User Adaptive Music Similarity with an Application to Playlist Generation

Diplomarbeit

by

Daniel Gärtner

ITI Waibel
Universität Karlsruhe (TH), Germany
Carnegie Mellon University, Pittsburgh, PA, USA

Advisors:
Dr.-Ing. Thomas Schaaf
Dipl.-Inform. Florian Kraft
Prof. Dr.rer.nat. Alexander Waibel

July 2006

Declaration

Hiermit versichere ich die vorliegende Diplomarbeit
selbständig und ohne unzulässige Hilfsmittel
verfasst zu haben. Alle verwendeten Quellen sind
im Literaturverzeichnis angegeben.

Karlsruhe, den 31.07.2006, ...

Abstract

This thesis is about the generation of playlists for music. The distances between songs are calculated based on spectral shape features. Minimal user feedback (approval or rejection of proposed songs) is used besides song distances to generate playlists. Automatic feature selection is used to better fit the user's needs. There are 4 different contributions to the current state of art:

- The comparison of different distance measures and statistical models in the context of playlist generation.
- A new way of using spectral shape features, extracting them in multiple dimensions and modelling them independently from each other instead of putting all spectral features together and just estimating one probability density function.
- Introduction of additional song selection strategies for playlist generation.
- Adaptive automatic feature selection in the context of playlist generation.

The results of research work indicate that using a likelihood ratio hypothesis test distance using single Gaussians returns results comparable to the results of the more complicated distances operating on Gaussian mixture models. Furthermore, results show that more-sophisticated selection strategies are capable of handling feature spaces including less important features. The quality of the playlists can be further improved by means of user-adaptive automatic feature selection.

Acknowledgments

First of all I would like to thank my advisors, Thomas Schaaf and Florian Kraft. Without Thomas Schaaf's encouragement it would not have been at all possible for me to do research in the very absorbing field of music information retrieval. My discussions with him about my research were the source of many interesting ideas. In the course of compiling my work, Florian Kraft gave me valuable assistance and the support necessary to finish this thesis.

I also wish to warmly thank Alex Waibel and interACT for giving me the opportunity to visit the Carnegie Mellon University in Pittsburgh. My stay in the United States was a great experience which positively influenced my work.

In addition, I would like to thank Roger Dannenberg and Tom Cortina for making it possible for me to participate in the computer music seminar at CMU. The discussions at the seminar proved to be very useful.

Thanks also go to all the people at the interactive systems labs at CMU and the ITI Waibel at Universität Karlsruhe for their helpful discussions and advice.

Further thanks go to George Tzanetakis for making the training data for his genre classification system available to me.

Finally, I want to thank my family for their support. They helped me at all times and made it easier for me to concentrate wholeheartedly on my thesis.

Contents

Chapter 1

Introduction

Every human being is musical. With its melodies, rhythms and timbres music affects mental sensation. Music offers pleasure and relaxation, and enables communication overriding language borders.

This chapter introduces the theme of music information retrieval and explains why it is necessary. This is followed by a literature review section giving an overview of the work that has already been done concerning automatic playlist generation. The chapter ends with a motivation of the used system and a short overview of how the system works.

1.1 Music information retrieval

Music information retrieval is a rather new field of computer science (the basic music information retrieval conference ISMIR [ISM] only exists since 2000). In a few years almost every musical piece that has ever been recorded will be digitally available. This fact, in combination with the triumphal procession of MP3 which has made it possible to distribute music over the internet, means that research on that topic has now become essential. Methods need to be developed that make this large amount of data manageable. Nowadays topics include:

- Transcription of audio files, e.g. writing scores to a given audio piece. This includes extraction of high level features like melody.
- Extraction of other high level features like tempo, rhythm, harmony.
- Sound classification. Extracting, for example, the instrumentation of a classical piece.

- Genre extraction. Genre is an attribute that is correlated to musical taste and often used for automatic playlist generation.
- Music recommendation. Systems able to recommend less-known artists and songs, given some music somebody likes.
- Music retrieval. What is the name of the song and who the performing artist of the music I'm currently listening to in a bar ?

1.2 Motivation and overview of the proposed system

This thesis addresses the generation of music playlists with as little user feedback as possible.

1.2.1 Motivation

Nowadays, MP3 players can store ten thousands of songs and it is almost impossible to manually create playlists without spending more time for playlist creation than for music listening. Generating playlists manually is a very time-consuming process and requires good knowledge of the data collection. Furthermore, playlist generation is made difficult by the limited controls which tiny MP3 players provide. The goal of automatic playlist generation is to generate playlists in a way the user would have done himself, leading to hours and hours of enjoyment without spending much time creating playlists. The decision which songs to play and which songs to miss out is made by the MP3 player itself.

Even if a personal MP3 player were to only contain songs liked by its user, playlist generation would still be essential. There will always be songs that will be rejected by the user, depending, for example, on his current mood. Somebody might not really be interested in listening to "their song" immediately after breaking up their relationship. Furthermore, online services like Napster[1] are starting to offer music flatrate services where members can download as much music as they want to and only pay a fixed amount of money, independent of the number of songs they download. Considering that and the capacity of MP3 players nowadays it is not unusual that MP3 collections on portable devices are very diverse.

[1]http://www.napster.com

To generate playlists, the generation system needs information about the songs. This information can be provided by meta data. As will be soon seen in the literature review, generating playlists using meta data information allows the playlist generation algorithms to incorporate as many attributes as the meta data provides. A major drawback of relying on meta data, however, is the fact that meta data is required whereby the meta data is hard to acquire. Labelling MP3 files is very time-consuming, and tags already included in MP3 files cannot always be trusted. This not only relates to promotional MP3 files provided by unknown local bands. Even commercial meta data databases like Gracenote do not have a perfect genre taxonomy, simply because there is no perfect genre taxonomy. A system has to collect the similarity information itself if it should not rely on meta data.

The system as described in this thesis employs a number of concepts which are similar to the system described in [PPW05a]. The only information needed to generate a playlist is a song, the so called seed-song. The songs included in the playlist should be similar to the chosen song. A skip button is the only control element. If the skip button is pressed, the system stops playing the current song and proposes the next one. The main advantages of a system like this are:

- No need of meta data.
- No sophisticated controls needed, one button is enough.
- No large user interaction needed, the user simply has to press the skip button, which is easy to do while on the move and without the need to change the focus over to the playlist generation process.

1.2.2 Overview

In Figure 1.1, the used components can be seen. Firstly, different spectral features are extracted from the audio signal. Then, statistical models are estimated to represent the distribution of the spectral frames. Based on the models, distances between songs on a certain feature space can be computed. The global distance function is a binary weighted sum of all the different distances. During the playlist generation process, the song selection component uses the global distance function and knowledge about the already classified songs (+rated and -rated) to select the next song that will be played. As soon as this song is played, the user can disagree with the selection by explicitly pressing the skip button, or he agrees implicitly with the choice

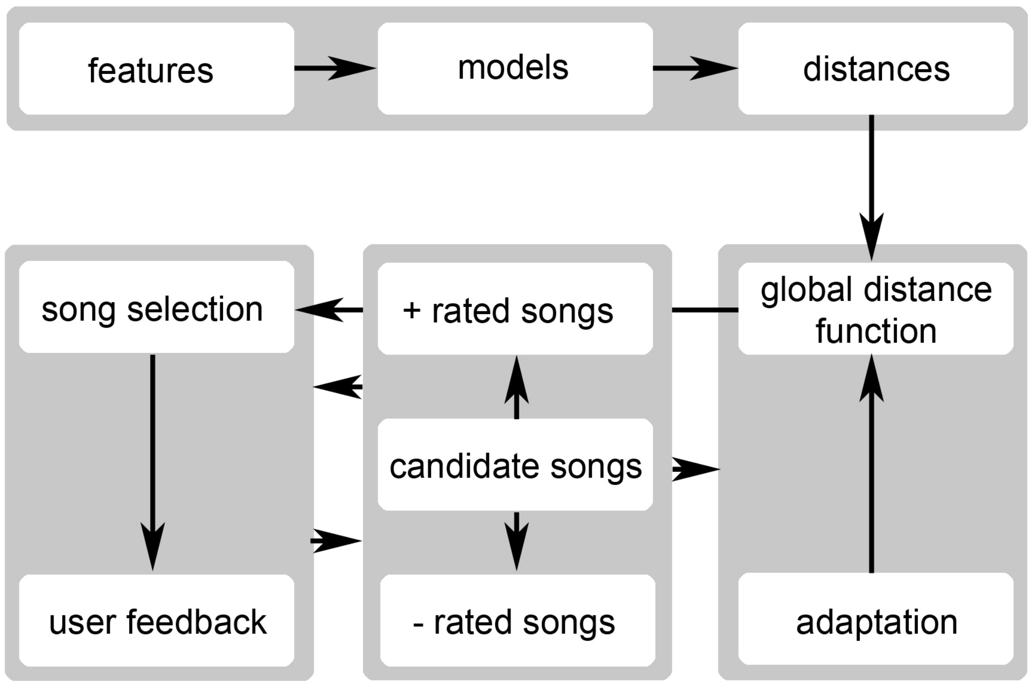

Figure 1.1: The components of the proposed system.

by just listening to the song. Depending on this user feedback, the song is either put from the candidate song set to the +rated song set or to the -rated song set. The next song selection step is then performed with the already updated song sets. Occasionally, an adaptation step is performed. In that adaptation step, automatic feature selection is performed, where the binary feature weighting of the global distance function is updated using an algorithm which was developed in this thesis, analysing the already classified songs. The adaptation step searches for a feature subset that best models the user's musical taste.

1.2.3 Objectives

The objectives of this thesis are to build an adaptive playlist generation system based on audio similarity. Within the scope of this process, acoustic similarity measurements and adaptation techniques are investigated.

Playlist standpoints like variation are not investigated. During the evaluation the user's needs are deliberately assumed to be simple. It is assumed that the user accepts songs from a certain genre and rejects songs from all the other genres, and that the user does not change the genre during the creation of a certain playlist. Time and and memory constraints were not explicitly made.

1.2.4 Structure

The structure of this thesis is as follows:

- Chapter 1: This introduction, including a related work section about playlist generation. Related work about certain components is included in the appropriate chapter.

- Chapter 2: Explanation of the extracted features. It is shown how each feature is computed. Examples are given of which features were used together with which music information retrieval task. Each feature is illustrated for 3 pieces of different genres.

- Chapter 3: Overview of the used statistical models. These are single Gaussians and Gaussian mixture models.

- Chapter 4: Explanation of the used distances. An introduction of how distances between songs which are represented by statistical models can be computed is given. Distances operating on single Gaussians and Gaussian mixture models are explained extensively.

- Chapter 5: The used classification scheme is explained. The algorithms which are used to select the next song are explained (selection strategies). Furthermore, an approach for automatic feature selection is investigated. For each section an introduction including literature review is given.

- Chapter 6: This is the experiments chapter. Models and distances are compared in the context of playlist generation. The selection strategies are compared and the influence of the automatic feature selection is investigated.

- Chapter 7: Conclusions are drawn and an outlook is provided.

1.3 Related work

In this section, an overview about the work already done to-date in playlist generation research is provided. In the second part some products are introduced which are already available.

1.3.1 Research

A distinction is made between two different approaches for playlist generation. The first approach assumes access to large meta data to find similar songs, while the second one relies on the quality of its audio similarity measurement. Since nowadays audio similarity measurements are still far away from the quality of descriptions given by humans, playlist generation based on audio similarity are usually evaluated under only one constraint - their coherency concerning audio similarity. Relying on the meta data attributes, playlists can be generated using numerous constraints. Some of them can be explicitly defined by the user, others are implicitly made. For instance, variation is important to enable a better listening experience over a long time.

