Faculty of Mathematics and Computer Sciences

FRIEDRICH-SCHILLER-
UNIVERSITÄT
JENA

Master's Thesis

Music Similarity Analysis Using the Big Data Framework Spark

presented by:	Johannes Schoder
born:	03. September 1994
ID:	169197
course of studies:	M.Sc. Informatik
supervisors:	Ralf Seidler, Prof. Dr. Martin Bücker

Contents

Abbreviations

MIR	music information retrieval
MFCC	mel frequency cepstral coefficients
FFT	fast fourier transformation
DCT	discrete cosine transformation
MSD	million song dataset
MIDI	musical instrument digital interface
KL divergence	Kullback-Leibler divergence
MP	mutual proximity
DTW	dynamic time warping
ESA	explicit semantic analysis
BPM	beats per minute

List of Figures

List of Tables

Abstract

Calculating music similarity metrics using map reduce algorithms.
This thesis is about the comparison of construction noise and modern day music. The field of music information retrieval (MIR) in computer science is mostly a data driven and purely mathematical topic. The goal of this thesis is to merge the fields of computer science with music theoretical knowledge and to find potential weak spots of current music similarity algorithms.

1. Introduction

The idea originated from Dr. T. Bosse from the chair for advanced computing at the Friedrich-Schiller-University (FSU) in Jena. When proposing the idea for a master thesis with the topic of "Music similarity measurement using genre specific features" using different guitar play styles in modern day metal music, he jokingly said that he would also like to know how metal music compares to construction building noise. The idea is actually not so groundless, considering that most people would agree on the fact that metal music is often described as noise by people not used to listening to genres like death and black metal. This thesis is meant to evaluate how music similarity algorithms compare construction noise to common musical genres and how to possibly improve these algorithms.

1.1 Why and how?

Why is music similarity research even necessary?

First of all, there is no fixed definition of music similarity so far. This is one of the first problems, dealing with music similarity. This topic offers multifaceted approaches. Merging multiple approaches with different weights can offer a more diverse music recommendation system. To do this, a lot of different data is required.
Content (music features) and context (listener behaviour) data can be fed into a big data framework to speed up operations. Collecting this data for large amounts of songs results in big datasets that need to be explored efficiently.

What to improve?

"[...] Spotify Radio, iTunes Radio, Google Play Access All Areas and Xbox Music. Recommendations are typically made using (undisclosed) content-based retrieval techniques, collaborative filtering data or a combination thereof." [1, p. 9] The goal of this work is, to propose a transparent music similarity retrieval method based on various weighted contextual and content-based data. Applying different weights to different

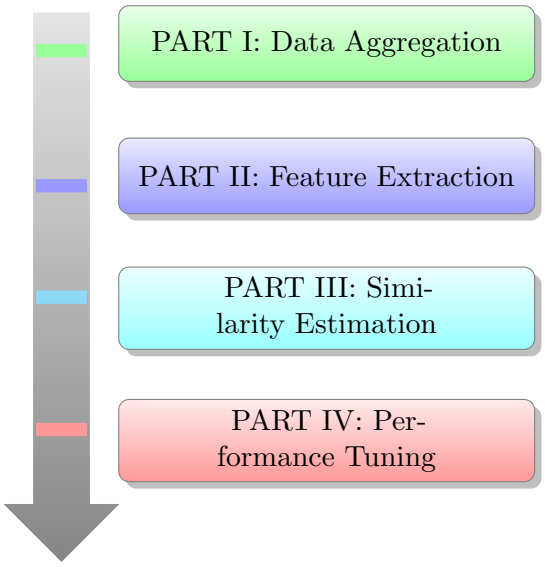
features allows similarity retrieval methods to search for different kind of similarity. E.g. weighing the tempo and beat of a song more then melodic similarity allows the creation of playlists for workout and sport, whilst melodic/ timbre etc. similarities allows to search for similar songs from musical subgenres. The user would get to decide what kind of playlist he wants to create. Adding contextual data and feeding it to an algorithm could add more or less popular music to the playlist with the goal to discover new upcoming artists or get other popular and most listened music. For this thesis however, the focus lies on content-based data/ audio features.

How to approach this?

First of all, a lot of data is required. In the first part, different scientific datasets are evaluated. Secondly, the available features are shown and explained. In the third part, different metrics are explained using the previously explored features. Lastly, a big data approach to efficiently use the gathered features is proposed and evaluated. A way to evaluate the results is also proposed.

1.2 Overview

Structure:



1.3 Conclusions

Why using a big data framework would help: music similarity is not well defined. It is a rather subjective value that differs from listener to listener. Two tracks could be considered as "similar" when they are equal in tempo, loudness, melody, instrumentation, key, rhythm mood, lyrics or a combination of more than a few of these features. The usage of a big data framework allows to create a variable/ fuzzy metric definition. Various parameters could easily be taken into consideration when calculating the musical distance of two different pieces. Using a Big Data framework, the problem of the fuzzy definition of music similarity could be avoided, if a metric can be found, that takes multiple of the accounted features of this thesis into consideration. Available information includes metadata, user data, audio features, sheet music and more.

The idea of using genre specific features could be evaluated any further
Another important question is how to measure similarity algorithms. There are a few possibilities like genre, composer/ interpret or cover song identification. Or actual user data. MSD Challenge Dataset usable?[2]

TODO:

Comparison of Echo Nest Pitch Features vs. Full MIDI notation?

Using genre specific features

Use extracted MIDI vs. professionally annotated sheet music

Echo Nest Features invariant to key

2. MIR Toolkits and audio features

2.1 Music Information Retrieval

The field of music information retrieval is a large research area combining studies in computer science like signal processing and machine learning as well as psychology and academic music study. The algorithm used in this thesis is described in [3, pp. 17ff] and in section 2.2.2. The framework used to determine the song similarity in chapter 4 is the MUSLY toolkit [4] To get started, a brief overview is given in the next section providing the most important information about publicly available datasets, MIR toolkits and approaches to music similarity.

2.2 Audio Features and Music Similarity Measurements

This section provides an overview about different music similarity measurements and metrics.
More in-depth information about the different metrics is given in chapter 7

2.2.1 Frequency based

As done by: [3, pp. 17ff]
Most of the algorithms start with switching from the time domain to the frequency domain, by performing a Discrete Fourier transform as described in equation 2.1 and then compute the power spectrum (equation 2.2)

$$X_m = \sum_{k=0}^{K-1} x_k \cdot e^{-\frac{K}{2 \cdot \pi \cdot i} \cdot k \cdot m} \tag{2.1}$$

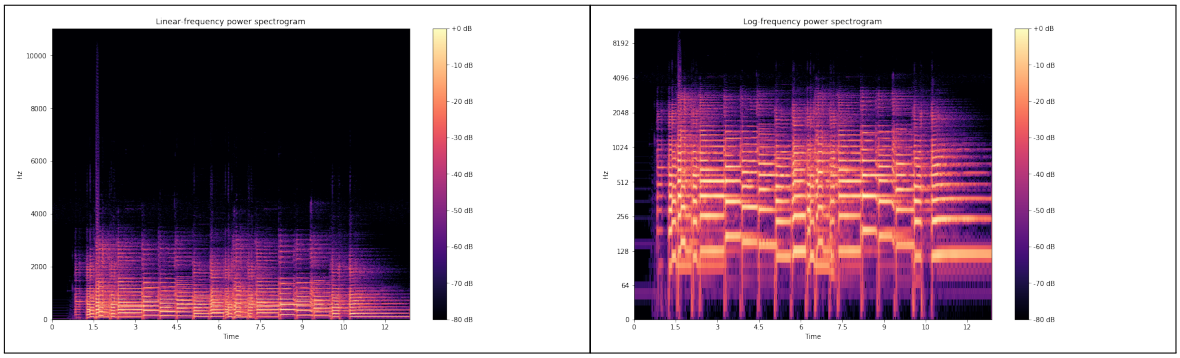
$$|X_m| = \sqrt{Re(X_m)^2 + Im(X_m)^2} \tag{2.2}$$

Figure 2.1a shows the spectrogram of the first bars of the song Layla by Eric Clapton.

The sound sample was recorded on an electric guitar to avoid copyright infringements. Due to the fact that the human ear perceives sound in a non-linear matter, a logarithmic or Mel-scale is better to represent different pitches.
For example the note A4 is perceived at a frequency of 440Hz, the A note of next octave (A5) is at 880Hz and the next one is at 1600Hz and so on. The Mel-scale [1, pp. 53f] was introduced to resemble the human perception of frequency (equation 2.3)

$$m = 1127 \cdot ln(1 + \frac{f}{700}) \tag{2.3}$$

The following plots were created with the librosa library [5].



(a) Spectrogram (b) Rachmaninoff Prelude in C Sharp Minor [6]

Figure 2.1: Frequency Space

The high dimensionality of the data is a problem for machine learning applications and music similarity tasks, as computation based on a vector with such a high dimensionality of the data would take to long. Given a sample rate of $f_s = 44,1kHz$ (usual CD sample-rate) and a length of a song of about $t = 180s$, the time domain contains 7938000 data points usually with 16-bit resolution.

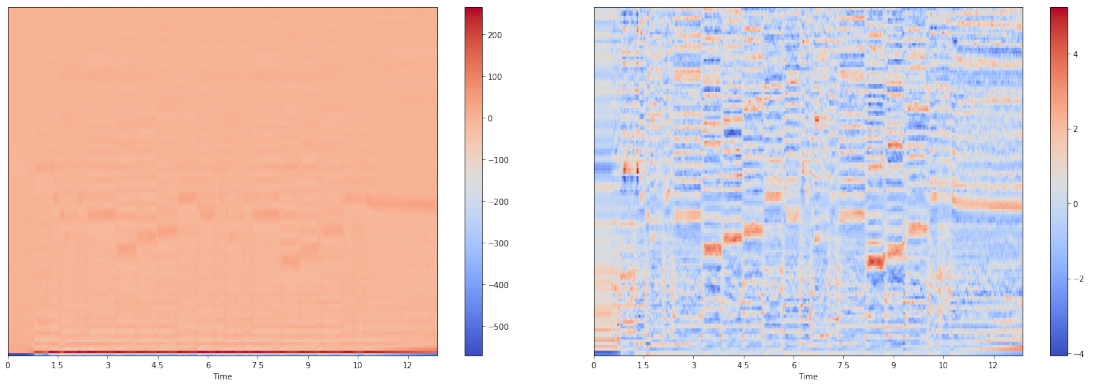
$$K = f_s \cdot t \tag{2.4}$$

Calculating a FFT with a window size of 1024 samples and a hop size of 512 samples (resulting in the factor 1.5 in equation 2.5)[1], the full resulting spectrogram would contain 11627 frames with 1024 frequency values per frame. (eq: 2.5)

$$N_{fv} = 1.5 \cdot (\frac{44100\ samples/s}{1024\ samples/frame}) \cdot t \tag{2.5}$$

To reduce the dimensionality of the feature vector, a typical approach in MIR would be to calculate the so called Mel Frequency Cepstral Coefficients (MFCCs). They are described in more detail in the next section.

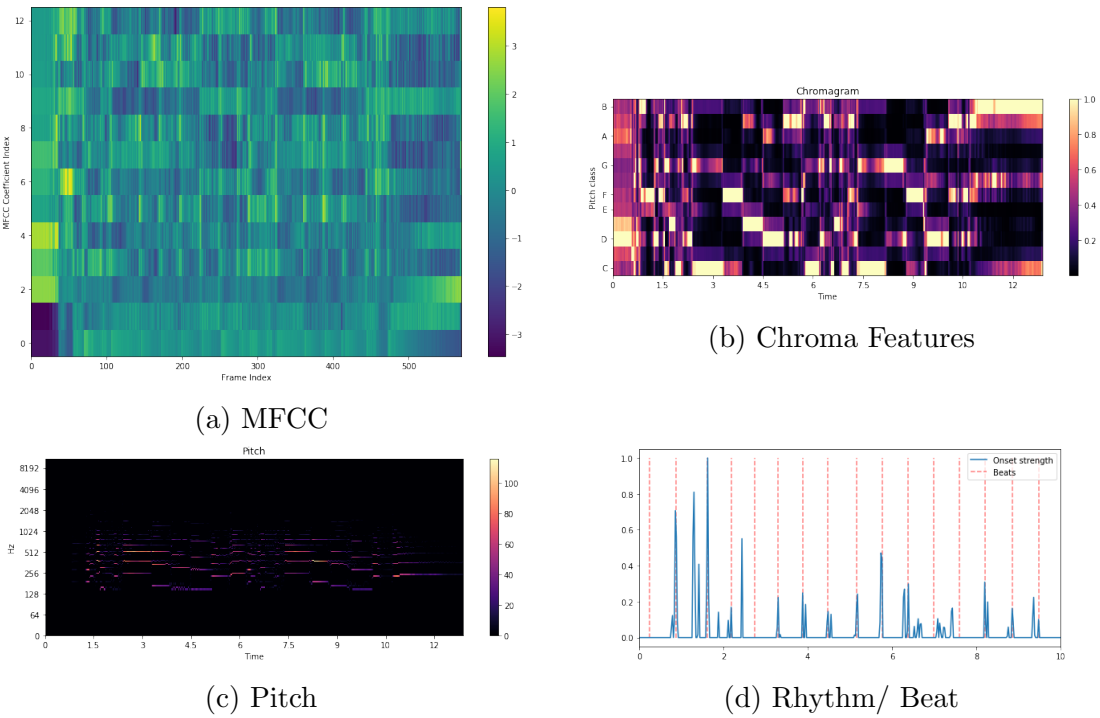
As another, better comprehensible feature set, the chroma features represent the tonal properties of a song. The chroma plot (Figure 2.3b) shows the distribution of the



(a) MFCC High Resolution (b) MFCC High Resolution Scaled

Figure 2.2: Features

different pitches mapped to the various keys in one octave. The values are normalized to one by the strongest dimension. So if all values are close to one, it is most likely that there is only noise at that frame in the recording, as depicted in the first frames of figure 2.3b. Figure 2.3c figures the pitch curve of the recording. None but the most dominant frequencies are shown. Pitches below a certain threshold are filtered out. These Pitch curves can be used to estimate and transcribe musical notes from audio data as presented in 2.3.3. The Plot in figure 2.3d shows the onsets (beginning of musical notes) and estimated beats.



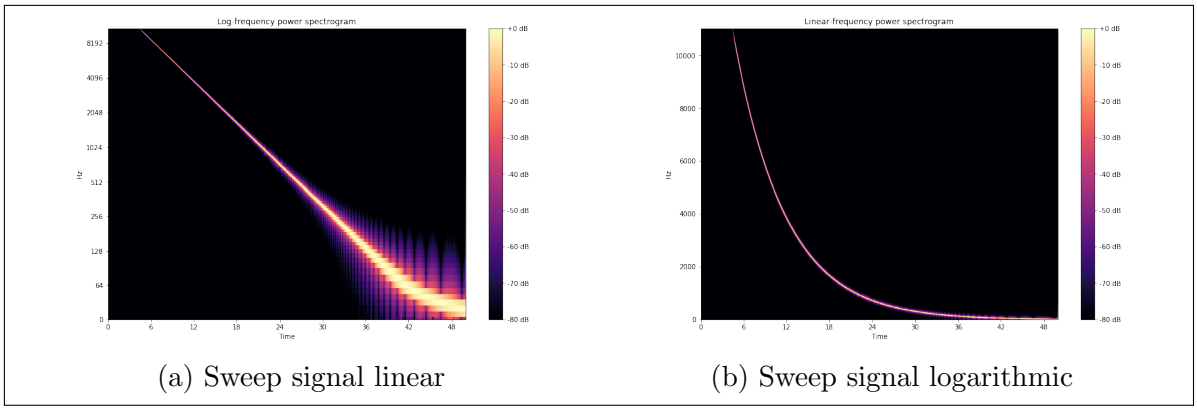
(a) MFCC (b) Chroma Features

(c) Pitch (d) Rhythm/ Beat

Figure 2.3: Features

2.2.2 MFCC

This section gives a brief overview over the computation of the MFCC as stated in [1, pp. 55ff]. Figure 2.4 shows the magnitude spectrum of a frequency sweep signal.



