv_hun*, *norm-other*, terms with at least 100 occurrences



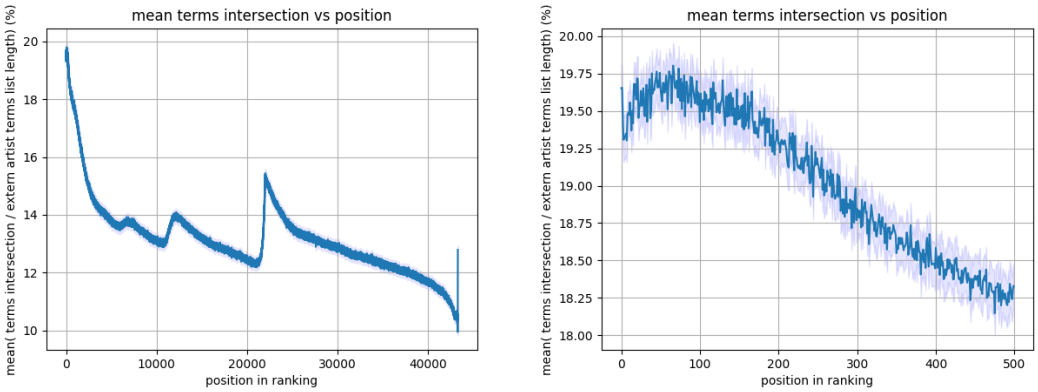
(a) Full ranking plot **(b)** First 500 ranking positions

Figure 5.9: intersection vs position mode: *m2*, metric: *biv_hun*, *not-norm*, terms with at least 100 occurrences



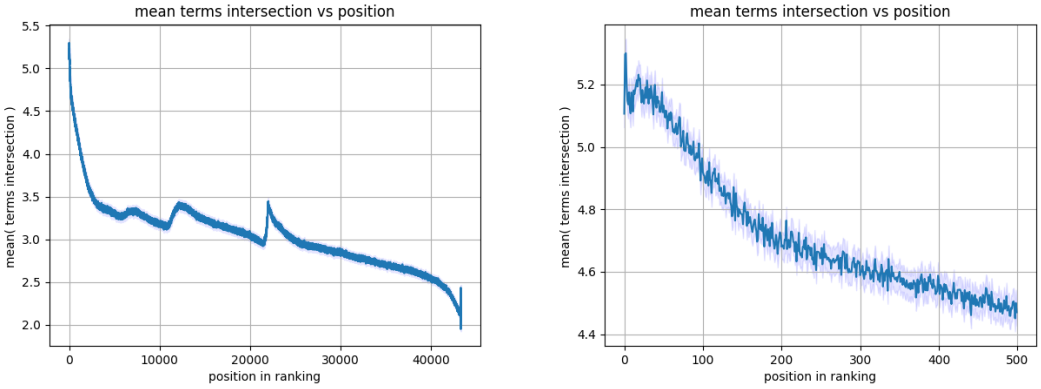
(a) Full ranking plot **(b)** First 500 ranking positions

Figure 5.10: intersection vs position mode: *m2*, metric: *cc_peak_1*, *norm-std*, terms with at least 100 occurrences



(a) Full ranking plot **(b)** First 500 ranking positions

Figure 5.11: intersection vs position mode: *m2*, metric: *cc_peak_1*, *norm-other*, terms with at least 100 occurrences



(a) Full ranking plot **(b)** First 500 ranking positions

Figure 5.12: intersection vs position mode: *m2*, metric: *cc_peak_1*, *not-norm*, terms with at least 100 occurrences

All the plots shown from figure 5.1 to figure 5.12 show a mostly decreasing trend. Globally, the results show that the metric *cc_peak_1* does a better job in terms of extension and regularity of the curve. All *m3* plots are characterized by a more regular curve than the *m2* ones as they generally present fewer spikes. On the other hand, since we are interested mostly in the left side of the plot, which represents just the top-ranked items, *m2* plots show a stronger descent at the beginning. The plots computed with *norm-std* have similar shape than the ones computed with no normalization *not-norm*. This phenomenon happens because *norm-std* approach (see 4.2.5) divides the intersection size by the number of tags of the reference artist,

which is constant along x-axis. The *norm-other* approach, instead, divides the intersection size by the tag-list size of the artist at position x. Therefore it could potentially modify the behaviour of the curve with respect to not-norm method. Similar results are obtained with configuration *m0* and *m1*, which are omitted for the sake of conciseness.