Playlist generation for multiple users

[HF01] proposes an algorithm that can be used to generate a playlist for an online radio with several listeners, using collaborative filtering. According to request histories from different users, all artists are rated, predictions for non-requested artists are made for each listener, popular artists among the listening audience are determined, and artists similar to the current playing artists are ascertained. Then a song is chosen which is both popular among the listening audience and similar to the previously played song.

There is a big difference between the described scenario and the scenario from this thesis. In this thesis, the playlist only has to satisfy one user. Nevertheless, there are at least two elements that can be compared.

- User rating. In contrast to the system of this thesis, the user rating is performed in advance. User ratings are calculated from the number of songs that are requested from an artist. In this thesis, the only information that has to be provided is the seed-song.
- Similarity measurement. The publication suggests to use an artist similarity rating, that is gained gained from the song requests. In this thesis, a song similarity measurement is used. This is more reasonable

since although the songs are from the same artist they can nevertheless be very different.

Playlist generation satisfying given constraints

[AT00, AT01] express constraints like "songs which are characterised by fast tempos", "at least 20% male singer", "at most 40% country pop" "one song by Susan Vega", "successive songs should differ at most by two style aspects", "at least 4 titles from universal" and "at least two new tracks" as a minimisation problem in a linear program that can be solved using a branch and bound algorithm.

In [Auc02] a given constraint like "a 10 title playlist" with "all different", increasing tempo, two cardinality constraints on genre (50% folk, 50% rock), genre continuity from item to item and genre distribution (items of the same genre should be as separated as possible from one another) is realised with automatic playlist generation. All the used attributes are given as meta data. Playlist generation is done by introducing a cost function for each constraint that represents how inefficiently the constraint is satisfied. A randomly-generated initial playlist is then optimised by minimising the cost functions.

Users can give a very detailed description of what they want the playlist to be like which is a big benefit of systems that satisfy user constraints. But this is also a major drawback, since that also requires that the user knows what he wants the playlist to be like. This is on one hand difficult to archieve, since musical tastes sometimes can be hard to convert into constraints. On the other hand, it would require more sophisticated controls to enter constraints while in this thesis playlists can be generated by only pressing a button from time to time, having defined a seed-song. Last but not least, meta data is required which can be hard to gain if attributes like tempo or gender of the artist are required which exceed common attributes like artist and genre.

Playlist generation based on meta data similarity

In [PE02] a meta data based similarity measurement is used to generate playlists depending on the similarity to a given song. Distances using different attributes are put together in a normalised weighted sum. During the playlist generation process, the proposed songs are rated by the user to provide information about his compilation strategy. The songs move in a two dimensional Euclidean space and form time dependent clusters. At a given time, the cluster including the given song is presented as a playlist. In a user evaluation, it has been shown that playlists generated in this way both contain more preferred songs and are rated higher by users. It is also reported that the learning of the compilation strategy leads to more preferred songs, but not to increasingly higher rated playlists.

This approach has very much in common with the one described in this thesis. Playlists are generated under the constraint that songs in the playlist should be similar to a seed-song. Furthermore, a user rating is used to learn about the compilation strategy and improve further playlists. In contrast to this thesis, where songs are rated positive or negative immediately, only complete playlists are judged with a score from 0 to 10 in [PE02]. A subtle rating like this requires more attention by the user than only accepting or rejecting a song. Beyond that, immediate feedback can be used immediately for the generation of the further playlist, and in addition every song is scored, not only a set. But of course, meta data provides more information than acoustic measurements probably ever will accomplish, and if meta data has been at one time assiduously collected it can always be trusted.

Playlist generation based on audio data similarity

In [PPW05b] the whole music collection is ordered by an acoustic similarity measurement using a travelling salesman algorithm. An input wheel is provided for scrolling through the whole collection, quickly finding a song that is preferred. From that song on, acoustically similar songs are played.

The system described in this thesis merely requires a single seed-song, which represents a major advantage over all the other approaches. Furthermore, a seed-song can be found quickly, but scrolling through 10,000 songs may also take some time. On the other hand, finding the seed-song is not addressed in any of the publications that require a seed-song. The similarity measure is fixed and the system does not adapt to the user at all, which is a major drawback of this approach. Furthermore, the order of the songs is always the same, which basically means that variation in a playlist only can be achieved by adding files to the collection.

The most important publication for this thesis is [PPW05a], since this thesis employs a number of concepts described in that publication. Given a seed-song, acoustically similar songs are played by the system. Proposed songs can be skipped by the user to signalise the rejection of a chosen song while listening to a song is seen as approval. Besides the "acoustic" distance, the song selection algorithm also incorporates information about the approved and declined songs.

As in the previous approach, the similarity measurement is fixed in this approach, too. This is a drawback, since a similarity measurement can be independent of the criteria the user uses to differentiate between good from bad songs. For example, if a user only wants to listen to slow songs, a timbre similarity measurement is not very useful. This thesis is basically an advancement of that publication. Among other things, it adds an adaptive similarity measurement to the proposed system, which is trying to learn the criteria that the user is using for his decisions.

Results of a user evaluation of a new interactive playlist generation concept merging meta data similarity and audio data similarity are reported in [PvdW05]. The "SatisFly" system allows a user to create a playlist by selecting songs one by one. The user can request additional "similar (audio)" songs based on a reference song and specify additional requirements like the number of songs, the duration of a playlist, the variety in genres, the artists, the albums, the tempo and the period of release of the songs. Song attributes like song title, artist, album, genre, duration, year of release and tempo are given for each song. The algorithm for playlist generation uses constraint propagation, construction, and backtracking. The reported results show that without any decrease in quality of the playlists, they could be created in less time and fewer actions.

This approach combines the diverse information that can be gained from meta data with an audio similarity measurement. Although it cannot be directly compared to the task described in this thesis, the conclusion is promising for this task, too. Similarity measurements which are derived from meta data or acoustic data can help a user to generate playlists of the same quality in less time and with fewer actions. This has been proven with a user test which is a great advantage over all the other approaches that merely use a ground truth that can be derived from meta data.

1.3.2 Products

This is a short overview of some systems already available which provide automatic playlist generation based on music similarity.

Last.fm

Last.fm[2] offers personalised radio streams. If a band or artist is defined, the personalised radio stream plays songs that are supposed to be liked by somebody who likes the seed artist. This information is gathered from audioscrobbler[3], a tool which collects playlist information from millions of users. Using an MP3 player with audioscrobbler support software on a computer, every song played by this software is disclosed to audioscrobbler. Collaborative filtering is used to build profiles by finding relationships between artists. Requesting a radio station with seed artist $\mathcal{A}$, last.fm plays songs that frequently occur in playlists from audioscrobbler users that include $\mathcal{A}$ as well.

Last.fm uses neither meta data (at least only indirectly) nor an acoustic similarity measurement to create playlists. Collaborative filtering is used to create personal similarities between artists. The quality of collaborative filtering techniques strongly depends on the profiles the collaborative filtering algorithm has access to. Since the profiles of Last.fm are based on songs played by a user with a software MP3 player on a computer, there are, for example, more likely to be profiles of computer science students listening to heavy metal than profiles of 70 year old senior citizens listening to folk music.

The Music Genome Project

Since January 2000, a team of about thirty analysts have been listening to music and analysing each of the songs for close to 400 attributes describing melody, harmony, instrumentation, rhythm, vocals, lyrics and more for the Music Genome Project[4]. Given a seed-song or seed artist, Pandora offers an online radio stream playing songs similar to the given seed. Starting with the Beatles's song "Yellow Submarine", pandora firstly recommends the song "Children" by "Family", featuring "mellow rock instrumentation", "folk influences", "acoustic sonority", "major key tonality" and "acoustic rhythm guitar". The next recommendation is "Itsy Bitsy Teenie Weenie Yellow Polka Dot Bikini" from "Brian Hyland", having "mellow rock instrumentation", "acoustic sonority", "major key tonality", acoustic rhythm guitars" and "humorous lyrics" in common with the previously played song. Both Last.fm and the Music Genome Project provide a rating system to judge recommendations provided by the respective systems.

[2]http://www.last.fm/
[3]http://www.audioscrobbler.net/
[4]http://www.pandora.com

A meta data database of that size and in that diversity offers lots of possibilities to playlist generation systems. Since all the attributes for each song have been analysed by experts, they can be trusted to a certain extent. On the other hand, since experts are needed to add songs to the database to ensure consistency, it is unlikely that the database will ever contain, for example, lots of German underground rap songs. From this point of view an audio based similarity measurement cannot be beaten by anything else.

Gracenote playlist / Gracenote playlist plus

Gracenote's products Playlist[5] and Playlist Plus[6] use probably the largest available music meta data database (the former CDDB) to generate playlists which, for example, fit the current mood of the user. Having millions of songs categorised in about 1,500 genres, relationships between songs can be drawn using additional information like the era of the recording artist, the release date or the geographic origin of the music. Another major application using this meta data database retrieves meta data information for a given CD for the labels and tags of MP3 files extracted from the CD in music organisation software like iTunes. Gracenote playlist plus is said to already work on certain portable devices.

The size of the database according to the number of songs is the basic advantage in comparison to the Music Genome Project, which on the other hand has other advantages as it uses a multiple of attributes. For all the introduced products, no information about how the playlists are generated exactly, based on the used similarity measure, is known to the author.

Portable MP3 players

Strategies that are already available on portable devices besides random playlist generation include playing choosing songs randomly from a given artist, a given genre or a given epoch. Furthermore, it is, for example, possible to play the top 25 most frequently played songs.

1.3.3 Conclusion

The general conditions of the proposed systems are rather different, each of them has advantages and drawbacks. Is one user to be satisfied or are many users to be satisfied? How exactly does the user know what he wants to listen

[5]http://www.gracenote.com/gn_products/playlist.html
[6]http://www.gracenote.com/gn_products/playlist_plus.html

to? Is access granted to meta data? Can the meta data be trusted? Are the systems adaptive? If so, do they adapt during the playlist generation process or only when a playlist is finished?

Obviously, the best system would unify access to large meta data, collaborative filtering techniques and millions of diverse profiles, adaptive audio similarity measurements, rating systems, and so on. In addition, components would be required which have the ability to decide which is the best scenario for a certain user.

By contrast, systems that are already available have rather simple playlist generation algorithms implemented so far. There is a large gap between ongoing research and available solutions. It is likely that adding only a few parts of the explained all-in-one solution would make playlist generation systems on portable devices seem much more intelligent.

Chapter 2

Features

In this chapter, an overview is given of the features calculated based on the audio signal data. Features are only used which characterise the spectral shape of the waveform. Features that describe rhythm, key, or melody are not addressed in this thesis. An extensive overview of the features used in music information retrieval can be found in [Poh05, Pee04].

In this thesis three pieces of music are used to visualise all of the features :

- Lara St. John - BWV 1043 II Largo, the first 5 seconds contain a string theme of a solo violin, accompanied by other strings as representative for the classical genre.
- DJ Markitos - Interplanetary Travel, a 5 second excerpt containing synthesised sounds, accompanied by bass, bass drum and hi-hats as representative for the electronic genre.
- Roots of Rebellion - Shift, 5 seconds of heavily distorted guitars with drums as representative for the rock genre.

All of the features are computed based on the power spectrum which can be derived from the audio signal by chopping it in overlapping segments, smoothing the borders with a hamming window, application of an FFT and squaring the coefficients. The imaginary part of these complex coefficients which stores the phase information is disregarded.
Let $\mathcal{A}$ denote song A, from which samples are extracted. 30 secs. from the middle of $\mathcal{A}$ are cut from the 16kHz ADC[1] files. Using 16 msec. Hamming windows with a shift of 10 msecs. leads to 3,000 power spectrum observation-vectors called frames $\mathcal{F}_{\mathcal{A}}^{\mathrm{POW}}(t)$, each with 128 coefficients $\mathcal{F}_{\mathcal{A}}^{\mathrm{POW}}(t,x)$, where x denotes the number of the coefficient.