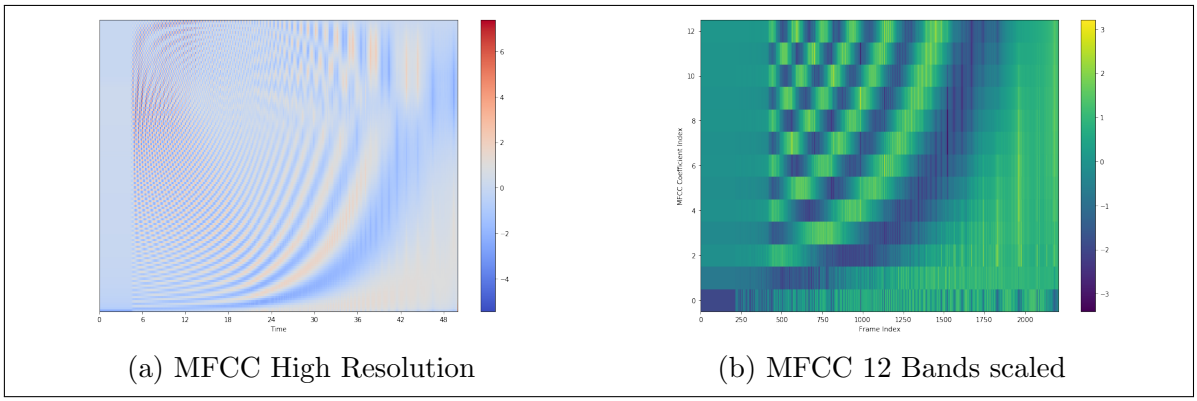
(a) Sweep signal linear (b) Sweep signal logarithmic

Figure 2.4: Sweep Sinal

First of all the magnitude spectrum is transformed to the Mel-scale by assigning each frequency value to a Mel- band. Doing this a dimension reduction can be done, by assigning multiple frequency values to one of typically 12 to 40 Mel-bands. The resulting vectors are then fed into a discrete cosine transformation (DCT) resulting in the MFCCs for each frame.

$$X_k = \sum_{n=0}^{N-1} x_n cos\left[\frac{\pi}{N}(n+\frac{1}{2})k\right] \tag{2.6}$$

Figure 2.5a shows the resulting MFCCs with a high resolution of 1024 Mel bands. This is not what in a usual application would be done, because this is nearly as high dimensional as the original spectrogram. Figure 2.5b shows the MFCC reduced to 12 Mel Bands.



(a) MFCC High Resolution (b) MFCC 12 Bands scaled

Figure 2.5: MFCCs

Onset and Chroma signal explained!!!!

2.2.3 Pitch based

As done by: [7]
The proposed approach by [7] is, to take mid-level features like chroma or pitch instead of high-level features like sheet music or low-level features like gaussian mixture models of MFCCs. A more detailed analysis of this topic is given in chapter 5

2.2.4 Rhythm based

As done by: [8]
Rhythm based music similarity algorithms use timing information of various events. The so called rhythm spectrum can be used as a feature vector. For example the onset and beat data from the plot in Figure 2.3d could be used as a starting point for rhythmic similarity retrieval. An in depth overview is given in chapter 6

2.2.5 Metadata based/ Collaborative Filtering

As done by: [9] and [10] using the Million Song Dataset (MSD) [11]
In 2012 the MSD Challenge was brought to the MIR community. The researches were challenged to give a list of song recommendations based on a large set of user data.
So if user X listened a lot to artist A and artist B and user Y listens mostly to artist A and artist C, then maybe user X would like artist C as well. These kind of metadata based recommendations are pretty common in large music streaming services, although not necessarily representing direct musical similarity. These kind of recommendation systems tend to propose commonly well known artists rather than not so well known ones biasing the result. On the other hand these kind of similarity algorithms can work very fast and efficient.

Genre and metadata, combinations and variable model, Collaborative Filtering, Lyrics

2.2.6 Note based

As done by: [12]
For comparing musical pieces by their symbolic represenation (notes, tabulatures etc.) different text retrieval methods could be used. The MIDI datatype could be used, as it is a form of digital representation of music information. [12] uses a variation of the

Levinshtein distance measurement. The problem with notation based algorithms is, that there are not many datasets available containing audio and MIDI information. As shown in 2.3.3 the automatic transcription of notes from raw audio does not work flawlessly. There is ongoing research to automatically annotate musical notes with the help of neural networks.[13]

2.2.7 Genre specific features

As done by: [14] for indian art music, by using 560 different combinations of different features. They state that: "We evaluate all possible combinations of the choices made at each step of the melodic similarity computation discussed in Section 2. We consider 5 different sampling rates of the melody representation, 8 different normalization scenarios, 2 possibilities of uniform time-scaling and 7 variants of the distance measures. In total, we evaluate 560 different variants" [14, p. 3]. This evaluation showed, that the choice of features and parameters for music similarity measurement is a critical point.
In Rock, Pop and Metal music, extraction of different guitar playstyles would be imaginable. Guitar Tab Extraction [15] Toolkits could be used to extract information if the guitar in a song is mostly plucked or strummed for instance. Or if there are Hammer-on/ Pull-off/ side bending or tapping techniques used. In classical music, the play style of the string section of an orchestra could be taken into consideration.

2.3 MIR Toolkits

2.3.1 Music Similarity

The easiest way to test state of the art music similarity algorithms is to use the open source toolkit Musly [4]
It is based on statistical models of MFCC features and calculates the distances between songs very fast, supporting OpenMP acceleration. It offers the classical mandel-ellis similarity method [16] and a timbre based improved version of the mandel-ellis algorithm [17]
The MIR Toolkit [18] is a toolbox for Matlab [19]. A port to GNU Octave [20] is also available [21]
The short code snippet below is all it takes to compute a similarity matrix based on MFCC features, but the calculation takes quite some time.

```
mydata = cell(1, numfiles);

for k = 1:numfiles
```

```
        myfilename = sprintf('%d.wav', k);
        mydata{k} = mirmfcc(myfilename);
        close all force
endfor

simmat = zeros(numfiles,numfiles);

for k = 1:numfiles
        for l = 1:numfiles
                simmat(k, l) = mirgetdata( ...
                mirdist(mydata{k}, ...
                mydata{l}));
        endfor
endfor
```

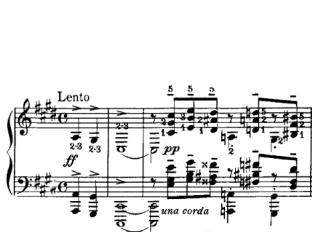
2.3.2 Audio feature extraction

To extract audio features, The YAAFE toolkit [22] is able to extract a lot of different audio features like energy, mfcc or loudness directly into the hadoop file format h5 making it ideal for big data frameworks to use. It can be used with C++, Python or Matlab.
The Essentia toolkit [23] is pretty similar to YAAFE, extending it by the calculation of the rhythm decriptors, bpm etc. It can also be used in C++ and Python
The Librosa Toolkit provides similar functionality [24] as Essentia. It is user-friendly and can be called from a Jupyter-Notebook [25], allowing rapid prototyping and testing of different algorithms. The plots from section 2.2.1 were created using librosa

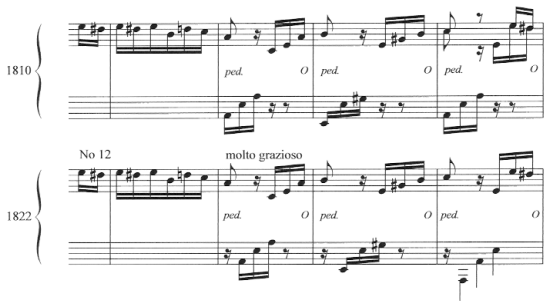
2.3.3 Melody/ pitch extraction

To test the various pitch extraction toolkits, a piece by Rachmaninoff and by Beethoven was used. The first three bars of Rachmaninoffs Prelude can be found in figure 2.6a Figure 2.6b shows the first five bars of Beethovens Bagatelle in A Minor (Für Elise).

The first toolkit tested is Aubio [27]. The result can be seen in figure 2.7a and figure 2.7b The upper subplot shows the waveform of the first few seconds of each piece. The second plot figures the estimated pitch with green dots. If the pitch is zero, then no pitch could be estimated, most likely because the associated frame contains silence. The blue dots resemble the estimated pitches, where the confidence (shown as the blue



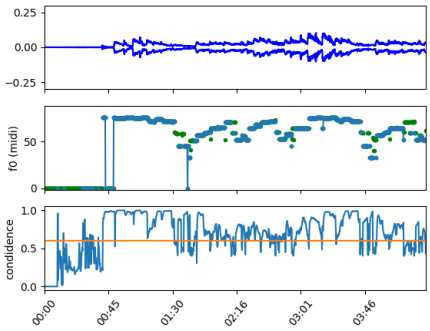
(a) Rachmaninoff Prelude in C Sharp Minor [6]



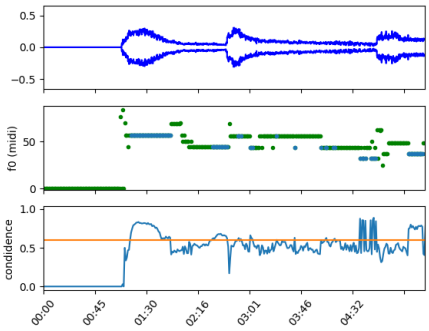
(b) Für Elise [26]

Figure 2.6: Original Scores

graph in the third subplot) is above a certain threshold (the orange line).



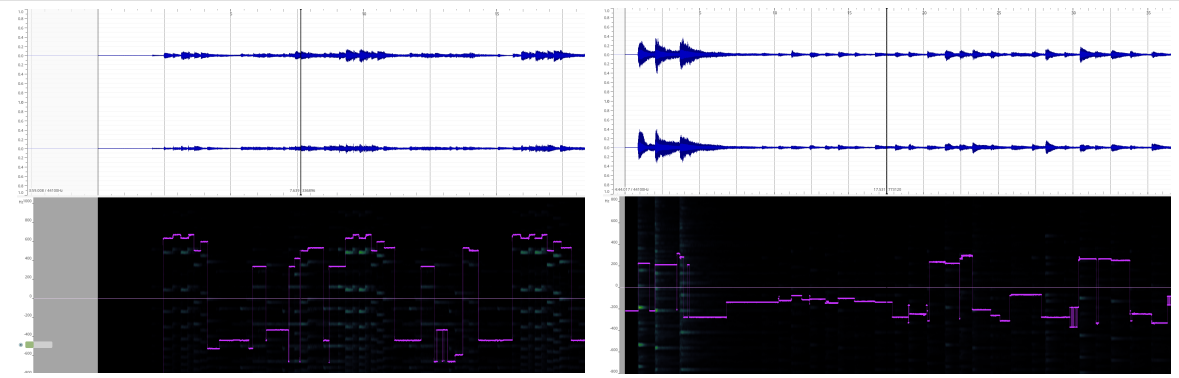
(a) Für Elise Aubio Pitch



(b) Rachmaninoff Prelude Aubio Pitch

Figure 2.7: Aubio

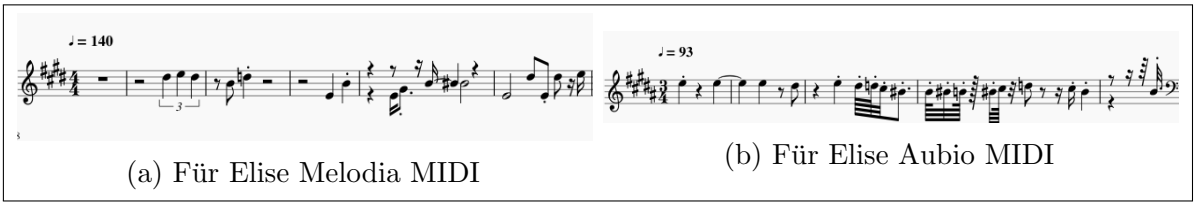
The other melody extraction tool is Melodia[28], which is available as a VAMP plugin and can be used together with the Sonic Visualizer[29]
The results are shown in figure 2.8a and 2.8b. The purple line is the estimated pitch, however there are large jumps between different octaves of the harmonics.



(a) Für Elise Melodia Pitch (b) Rachmaninoff Melodia Pitch

Figure 2.8: Melodia

Sadly the conversion to MIDI does not work flawlessly.
Figure 2.9a shows the output of a python script using the Melodia VAMP plugin to calculate a MIDI file containing the main melody line.



(a) Für Elise Melodia MIDI (b) Für Elise Aubio MIDI

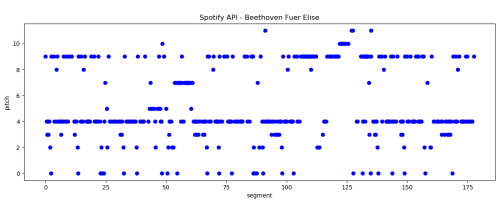
Figure 2.9: Transcription

Work in progress: A script to convert Aubio Pitch to MIDI and the output shown in figure 2.9b

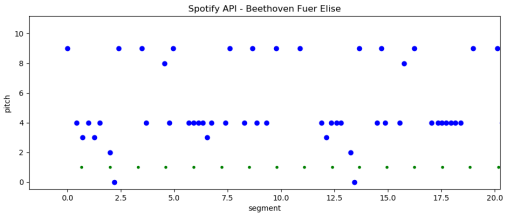
2.3.4 Spotify API/ Echonest

Another way of getting music information, audio analysis and metadata is by using the Spotify API[30] Part of the available audio features comes from the Echo Nest[31] For each song available in Spotify, there are audiofeatures like loudness, speechiness, liveness, instrumentalness, energy, danceability, acousticness, pitch, rhythm bars etc. With a small Python library named Spotipy, the available information can very easily be used and accessed. [32]
In figure 2.10a the pitch of the piano piece Für Elise by Beethoven is shown and figure 2.10b shows the beginning of the piece in more detail, including green dots, that resemble estimated bar markings. The blue dots represent the note values of one octave. That means they can resemble a value between zero and eleven with zero representing the key C and 11 is representing a B. The Spotify API actually returns a value for every single one of the keys per segment, while one segment is a section of samples that are

relatively uniform in timbre and harmony. In the plots only the most dominant key per segment is shown. The pitch values in one segment are normalized to one by their strongest dimension, so the more evenly distributed the pitch values are and the closer they all get to the value one, the more likely it is, that the sample contains only noise. The Downside using the Spotify API is, that there is no packed and ready to use test dataset containing the relevant features. So for scientific purposes, a test dataset would have to be created first.



(a) Für Elise Spotify pitch



(b) Für Elise detail

Figure 2.10: Spotify API

The pitch graph has one significant downside, because it is reduced to one octave and thus can not represent the melody of a song to its full extend.

2.3.5 Cover song identification

The ability to detect cover songs or different versions/ recording of a musical piece could be a good measurement for the efficiency of music similarity algorithms. In the next section one test case is presented, showing, that a MFCC based music similarity algorithm isn't able to detect different recordings of the same piano piece as most similar to each other.

3. Data aggregation

To evaluate the music similarity algorithms and metrics a lot of music data is needed.

3.1 Datasets

3.1.1 Free Music Archive

The largest dataset is the Free Music Archive- dataset (fma) consisting of 106733 different songs totalling an amount of nearly one terabyte of music data from all kinds of different music genres.[33] There is also a lot of metadata information available for most of the songs.