5.2 Rankings comparison

In this section, the rankings retrieved through the process described in section 4.2.4 are compared with the rankings provided by the MSD contained in the field *similar_artists* attached to each *Artist* object. As shown before, the field *similar_artists* contains an ordered list of *100* artist ids. Since not all of these ids are present in the MSD, a filter was applied to remove those ids that have no occurrences in the MSD changing the relative order of the ids in the list. According to the notation used in section 4.2.5, for each artist, we have its ranking of similar artists attached GT and 8 full-length rankings computed with all combinations of the parameters below:

- 4 sets of t-SNE coordinates: *m0*, *m1*, *m2* and *m3* (described in 4.2.2).
- 2 heatmap distance metrics: *biv_hun* and *cc_peak_1* (described in 4.2.4).

5.2.1 Comparison through RBO

As mentioned before, RBO (see 3.3.1) is a method that takes into account the order of the two ranked lists to be compared giving more importance to first positions rather than the last ones. For each artist, the RBO between GT and its *sub-ranking* of size 100 is computed. Note that, given an artist and its distance values with respect to other artists, its *full-length ranking* is obtained by sorting all these values in ascending order. A *sub-ranking* of size n is a subset of the full-length ranking truncated at position n. The final result is displayed in table 5.1 as an average. The results are not satisfactory at all. Results for configurations *m0* and *m1* are not reported as they are even less significant. Unfortunately, there is no available information on the methodological approach used to build up the GT rankings. As far as we know, the GT rankings are provided by The Echo Nest[1] and the algorithm used to perform such this operation is not public. Since the criterion used to build up the GT rankings is not known, it would seem unlikely that the similarity criteria involves only the musical features. It's much more likely that this process involves a mix of musical features plus other data coming from different

[1] `http://millionsongdataset.com/pages/example-track-description/`

contexts such as the users' listening history. It is known that Spotify[2], which acquired The Echo Nest, uses a recommendation algorithm based on a mixture of collaborative filtering models, natural language processing and audio models.

mode / metric	**biv_hun**	**cc_peak_1**
m3	0.003532	0.009022
m2	0.004604	0.009569

Table 5.1: RBO average scores

5.2.2 Comparison through intersection

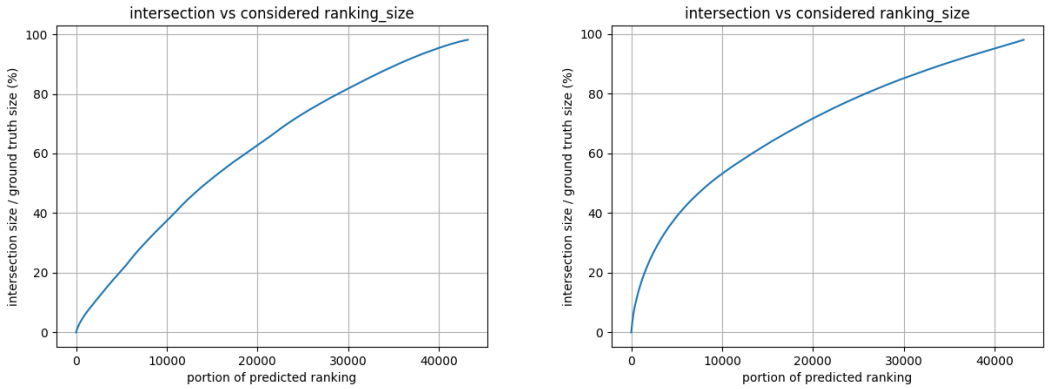
The plots shown below represent the trend of the intersection percentage with respect to the GT. The x-axis represents the considered size x of the sub-ranking (i.e. the full-length ranking truncated at position x). Of course, the sub-ranking tends to the full-length ranking as x-value reaches the maximum value. The value read on y-axis is given by the formula

$$y(x) = \frac{1}{N} \sum_{i=1}^{N} \frac{|GT_i \cap R_x^i|}{|GT_i|}, \tag{5.1}$$

where N represents the total number of artists in the dataset, GT_i is the ground truth ranking of artist i and R_x^i is the sub-ranking of size x relative to artist i.