[1]raw audio files with a linear encoding of 16 bit

2.1 Feature space transformation / LDA

Features that are extracted from the power spectrum can be highly correlated and it is probable that effects which occur in a frequency band also occur in the neighbouring frequency bands. Therefore, not all the coefficients are necessary to separate different classes (songs) from each other. With the help of linear discriminant analysis (LDA, [Fuk90]) one can perform a feature space transformation that puts samples from the same class closer together while samples from different classes are put further apart from each other. In addition, the transformed coefficients are sorted by their importance in the classification task, hence the dimension of the feature space can be reduced by omitting higher coefficients without losing much separability.

Assuming k songs, we have k classes. Let $x_j^{(i)}$ be the samples which are feature frames from class i, n_i the number of samples from class i, μ_i the mean of all samples from class i, n the number of all samples, μ the mean of all samples, W_i the within scatter matrix of class i, and T the total scatter matrix.

$$W_i = \frac{1}{n_i} \sum_{j=1}^{n_i} (x_j^{(i)} - \mu_i)(x_j^{(i)} - \mu_i)^\top$$

$$T = \frac{1}{n} \sum_{j=1}^{n} (x_j - \mu)(x_j - \mu)^\top$$

W is the average within the scatter matrix, the mean of all W_i weighted with their class size n_i.

$$W = \sum_{i=1}^{k} \frac{n_i}{n} W_i$$

The goal is to find a linear transformation A that maximises the total scatter and minimises the average within scatter.

$$\underset{A}{\operatorname{argmax}} = \frac{|T_A|}{|W_A|} = \underset{A}{\operatorname{argmax}} \, |T_A W_A^{-1}|,$$

where T_A is the total scatter matrix and W_A is the average within scatter matrix after applying transformation A. This problem can be solved with simultaneous diagonalisation - searching for a matrix A that simultaneously diagonalises W and T [Fuk90].

$$AWA^\top = I, ATA^\top = \Lambda, \Lambda = (\theta_1 \theta_2 \ldots \theta_d),$$

where θ_i is the i-th eigen value of TW^{-1} and d is the dimensionality of the feature space. The new feature vectors can then be computed with $y = Ax$, and the dimensionality can be reduced by omitting the coefficients of A, those with the smallest eigen values.

Remark: We use an LDA with k classes, where k is the number of songs. For genre classification it would be advisable to perform an LDA with l classes, where l is the number of genres, and compute the within scatter matrices based on the samples of all the tracks from a genre. The resulting subspace would have a dimensionality of $l - 1$. Although the system of this thesis will be evaluated assuming a genre ground truth, transforming the features with a genre class LDA transformation would be cheating since in a real playlist generation process the clustering intention of the user is not known in advance. Nevertheless, applying a genre class LDA transformation to the data and evaluating the system with real users is an interesting investigation.

2.2 Basic features

MFCCs are the most-used feature in music similarity tasks. Their relevance for music modelling has been extensively examined in [Log00]. We use MFCCs and MFLCs respectively as the two basic features. Then we try to add side features to improve the performance.

2.2.1 MFCCs

Mel-frequency cepstral coefficients (MFCCs) are currently the most common features used in speech recognition. The following steps have to be performed to obtain Mel-frequency cepstral coefficients from power spectrum coefficients:

$$\text{power spectrum} \rightarrow \text{melscale} \rightarrow \text{logarithm} \rightarrow \text{cosine transform} \rightarrow \text{MFCCs}$$

1. Melscale
   The mel-frequency scale was originally developed in phonetics to help model the non-linear nature of the human auditory system. A mel filterbank groups power spectrum frequency bins in new bins, following a model based on what we know from human pitch perception. Mel-frequency bins overlap which results in a smoothing of the signal. [Log00] showed that mel cepstral features performed significantly better in a speech / music classification task than linear cepstral features, but they do not answer the question of whether these results are due to a better modelling of only speech or of both, speech and music.

2. Logarithm
   Since humans perceive loudness logarithmically, a logarithm function is applied to the mel scale coefficients.

3. Cosine transformation
   The features obtained after applying a mel filter are highly correlated. A cosine-transform is applied to de-correlate the features, which is an approximation of a principal component analysis [Log00]. Choosing only the first n coefficients reduces the dimensionality of the feature space to n without losing much of the required information - it is basically a smoothing of the spectral envelope. [Pam06] calls this smoothing "a side effect [..] which can be interpreted as a simple approximation of the spectral masking in the human auditory system".
   [AP04a] reported best performance in a timbre similarity task for around the first 20 coefficients. They also say that further coefficients are unwanted, since they are correlated, e.g. with pitch, and in a timbre similarity system, similar timbres with different pitch should nevertheless be matched. [LS01b] reports increase of performance when omitting the first coefficient, which coheres with the signal's energy.

MFCCs are widely used in the field of music information retrieval. [LT03], [TEC01] use MFCCs for musical genre classification, [AHH$^+$03] do fingerprinting with MFCCs, MFCCs are used for music summarisation and segmentation in [TC99] and [LC00] and for music similarity research in [AP04b], [HAE03], [LST04], [AP02], [Foo97], [BLEW03], [Pye00], and [LO04], which also addresses emotion detection. [TTK05] analyse mood-based navigation through musical data with MFCC features.

2.2.2 MFLCs

MFLCs (mel-frequency LDA coefficients) can be similarly derived from the power spectrum as the MFCCs, except the cosine transformation step for de-correlation is replaced with an LDA [EHUL96]:

$$\text{power spectrum} \rightarrow \text{melscale} \rightarrow \text{logarithm} \rightarrow \text{LDA} \rightarrow \text{MFLCs}$$

40 mel scale coefficients are used and LDA-transformed to a 20 dimension feature vector $\mathcal{F}_{\mathcal{A}}^{\text{MFLC}}(t)$. Figure 2.2 shows the three excerpts in MFLCs.

Remark: Properties of MFCCs like the coherency between the first coefficient and the signal's energy or the correlation between pitch and coefficients larger than 20 do not apply for MFLCs.

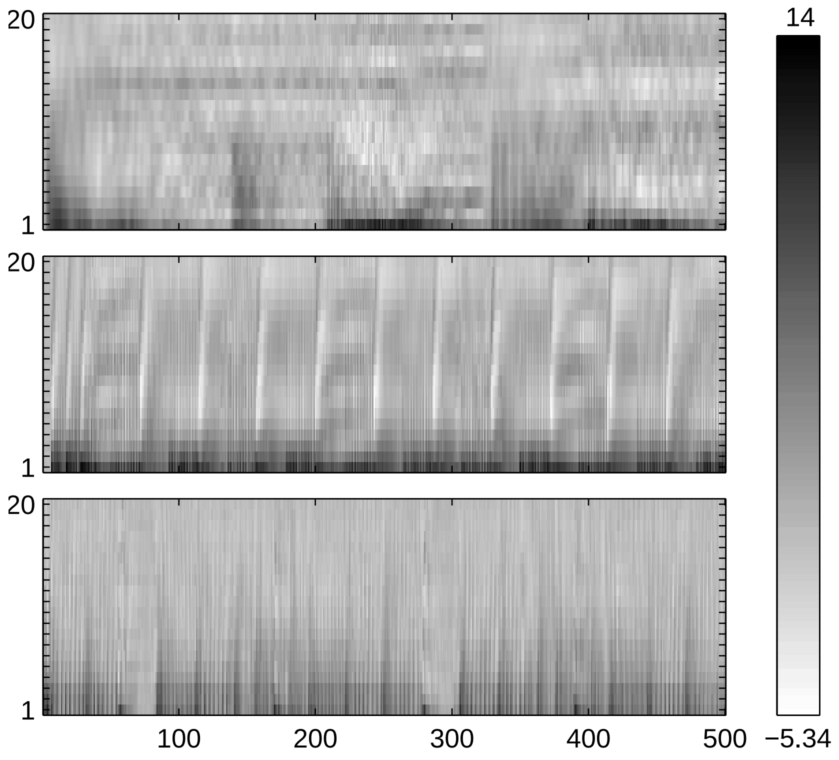

Figure 2.1: MFCCs (x samples, y coefficients). The top illustration shows the frames from the classical piece, the illustration in the middle shows the frames of the electronic piece, and the illustration on the bottom shows the frames of the rock piece. The bar on the right side shows the scale, valid for all 3 pieces.

2.2.3 Basic feature parameters

As already mentioned, a power spectrum is computed on 30 secs. of audio data from the middle of each piece, leading to 3,000 frames with 128 power spectrum coefficients each. A mel-filterbank with 40 bins is applied and all the values are logarithmised. From those 40 coefficients the first 21 MFCCs are extracted and the first one is omitted, leading to a 20-dimensional feature vector $\mathcal{F}_{\mathcal{A}}^{\text{MFCC}}(t)$. MLFCs are also computed from those 40 coefficients. The first 20 MFLC coefficients are taken.

2.3 Side features

In addition to MFCCs/MFLCs, side features are used to improve the performance of the music similarity system. It is a common approach to extract

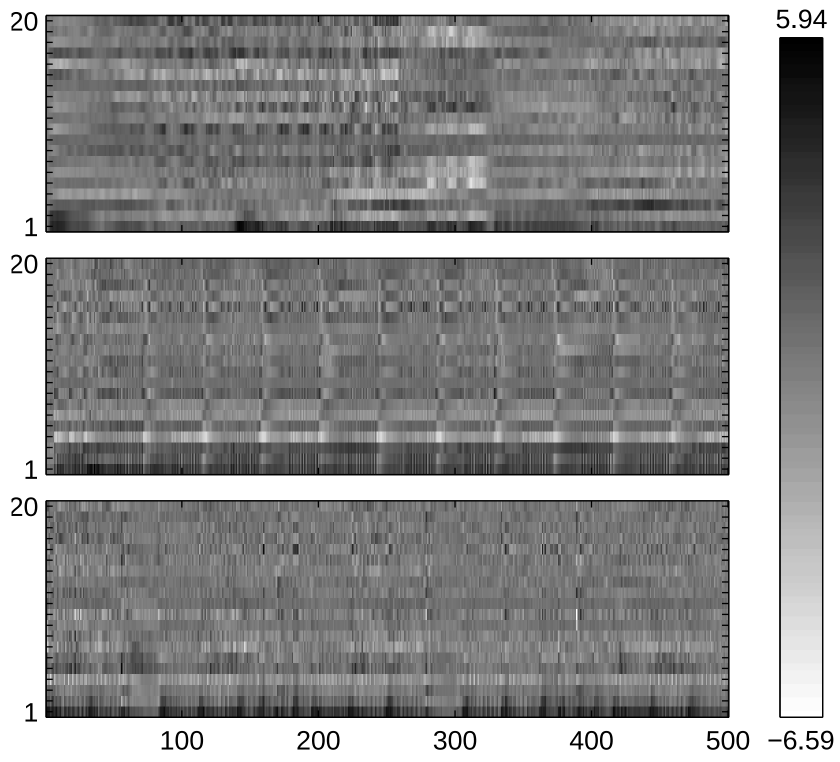

Figure 2.2: MFLCs (x samples, y coefficients).

side features over the whole frequency range, obtaining one coefficient for each side feature and each frame. All the side feature coefficients and the MFCCs are then merged in a "spectral similarity feature". In [HAH01], spectral flatness and spectral crest factor (see the corresponding sections later in this chapter) are extracted in sub-bands "since the desired characteristics [..] are attributed to specific frequency bands rather than the entire spectrum". This thesis investigates the possibility of extracting **all** side features in linearly spaced sub-bands and modelling each of the side features separately with a statistical model. The sub-band n used for one dimension of a side feature is defined by the lowest power spectrum coefficient n_l and the highest power spectrum coefficient n_u, used for this sub-band.