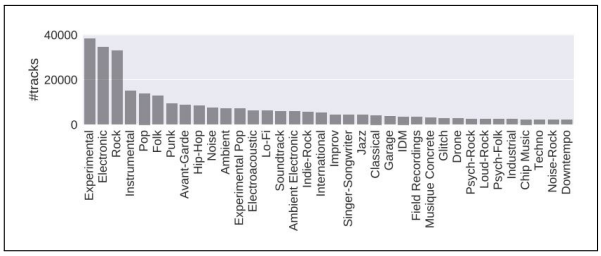


Figure 3.1: Number of songs in the free music archive dataset per genre

3.1.2 Musicnet and 1517 artists

Another source of music is the Musicnet dataset.[34] It includes 330 pieces of classical music with musical notes as annotations. Other sources of musical information would be the 1517-Artists dataset containing 3180 songs of multiple genres. [35]

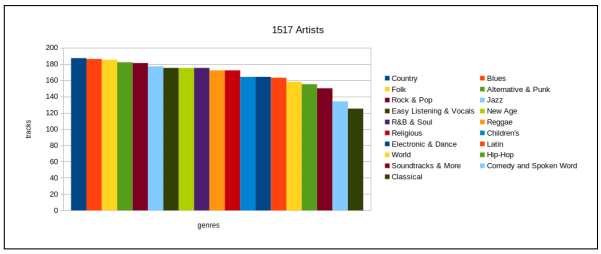


Figure 3.2: Number of songs in the 1517 artists dataset per genre

3.1.3 MedleyDB

For a melody/ pitch based similarity analysis, multitrack datasets could provide useful data, due to the fact that the pitch estimation can be done instrument by instrument. Datasets available are the MedleyDB[36] and MedleyDB2[37] datasets as well as the Open Multitrack Dataset[38] currently consisting of 593 multi-tracks in which the MedleyDB dataset is already included, leaving 481 other tracks for analysis.

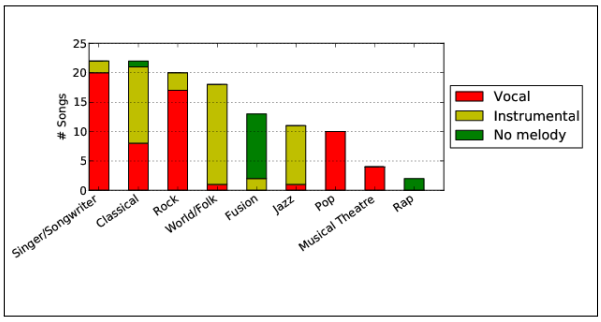


Figure 3.3: Number of songs in the MedleyDB dataset per genre

3.1.4 Private music collection

The private music collection used in this work consist mainly of metal music. The music was legally purchased, all rights belong to the respective owners. The distribution of different songs per genre for this dataset is visualized in figure 3.4.

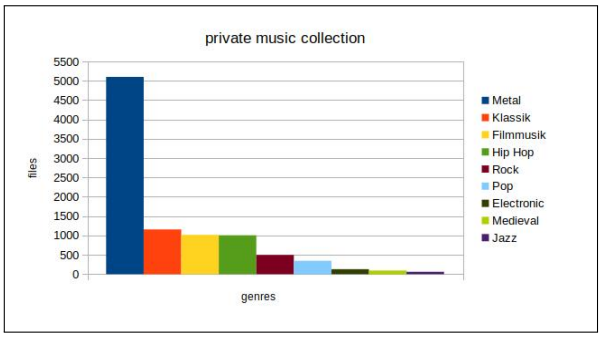


Figure 3.4: Number of songs in the private music collection per genre

Additionally a private recording dataset was used, consisting of ambient recordings and self produced music. Most of this music can be downloaded from soundcloud[39]

3.1.5 Overview and other sources

The music sources and amounts of songs used for the task at hand is listed in table 3.1.

Table 3.1: available music datasets

fma	106.733 Songs
private	8484 Songs
1517 artists	3180 Songs
Maestro	1184 Songs (piano) + MIDI
musicnet	330 Songs (classical) + note annotation
Open Multitrack Testbed	593(481) Songs/ Multitracks
MedleyDB	122 Songs/ Multitracks
MedleyDB2	74 Songs/ Multitracks

A special and very large dataset is available with the Million Song Dataset (MSD)[11]. It contains a large set of metadata per track as well as a lot of supplementary datasets, like the Tagtraum genre annotation (figure 3.5)[40], the last.fm dataset[41] and the Echo Nest API dataset[42]. Although the MSD does not contain any music files in the first place, 30 second samples can be gathered through simple scripts from 7digital.com. On top of that the Echo Nest API data already contains a lot of audio features like pitch, loudness, energy and danceability to name just a few. Another addition is the secondhand dataset, containing a list of cover songs in the million song dataset[43]

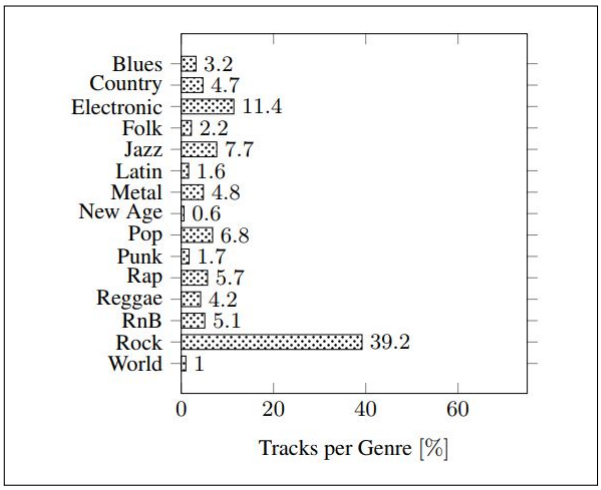


Figure 3.5: Number of songs in the million song dataset per genre

Due to the fact that the Spotify API[30] also works with audio features from the

Echo Nest[31], the MSD could be used in a big data environment to simulate the work with Spotify data, without manually mining the actual data. The MSD was actually already used in Big Data frameworks for music similarity retrieval based on metadata and user information[10]

3.2 Spotipy

For the purpose of this thesis, the option of creating an own dataset using the spotify API and spotipy (section 2.3.4) was considered. Ten very small test playlists of different genres were created using the Spotify Playlist Miner [44]. Appendix 10.1 lists a small script, that downloads all audio features and analysis data from all of the songs of a playlist, that contains a preview URL with a 30 second audio snippet.
The audio features and analysis data ist saved as a JSON file containing information about (beyond others)

- acousticness
- danceability
- instrumentalness
- liveness
- loudness
- speechiness
- valence
- predicted key
- tempo

as well as pitch and timbre information, beats and bars.

Together with the 30 second audio sample from which more data like MFCCs could be extracted, these crawler could provide all the information needed to build a large dataset for MIR. However the terms and conditions explicitly prohibits crawling the Spotify service. As stated by the Spotify Terms and Conditions of Use, section 9 (User guidelines):
"*The following is not permitted for any reason whatsoever: [...]*
12. "crawling" the Spotify Service or otherwise using any automated means (including bots, scrapers, and spiders) to view, access, or collect information from Spotify or the Spotify Service;" [45]

Therefore a larger user created dataset can not be used without the risk of legal infringements. However one could argue, that there is a difference between data mining and data crawling and for small datasets with the purpose of creating Spotify playlists, these restrictions may not apply.
In the sense of the Spotify Developer Terms of Service [46] there may be no legal infringements by creating a non-commercial playlist creation tool. [47] states, that by creating algorithmically-generated playlists similar to the "Discover Weekly" Playlists one may run into challenges if using such features commercially. However it does not prohibit the usage for non-commercial cases. Upon request the Spotify API developer team did not respond and thus in this thesis the Spotify API wont be used to create a test dataset.

4. Timbre Similarity

MFCCs explained in chapter 1, here: how many melbands, visualize GMM
Proposed methods according to [1, pp.51ff]
Mel Frequency Cepstral Coefficients have already been introduced in chapter 2.2.2. This chapter focuses on the different similarity metrics.
To reduce the dimensionality of the data even further, a statistical summarization of the MFCC feature can be calculated [1, p. 58]. For each of the Mel- Bands (12 in this case) the mean and standard deviation over all frames is calculated, resulting in a vector of 12 mean values, a 12 by 12 co-variance matrix (actually $\frac{12*11}{2}$ values, because of the triangular shape - the upper triangle contains the co-variances and the main diagonal contains the variances) and 12 variances. These vectors are therefore not dependent on the length of the actual song. Using such a model, the distance between two songs can be calculated as in equation 4.1, where x and y are the n-dimensional feature vectors of two different musical pieces:

$$d(x, y) = ||x - y||_p = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}} \tag{4.1}$$

also known as the L_p distance. Most of the times, the Euclidean (L_2) or the Manhattan (L_1) distance would be used in real world scenarios. This very basic approach has been refined and improved over the past years.

4.1 Single Gaussian Model

This approach was first proposed by Mandel and Ellis [16] in 2005 and is briefly summarized in [1, pp. 65f].
After computing the mean value of each MFCC (μ_P and μ_Q) and the covariance matrix of the different MFCC vectors (Σ_P and Σ_Q) of two musical pieces P and Q, the Kullback-Leibler divergence (KL divergence) can be calculated as follows, with $Tr(\cdot)$ being the trace (i.e. the sum of the diagonal of a matrix), d being the dimensionality (number of MFCCs) and $|\Sigma_P|$ being the determinant of Σ_P

$$KL_{(P||Q)} = \frac{1}{2}[log\frac{|\Sigma_P|}{|\Sigma_Q|} + Tr(\Sigma_P^{-1}\Sigma_Q) + (\mu_P - \mu_Q)^T\Sigma_P^{-1}(\mu_Q - \mu_P) - d] \tag{4.2}$$

As a second step the result has to be symmetrized.

$$d_{KL}(P,Q) = \frac{1}{2}(KL_{(P||Q)} + KL_{(Q||P)}) \tag{4.3}$$

This approach is one of the two available similarity metrics of the musly [4] toolkit introduced in section 2.3. This can be simplified and written as a closed form:

$$d_{SKL}(P,Q) = \frac{1}{4}(tr(\Sigma_P\Sigma_Q^{-1}) + tr(\Sigma_Q\Sigma_P^{-1}) + tr((\Sigma_Q^{-1}\Sigma_P^{-1})(\mu_P - \mu 2)^2) - 2d) \tag{4.4}$$

4.2 Other Similarity Metrics

The second available metric in the musly toolkit by Schnitzer is using the Jenson-Shannon Divergence (in an slightly adapted way). "The Jensen-Shannon (JS) divergence is another symmetric divergence derived from the Kullback-Leibler divergence. To compute it, a mixture X_m of the two distributions is defined" [3, p. 43]. "To use the Jensen-Shannon divergence [...] to estimate similarities between Gaussians, an approximation of X_m as a single multivariate Gaussian can be used [...] This approximation of X_m is exactly the same as the left-type Kullback-Leibler centroid of the two Gaussian distributions [...]" [3, p. 44]

$$\mu_m = \frac{1}{2}\mu_P + \frac{1}{2}\mu_Q \tag{4.5}$$

$$\Sigma_m = \frac{1}{2}(\Sigma_P + \mu_P\mu_P^T) + \frac{1}{2}(\Sigma_Q + \mu_Q\mu_Q^T) - \mu_m\mu_m^T \tag{4.6}$$

$$JS(P,Q) = \frac{1}{2}log|\Sigma_m| - \frac{1}{4}log|\Sigma_P| - \frac{1}{4}log|\Sigma_Q| \tag{4.7}$$

After calculating a similarity matrix for all songs, it normalizes the similarities with mutual proximity (MP). [17] This method wants to reduce the effect of a phenomenon called hubness that appears as a general problem of machine learning in high-dimensional data spaces. "Hubs are data points which keep appearing unwontedly often as nearest neighbors of a large number of other data points." [3, p. 66].

Schedl and Knees state: "To apply MP to a distance matrix, it is assumed that the distances $D_{x,i=1..N}$ from an object x to all other objects in the data set follow a certain probability distribution; thus, any discance $D_{x,y}$ can be reinterpreted as the probability of y being the nearest neighbor of x, given the distance $D_{x,y}$ and the probability distribution $P(x)$ [...] MP is then defined as the probability that y is the nearest neighbor of x given $P(x)$ and x is the nearest neighbor of y given $P(y)$" [1, p. 80]

Resulting in:

$$P(X > D_{x,y}) = 1 - P(X \leq D_{x,y}) = 1 - \mathscr{F}(D_{x,y}) \tag{4.8}$$

$$MP(D_{x,y}) = P(X > D_{x,y} \cap Y > D_{x,y}) \tag{4.9}$$

according to [1, p. 80]
Another, more compute heavy distance measurement would make use of Gaussian Mixture Models of MFCCs. As Knees and Schedl state "Other work on music audio feature modeling for similarity has shown that aggregating the MFCC vectors of each song via a single Gaussian may work almost as well as using a GMM [...] Doing so decreases computational complexity by several magnitudes, in comparison to GMM-based similarity computations." [1, p. 65] Therefore the usage of GMMs is not further considered in this thesis.
The last method mentioned in this thesis for timbral similarity is to use block-level features as proposed by Seyerlehner [48] and described in short by Knees and Schedl [1, p. 67]. Instead of using single frames and summarizing them into statistical or probabilistic models, block-level features use larger, e.g. multiple second long, audio frames and features like fluctuation patterns are computed for these frames.

4.3 Validation

For this thesis the approach by Mandel and Ellis is used to compute the similarity matrix, because it is . There is always a trade-off of complexity and functionality. Using the musly toolkit, a first evaluation is presented in the next section. The feature extraction and distance calculation can also be done in python using the librosa library and a small reimplementation of the Mandel-Ellis approach was tested.
A possible measure for the efficiency of a timbre similarity algorithm is the ability to find songs of the same genre.

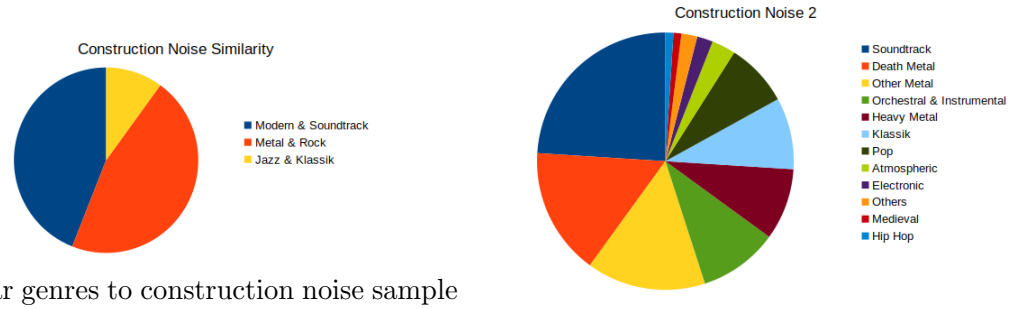
4.3.1 Construction Noise

Comparing a construction noise sound sample with the private music collection containing mostly metal, rock, pop, classical and hip hop music, the following six best results could be achieved in descending order:

- Ziegenmühlen Session - Down On The Corner (Folk Musik)
- While She Sleeps - The Divide (Metalcore)
- Delain - Mother Machine (Live) (Symphonic Metal)
- Within Temptation - Sanctuary (Intro Live) (Symphonic Metal)

- Without A Martyr - Medusa's Gaze (Death Metal)
- 100 Meisterwerke der Klassik - Orpheus In The Underworld (Orphée aux enfers) - Can-Can (Live At Grosser Saal, Musikverein) (Klassik)

Figure 4.1a and 4.1b show the distribution of the genres of 100 most similar songs compared to the construction noise sample.