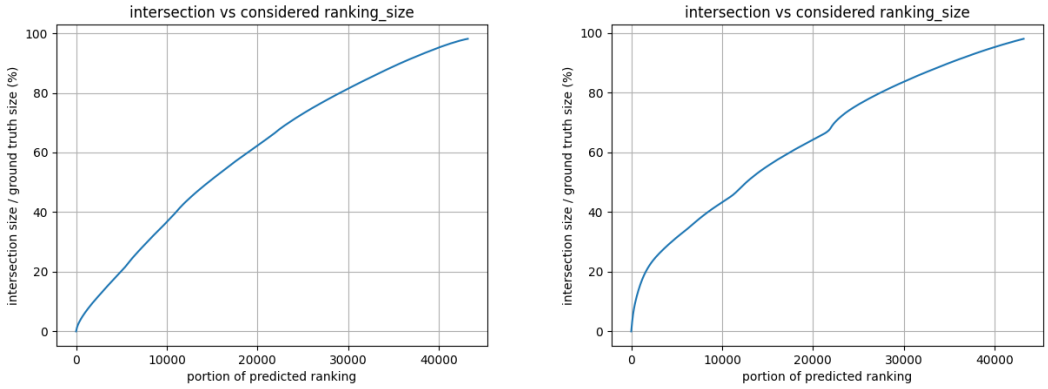
As described in the formula above, the curve represents an average over all artists. These data are reasonably encouraging because, if the full-length rankings were randomly populated, the curve would degenerate into a straight line. Indeed, the more concave the curve is, the better the results are.

[2] `https://www.spotify.com/`



(a) metric: *biv_hun* **(b)** metric: *cc_peak_1*

Figure 5.13: Intersection percentage with respect to portion of predicted ranking (mode: *m3*)



(a) metric: *biv_hun* **(b)** metric: *cc_peak_1*

Figure 5.14: Intersection percentage with respect to portion of predicted ranking (mode: *m2*)

The plots in figures 5.13 and 5.14 show that, for all configuration seen so far, the proposed model gives promising results as the curve maintain its concavity without degenerating into a straight line. Also in this case the best results are obtained with the metric *cc_peak_1*, which shows a more evident concavity. However, these results are not sufficient to say that one metric is better than the other. In fact, the metric *biv_hun*, as described in section 4.2.4, allows one to modify a weight parameter to give greater value to the differences between the non-zero values of the heatmap rather than to the relative positions of the non-zero points on the heatmap.

Chapter 6

Conclusions

In this thesis, we addressed the problem of comparing artists based on their musical production. We deliberately chose to not consider other kinds of features except that the one that comes from raw audio. The task was to compare our results with a ground truth that was not necessarily generated with inputs of the same type as ours. A discussion on different approaches has been provided. In particular, the main process was, firstly, to arrange features extracted from raw audio in a lighter and readable way using t-SNE. Secondly, the task was to use those preprocessed data to estimate distance relationships between artists by using two different novel approaches. After generating our rankings of similarity, we chose two ways to evaluate our results: one based on metadata attached to each artist, in particular on tag-lists; the other based on rankings already provided by the Million Song Dataset used for our study. The evaluation based on tag lists has led to encouraging results since it is evident that as the position within the ranking increases, the number of tags in common decreases accordingly. The evaluation based on a comparison between rankings didn't go as well. Generating a ranking with the same elements as the reference one (the GT one) turned to be not an easy task, especially because of the large number of artists involved. A solution to this problem was to consider the full-length ranking in order to see if Artists present in GT list were concentrated mostly in the head of the ranking rather than the tail. This approach led us to a partially positive confirmation. The comparison between GT rankings and ours using RBO (see 3.3.1) produced poor results with no exceptions. One of the main causes could be the different nature of the ranking generation process.

6.1 Future works

Future works should aim to fine-tune the parameter w of metric *biv_hun* to find an optimal trade-off between shift and difference of non-zero values of two heatmaps.

Future research could also examine metric *cc_peak_1* to explore different scenarios where multiple peaks of the cross-correlation matrix are handled. Moreover, since the number of unique tags is more than 7500, it often happens that many tags have almost the same meaning for discovering similarities (e.g. "jazz piano" and "piano jazz"). Mapping those tags to a set of *macro*-tags might lead to more consistent results Another interesting research could use this framework on other datasets to provide benchmarking results.

Acronyms

MIR
Music Information Retrieval

GT
Ground Truth

RS
Recommendation System

MSD
Million Song Dataset

API
Application Program Interface

MFCCs
Mel-frequency cepstral coefficients

Bibliography

[1] *Music information retrieval*. URL: https://en.wikipedia.org/wiki/Music_information_retrieval (cit. on p. 3).

[2] Markus Schedl, Arthur Flexer, and Julián Urbano. «The Neglected User in Music Information Retrieval Research». In: 41.3 (Dec. 2013), pp. 523–539. ISSN: 0925-9902. DOI: 10.1007/s10844-013-0247-6. URL: https://doi.org/10.1007/s10844-013-0247-6 (cit. on p. 4).