Remark: The used sub-bands, from which side features are extracted, are linearly spaced and non-overlapping. Having 128 power spectrum coefficients, a 16 sub-band side feature would use power spectrum coefficients 1 to 16 for the computation of the first side feature coefficient, power spectrum coefficients 17 to 32 for the computation of the second side feature coefficient, and so on. A promising approach from other publications, motivated by the human auditory system, is to use log-spaced frequency sub-bands for the computation of the side features (e.g. [HAH01, RK]. On the other hand, spectral power coefficients are linearly spaced. Having 128 coefficients for each frame, a very low number of coefficients would be assigned to the lower sub-bands in a logarithmically spaced spectrum. The question of which side features are better computed on linearly spaced bands and which side features are better computed on logarithmic spaced bands requires further investigation.

2.3.1 Side feature dimensionality

The following approach is used to determine the dimensionality of a side feature:

- Each side feature is initially extracted in 16 sub-bands from the 128 power spectrum coefficients.
- Λ, the vector of eigen values of the LDA transformation is computed.
- Determine r, the number of the last eigen value θ_r larger than τ.
- Transform the feature space from 16 dimensions in an r-dimensional space using LDA. The resulting feature will be referred to as the LDA side feature.
- Extract the side feature again from the power spectrum in r sub-bands. The resulting feature will be referred to as the side feature.

Figure 2.3 shows the 16 eigen values of each of the side features, table 2.1 the according θ_r and r. In the experiments chapter (6), the performance of side features and LDA - side features is compared.

Remark: 1.1 has been empirically determined to be a reasonable value for τ. $\tau = 1.1$ has been used for all side features. Table 2.2 shows the eigen values of kurtosis, extracted in 2, 4, 8, and 16 sub-bands. Using our approach in determining the dimensionality of a side feature, initial extraction of kurtosis in 4 or 8 sub-bands would lead to a three dimensional kurtosis instead of only one dimension when extracting it initially in 16 sub-bands. This needs further investigation which has not been performed in this thesis.

Side feature	θ_r	r
Centroid	1.10	10
Bandwidth	1.10	4
Skewness	1.10	5
Kurtosis	1.20	1
Flatness	1.15	4
Crest factor	1.10	4
Shannon entropy	1.11	6
Renyi entropy	1.10	13
Flux	1.36	1

Table 2.1: θ_r and r, the number of sub-bands, a side feature is eventually extracted in.

#sub-bands	θ_1	θ_2	θ_3	θ_4	θ_5
2	1.39	1.10			
4	1.41	1.16	1.13	1.08	
8	1.34	1.17	1.12	1.07	1.06
16	1.2	1.07	1.04	1.04	1.03

Table 2.2: first 5 eigen values θ_i of Kurtosis, extracted in different dimensions.

2.3.2 Central spectral moments

In statistics, the kth central moment of a random variable X is the quantity

$$E[(X - E[X])^k],$$

where E is the expectation operator. The first 4 central spectral moments have been examined, spectral centroid, spectral bandwidth, spectral skewness and spectral kurtosis. They all provide information about the distribution of the coefficients of a spectral sub-band frame.

Spectral centroid

In [Pee04], spectral centroid which is the first central moment of a distribution is defined as:

$$\mathcal{F}_{\mathcal{A}}^{\mathrm{CENT}}(t,n) = \frac{\sum_{k=n_l}^{n_u} (k - n_l + 1) * \mathcal{F}_{\mathcal{A}}^{\mathrm{POW}}(t,k)}{\sum_{k=n_l}^{n_u} \mathcal{F}_{\mathcal{A}}^{\mathrm{POW}}(t,k)}$$

Spectral centroid is the centre of the power spectrum of a sub-band. Centroid is a measure of the brightness of a sub-band. When the greater part of

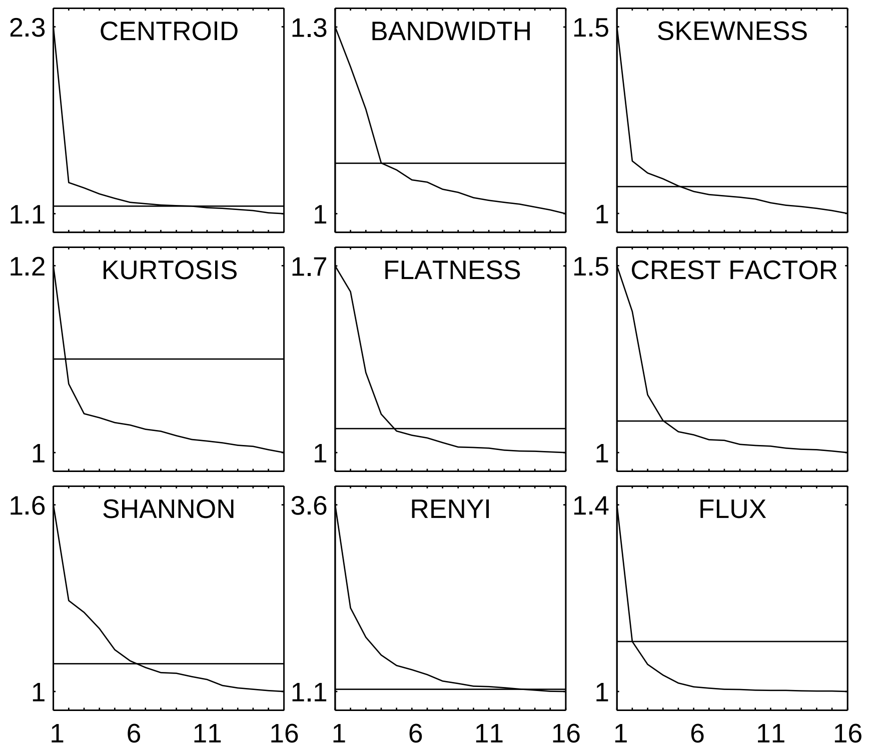

Figure 2.3: The 16 eigen values of each of the side features. The last eigen value before the graph crosses the 1.1 - line is taken as the dimension for this side feature. Renyi entropy crosses this line late, flux and kurtosis cross this line between the first and the second eigen value.

the sub-band energy is located in the lower coefficients of the sub-band, the centroid value is correspondingly lower. Figure 2.4 shows the centroid extracted in 10 sub-bands from our three sample excerpts, Figure 2.5 shows the 10 dimensional LDA centroid. Spectral centroid is used for automatic genre classification in [LT03, TEC01, BL03], for music similarity modelling in [LST04, LO04], for mood-based similarity modelling in [TTK05], for audio fingerprinting in [RK] and for audio segmentation in [TC99].

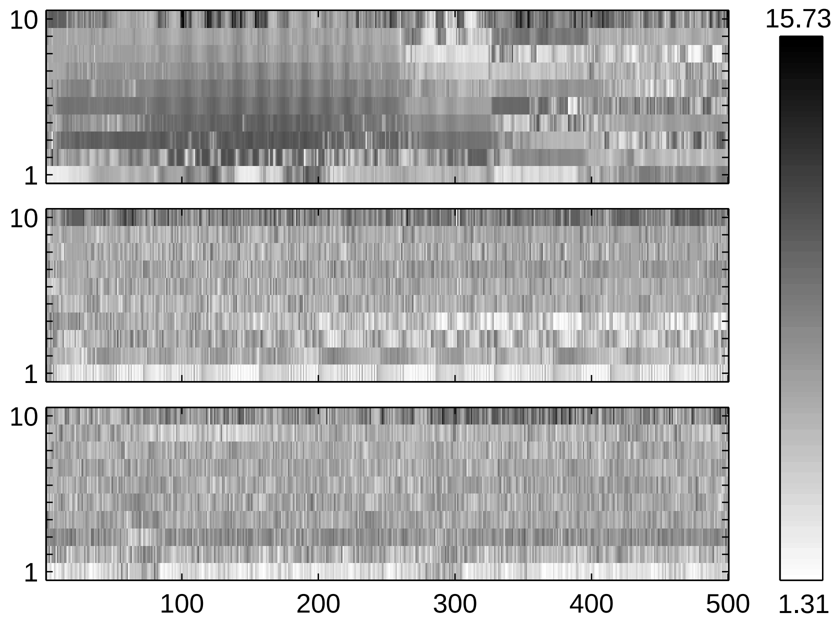

Figure 2.4: Spectral centroid (x time in frames, y coefficients), extracted in 10 sub-bands.

Spectral bandwidth

Spectral bandwidth is the second central moment of a distribution. In [Pee04] it is defined as:

$$\mathcal{F}_{\mathcal{A}}^{\text{BAND}}(t,n) = \frac{\sum_{k=n_l}^{n_u} ((k - n_l + 1) - \mathcal{F}_{\mathcal{A}}^{\text{CENT}}(t,n))^2 * \mathcal{F}_{\mathcal{A}}^{\text{POW}}(t,k)}{\sum_{k=n_l}^{n_u} \mathcal{F}_{\mathcal{A}}^{\text{POW}}(t,k)}$$

Spectral bandwidth provides information about the scatter of the distribution. It is the normalised weighted sum over the squares of the distances between the coefficients and the arithmetic mean of the coefficients. The bandwidth gets larger if the coefficients are scattered to a greater extent, having many coefficients much larger and much smaller than the centroid of the sub-band. Figure 2.6 shows the bandwidth extracted in 4 sub-bands from our three sample excerpts, Figure 2.7 shows the 4 dimensional LDA bandwidth. Spectral bandwidth is used for fingerprinting in [RK]

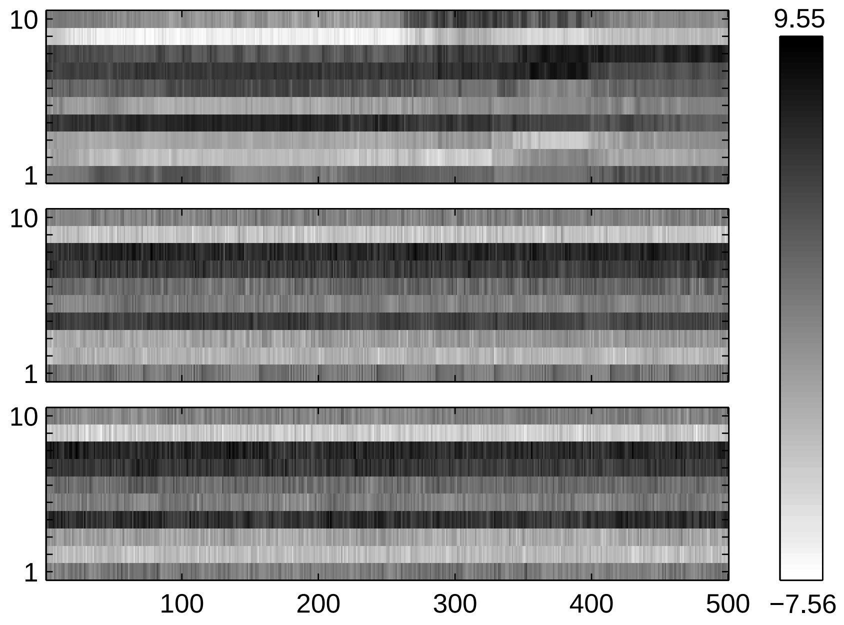

Figure 2.5: Spectral centroid (x time in frames, y coefficients), extracted in 16 sub-bands, LDA transformed to 10 sub-bands.

Spectral skewness

Spectral Skewness is the third central moment of a distribution. Following definition is given in [Pee04]:

$$\mathcal{F}_{\mathcal{A}}^{\mathrm{SKEW}}(t,n) = \frac{\sum_{k=n_l}^{n_u}((k-n_l+1)-\mathcal{F}_{\mathcal{A}}^{\mathrm{CENT}}(t,n))^3 * \mathcal{F}_{\mathcal{A}}^{\mathrm{POW}}(t,k)}{\mathcal{F}_{\mathcal{A}}^{\mathrm{BAND}}(t,n)^{\frac{3}{2}} \sum_{k=n_l}^{n_u} \mathcal{F}_{\mathcal{A}}^{\mathrm{POW}}(t,k)}$$

The skewness gives a measure of the asymmetry of a distribution around its mean value. A skewness of 0 in a sub-band indicates that the bins of the power spectrum's sub-band are symmetrically distributed, whereby a skewness below 0 indicates that the tail of the left side of the distribution is longer than the tail of the right side of the distribution and vice versa. Figure 2.8 shows the skewness extracted in 5 sub-bands from our three sample excerpts, Figure 2.9 shows the 5 dimensional LDA skewness.