(a) Similar genres to construction noise sample file

(b) Similar genres to construction noise sample file in detail

Figure 4.1: Construction Noise

Using the full dataset consisting of the private music collection, private field recordings, the full fma-dataset and the musicnet data, the following results could be achieved:

- Born Pilot - Birds Fell (FMA, Electronic, Noise)
- mrandmrsBrian - sun is boring (FMA, Avant-Garde, field recordings)
- steps in snow (private field recording)
- Sawako - Paris Children (FMA, field recordings)
- Jeremy Gluck and Michael Dent - Olivier (FMA, Ambient Electronic)

4.3.2 Different recordings and cover versions

Another experiment was, to get the most similar songs to the famous 'Rondo alla Turca' by Mozart. The recording used as a starting point is from the CD "100 Meisterwerke der Klassik" and has a length of 3:33 minutes. This piece by Mozart appears overall four times in the dataset and is recorded by different pianists. Every recording has a different length as listed in the following overview of the recordings by CD

- 100 Meisterwerke der Klassik (3:33)
- Piano Perlen (3:30)
- The Piano Collection - Disk 18 (3:28)

- Mozart Premium Edition - Disk 31 (4:29)

The top ten most similar songs to the 3 minutes and 33 seconds version are listed below:

- Mozart - Concert No. 10 for 2 Pianos and Orchestra in E Flat Major, KV 365 - 2. Andante
- Schubert - Sonata in B Flat, D. 960 - III. Scherzo (Allegro vivace con delicatezza)
- Albeniz - Iberia, Book I - Evocación
- Mozart Sonate Nr. 11 in A-Dur, K. 33 - Mozart - Alla Turca Allegretto (3:28)
- Beethoven - Bagatellen Op 119 -Allemande in D major
- Mozart - Rondo No. 1 in D Major, K. 485
- Mozart - Sonata For Piano No. 8 KV 310 A Minor - Allegro Maestoso
- Sonata For Piano No. 16 KV 545 C Major - Rondo: Allegretto
- Mozart Sonate Nr. 11 in A-Dur, K. 33 - III. Tuerkischer Marsch (3:30)
- Mozart - Piano Sonata No. 13 in B flat major, K. 333 (K. 315c): Allegretto grazioso

The interesting conclusion is that only 2 out of the 3 other versions were considered as most similar songs. The slower recording wasn't even in the top 30 list of the most similar songs. The same was observable for other cover versions of songs in the dataset like Serj Tankians song "Lie Lie Lie" from the CD "Harakiri" and an orchestral recording of the same piece. This is probably due to the usage of GMMs of MFCCs representing and valuing the timbre of the music predominantly instead of the pitches and melody movements.

4.4 summary

Three different similarity metrics are chosen:

- Euclidean Distance
- Symmetric Kullback-Leibler Divergence
- Jensen-Shannon like Divergence

To calculate the distances, for each song the mean vector, variance and covariance matrix has to be computed for each mfcc band. These are stored in two different output text files:

- out.mfcc (containing mean vector (length b), variance vector (length b) and vectorized upper triangular covariance matrix (length b))
- out.mfcckl (containing mean vector (length b) and full covariance matrix as a vector (length $\frac{b\cdot(b-1)}{2}$))

The amount of chosen mfcc bands is $b = 13$.
The second file is created to get rid of the necessity to rearrange the covariance matrix inside the Big Data Framework and reduce the computation time when a similarity computation request is processed. The text files contain strings like the following:
music/WALLS1.mp3; [-498.03763, ... ,4.321189]; [8943.487,... ,61.624344]; [8944.3907652, ... ,74.17548092]
music/WALLS1.mp3; [-498.03763, ... ,4.321189]; [8943.487,... ,61.624344]; [[6568.27958735, ... ,74.64776425], ... , [74.64776425, ... ,69.1589048]]

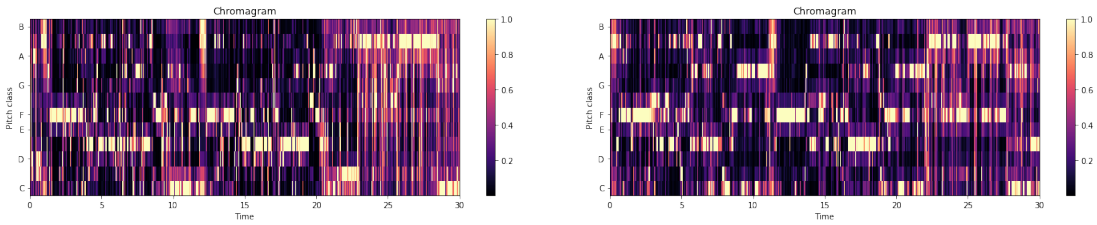
5. Melodic Similarity

5.1 Representation

As presented in section 2.3.3 there are tools for pitch curve extraction of the main melody line. However in polyphonic music these kind of algorithms struggle to get reasonable results, even in pop music. In musical genres like Metal it gets even worse. In conclusion the main pitch-line extraction and the following conversion of a song with multiple concurrent audio tracks to MIDI using up-to-date open source toolkits doesn't produce very reasonable results as shown in 2.3.3. Another possible representation for melodic features could be to transform the structural information to graphs and use graph comparing algorithms to estimate the similarity between songs. [49] A better and widely used approach is to use chroma features as described in the next section 5.2.

5.2 Chroma Features

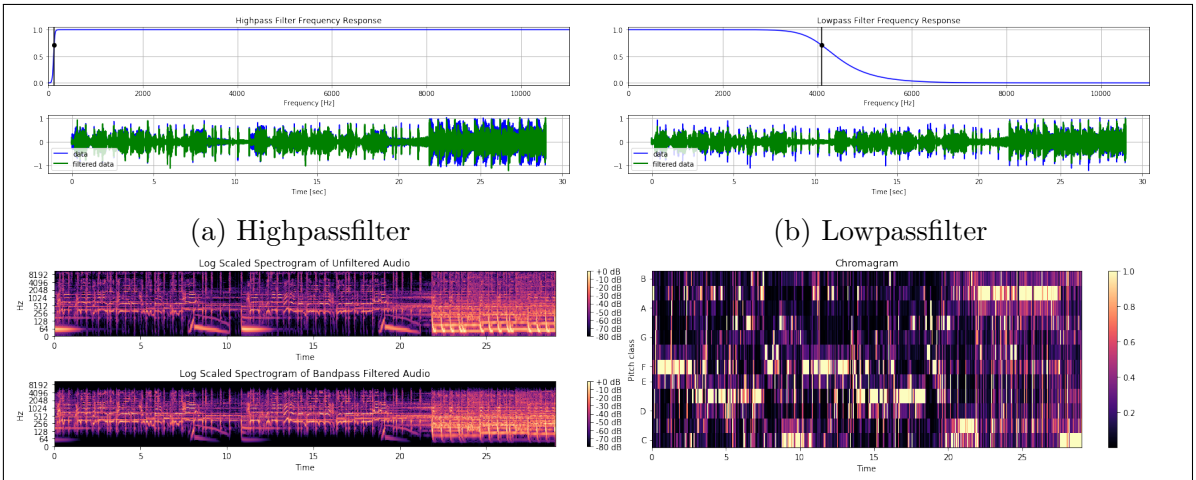
Chroma Features as described in section 2.2.1 are a good and low dimensional way to describe the melodic features of a song. The reduction of dimensionality however comes with a loss of information, especially what octaves the notes are played in. To compute chroma features, most MIR toolkits already offer methods to do so. The plots in this chapter were created using the essentia [23] and librosa [5] toolkits. In addition to the pure computation of the chroma features, some pre- and post-processing steps were implemented and tested and will be presented later in this chapter. First of all figure 5.1 shows the chroma feature plots from 2 recordings of the first thirty seconds of the song "Chandelier". Figure a shows the original version sung by the artist Sia and figure b shows the features of a cover version from the band Pvris. In the last third of each sample, the chroma features seemingly get noisier. At these timings in both songs the bass and drum set in. To reduce the effect of rhythm elements over the melodic voice and instrument lines, the audio signal was filtered firstly by a high-pass filter with a cut-off frequency of 128Hz (nearly equal to C3 Key) and secondly by a low-pass filter with a cut-off frequency of 4096Hz (C8 Key). This limits the frequency range to about 5 octaves. In figure 5.2 the filter frequency and the original audio signals are visualized



(a) Chroma Features Sia - Chandelier (b) Chroma Features Pvris - Chandelier

Figure 5.1: Chroma Features

in blue color and the filtered audio signal is green. The FFT plot before and after filtering the audio signal is also shown. In the chromagram of the bandpass filtered



(a) Highpassfilter (b) Lowpassfilter

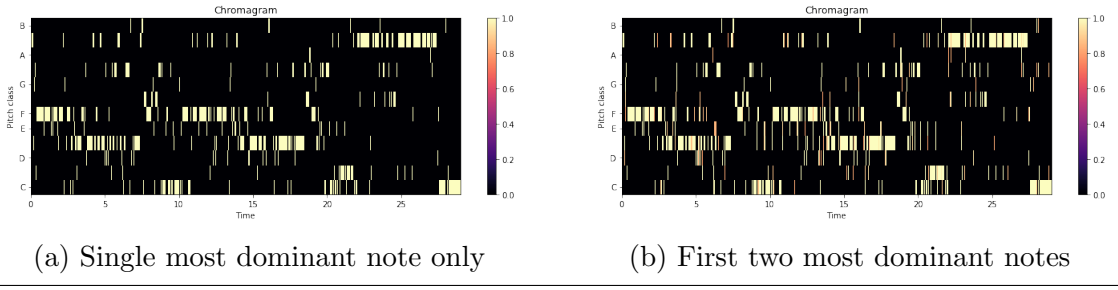
(c) FFT Bandpassfilter Sia (d) Bandpass Filtered Chromagram

Figure 5.2: Bandpass - Sia

audio signal the last 10 seconds look cleaner and the melody line is more distinct from the rest in comparison to the chromagram of the unfiltered audio 5.1. The next step is to calculate the most dominant note value for each timeframe. Due to the fact that the chromagram normalizes every timeframe to the maximum note value, the most dominant note always gets the value 1. The closer the rest of the notes are to zero the more likely the timeframe contains silence. If only a few values are close to one, then a chord or harmony is played. To filter out silence the sum over all note values of every timeframe is calculated and if this sum is twice as high as the average sum of notes of the whole song, then the frame is considered as silence. Otherwise the most dominant pitch is set to a fixed value while the rest of the notes are set to zero.

To extract the main melody in most cases only the most dominant pitch is needed, but sometimes the main melody is superimposed by other accompanying instruments. To prevent this the second most dominant pitch is also taken into consideration if its value is greater then a specific threshold. The result is shown in figure with the threshold 0.8

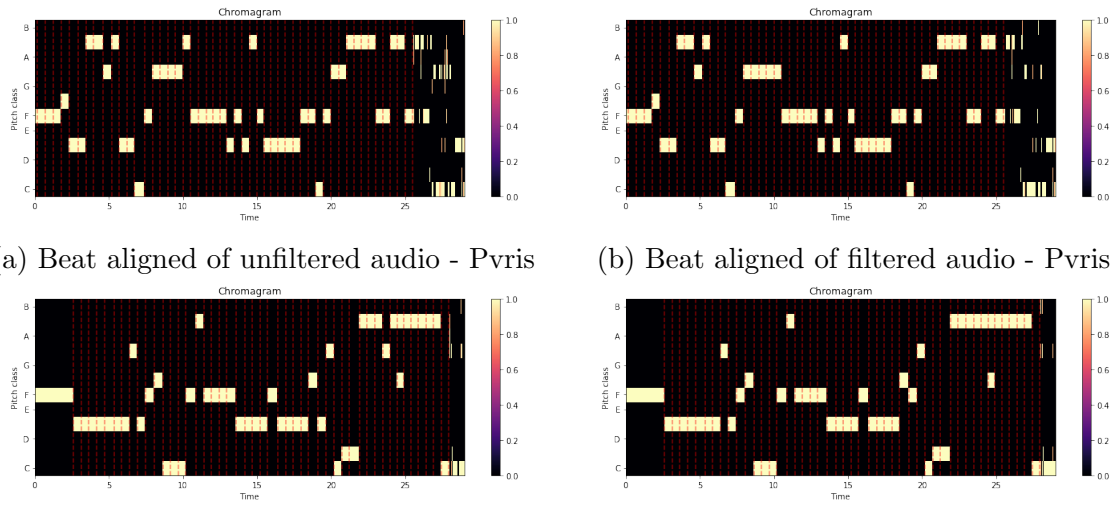
5.3.



(a) Single most dominant note only (b) First two most dominant notes

Figure 5.3: Thresholded Chroma Features - Sia

After that a beat tracking algorithm is applied to the song and the count of appearances of each note between two beats is calculated. The notes that appear the most are then set 1 while the rest is set to 0 for each section between two beats. This beat-alignment serves to make the similarity measurement invariant to the global tempo of the song. Even if a cover of a song is played with half the tempo of the original song, then the melody of each bar is still the same as in the faster version. Figure 5.4 show the different beat aligned features of both songs with bandpass filtered audio and unfiltered audio. The red lines are the detected beat events. Another option would be to separate the frames between the beats in more smaller sections. This would result in a better resolution of the melodic movement but at the same time increase the length of the data vectors that have to be compared to each other. The last processing step is



(a) Beat aligned of unfiltered audio - Pvris (b) Beat aligned of filtered audio - Pvris

(c) Beat aligned of unfiltered audio - Sia (d) Beat aligned of filtered audio - Sia

Figure 5.4: Processed Chroma Features - Sia

to key shift the chroma features to make the similarity analysis key invariant. One way to do so would be to estimate the key in which the song is played in and then shift all chroma features to the same base key, e.g. C Major or A Minor. Due to the structure

of the chroma features this can easily be done by assigning all estimated notes a new value a few keys higher or lower and thus shifting the whole song by a few semitones. The whole workflow to extract the chroma features for this thesis is shown in figure 5.5. Another consideration is to use the original chromagram without filtering out the

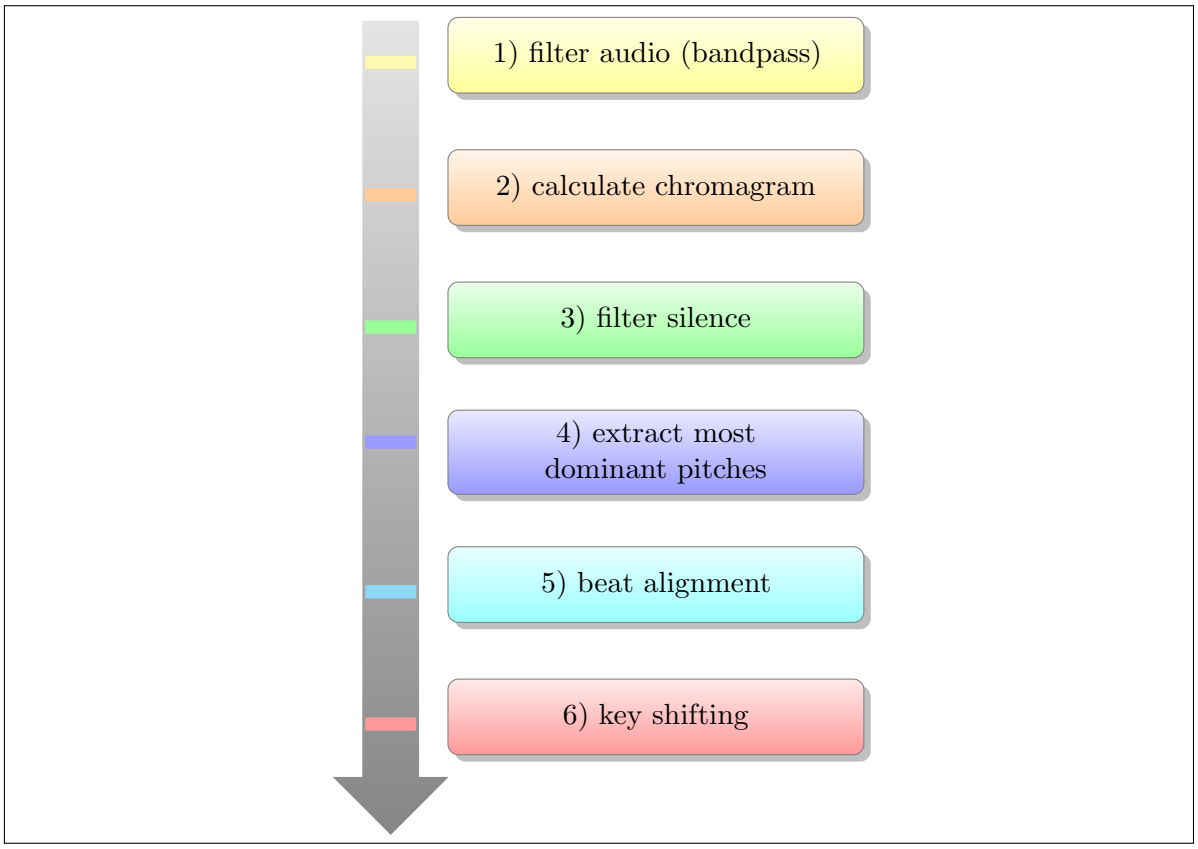
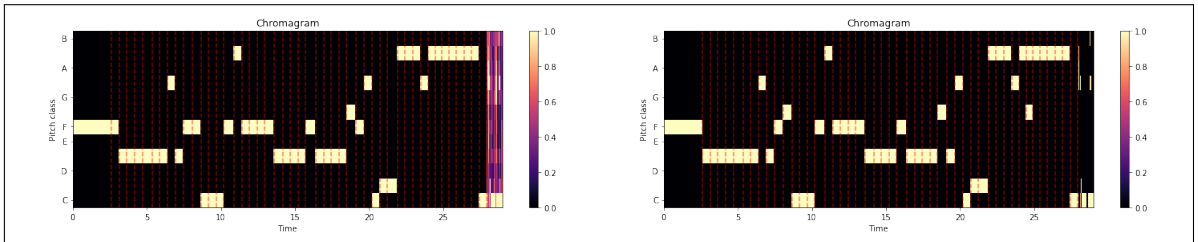


Figure 5.5: Workflow chroma feature extraction

least dominant keys and thus leaving the processing step 3 out. This means a possible tradeoff between accuracy and computation time. The result in the example song by Sia doesn't show a major impact, as can be seen in figure 5.6. In this thesis step 3 will be used in an attempt to get rid of the pitches of the accompaniment from the main melody line.