[3] Markus Schedl, Emilia Gómez, and Julián Urbano. «Music Information Retrieval: Recent Developments and Applications». In: *Foundations and Trends® in Information Retrieval* 8.2-3 (2014), pp. 127–261. ISSN: 1554-0669. DOI: 10.1561/1500000042. URL: http://dx.doi.org/10.1561/1500000042 (cit. on pp. 3, 4, 6–8, 10, 11).

[4] Avery Li-chun Wang and Th Floor Block F. «An industrial-strength audio search algorithm». In: *Proceedings of the 4 th International Conference on Music Information Retrieval*. 2003 (cit. on p. 4).

[5] J. Serrà, Emilia Gómez, and Perfecto Herrera. «Audio cover song identification and similarity: background, approaches, evaluation, and beyond». In: *Advances in Music Information Retrieval*. Ed. by Z. Ras and A. A. Wieczorkowska. Vol. 274. Studies in Computational Intelligence. Springer-Verlag Berlin / Heidelberg, 2010. Chap. 14, pp. 307–332. ISBN: 978-3-642-11673-5. DOI: 10.1007/978-3-642-11674-2_14. URL: files/projectsweb/jserra10coverireview.pdf (cit. on p. 4).

[6] Òscar Celma. *Music Recommendation and Discovery – The Long Tail, Long Fail, and Long Play in the Digital Music Space*. Ed. by Springer. Berlin, Heidelberg, Germany, 2010 (cit. on p. 5).

[7] Bracha Shapira Francesco Ricc Lior Rokach and Paul B. Kantor. *Recommender Systems Handbook*. Ed. by Springer. Berlin, Heidelberg, Germany, 2011 (cit. on p. 5).

[8] Markus Schedl, Emilia Gómez, and Julián Urbano. «Music Information Retrieval: Recent Developments and Applications». In: *Found. Trends Inf. Retr.* 8.2–3 (Sept. 2014), pp. 127–261. ISSN: 1554-0669. DOI: 10.1561/1500000042. URL: https://doi.org/10.1561/1500000042 (cit. on p. 5).

[9] Markus Schedl, David Hauger, and Dominik Schnitzer. «A Model for Serendipitous Music Retrieval». In: *Proceedings of the 2nd Workshop on Context-Awareness in Retrieval and Recommendation*. CaRR '12. Lisbon, Portugal: Association for Computing Machinery, 2012, pp. 10–13. ISBN: 9781450311922. DOI: 10.1145/2162102.2162105. URL: https://doi.org/10.1145/2162102.2162105 (cit. on p. 5).

[10] Justin Salamon and Emilia Gómez. «A Chroma-based Salience Function for Melody and Bass Line Estimation from Music Audio Signals». In: July 2009 (cit. on pp. 9, 10).

[11] Sung-Hyuk Cha. «Comprehensive Survey on Distance/Similarity Measures Between Probability Density Functions». In: *Int. J. Math. Model. Meth. Appl. Sci.* 1 (Jan. 2007) (cit. on p. 11).

[12] Laurens van der Maaten and Geoffrey Hinton. «Viualizing data using t-SNE». In: *Journal of Machine Learning Research* 9 (Nov. 2008), pp. 2579–2605 (cit. on pp. 13, 16).

[13] William Webber, Alistair Moffat, and Justin Zobel. «A Similarity Measure for Indefinite Rankings». In: *ACM Trans. Inf. Syst.* 28.4 (Nov. 2010). ISSN: 1046-8188. DOI: 10.1145/1852102.1852106. URL: https://doi.org/10.1145/1852102.1852106 (cit. on p. 17).

[14] *Cross-correlation*. URL: https://en.wikipedia.org/wiki/Cross-correlation (cit. on p. 19).

[15] *Cross-correlation on 2d matrices*. URL: https://www.mathworks.com/help/signal/ref/xcorr2.html (cit. on p. 19).

[16] *Hungarian method*. URL: https://glossary.informs.org/ver2/mpgwiki/index.php?title=Hungarian_method (cit. on p. 20).

[17] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. «The Million Song Dataset». In: *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*. 2011 (cit. on p. 21).

[18] *The Echo Nest*. URL: https://en.wikipedia.org/wiki/The_Echo_Nest (cit. on p. 21).

[19] *Spotify acquires music data firm The Echo Nest*. URL: https://www.theguardian.com/technology/2014/mar/06/spotify-echo-nest-streaming-music-deal (cit. on p. 21).

[20] *Pyconest*. URL: `https://pypi.org/project/pyechonest/` (cit. on p. 21).