Spectral kurtosis

Spectral kurtosis, the fourth central moment is another measure of the distribution of a feature vector sample. A normal distribution has a kurtosis of 3. Distributions with a kurtosis of 3 are called mesokurtic, distributions with

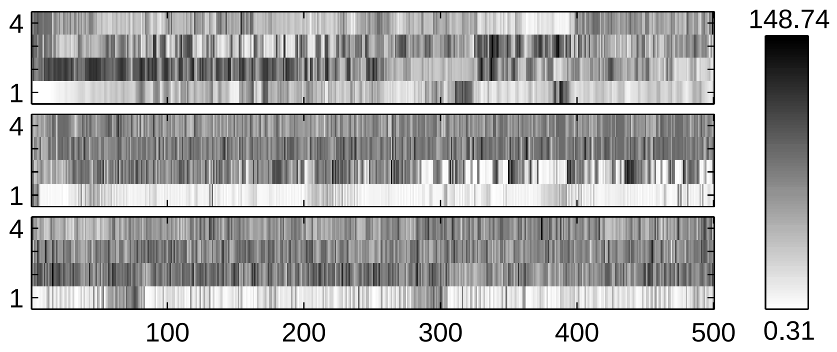

Figure 2.6: Spectral bandwidth (x time in frames, y coefficients), extracted in 4 sub-bands.

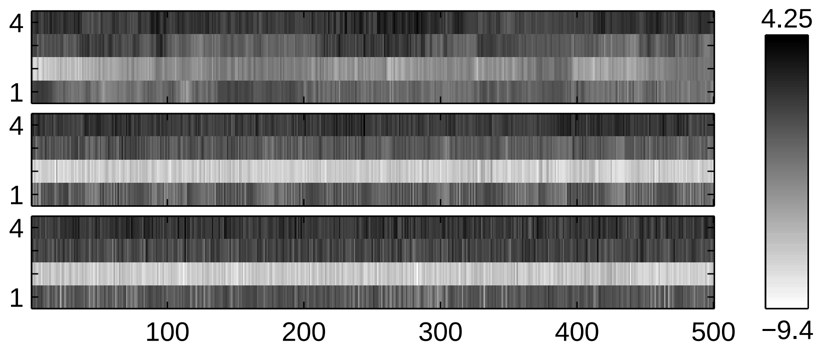

Figure 2.7: Spectral bandwidth (x time in frames, y coefficients), extracted in 16 sub-bands, LDA transformed to 4 sub-bands.

a kurtosis larger then 3 are called leptokurtic, distributions with a kurtosis smaller then 3 are called platykurtic. In [Pee04] the following definition is given:

$$\mathcal{F}_{\mathcal{A}}^{\mathrm{KURT}}(t,n) = \frac{\sum_{k=n_l}^{n_u}((k-n_l+1)-\mathcal{F}_{\mathcal{A}}^{\mathrm{CENT}}(t,n))^4 * \mathcal{F}_{\mathcal{A}}^{\mathrm{POW}}(t,k)}{\mathcal{F}_{\mathcal{A}}^{\mathrm{BAND}}(t,n)^2 \sum_{k=n_l}^{n_u} \mathcal{F}_{\mathcal{A}}^{\mathrm{POW}}(t,k)}$$

In this thesis an offset of -3 is added to all kurtosis values. This is a common principle because the normal distributions then have a kurtosis of 0. A leptokurtic distribution has a more acute peak in the region very close around the centroid of the signal compared to a normal distribution. In addition, such a distribution has ”fat tails” which means that samples far away from the mean are more likely to occur. In contrast to this, platykurtic distributions have thin tails and a relatively flat middle. Compared with a normal distribution, a larger fraction of its observations are clustered within two standard deviations of the mean. Figure 2.10 shows the kurtosis ex-

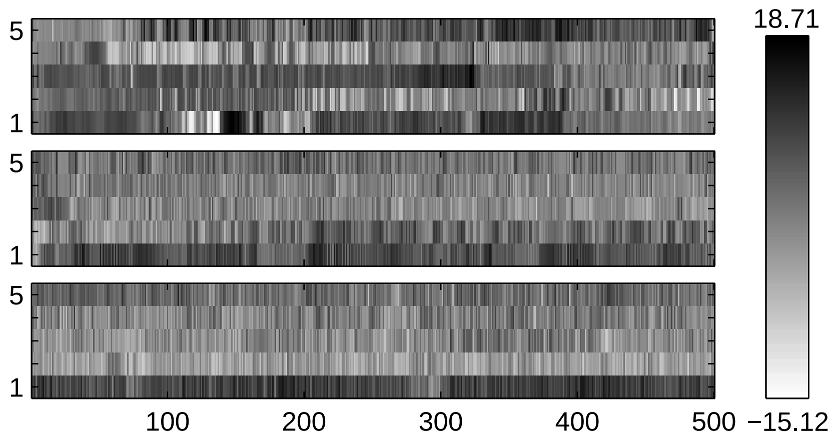

Figure 2.8: Spectral skewness (x time in frames, y coefficients), extracted in 5 sub-bands.

tracted in 1 sub-band from our three sample excerpts, Figure 2.11 shows the 1 dimensional LDA - kurtosis.

2.3.3 MPEG7 low level audio descriptors

Spectral flatness measure and spectral crest factor are part of the MPEG7 low level audio description framework for standardised handling of files with audio and multimedia [mpe].

Spectral flatness measure

$$\mathcal{F}_{\mathcal{A}}^{\mathrm{FLAT}}(t,n) = \frac{\left(\prod_{k=n_l}^{n_u} \mathcal{F}_{\mathcal{A}}^{\mathrm{POW}}(t,k)\right)^{\frac{1}{n_u-n_l+1}}}{\frac{1}{n_u-n_l+1}\sum_{k=n_k}^{n_u} \mathcal{F}_{\mathcal{A}}^{\mathrm{POW}}(t,k)}$$

Spectral flatness [JN84] is the geometric mean divided by the arithmetic mean of a sub-band. A flatness close to 1 means that the frequency components are very similar, which indicates a noise-like sound. By contrast, a flatness close to 0 indicates tonal-like signal. Spectral flatness is used for music similarity modelling in [RK, AHH$^+$01] and for music recommendation in [HJ04]. Figure 2.12 shows the spectral flatness extracted in 4 sub-bands from our three sample excerpts, Figure 2.13 shows the 4 dimensional LDA flatness.

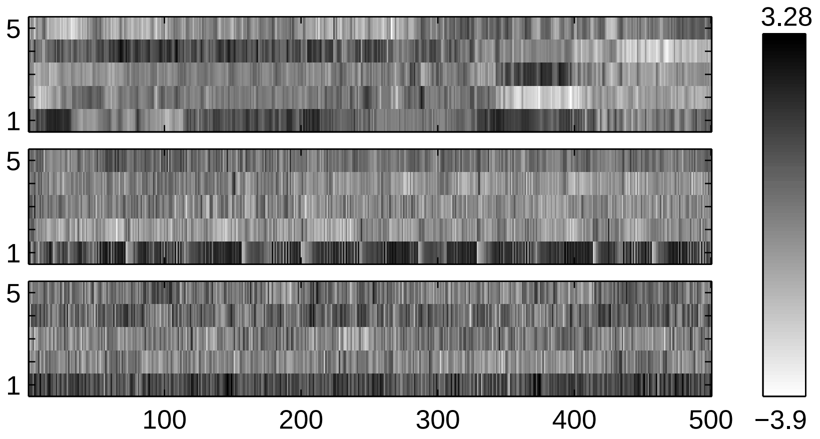

Figure 2.9: Spectral skewness (x time in frames, y coefficients), extracted in 16 sub-bands, LDA transformed to 5 sub-bands.

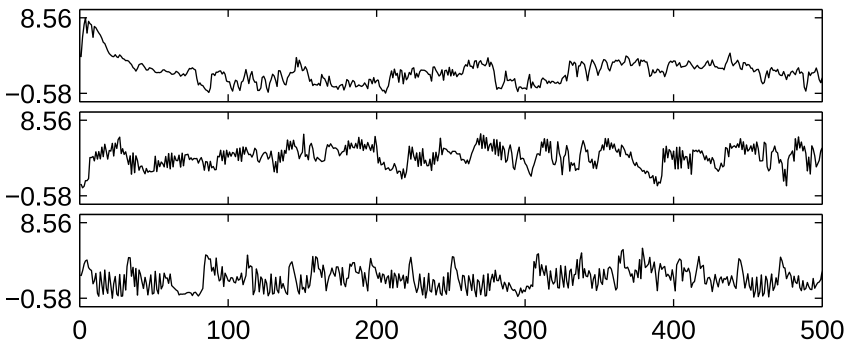

Figure 2.10: Spectral kurtosis (x time in frames, y kurtosis, scaled with $\log(2 + \mathcal{F}_{\mathcal{A}}^{\text{KURT}}(t, n)))$. Values below $\log(2) = 0.69$ indicate a platykurtic distribution of the coefficients in the sub-band frame.

Spectral crest factor

$$\mathcal{F}_{\mathcal{A}}^{\text{CREST}}(t, n) = \frac{\max_k(\mathcal{F}_{\mathcal{A}}^{\text{POW}}(t, k))}{\frac{1}{n_u - n_l + 1}\sum_{k=n_k}^{n_u} \mathcal{F}_{\mathcal{A}}^{\text{POW}}(t, k)}$$

Spectral crest factor (e.g. [HAH01]) is the maximum spectral power in a sub-band divided by the arithmetic mean of the spectral power coefficients in the sub-band. Like spectral flatness, spectral crest factor is also an indication of whether an audio signal is noisy or tonal. Having the same energy in all sub-band coefficients leads to a Crest factor of 1, having almost no energy in all sub-band coefficients but one coefficient with a large power value makes crest factor become large as well. Figure 2.14 shows the spectral crest factor extracted in 4 sub-bands from our three sample excerpts, Figure 2.15 shows

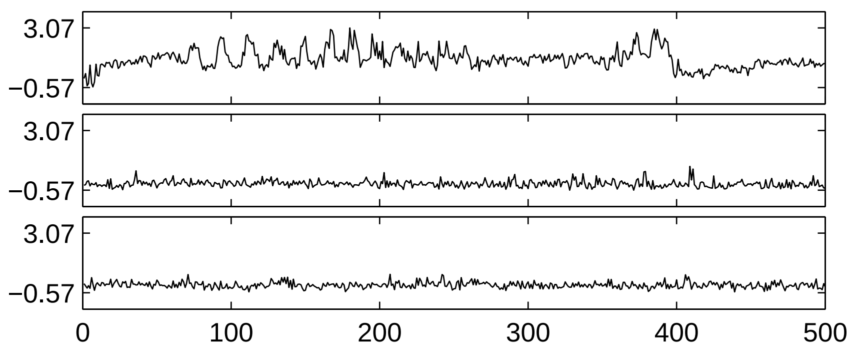

Figure 2.11: Kurtosis (x time in frames, y kurtosis), extracted in 16 sub-bands, LDA transformed to 1 dimension.