(a) Using full chromagram (b) Using most dominant pitches

Figure 5.6: Processing Step 3 Chroma Features

5.3 similarity of chroma features

In this section, two completely different approaches to measure the melodic similarity of two songs will be presented. The first one as proposed by [50] or [51] uses text retrieval methods to compare the chroma features of two songs and the second evaluates the usage of cross-correlation as a signal processing approach [52] and [53]

5.3.1 text retrieval

One possibility to process the chromagrams and to estimate similarity between the features of different songs is by handling the features like texts. Due to the extraction of only the main melody line in our feature vector there is only one note for every detected beat. The beat- and pitch-alignment done in the previous steps makes the features relatively time- and key invariant. One problem that remains is the different length of the various feature vectors. [51] mentions that this is indeed a problem when using the levenshtein distance to compute similarity. In their paper they use MIDI files instead of chroma features, but both contain information about the melody of songs so an adaption to chroma features is not an issue, because they can also easily be interpreted as simple strings. The levenshtein distance between the first i characters of a string S and the first j characters of T can be calculated as follows [51, p. 7]

$$lev_{S,T}(i,j) = \begin{cases} max(i,j), & \text{if } min(i,j) = 0 \\ min \begin{cases} lev_{S,T}(i-1,j) + 1 \\ lev_{S,T}(i,j-1) + 1 \\ lev_{S,T}(i-1,j-1) \\ +cost[S_i \neq T_j] \end{cases} & \text{else} \end{cases} \tag{5.1}$$

Xia (et al.) made some adjustments to this to be able to handle musical information.[51, pp. 7ff] For example to get rid of the problem of various lenghts between the songs, they only took the first 200 and the last 200 notes of every song because it could be observed that cover songs tend to share more common notes in the beginning and in the end of each song.

Englmeier (et al.) use more advanced information retrieval techniques called TF-IDF weights and explicit semantic analysis (ESA). "The TF-IDF weight is a measure which expresses the meaning of a term or a document within a collection of documents." [50, p. 186] To do so, they have to create "audio words" from the song database by splitting the audio signal into snippets, creating chroma features and clustering them with the k-means algorithm. The centroids are then added to a database. These audio words

can then be evaluated using the TF-IDF weights and ESA. Although their approach looks promising, a re-implementation of their algorithms would exceed the frame of this thesis.

5.3.2 cross-correlation

Another possibility to handle the extracted chroma features is by viewing them as ordinary signals and creating opportunities to apply classical signal processing algorithms. Ellis and Poliner use cross-correlation in their 2007 published paper [53]. Serra (et al.) also reference the work of Ellis and Poliner and discuss different weak points and influences of processing steps like beat tracking and key transposition to the overall performance of this similarity measurement. They also discuss and improve an other approach called dynamic time warping (DTW) further in their paper [52]. The focus in this thesis is set on the cross-correlation method. Given two discrete time signals $x[n]$ and $y[n]$ the cross-correlation between the both signals $k[n] = (x \star y)[n]$ can be denoted as follows:

$$k[n] = (x \star y)[n] = \sum_{m=-\infty}^{\infty} x[m]y[m-n] \tag{5.2}$$

For two 2-dimensional input matrices X with the dimensions M by N and Y as an P by Q matrix the cross-correlation result is a matrix C of size $M + P - 1$ rows and $N + Q - 1$ columns. Its elements are given by equation 5.3 [54], the bar over H denotes complex conjugation (in this case H is a matrix with real values only).

$$C(k,l) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X(m,n)\overline{H}(m-k, n-l) \tag{5.3}$$

with

$$-(P-1) \leq k \leq M-1 \tag{5.4}$$

$$-(Q-1) \leq l \leq N-1 \tag{5.5}$$

An example for the one dimensional cross-correlation is shown in figure 5.7 and the full two dimensional cross-correlation of two songs is figured in figure 5.8 and 5.9. Ellis and Poliner did not transpose the songs in the pre-processing step to match the keys of both audio files. Instead they calculated the full cross-correlation for all 12 possible transpositions and chose the best one. As input matrices they averaged all notes of the chroma features per beat and scaled them to have unit norm at each time slice/ beat frame. The cross-correlation is normalized by the length of the shorter song segment to bind the correlation result to an interval between 0 and 1. Additionally they filtered the result of the correlation with a high-pass filter. "We found that genuine matches were indicated not only by cross-correlations of large magnitudes, but that these large values

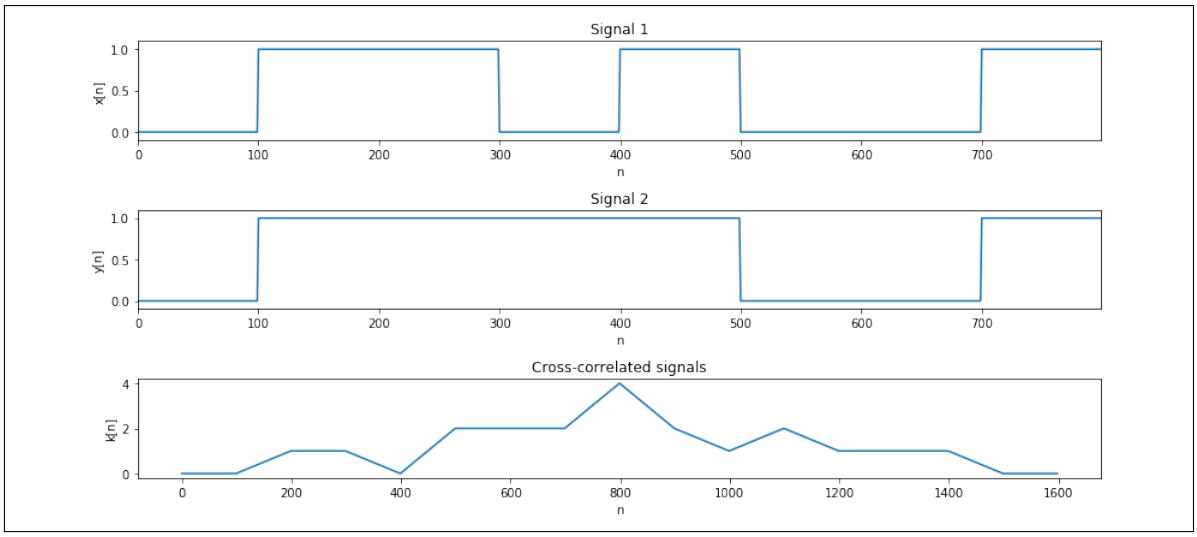


Figure 5.7: cross-correlation

occurred in narrow local maxima in the cross-correlations that fell off rapidly as the relative alignment changed from its best value. To emphasize these sharp local maxima, we choose the transposition that gives the largest peak correlation then high-pass filter that cross-correlation function with a 3dB point at 0.1 rad/sample" [53, p. 1431]
Serra (et al.) discussed various effects of pre-processing steps. E.g they note that a higher chroma resolution of 3 octaves gives better results. Also a key detection and transposition before cross-correlation gives slightly worse results in comparison to the method Ellis and Poliner used. This shows that there is still room for improvement of the algorithm used in this thesis. In summary, the implementation of Ellis and Poliners method in this thesis differs slightly from the approach by Ellis and Poliner. The chroma features are beat aligned, averaged per beat and normalized to unit length as well but additionally all chroma features are transposed to a common key (A in this case) in the pre-processing step. The full cross-correlation according to equation including key shifts by letting k run from $-(P-1) \leq k \leq M-1$ is shown in the figures 5.8 and 5.9 but due to the previous pre-processing key shift and the fact that both input matrices share the same amount of rows (12, one per key) these are not necessary and computation time can be safed by altering the equation to equation 5.6 resulting in a vector C with the correlation results.

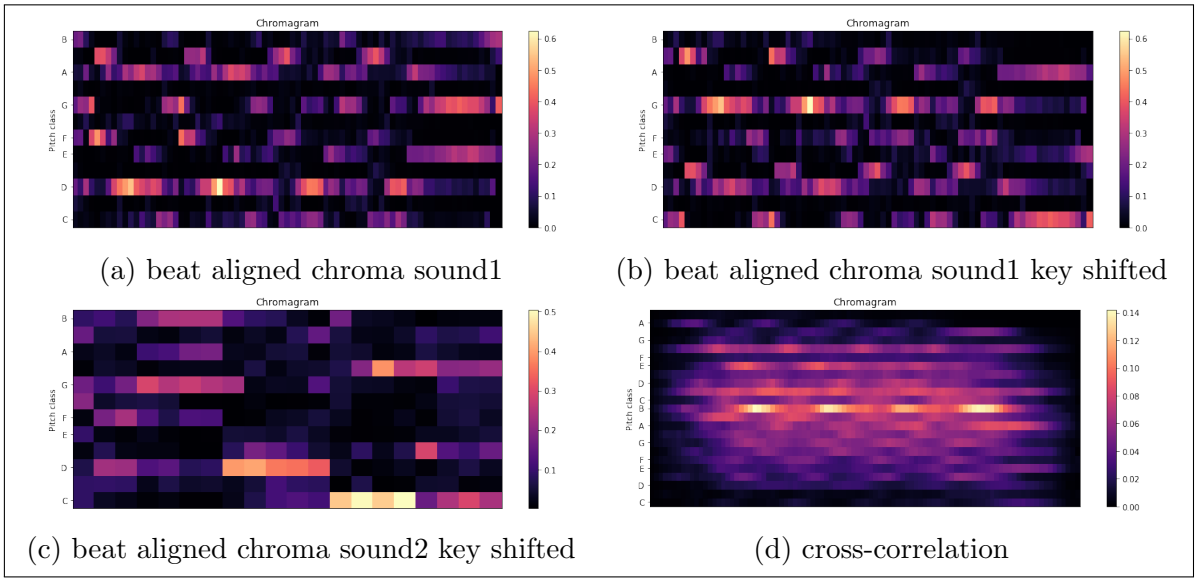
$$C(l) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X(m,n)\overline{H}(m,n-l) \tag{5.6}$$

$$-(Q-1) \leq l \leq N-1 \tag{5.7}$$

or even faster without calculating the edges of the matrix.

$$0 \leq l \leq N - Q \tag{5.8}$$

The post-processing step from Ellis and Poliner, namely the high-pass filtering of the result are left out in the implementation of this thesis as well.
Figure 5.8 shows two beat aligned, key shifted and per beat averaged chroma features of two short guitar snippets and their cross-correlation. The interesting row of the cross-correlation matrix is the middle row marked with the C key. It shows that both already key shifted melodies do not correlate well. In figure 5.9 the cross-correlation of



(a) beat aligned chroma sound1

(b) beat aligned chroma sound1 key shifted

(c) beat aligned chroma sound2 key shifted

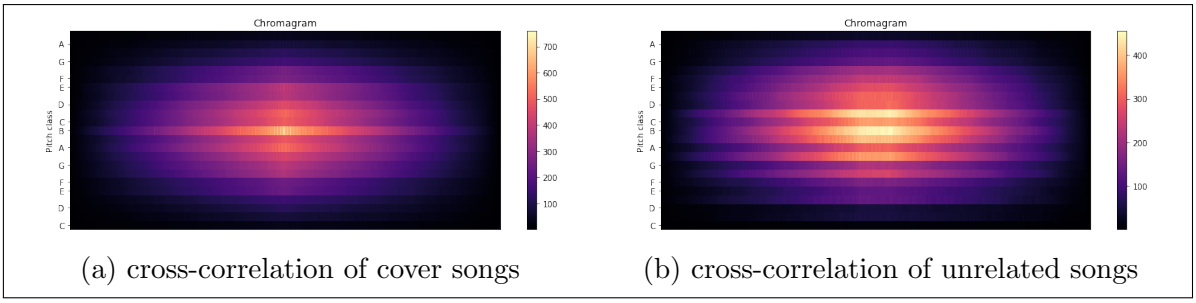
(d) cross-correlation

Figure 5.8: beat-aligned chromagram

the song "Rock you like a Hurricane" by the Scorpions and covered by Knightsbridge are shown in 5.9a and in contrast to this the cross-correlation of "Rock you like a Hurricane" with the song "Africa" by Toto are shown. The relevant middling line in the resulting matrix shows, that the covers do correlate much better than the unrelated songs. The other rows of the resulting matrix in 5.9b show that if one song would be key shifted again (possibly resulting in a manipulation of the melody line, which is not wanted) better correlations could be achieved. The important C-key row of the cross-correlation matrices from figure 5.9 are separately pictured in figure 5.10. The peak in 5.10a when cross-correlating the cover songs is clearly visible.

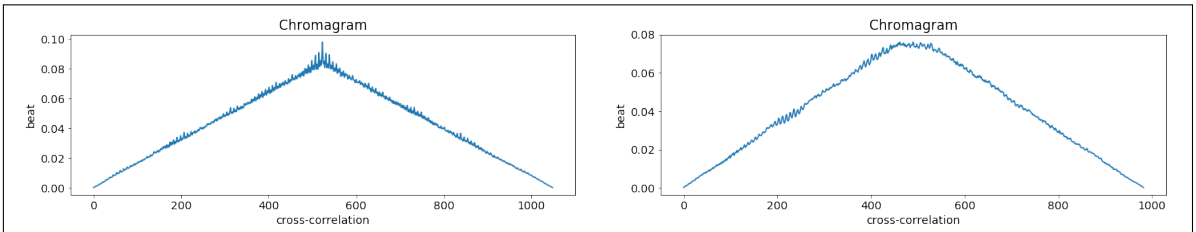
5.4 Validation

A good measure for the efficiency of a melodic similarity algorithm is the ability to find cover songs, remixes and recordings of the same song from different artists.



(a) cross-correlation of cover songs (b) cross-correlation of unrelated songs

Figure 5.9: Cross-correlation



(a) cross-correlation of cover songs (b) cross-correlation of unrelated songs

Figure 5.10: Cross-correlation

5.5 summary

Two different similarity metrics are chosen:

- Levenshtein Distance
- cross-correlation on full beat aligned and per beat averaged chroma features

These are stored in two different output text files. The vector length is dependent on the numbers of detected beats n

- out.notes (containing the estimated original key, the scale and a list of most dominant key per beat, key shifted to the A key (length n))
- out.chroma (full beat aligned chromagram, containing a $12xn$ matrix)

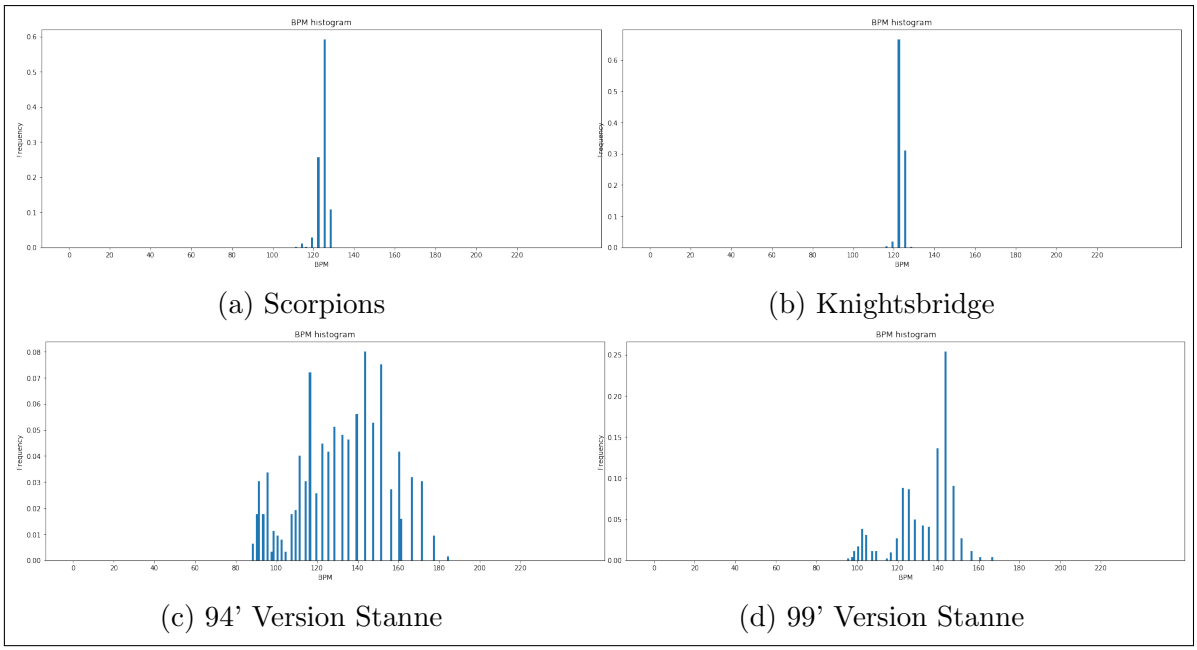
The output files contain strings like follows:
music/guitar2.mp3; G; minor; [6, 2, 5, 7, 7, 7, 7, 2, 2, 2, 2, 2, 0, 0, 0, 0, 3, 0, 0, 0]
music/guitar3.mp3; [[0.5209161 ,0.82440507,... ,0.68443549] ... [0.31470749,0.02552716,... ,0.01234249]] for the out.notes files and out.chroma files.

6. Rhythmic Similarity

This chapter provides an overview over some of the possibilities for computing music similarity by focusing on rhythmic features of different songs.
Nearly every MIR Toolkit provides an extraction tool for the beats per minute (BPM) and thus the tempo of each song. The most trivial solution to computing very low level rhythmic similarities is by sorting and comparing songs by their tempo. Of course there are far better and more accurate solutions. By just comparing the tempo of songs a lot of rhythm information is lost e.g. the rhythmic structures of songs like the time signature, up- and downbeats, etc.
This chapter presents some of the most promising approaches to compute rhythm similarities regarding the applicability in a big data framework.

6.1 Beat histogram

Other similarity measurements are e.g. the usage of beat histograms as proposed by Tsanetakis and Cook [55]. These are relatively similar to the later evaluated Rhythm Histograms. Gruhne (et al.) further improved the beat histograms and suggested an additional post processing step for the beat histogram before calculating the similarity between songs with the euclidean distance, to improve a comparison of two songs with different tempi by transforming the beat histograms into the logarithmic lag domain. They found, that logarithmic re-sampling of the lag axis of the histogram and cross-correlation with an artificial rhythmic grid improves the performance of this similarity measurement further [56, p. 182]. The essentia toolkit offers methods to extract the beat histogram. The different detected potential bpms are normalized to 1. If a song changes its tempo then multiple peaks can be seen.
Figure 6.1 shows the beat histograms of the song ”Rock you like a hurricane” by the Scorpions and covered by Knightsbridge as well as two different versions of the song ”Behind Space” from the swedish metal band In Flames, one is sung by Stanne Mikkels in 1994 and the second version was recorded with Anders Friden as a singer in 1999. The 1994 version changes its tempo in the outro of the song as can be seen in the figure.



(a) Scorpions

(b) Knightsbridge

(c) 94' Version Stanne

(d) 99' Version Stanne

Figure 6.1: Beat Histogram

Another feature that will just be mentioned here (one of the older ones from 2002) uses the beat spectrum as a feature [8].

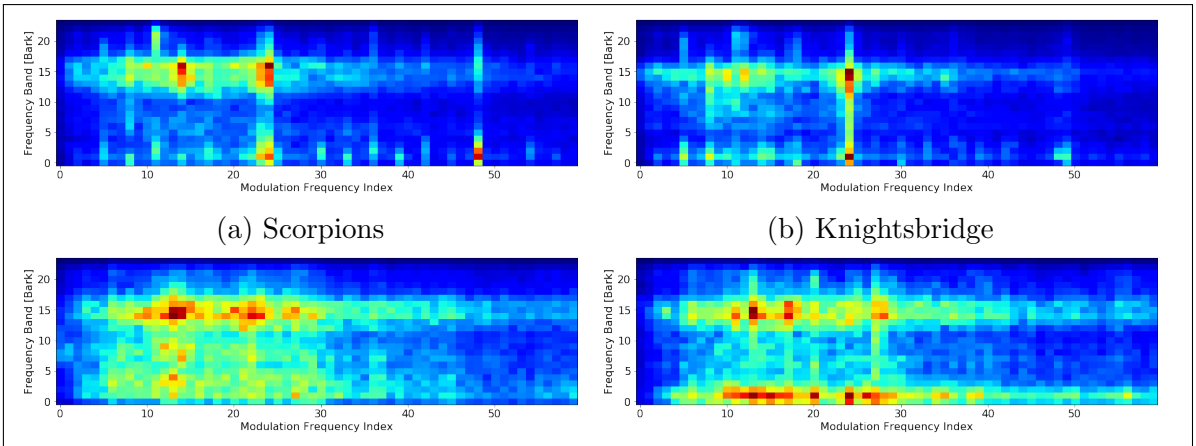
6.2 Rhythm patterns

A more state-of-the-art feature is the so called rhythm pattern, also known as fluctuation patterns for instance mentioned by [57]. To extract these features the rp_extractor library [58] was made publicly available by the TU Vienna [59]. Figure 6.2 shows the extracted rhythmic patterns of the previously mentioned songs "Rock you like a Hurricane" and "Behind Space". The similarities of the different versions from the same songs are quite visible while at the same time substantial differences between the different songs are recognizable.

The x-axis represents the frequency band converted to the bark scale (a scale representing the human auditory system comparable to the mel scale) and the y-axis represents the modulation frequency index representing the modulation frequencies up to 10Hz (600 BPM). The Bark of a frequency f can be determined using formula 6.1.

$$Bark = 13 \arctan(0.00076f) + 3.5 \arctan((f/7500)^2) \tag{6.1}$$

The algorithm to extract the rhythm patterns as well as the rhythm histogram and statistical spectrum descriptors measuring the variations over the critical frequency bands can be seen in figure 6.3.



(a) Scorpions (b) Knightsbridge

(c) 94' Version Stanne (d) 99' Version Anders

Figure 6.2: Rhythmic Patterns

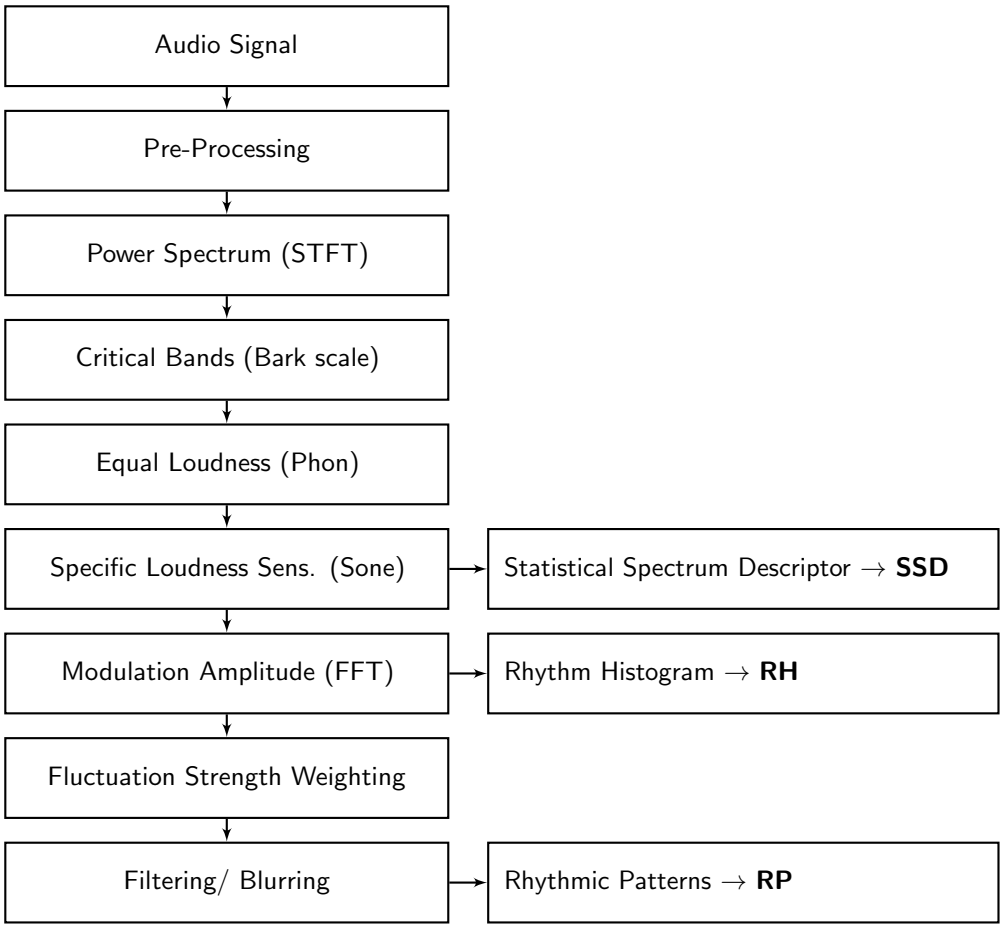


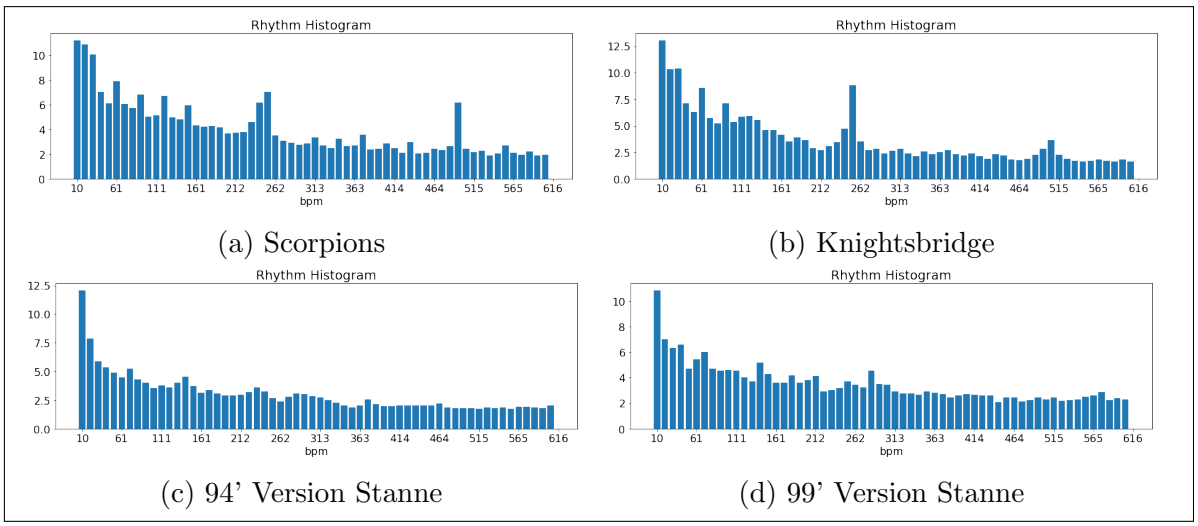
Figure 6.3: Rhythm Pattern extraction [59]

So in conclusion the Rhythm Patterns basically represent the BPM of various frequency bands. To compare two different songs the euclidean distance between the vectorized rhythm pattern matrices can be calculated as Pampalk suggests [60, p. 40] Pohle, Schnitzer et al. refined Fluctuation Patterns into Onset Patterns e.g. by using

semitone bands instead of fewer critical bands to detect onsets. [61]
This thesis however focuses on Fluctuation/ Rhythm patterns extracted with the rp_extractor library.

6.3 Rhythm Histogram

A more simplistic and lower dimensional feature coming with the rp_extract toolkit is the Rhythm histogram. "The Rhythm Histogram features we use are a descriptor for general rhythmics in an audio document. Contrary to the Rhythm Patterns and the Statistical Spectrum Descriptor, information is not stored per critical band. Rather, the magnitudes of each modulation frequency bin of all 24 critical bands are summed up, to form a histogram of "rhythmic energy" per modulation frequency. The histogram contains 60 bins which reflect modulation frequency between 0 and 10 Hz." [57, p. 3]. The difference in comparison to the beat histogram mentioned earlier in section 6.1 appears to be, that the beat histogram focuses on the basic tempo of the whole song while the rhythm histogram takes all frequency bands and therefor the sub-rhythms of single instruments into account.



(a) Scorpions (b) Knightsbridge

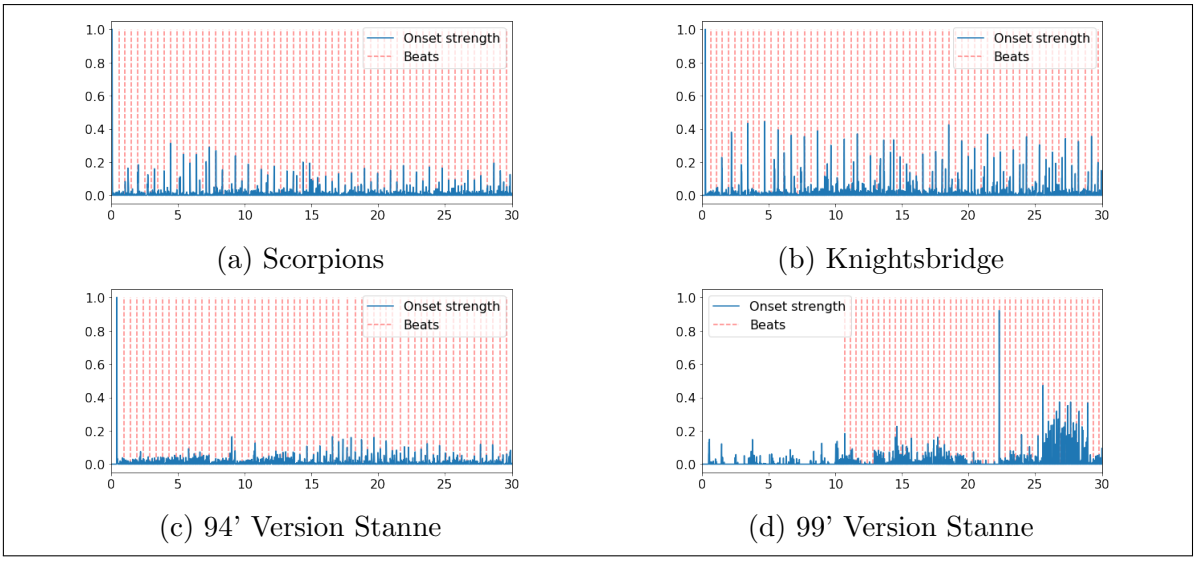
(c) 94' Version Stanne (d) 99' Version Stanne

Figure 6.4: Rhythm Histogram

6.4 cross-correlation

Estimating the onset strength per beat and creating a discrete-time signal for each song is an other option. Similar to the chroma features the cross-correlation of the onset functions could be used as a similarity measurement.