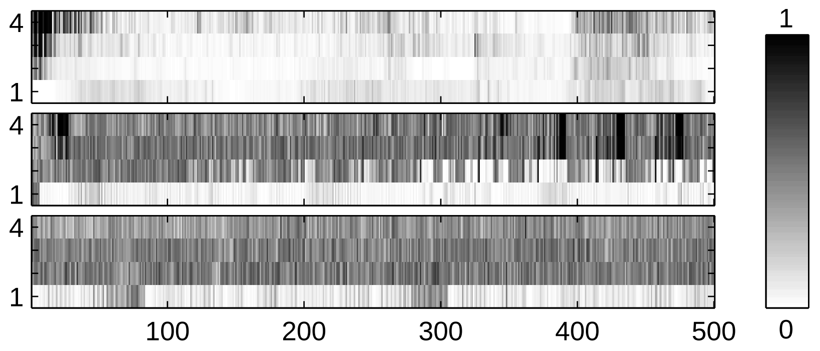

Figure 2.12: Spectral flatness (x time in frames, y coefficients), extracted in 4 sub-bands.

the 4 dimensional LDA crest factor. Spectral crest factor is used for music similarity modelling in [RK, AHH+01]

2.3.4 Entropies

Entropies are a measure of the uncertainty of a distribution.

Shannon entropy

In information theory Shannon entropy is used to estimate the number of bits that are needed to encode a sequence of symbols. In [RK] it is defined as

$$\mathcal{F}_{\mathcal{A}}^{\mathrm{SHAN}}(t,n) = \sum_{k=n_l}^{n_u} \mathcal{F}_{\mathcal{A}}^{\mathrm{POW}}(t,k) * \log_2(\mathcal{F}_{\mathcal{A}}^{\mathrm{POW}}(t,k)).$$

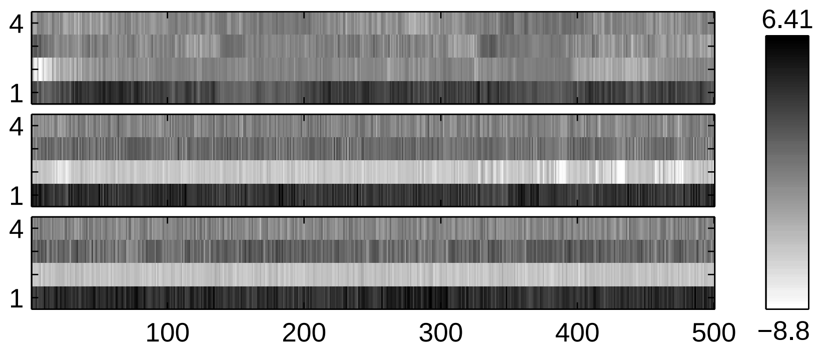

Figure 2.13: Spectral flatness (x time in frames, y coefficients), extracted in 16 sub-bands, LDA transformed to 4 sub-bands.

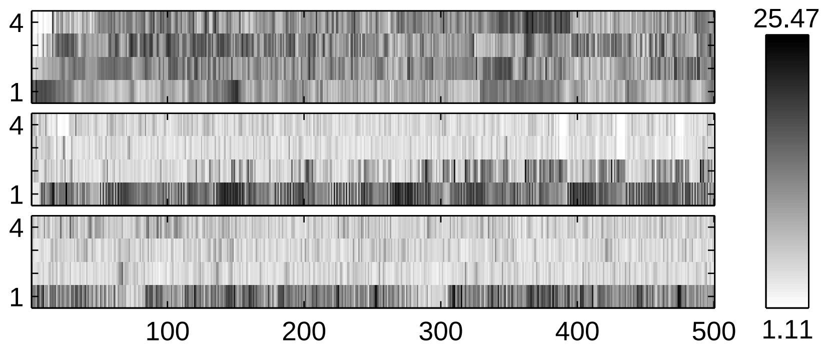

Figure 2.14: Crest factor (x time in frames, y coefficients), extracted in 4 sub-bands.

Figure 2.16 shows the Shannon entropy extracted in 6 sub-bands from our three sample excerpts, Figure 2.17 shows the 6 dimensional LDA Shannon entropy. Shannon Entropy is used for fingerprinting in [RK].

Renyi entropy

The Renyi entropy is a generalisation of the Shannon entropy, defined in [RK] to be

$$\mathcal{F}_{\mathcal{A}}^{\mathrm{REN}}(t,n) = \frac{1}{1-r}\log(\sum_{k=n_l}^{n_u} \mathcal{F}_{\mathcal{A}}^{\mathrm{POW}}(t,k)^r).$$

For $\lim_{r\to 1}$ Renyi entropy is the Shannon entropy. Renyi entropy is used with r=2 [RK], which is also known as correlation entropy. Figure 2.18 shows the Renyi entropy extracted in 13 sub-bands from our three sample excerpts, Figure 2.19 shows the 13 dimensional LDA Renyi entropy. Renyi entropy is

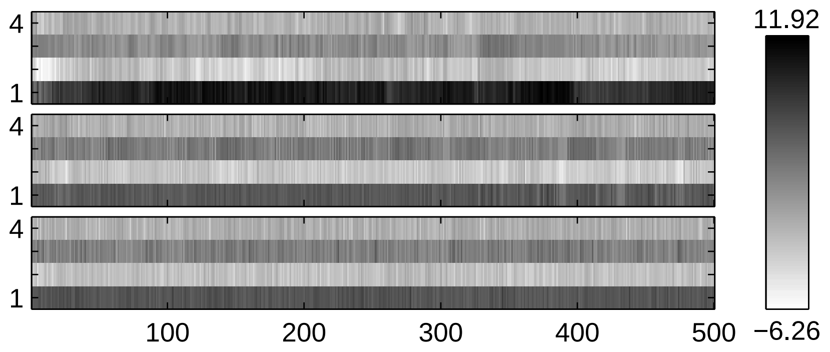

Figure 2.15: Crest factor (x time in frames, y coefficients), extracted in 16 sub-bands, LDA transformed to 4 sub-bands.

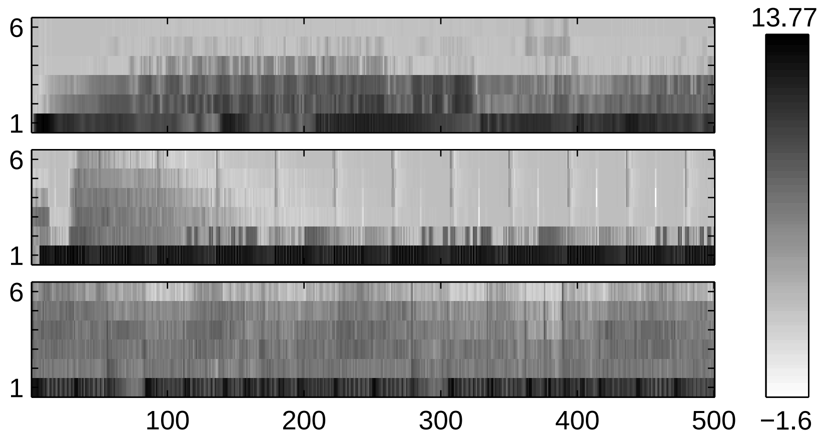

Figure 2.16: Shannon entropy (x time in frames, y coefficients), extracted in 6 sub-bands, scaled with $\log(11 + \mathcal{F}_{\mathcal{A}}^{\text{SHAN}}(t, n))$ for visualisation.

used for fingerprinting in [RK].

Remark: In the original definition of Shannon entropy [SW59], the result of Shannon entropy is multiplicated with -1. Furthermore, probabilities are used instead of power spectrum coefficients, and probabilities of all possible observations sum up to 1. Values of the original entropy can range from 0 to 1. Equally distributed probabilities lead to an entropy of 1 whereas not equally distributed probabilities lead to an entropy smaller than 1. Using the definition of [RK], which is an audio fingerprinting task related to music similarity, a low shannon entropy value for a sub-band frame could be caused by all power spectrum coefficients having similar values (noisy signal), but it is also affected by the signals energy itself. Therefore, the original sense of the entropies cannot be directly transferred. This could be solved by nor-

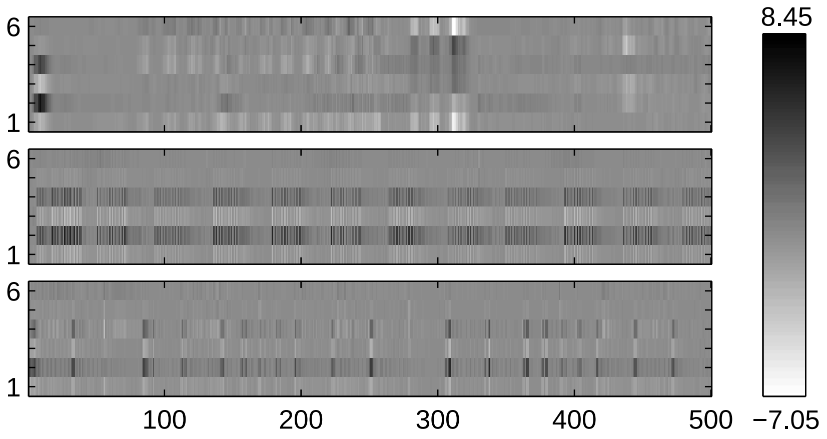

Figure 2.17: Shannon entropy (x time in frames, y coefficients), extracted in 16 sub-bands, LDA transformed to 6 sub-bands.

malising the power spectrum sub-band frames in a way that they sum up to 1.

2.3.5 Spectral flux

The squared difference between two spectral sub-band distributions of successive frames is called spectral flux.

$$\mathcal{F}_{\mathcal{A}}^{\text{FLUX}}(t,n) = \sum_{k=1}^{N} (\mathcal{F}_{\mathcal{A}}^{\text{POW}}(t,k) - \mathcal{F}_{\mathcal{A}}^{\text{POW}}(t-1,k))^2$$

Figure 2.20 shows the spectral flux extracted in 1 sub-band from our three sample excerpts, Figure 2.21 shows the 1 dimensional LDA flux. If two consecutive sub-band frames are similar, the flux value is 0, if they are dissimilar, the flux value is large. Spectral flux is used for genre classification in [TEC01, BL03], for music similarity modelling in [LST04, LO04], for music segmentation in [TC99] and for mood-based music retrieval in [TTK05].

2.4 Summary

This chapter shows how the different features are extracted from the audio signal. The following features are extracted from $\mathcal{A}$:

- Basic features
  - Mel-frequency cepstral coefficients $\mathcal{F}_{\mathcal{A}}^{\text{MFCC}}(t,n)$

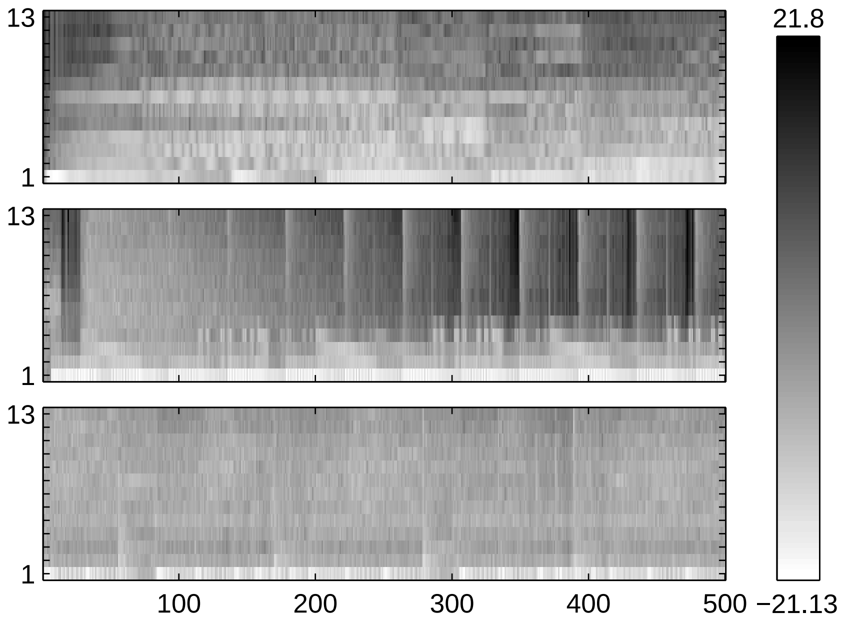

Figure 2.18: Renyi entropy (x time in frames, y coefficients), extracted in 13 sub-bands.