Looking at the extracted onset features of the Song "Behing Space" by In Flames (sung by Anders Frieden 99' and Stanne Mikkels 94') in figure 6.5, one can see that the



(a) Scorpions

(b) Knightsbridge

(c) 94' Version Stanne

(d) 99' Version Stanne

Figure 6.5: Detected Onsets (first 30 seconds)

quality of these signals are greatly dependent on the underlying beat extraction and onset detection algorithms. E.g. the librosa toolkit struggles to detect beats in the first 10 seconds of the version of the song from the year 1999. Also this representation seems to contain a lot less valuable and comparable information in contrast to Fluctuation Patterns. In conclusion this approach is discarded and not further considered and tested in this thesis.

6.5 summary

Three different similarity metrics are chosen:

- euclidean distance between beat histograms
- euclidean distance between rhythm histograms
- euclidean distance between rhythm patterns

These are stored in three different output text files.

- out.bh (containing the estimated overall bpm and a vector for the beat histogram normalized to one (length 250))
- out.rh (containing a vector for the rhythm histogram extracted with rp_extract (length 60))
- out.rp (containing a vectorized matrix for the rhythm patterns extracted with rp_extract (length $24 \cdot 60$))

The text files contain strings like:
music/CHANDELIER2.mp3; 86.9380264282; [0. ... 0.01453488 ... 0. 0.]
music/CHANDELIER2.mp3,15.2521291416,10.10441871, ... ,2.2519330706
music/CHANDELIER2.mp3,0.0237481782333,0.0208784207788, ... ,0.00204177442894

7. Other Similarity metrics and Performance

7.1 Test Datasets

Chapter 3.1 introduced a range of MIR datasets but not all are fitting to the problems this thesis evaluates. To test the algorithms on the one hand a lot of data is needed, so the Free Music Archive with its over 100000 songs is a solid option for performance tests. However on the other hand the genre distribution in the FMA dataset is quite one sided. Most of the songs are tagged as experimental or rock. Also this dataset may not be really representative for actual popular music, a lot of the songs are live recordings with poor audio quality, possibly influencing the results. The 1517 artists dataset offers 19 different genres with songs relatively equally distributed. For an objective evaluation of the proposed algorithms e.g. by genre recall this dataset is ideal. The last source used in this thesis is the private music collection. This collection is biased towards metal music but due to the match with personal taste, it offers a subjective evaluation of the results of the similarity analysis. In conclusion that sums up to about 117000 songs for performance tests and about 12000 songs for a detailed evaluation of the algorithms

Table 7.1: used music datasets

fma	106.733 Songs
private	8484 Songs
1517 artists	3180 Songs

7.2 Feature Extraction Performance

After evaluating the different features in the last three chapters, this one discusses the feature extraction process in detail. The post processing of the features like the note estimation from the chroma features and the calculation of statistic features from the

mfccs was already explained in the previous chapters and is left out here. The full code is in the appendices. For the post processing part the python numpy library was used.

7.2.1 Librosa

For most of the plots in the introduction section 2 the python toolkit librosa was used because of its ease of use and very good documentation. The code example shows the necessary methods to extract the most important features like mfcc, chromagram and beats/ onsets.

```
path = ('music/guitar2.mp3')
x, fs = librosa.load(path)
mfcc = librosa.feature.mfcc(y=x, sr=fs, n_mfcc=12)
onset_env = librosa.onset.onset_strength(x, fs, aggregate=np.median)
tempo, beats = librosa.beat.beat_track(onset_envelope=onset_env,sr=fs)
hop_length = 512
times = librosa.frames_to_time(np.arange(len(onset_env)), sr=fs, hop_length=hop_length)
chroma = librosa.feature.chroma_stft(x, fs)
```

But when extracting features from batches of audio data the librosa library showed to be very slow. For a very small dataset of 100 songs, the extraction of just the mean, variance and covariance of the mfccs and the estimated notes from the chromagram took about 48 minutes. For larger datasets like the 1517 artists dataset the feature extraction process would have taken about 22 hours.

7.2.2 Essentia

[62] compares different Audio feature extraction toolboxes and shows that essentia is a much faster alternative due to the underlying C++ Code and provides even more features, but it is a bit less well documented and requires more effort in implementation at the same time.
In the end the code to extract the necessary features had to be rewritten for the usage of essentia due to the slow performance of librosa. Essentia offers two different ways to handle audio files. The first one is to use the essentia standard library. It offers similar methods to librosa and uses an imperative programming style. The audio file has to be read, sliced and preprocessed by hand. The second way is to use essentia streaming. Basically a network of connected algorithms is created and they handle and schedule the ”how and when” a process is called.

Essentia Standard

In the final extractor code the mfcc calculation and beat histogram estimation is done with the essenia standard library, because it offers a fast way to implement feature extraction.

```
audio = es.MonoLoader(filename=path, sampleRate=fs)()
hamming_window = es.Windowing(type='hamming')
spectrum = es.Spectrum()
mfcc = es.MFCC(numberCoefficients=13)
frame_sz = 2048
hop_sz = 1024
mfccs = numpy.array([mfcc(spectrum(hamming_window(frame)))[1]
  for frame in es.FrameGenerator(audio, frameSize=frame_sz, hopSize=hop_sz)])
rhythm_extractor = es.RhythmExtractor2013(method="multifeature")
bpm, beats, beats_confidence, _, beats_intervals = rhythm_extractor(audio)
peak1_bpm, peak1_weight, peak1_spread, peak2_bpm, peak2_weight, peak2_spread, histogram =
  es.BpmHistogramDescriptors()(beats_intervals)
```

Essentia Streaming

The essentia streaming library is used to calculate the chroma features in the final code. It eases up the filtering with a high- and a lowpass filter.

```
loader = ess.MonoLoader(filename=path, sampleRate=44100)
HP = ess.HighPass(cutoffFrequency=128)
LP = ess.LowPass(cutoffFrequency=4096)
framecutter = ess.FrameCutter(frameSize=frameSize, hopSize=hopSize, silentFrames='noise')
windowing = ess.Windowing(type='blackmanharris62')
spectrum = ess.Spectrum()
spectralpeaks = ess.SpectralPeaks(orderBy='magnitude', magnitudeThreshold=0.00001,
  minFrequency=20, maxFrequency=3500, maxPeaks=60)
hpcp = ess.HPCP()
hpcp_key = ess.HPCP(size=36, referenceFrequency=440, bandPreset=False, minFrequency=20,
  maxFrequency=3500, weightType='cosine', nonLinear=False, windowSize=1.)
key = ess.Key(profileType='edma', numHarmonics=4, pcpSize=36, slope=0.6,
  usePolyphony=True, useThreeChords=True)
pool = essentia.Pool()
loader.audio >> HP.signal
HP.signal >> LP.signal
LP.signal >> framecutter.signal
framecutter.frame >> windowing.frame >> spectrum.frame
spectrum.spectrum >> spectralpeaks.spectrum
spectralpeaks.magnitudes >> hpcp.magnitudes
```

```
21 spectralpeaks.frequencies >> hpcp.frequencies
22 spectralpeaks.magnitudes >> hpcp_key.magnitudes
23 spectralpeaks.frequencies >> hpcp_key.frequencies
24 hpcp_key.hpcp >> key.pcp
25 hpcp.hpcp >> (pool, 'tonal.hpcp')
26 essentia.run(loader)
27 chroma = pool['tonal.hpcp'].T
```

Essentia performance

The calculation with the essentia library for 100 songs took less than half of the time librosa needed. This is significant improvement, however the essentia library uses only one CPU core so that performance was further improved by using the parallel python library as presented in the next code snippet.

7.2.3 Essentia parallel

Multiple CPU cores get a part of the filelist of all songs and can compute the features fully parallel.

```
1  job_server = pp.Server()
2  job_server.set_ncpus(ncpus)
3  jobs = [ ]
4  for index in xrange(startjob, parts):
5     starti = start+index*step
6     endi = min(start+(index+1)*step, end)
7     jobs.append(job_server.submit(parallel_python_process, (index, filelist[starti:endi],1,1,1,1,1)))
8     gc.collect()
9  times = sum([job() for job in jobs])
10 job_server.print_stats()
```

The computation time takes about 33 seconds for batches of 4 songs from the list on one CPU core. Using 4 CPU cores for 32 songs, the overall processing time could be reduced to one minute.

$$time = \frac{\#songs}{4 \cdot \#CPUs} \cdot 33s \tag{7.1}$$

Parallel python also opens up the possibility to use a cluster instead of a single node PC.

7.2.4 rp_extractor

For the extraction of the rhythm patterns and rhythm histrogram features as described in chapter 6 the rp_extractor tool provided by the TU Wien was used. Although running in parallel on all CPU cores on a single node, the extraction of the features from 29 songs takes about 90 seconds.

7.2.5 performance

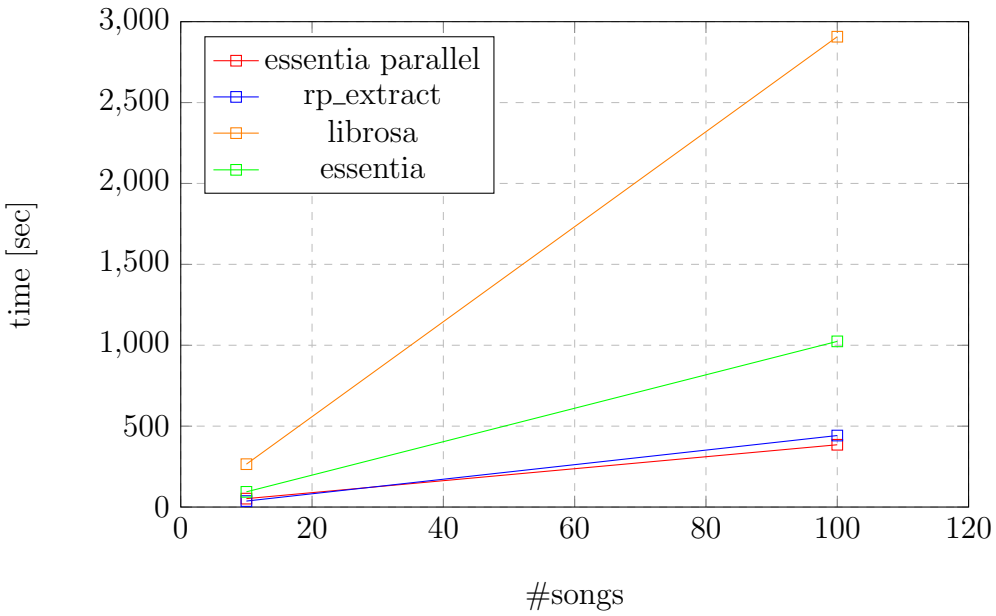


Figure 7.1: Performance of various toolkits

In summary the estimated time based on the performance measurements can be calculated leading to the conclusion that the extraction of the features for the full dataset including the FMA dataset can only be done with the help of a computing cluster or the FMA has to be left out. For this thesis the extracted features from the private music collection and the 1517 artists dataset summing up to over 10000 songs is more than sufficient and the FMA dataset is left out. For future work and performance testing however the inclusion of larger datasets has to be considered.

- 3h24 - 1517 artists - essentia parallel single node
- 3h54 - 1517 artists - rp_extract
- 9h06 - private dataset - essentia parallel single node
- 10h24 - private dataset - rp_extract
- 125h - full dataset - essentia parallel single node

- 143h - full dataset - rp_extract

8. Big Data Framework Spark

After accumulating the Data and presenting various methods to extract different audio features, the following chapters describe the data analysis with Apache Spark [63]. Chapter 8 deals with the implementation of the various similarity measurements while chapter 9 deals with the handling of larger amounts of data, runtime analysis and the combination of multiple similarity measurements.

8.1 Apache Hadoop and Spark

First of all a small introduction to Big Data processing with Spark is given. With the ever growing availability of huge amounts of high dimensional data the need for toolkits and efficient algorithms to handle these grew as well over the past years. Map-Reduce as a

8.1.1 Hadoop and Map Reduce

8.1.2 Spark and RDDs

8.1.3 Spark DataFrame

8.2 Data aggregation

To work with the features a few transformations have to be done first.

```
1 song = "music/TURCA1.wav"
2 chroma = sc.textFile("features/out[0-9]*.notes")
3 chroma = chroma.map(lambda x: x.split(';'))
4 chroma = chroma.map(lambda x: (x[0], x[1], x[2], x[3].replace("10",'K').replace("11",'L').replace
  ("0",'A').replace("1",'B').replace("2",'C').replace("3",'D').replace("4",'E').replace("5",'F').replace("6",
  'G').replace("7",'H').replace("8",'I').replace("9",'J')))
5 chroma = chroma.map(lambda x: (x[0], x[1], x[2], x[3].replace(',','').replace(' ','')))
6 df = spark.createDataFrame(chroma, ["id", "key", "scale", "notes"])
```

8.3 Euclidean Distance

```
1 distance_udf = F.udf(lambda x: float(distance.euclidean(x, comparator_value)), FloatType())
2 result = df_vec.withColumn('distances', distance_udf(F.col('features')))
3 result = result.select("id", "distances").orderBy('distances', ascending=True)
4 result = result.rdd.flatMap(list).collect()
```

8.4 Bucketed Random Projection

```
1 brp = BucketedRandomProjectionLSH(inputCol="features", outputCol="hashes", seed=12345,
  bucketLength=1.0)
2 model = brp.fit(df_vec)
3 comparator_value = Vectors.dense(comparator[0])
4 result = model.approxNearestNeighbors(df_vec, comparator_value, df_vec.count()).collect()
5 rf = spark.createDataFrame(result)
6 result = rf.select("id", "distCol").rdd.flatMap(list).collect()
```

8.5 Cross-correlation

```
1 def chroma_cross_correlate(chroma1, chroma2):
2     corr = sc.signal.correlate2d(chroma1, chroma2, mode='full')
3     transposed_chroma = np.transpose(corr)
4     mean_line = transposed_chroma[12]
5     return np.max(mean_line)
6 distance_udf = F.udf(lambda x: float(chroma_cross_correlate(x, comparator_value)), DoubleType()
  )
7 result = df_vec.withColumn('distances', distance_udf(F.col('chroma')))
8 result = result.select("id", "distances").orderBy('distances', ascending=False)
9 result = result.rdd.flatMap(list).collect()
```

8.6 Kullback-Leibler Divergence

```
1 def symmetric_kullback_leibler(vec1, vec2):
2     mean1 = np.empty([13, 1])
3     mean1 = vec1[0:13]
4     cov1 = np.empty([13,13])
```

```
    cov1 = vec1[13:].reshape(13, 13)
    mean2 = np.empty([13, 1])
    mean2 = vec2[0:13]
    cov2 = np.empty([13,13])
    cov2 = vec2[13:].reshape(13, 13)
    d = 13
    div = 0.25 * (np.trace(cov1 * np.linalg.inv(cov2)) + np.trace(cov2 * np.linalg.inv(cov1)) + np.trace( (np.linalg.inv(cov1) + np.linalg.inv(cov2)) * (mean1 - mean2)**2) - 2*d)
    return div
distance_udf = F.udf(lambda x: float(symmetric_kullback_leibler(x, comparator_value)), DoubleType())
result = df_vec.withColumn('distances', distance_udf(F.col('features')))
result = result.select("id", "distances").orderBy('distances', ascending=True)
result = result.rdd.flatMap(list).collect()
```

8.7 Jensen-Shannon Divergence

```
def jensen_shannon(vec1, vec2):
    mean1 = np.empty([13, 1])
    mean1 = vec1[0:13]
    cov1 = np.empty([13,13])
    cov1 = vec1[13:].reshape(13, 13)
    mean2 = np.empty([13, 1])
    mean2 = vec2[0:13]
    cov2 = np.empty([13,13])
    cov2 = vec2[13:].reshape(13, 13)
    mean_m = 0.5 * (mean1 + mean2)
    cov_m = 0.5 * (cov1 + mean1 * np.transpose(mean1)) + 0.5 * (cov2 + mean2 * np.transpose(mean2)) - (mean_m * np.transpose(mean_m))
    div = 0.5 * np.log(np.linalg.det(cov_m)) - 0.25 * np.log(np.linalg.det(cov1)) - 0.25 * np.log(np.linalg.det(cov2))
    if np.isnan(div):
        div = np.inf
    return div
distance_udf = F.udf(lambda x: float(jensen_shannon(x, comparator_value)), DoubleType())
result = df_vec.withColumn('distances', distance_udf(F.col('features')))
result = result.select("id", "distances").orderBy('distances', ascending=True)
result = result.rdd.flatMap(list).collect()
```

Problem mit den "skyrocketing determinanten" -> Lösung: cholesky zerlegung, geht nicht - nicht positiv definit[3, p.45]

8.8 Levenshtein distance

```
1 def get_neighbors_notes(song):
2     filterDF = df.filter(df.id == song)
3     filterDF.first()
4     comparator_value = filterDF.collect()[0][3]
5     print comparator_value
6     df_merged = df.withColumn("compare", lit(comparator_value))
7     df_levenshtein = df_merged.withColumn("word1_word2_levenshtein", levenshtein(col("notes"),
  col("compare")))
8     df_levenshtein.sort(col("word1_word2_levenshtein").asc()).show()
```

8.9 Andere MapReduce tasks

8.9.1 Alternating Least Squares

8.9.2 TF-IDF weights

8.9.3 DIMSUM all-pairs similarity

8.10 Combining different measurements

8.11 performance

high data locality, full parallelizable, very low replication rate

9. Results

9.1 Runtime

9.2 Cover song identification

9.3 Genre similarity

9.4 Improvements and outlook

9.5 Definition of similarity

Rhythm, Timbre, Melodic, Genre and metadata, Genre specific features, combinations and variable model, Collaborative Filtering, Lyrics

reduce hubness

References

[1] Peter Knees and Markus Schedl. Music Similarity and Retrieval. An introduction to web audio- and web-based strategies. In: Springer, 2016. ISBN: 9783662497203.