  - Mel-frequency LDA coefficients $\mathcal{F}_{\mathcal{A}}^{\mathrm{MFLC}}(t,n)$

- Side features
  - Spectral centroid $\mathcal{F}_{\mathcal{A}}^{\mathrm{CENT}}(t,n)$
  - Spectral bandwidth $\mathcal{F}_{\mathcal{A}}^{\mathrm{BAND}}(t,n)$
  - Spectral skewness $\mathcal{F}_{\mathcal{A}}^{\mathrm{SKEW}}(t,n)$
  - Spectral kurtosis $\mathcal{F}_{\mathcal{A}}^{\mathrm{KURT}}(t,n)$
  - Spectral flatness $\mathcal{F}_{\mathcal{A}}^{\mathrm{FLAT}}(t,n)$
  - Spectral crest factor $\mathcal{F}_{\mathcal{A}}^{\mathrm{CREST}}(t,n)$
  - Shannon entropy $\mathcal{F}_{\mathcal{A}}^{\mathrm{SHAN}}(t,n)$
  - Renyi entropy $\mathcal{F}_{\mathcal{A}}^{\mathrm{REN}}(t,n)$
  - Spectral flux $\mathcal{F}_{\mathcal{A}}^{\mathrm{FLUX}}(t,n)$

The side features are extracted both correlated and de-correlated with LDA.

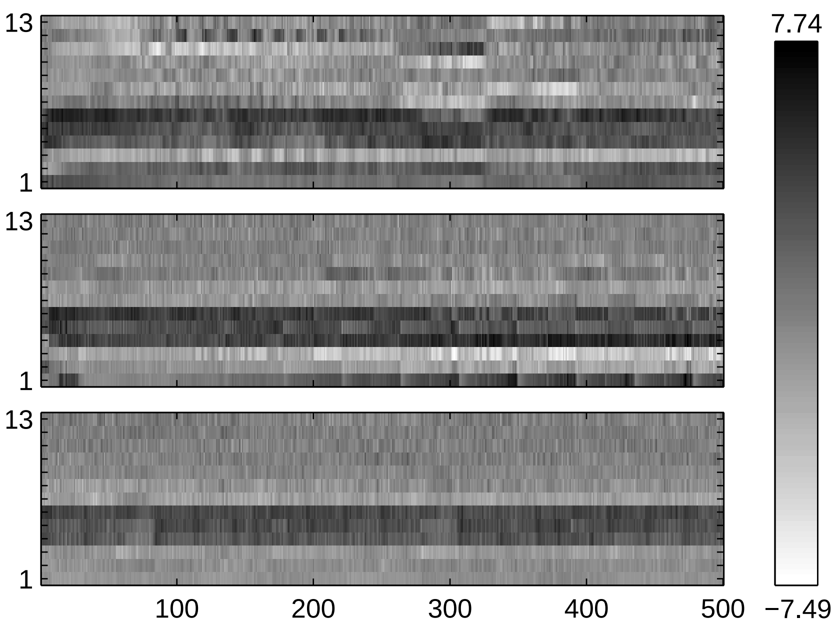

Figure 2.19: Renyi entropy (x time in frames, y coefficients), extracted in 16 sub-bands, LDA transformed in 13 sub-bands.

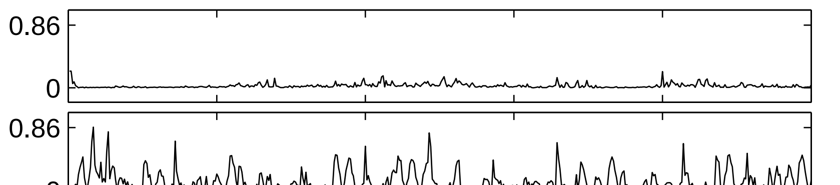

Figure 2.20: Spectral flux (x time in frames, y flux), extracted in 1 sub-band. For the classical piece, flux is moving constantly in the lower quarter. The electronic and the rock piece have more distinctive peaks. In the case of the excerpts, the flux of the rock piece seems to flutter more than the electronic piece. This is something that probably cannot be represented in the statistical models that are used since the ordering of the frames is not taken into account. Additional Δflux coefficients could solve this.

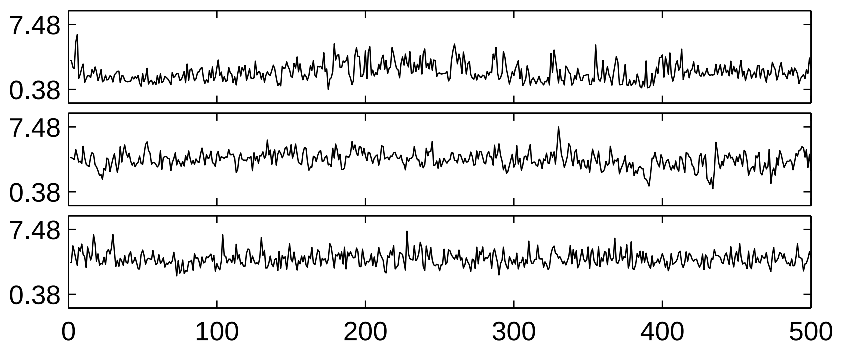

Figure 2.21: Spectral flux (x time in frames, y flux), extracted in 16 sub-bands, LDA transformed in 1 sub-band.

Chapter 3

Models

This chapter addresses the question of how feature vectors can be summarised in a model that can be used to compute distances between pieces of music. Direct comparisson of features of piece $\mathcal{A}$ with features of piece $\mathcal{B}$ would likely fail due to lack of knowledge of which feature frames from piece $\mathcal{A}$ should be compared with which feature frames from piece $\mathcal{B}$. The problem is solved by estimating time-invariant statistical models using the sample frames. The statistical models usually consist of one or several normally distributed clusters. Distances between songs can then be computed by calculating the distances between the estimated statistical models of the two songs.

3.1 Introduction

In this introductory section, several different ways to model distributions are introduced, along with literature references from music information retrieval.

3.1.1 K-means

K-means [HW79] assigns given feature frames in k clusters, represented by their centroids. The initial cluster centroids can be chosen randomly from the given samples. The quality of the clusters is then iteratively improved in two steps :

- Each sample is assigned to the nearest of the centroids.
- After all samples have been assigned to centroids, new centroids are calculated from all the samples that belong to the cluster.

Usually, the number of iterations is a fixed number, provided by experience. A generalisation of k-means is the neural gas algorithm [MS91], where samples can be pro-rated to several clusters. The new centroids are then computed as the weighted mean of the assigned samples. K-means clustering is used, e.g. for music recommendation in [Log04].

3.1.2 Gaussian distributions

Instead of representing clusters only by their means, Gaussian distributions have the capacity to carry information about the correlation of the coefficients. The data is assumed to be normally distributed. The mean and the covariances then can be directly computed from the given samples. In [ME05] Gaussian distributions are used for modelling songs for music classification.

3.1.3 Gaussian mixture models

Single Gaussian distributions have the drawback that they assume only one centroid and that the samples are normally distributed around it. The weighted sum of single Gaussian distributions is called the Gaussian mixture model (GMM). They can be estimated iteratively with the expectation maximisation algorithm. Using a GMM with unlimited number of Gaussians can approximate any meaningful distribution as accurately as is required. GMMs are used, for example, in [PPW05b] for music similarity.

3.1.4 Hidden Markov models

A major drawback of the already introduced models is the lack of modelling time progression. As well as only estimating probability density functions, hidden Markov models (HMMs) involve moving from clusters to different clusters during the song.

In this thesis, the estimation of the distribution of the feature frames of a given song is achieved with a multivariate Gaussian distribution (single Gaussian, SG) or with a GMM, depending on the used distance. HMMs have not been investigated since [AP04a] reports no improvements in a timbre similarity task by replacing static GMMs with dynamic HMMs. However, they provide a useful tool in other music information retrieval tasks, e.g. music segmentation [GML03].

3.2 Gaussian distributions

A common way to estimate a distribution is to assume the data is normally distributed and then estimate a Gaussian distribution. An observation is a d-dimensional vector $x_i = (x_{i1}, x_{i2}, \ldots, x_{id})^\top \in \mathcal{F}_\mathcal{A}(t)$. Given T observations from song $\mathcal{A}$, the distribution parameters can be estimated by:

$$\mu = \frac{1}{T}\sum_{t=1}^{T} x_t$$

and

$$\Sigma = \frac{1}{T}\sum_{t=1}^{T}(x_t - \mu)(x_t - \mu)^\top.$$

μ is the d-dimensional mean of the distribution and Σ is the $d \times d$-dimensional covariance matrix. The probability density function of a normal distribution with mean μ and covariance Σ is given by:

$$N_{\mu,\Sigma} = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} e^{-\frac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)}.$$

A single Gaussian model, being the maximum likelihood estimate of $\mathcal{A}$ with the given frames $\mathcal{F}_\mathcal{A}(t)$ will be denoted by $\Phi_\mathcal{A}$. The order of the sample frames does not have any influence on the model parameters. Characteristics such as attack and decay of a sound which are very important for human perception of sound are not modelled at all. A song played backwards would lead to almost the same probability density function as the song played from front to back.

3.3 Gaussian mixture models

A Gaussian mixture model (GMM) is the weighted sum of single Gaussians. The probability density function of class j, where each class is a song, is defined by :

$$N_j(x) = \sum_{i=1}^{k_j} c_{ji} N_{\mu_{ji}\Sigma_{ji}} = \sum_{i=1}^{k_j} c_{ji} \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} e^{-\frac{1}{2}(x-\mu_{ji})^\top \Sigma_{ji}^{-1}(x-\mu_{ji})},$$

where μ_{ji} is the mean of the i-th Gaussian of class j, Σ_{ji} is the covariance matrix of the i-th Gaussian of class j, k_j is the number of Gaussians of class j, c_{ji} is the distribution weight from Gaussian i in class j, and d the dimension of the feature space. It can be proven that any meaningful distribution can be approximated as accurately as is reqired with a GMM (using many components), but on the other hand, the number of components is obviously limited by the amount of training samples. For GMMs, diagonal covariance matrices have been used in this thesis.

3.3.1 Expectation maximisation algorithm

There is no analytical method to estimate the parameters of a GMM. Using a maximum likelihood estimation to improve the log-likelihood of generating the given training samples with the model would require the information of which samples belong to which Gaussian, and this information is not given. On the other hand, the information of which sample belongs to which Gaussian cannot be acquired without the parameters of the Gaussian distribution. A simultaneous optimisation of Gaussian parameters and training samples to Gaussian assignment is not possible. The expectation maximisation algorithm (EM, e.g. [DLR77]) is an iterative algorithm, where both steps are performed in alternation, always using the information from the previous iteration. There is no guarantee that the EM algorithm converges to the global maximum. EM can also stop in a local maximum. Therefore, the EM algorithm is basically no maximum likelihood estimator.

Expectation step

In the expectation step, the expectation value of an sample frame x_t belonging to class k which is the current model $\Phi_{\mathcal{K}}$ is computed:

$$\gamma_{tk} := E[x_t \in k] = \frac{c_k N_k(x_t)}{\sum_j c_j N_j(x_t)}$$

The c_j and N_j are given by the previous maximisation step.