[2] B. McFee et al. The Million Song Dataset Challenge, AdMIRe '12. In:

[3] Dr. Dominik Schnitzer. Dealing with the Music of the World: Indexing Content-Based Music Similarity Models for Fast Retrieval in Massive Databases, 1st ed. In: 2012. ISBN: 9781477494158.

[4] Dr. Dominik Schnitzer. Audio Music Similarity. In: URL: http://www.musly.org/index.html.

[5] Brian McFee et al. LibROSA: Audio and Music Signal Analysis in Python. In: Proceedings of the 14th Python in Science Conference.

[6] Prélude cis-Moll (Rachmaninow). In: URL: https://imslp.org/wiki/Morceaux_de_fantaisie\%2C_Op.3_(Rachmaninoff\%2C_Sergei).

[7] Matija Marolt. A Mid-level Melody-based Representation for Calculating Audio Similarity. In: ISMIR 2006, 7th International Conference on Music Information Retrieval, Victoria, Canada, 8-12 October 2006. 2006.

[8] Jonathan Foote, Matthew Cooper, and Unjung Nam. Audio Retrieval by Rhythmic Similarity. In: Jan. 2002.

[9] Zhao Yufeng and Li Xinwei. Design and Implementation of Music Recommendation System Based on Hadoop. In: 2018. DOI: 10.2991/icsnce-18.2018.36..

[10] B. McFee and G.R.G. Lanckriet. Large-scale music similarity search with spatial trees. In: ISMIR '11. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.226.5060.

[11] Thierry Bertin-Mahieux et al. The Million Song Dataset. In: Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011). 2011.

[12] Guangyu Xia et al. MidiFind: Similarity Search and Popularity Mining in Large MIDI Databases, 259-276. In: 2013. DOI: 10.1007/978-3-319-12976-1_17..

[13] Jong Wook Kim et al. CREPE: A Convolutional Representation for Pitch Estimation. In: Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 2018.

[14] Sankalp Gulati, Joan Serra, and Xavier Serra. An evaluation of methodologies for melodic similarity in audio recordings of Indian art music, 678-682. In: 2015. DOI: 10.1109/ICASSP.2015.7178055..

[15] Cumhur Erkut et al. Extraction of Physical and Expressive Parameters for Model-based Sound Synthesis of the Classical Guitar. In: 2002.

[16] M. Mandel and D. Ellis. Song-level features and support vector machines for music classification. In: Proceedings of the 6th International Conference on Music Information Retrieval, ISMIR, 2005.

[17] D. Schnitzer et al. Using mutual proximity to improve content-based audio similarity. In: Proceedings of the 12th International Society for Music Information Retrieval Conference, ISMIR, 2011.

[18] Olivier Lartillot and Petri Toiviainen. MIR in Matlab (II): A Toolbox for Musical Feature Extraction from Audio. In: Proceedings of the 8th International Conference on Music Information Retrieval, ISMIR 2007, Vienna, Austria, September 23-27, 2007.

[19] MATLAB. in: URL: https://de.mathworks.com/help/matlab/index.html.

[20] John W. Eaton et al. GNU Octave version 4.2.0 manual: a high-level interactive language for numerical computations. In: 2016. URL: http://www.gnu.org/software/octave/doc/interpreter/.

[21] Martin Ariel Hartmann. A port of MIRToolbox for Octave. In: 2016. URL: https://github.com/martinarielhartmann/mirtooloct.

[22] B.Mathieu et al. YAAFE, an Easy to Use and Efficient Audio Feature Extraction Software. In: Proceedings of the 11th ISMIR conference, Utrecht, Netherlands, 2010.

[23] D. Bogdanov et al. ESSENTIA: an Audio Analysis Library for Music Information Retrieval. In: International Society for Music Information Retrieval Conference (ISMIR'13). 2013, pp. 493–498.

[24] Michael I. Mandel and Daniel P. W. Ellis. Labrosa's audio music similarity and classification submissions. In: 2007.

[25] Jupyter. In: URL: https://jupyter.org/index.html.

[26] Für Elise. In: URL: https://upload.wikimedia.org/wikipedia/commons/a/a9/BH_116_Vergleich.png.

[27] Paul Brossier et al. aubio/aubio: 0.4.8 (Version 0.4.8). In: 2018. URL: http://doi.org/10.5281/zenodo.1494152.

[28] J. Salamon and E. Gómez. Melody Extraction from Polyphonic Music Signals using Pitch Contour Characteristics. In: IEEE Transactions on Audio, Speech and Language Processing, 20(6):1759-1770. 2012.

[29] C. Cannam, C. Landone, and M. Sandler. Sonic Visualiser: An Open Source Application for Viewing, Analysing, and Annotating Music Audio Files. In: Proceedings of the ACM Multimedia 2010 International Conference. Firenze, Italy, 2010, pp. 1467–1468.

[30] Spotify API. in: URL: https://developer.spotify.com/documentation/.

[31] The Echo Nest. In: URL: http://the.echonest.com/.

[32] Spotipy - a Python client for The Spotify Web API. in: URL: https://github.com/plamere/spotipy.

[33] Michäel Defferrard et al. FMA: A Dataset for Music Analysis. In: 18th International Society for Music Information Retrieval Conference, 2017. URL: https://arxiv.org/abs/1612.01840.

[34] John Thickstun, Zaid Harchaoui, and Sham M. Kakadetitle. Learning Features of Music from Scratch. In: International Conference on Learning Representations (ICLR). 2017. URL: https://arxiv.org/abs/1611.09827.

[35] Klaus Seyerlehner. 1517-Artists Dataset. In: 2010. URL: http://www.seyerlehner.info/index.php?p=1_3_Download.

[36] R. Bittner et al. "MedleyDB: A Multitrack Dataset for Annotation-Intensive MIR Research". In: 15th International Society for Music Information Retrieval Conference, 2014.

[37] R. Bittner et al. MedleyDB 2.0: New Data and a System for Sustainable Data Collection. In: New York, NY, USA: International Conference on Music Information Retrieval (ISMIR-16). 2016.

[38] B. De Man et al. The Open Multitrack Testbed. In: 137th Convention of the Audio Engineering Society, 2014.

[39] Soundcloud bqpd. In: URL: https://soundcloud.com/bq_pd.

[40] Hendrik Schreiber. Improving Genre Annotations for the Million Song Dataset. In: In Proceedings of the 16th International Society for Music Information Retrieval Conference (ISMIR), pages 241-247, Málaga, Spain, Oct. 2015.

[41] Last.fm dataset, the official song tags and song similarity collection for the Million Song Dataset. In: URL: http://labrosa.ee.columbia.edu/millionsong/lastfm.

[42] The Echo Nest Taste profile subset, the official user data collection for the Million Song Dataset. In: URL: http://labrosa.ee.columbia.edu/millionsong/tasteprofile.

[43] SecondHandSongs dataset, the official list of cover songs within the Million Song Dataset. In: URL: http://labrosa.ee.columbia.edu/millionsong/secondhand.

[44] Spotify Playlist Miner. In: URL: https://developer.spotify.com/community/showcase/playlist-miner-spotify/.

[45] Spotify Terms and Conditions of Use. In: URL: https://www.spotify.com/lt/legal/end-user-agreement/plain/#s9.

[46] Spotify Developer Terms of Service. In: URL: https://developer.spotify.com/terms/.

[47] Spotify Commercial Restrictions. In: URL: https://developer.spotify.com/legal/commercial-restrictions/.

[48] Klaus Seyerlehner and Markus Schedl. Block-Level Audio Features for Music Genre Classification. In: 2009.

[49] Nicola Orio and Antonio Rodà. A Measure of Melodic Similarity based on a Graph Representation of the Music Structure. In: ISMIR. 2009.

[50] David Englmeier et al. Musical similarity analysis based on chroma features and text retrieval methods. In: Datenbanksysteme für Business, Technologie und Web (BTW 2015) - Workshopband. Ed. by Norbert Ritter et al. Bonn: Gesellschaft für Informatik e.V., 2015, pp. 183–192.

[51] Guangyu Xia et al. MidiFind: Similarity Search and Popularity Mining in Large MIDI Databases. In: Oct. 2013, pp. 259–276. ISBN: 978-3-319-12975-4. DOI: 10.1007/978-3-319-12976-1_17.

[52] Joan Serra et al. Chroma Binary Similarity and Local Alignment Applied to Cover Song Identification. In: Audio, Speech, and Language Processing, IEEE Transactions o 16 (Sept. 2008), pp. 1138 –1151. DOI: 10.1109/TASL.2008.924595.

[53] Daniel P.W. Ellis and Graham E. Poliner. Identifying 'Cover Songs' with Chroma Features and Dynamic Programming Beat Tracking. In: vol. 4. May 2007, pp. IV–1429. DOI: 10.1109/ICASSP.2007.367348.

[54] Mathworks. xcorr2. In: URL: https://www.mathworks.com/help/signal/ref/xcorr2.html.

[55] George Tzanetakis and Perry Cook. Musical Genre Classification of Audio Signals. In: IEEE Transactions on Speech and Audio Processing 10 (Aug. 2002), pp. 293 –302. DOI: 10.1109/TSA.2002.800560.

[56] Matthias Gruhne, Christian Dittmar, and Daniel Gärtner. Improving Rhythmic Similarity Computation by Beat Histogram Transformations. In: Jan. 2009, pp. 177–182.

[57] Thomas Lidy and Andreas Rauber. Evaluation of Feature Extractors and Psycho-Acoustic Transformations for Music Genre Classification. In: Jan. 2005, pp. 34–41.

[58] Audio Feature Extraction. In: URL: https://github.com/tuwien-musicir/rp_extract.

[59] Audio Feature Extraction - Rhythm Patterns. In: URL: http://www.ifs.tuwien.ac.at/mir/audiofeatureextraction.html.

[60] Elias Pampalk. Computational Models of Music Similarity and their Application in Music Information Retrieval. In: 2006.

[61] Tim Pohle et al. On Rhythm and General Music Similarity. In: Jan. 2009, pp. 525–530.

[62] David Moffat, David Ronan, and Joshua Reiss. An Evaluation of Audio Feature Extraction Toolboxes. In: Nov. 2015. DOI: 10.13140/RG.2.1.1471.4640.

[63] Matei Zaharia et al. Apache Spark: A Unified Engine for Big Data Processing. In: Commun. ACM 59.11 (Oct. 2016), pp. 56–65. ISSN: 0001-0782. DOI: 10.1145/2934664. URL: http://doi.acm.org/10.1145/2934664.

10. Appendix

10.1 Spotipy Crawler

```
from __future__ import print_function
from spotipy.oauth2 import SpotifyClientCredentials
import json, sys, spotipy, time, os.path
import requests, urllib
import matplotlib.pyplot as pl
import h5json, scipy
import numpy as np
from scipy.spatial import distance

reload(sys)
sys.setdefaultencoding('utf8')
client_credentials_manager = SpotifyClientCredentials()
sp = spotipy.Spotify(client_credentials_manager=client_credentials_manager)
if len(sys.argv) > 1:
  uri = sys.argv[1]
else:
  uri = 'spotify:user:bqpd:playlist:5oF8D71X38WwzeRUdyvpmd'
username = uri.split(':')[2]
playlist_id = uri.split(':')[4]
playlist = sp.user_playlist(username, playlist_id)
results = sp.user_playlists(username, limit=50)
playlist_length = playlist['tracks']['total']
path = os.getcwd()
path = path + "/crawled_data"
playlist_name = playlist['name']
directory = path + "/" + playlist_name
if not os.path.exists(directory):
```

```
os.makedirs(directory)
t_start = time.time()
f_feat = open(path + "/" + playlist_name + "/featurevector.txt","w")
f_feat.write("Features: \n")
f_feat.close()
feat_vec = []
feat_num = []
feat_name = []

for num in range(0, playlist_length, 100):
  results = sp.user_playlist_tracks(username, playlist_id, limit=100, offset=int(num))
  tracks = results
  for i, item in enumerate(tracks['items']):
    track = item['track']
    track_id = str(track['id'])
    path = os.getcwd()
    path = path + "/crawled_data"
    artist = str(track['artists'][0]['name'])
    songtitle = str(track['name'])
    artist = artist.replace("/", "")
    songtitle = songtitle.replace("/", "")
    artist = artist.replace("$", "")
    songtitle = songtitle.replace("$", "")
    number = i + num
    name = str(number) + " - " + artist + " - " + songtitle
    directory = path + "/" + playlist_name + "/" + name
    prev_url = track['preview_url']
    if not prev_url == None:
      if not os.path.exists(directory):
        os.makedirs(directory)
      filename = directory + "/" + artist + " - " + songtitle + ".mp3"
      urllib.urlretrieve(prev_url, filename)

      tid = 'spotify:track:' + track['id']

      analysis = sp.audio_analysis(tid)
      with open(directory + "/" + songtitle + '_analysis.json', 'w') as outfile:
        json.dump(analysis, outfile)
```

```
        outfile.close()
        segments = analysis["segments"]
        bars = analysis["bars"]
        beats = analysis["beats"]

        tid = str(tid)

        features = sp.audio_features(tid)
        with open(directory + "/" + songtitle + '_features.json', 'w') as outfile:
          json.dump(features, outfile)
        outfile.close()
        acousticness = features[0]['acousticness']
        danceability = features[0]['danceability']
        energy = features[0]['energy']
        instrumentalness = features[0]['instrumentalness']
        liveness = features[0]['liveness']
        loudness = features[0]['loudness']
        speechiness = features[0]['speechiness']
        valence = features[0]['valence']
        feat_vec.append(scipy.array([acousticness, danceability, instrumentalness,
liveness, loudness, speechiness, valence]))

    else:
        print("no url - entry: " + artist + " - " + songtitle)
        print(track_id + "\n")

t_delta = time.time() - t_start
print ("features retrieved in %.2f seconds" % (t_delta,))
dist = distance.euclidean(feat_vec[0], feat_vec[1])
```

Acknowledgement

Declaration of Authorship

I hereby declare that the thesis submitted is my own unaided work. All direct or indirect sources used are acknowledged as references. This thesis was not previously presented to another examination board and has not been published in German, English or any other language.

Hermsdorf, July 4, 2019

Johannes Schoder