Maximisation step

In the maximisation step, the parameters of the GMM are updated.

$$\bar{c}_k = \frac{1}{T} \sum_{t=1}^{T} \gamma_{tk}$$

$$\bar{\mu}_k = \frac{1}{\sum_{t=1}^{T} \gamma_{tk}} \sum_{t=1}^{T} \gamma_{tk} x_t$$

$$\bar{\Sigma}_k = \frac{1}{\sum_{t=1}^{T} \gamma_{tk}} \sum_{t=1}^{T} \gamma_{tk}(x_t - \mu_k)(x_t - \mu_k)^\top$$

γ_{tk} is given by the previous expectation step, the model parameters c_k, μ_k and Σ_k are given by the previous maximisation step.

The maximum likelihood estimate given feature frames $\mathcal{F}_\mathcal{A}$ using a GMM is denoted $\Theta_\mathcal{A}$, including the means μ, the variances Σ and the distribution weights c. The set of Gaussians in $\Theta_\mathcal{A}$ is denoted $\Gamma(\Theta_\mathcal{A})$.

There are different ways to train a GMM. One is to initially estimate k clusters with K-means algorithm and then perform some EM - iterations to improve the parameters of the cluster (kMEANS). Another strategy is to start with one cluster and then alternately split clusters into subclusters, do EM-training, and then merge starving clusters. Starving clusters are clusters that do not get assigned enough samples in the expectation step (merge and split, MAS).

3.3.2 Number of Gaussians

MAS and kMEANS training both require the number of Gaussian components in advance. In this work, several values of k have been investigated. K is given and a probability density function is estimated. If the model contains Gaussians with a variance below 1.0E-19 or at least one distribution weight has a value of "not a number" due to numerical complications, the model is retrained with an initial cluster number of k-1. This is repeated until the GMM meets the given requirements.

Remark: There are several other methods to estimate the number of clusters. One that could be used in combination with the kMEANS training is to replace the k-means initialisation step with x-means [PM00], where also the number of clusters is estimated by splitting up certain clusters, using the Bayesian information criterion (BIC) for the splitting decision. BIC could also be used directly in all of the algorithms.

3.3.3 kMEANS training

Algorithm 1 shows how kMEANS training is performed. kMEANS training leads to a slightly different probability density function every time a model is trained, since the initial samples for the k-means initialisation are chosen randomly, and the EM algorithm converges in a local maximum that does not have to be the global maximum.

Algorithm 1 GMM EM-training with k-means initialisation.

```
Choose k samples randomly from training samples
Form k clusters with k-means
Improve cluster parameters with $d$ EM iterations
```

3.3.4 MAS training

Algorithm 2 shows how merge and split training is performed. Starting with one Gaussian component, components are split up into more components meanwhile components can also be merged together (e.g. [KFN98]).

Algorithm 2 GMM merge and split training.

```
Train single Gaussian with all the training samples
for $log_2(k)$ times do
    Split Gaussians
    Improve cluster parameters with $d$ EM iterations
    Merge Gaussians
end for
Improve cluster parameters with $d$ EM iterations
```

In the split step, Gaussians that have been assigned more than $2t$ samples in the previous step are split into two Gaussians, with their means slightly different from the originally mean.
In the merge step, each Gaussians that has been assigned less than t samples in the previous step, is merged with the one of the other Gaussians where the log-likelihood is least compromised.
t is set to 70, d is set to 5.

3.3.5 Problems during modelling

Sometimes the last track of an album contains some minutes of artificial silence between the last track and a hidden track, which is put into the same file. It may transpire that during training one Gaussian cluster is only used for all the frames of this artificial silence. The result is a Gaussian with almost zero variance depending on how equal those frames are. This can lead to numerical problems. To avoid such effects, regions from the power spectrum are removed where the sum over the Euclidean distance of at least 100 neighbouring frames is below 10.

3.4 Summary

In this chapter different methods for statistical modeling of feature frames have been proposed. In the further course of this thesis, the following statistical models will be used (modelling $\mathcal{A}$):

- Single Gaussians, denoted $\Phi_{\mathcal{A}}$
- Gaussian mixture models, denoted $\Theta_{\mathcal{A}}$

Two different training methods for Gaussian mixture models with k Gaussian components are introduced:

- kMEANS, where k initial clusters are formed using the k-means algorithm. These initial clusters are then re-estimated with EM iterations.
- MAS, where one initial cluster is alternately split and merged following certain guidelines.

Chapter 4

Distances

In the features chapter, features were extracted to form multidimensional feature vectors. In the model chapter, probability density functions were estimated based on the feature frames of each song. This chapter is about the question of how distances can be computed between the probability density functions of two songs in the same feature space.

4.1 Introduction

The choice of the distance mainly depends on the underlying model. If each song is represented by only one vector, a simple Euclidean distance might work. For single Gaussians, one might suggest choosing, e.g. relative entropy (Kullback-Leibler divergence) to compute distances between models. For GMMs, there is no closed form of the relative entropy. Therefore, for example, it is necessary to find a mapping between single Gaussian components in order to then compute a distance based on that mapping and the space between the pairs of Gaussian components which the mapping suggests.

4.1.1 Distances between representative vectors

In [LO04], each song is represented by a 35-dimensional feature vector consisting of MFCCs, centroid, flux and some more features, which are computed from a 30 secs piece from each track. Distances between feature vectors, which are mean-subtracted and standard deviation normalised are then computed with the Euclidean distance.

4.1.2 Distances between single Gaussians

Kullback-Leibler divergence

[ME05] extracts MFCCs from songs and models the distribution of a song with a single Gaussian. Then, distances between probability density functions are calculated using Kullback-Leibler divergence. Those distances are then used to carry out artist identification with support vector machines.

Likelihood ratio hypothesis test

[TTK05] use a likelihood ratio hypothesis test to decide whether an observation was rather produced by a certain mood model (happy, melancholic, aggressive) or not. This is used for mood-based navigation through music collections.

4.1.3 Distances between GMMs

Log-likelihood distance

The log-likelihood distance can be used for all kinds of probability density functions, where the log-likelihood that the feature frames from $\mathcal{A}$ were generated by the model from $\mathcal{B}$ is computed. In [AP02], this distance is used in a timbre similarity task. Instead of using the original samples from $\mathcal{A}$, the model from $\mathcal{A}$ is used to generate samples.

kNN distances

In the kNN distances, distances between single Gaussians of two GMMs are known. Using the sum of the distances between each Gaussian $\Phi_A \in \Gamma(\Theta_{\mathcal{A}})$ to the k nearest neighbour Gaussians from $\Theta_{\mathcal{B}}$ is a distance based on a simple mapping between Gaussians of two probability density functions.

Earth mover's distance

A more complex mapping, derrived from the solution of a transportation problem, is used in the earth mover's distance, used in [LS01b] to compute distances between GMMs, trained with MFCC feature frames which are later used to generate playlists.

4.2 Distances on single Gaussians

$\Phi_{\mathcal{A}}$ is given by its mean $\mu_{\mathcal{A}}$ and its covariance matrix $\Sigma_{\mathcal{A}}$. Based on those parameters, distances between single Gaussians can be computed.

4.2.1 Euclidean distance

Considering only the means of two distributions and disregarding the scatter, a reasonable distance is the Euclidean distance:

$$D_{\mathrm{EUCL}}(\Phi_{\mathcal{A}}, \Phi_{\mathcal{B}}) = \sqrt{(\mu_{\mathcal{A}} - \mu_{\mathcal{B}})'(\mu_{\mathcal{A}} - \mu_{\mathcal{B}})}$$

Since we disregard the scatters, very different distributions can still be very close as long as their means are close.

4.2.2 Kullback-Leibler distance

Let p and q be two probability distributions. The Kullback-Leibler divergence between p and q is defined as

$$D(p\|q) = \int_{x=-\infty}^{\infty} p(x) log \frac{p(x)}{q(x)}.$$

It is a measure of the inefficiency of assuming that the distribution is q when the true distribution is p. $D(p\|q) = 0$ if $p = q$. The Kullback-Leibler divergence is not symmetrical. As the distance is required between $\Phi_{\mathcal{A}}$ and $\Phi_{\mathcal{B}}$ be the same than the distance between $\Phi_{\mathcal{B}}$ and $\Phi_{\mathcal{A}}$, we have to define a modified Kullback-Leibler divergence

$$D_{\mathrm{KL}}(\Phi_{\mathcal{A}}, \Phi_{\mathcal{B}}) = D(\Phi_{\mathcal{A}}\|\Phi_{\mathcal{B}}) + D(\Phi_{\mathcal{B}}\|\Phi_{\mathcal{A}}).$$

The modified Kullback-Leibler distance will be called the Kullback-Leibler distance (KL). However, the KL distance is no metric, since the triangle inequality generally fails.

The following equation applies for multivariate Gaussian distributions [Kul59]:

$$D_{\mathrm{KL}}(\Phi_{\mathcal{A}}\|\Phi_{\mathcal{B}}) = \frac{1}{2} log \frac{|\Sigma_B|}{|\Sigma_A|} + \frac{1}{2} tr \Sigma_A(\Sigma_B^{-1} - \Sigma_A^{-1}) + \frac{1}{2} tr \Sigma_B^{-1}(\mu_A - \mu_B)(\mu_A - \mu_B)'$$

If the coefficients are statistically independent and we can use diagonal covariances, the relative entropy can be reduced to an equation which allows a simpler computation of the KL distance:

$$D(\Phi_{\mathcal{A}}\|\Phi_{\mathcal{B}}) + D(\Phi_{\mathcal{B}}\|\Phi_{\mathcal{A}}) = \underbrace{\frac{1}{2}\log\frac{|\Sigma_B|}{|\Sigma_A|} + \frac{1}{2}\log\frac{|\Sigma_A|}{|\Sigma_B|}}_{(1)}$$

$$+\underbrace{\frac{1}{2}tr(\Sigma_A(\Sigma_B^{-1} - \Sigma_A^{-1})) + \frac{1}{2}tr(\Sigma_B(\Sigma_A^{-1} - \Sigma_B^{-1}))}_{(2)}$$

$$+\underbrace{\frac{1}{2}tr(\Sigma_B^{-1}(\mu_1-\mu_2)(\mu_1-\mu_2)') + \frac{1}{2}tr(\Sigma_A^{-1}(\mu_2-\mu_1)(\mu_2-\mu_1)')}_{(3)}$$

1.) $\frac{1}{2}\log\frac{|\Sigma_B|}{|\Sigma_A|} + \frac{1}{2}\log\frac{|\Sigma_A|}{|\Sigma_B|} = \frac{1}{2}(\log|\Sigma_B| - \log|\Sigma_A| + \log|\Sigma_A| - \log|\Sigma_B|) = 0$

2.) $\frac{1}{2}tr\Sigma_A(\Sigma_B^{-1} - \Sigma_A^{-1}) + \frac{1}{2}tr\Sigma_B(\Sigma_A^{-1} - \Sigma_B^{-1})$
$= \frac{1}{2}\sum_{i=1}^{\dim}\sigma_{Ai}^2(\frac{1}{\sigma_{Bi}^2} - \frac{1}{\sigma_{Ai}^2}) + \frac{1}{2}\sum_{i=1}^{\dim}\sigma_{Bi}^2(\frac{1}{\sigma_{Ai}^2} - \frac{1}{\sigma_{Bi}^2}) = \frac{1}{2}\sum_{i=1}^{\dim}(\frac{\sigma_{Ai}^2}{\sigma_{Bi}^2} + \frac{\sigma_{Bi}^2}{\sigma_{Ai}^2} - 2)$

3.) $\frac{1}{2}tr(\Sigma_B^{-1}(\mu_A-\mu_B)(\mu_A-\mu_B)') + \frac{1}{2}tr(\Sigma_A^{-1}(\mu_A-\mu_B)(\mu_A-\mu_B)')$
$= \frac{1}{2}\sum_{i=1}^{\dim}(\frac{1}{\sigma_{Bi}^2}(\mu_{Ai}-\mu_{Bi})^2 + \frac{1}{\sigma_{Ai}^2}(\mu_{Ai}-\mu_{Bi})^2)$
$= \frac{1}{2}\sum_{i=1}^{\dim}((\frac{1}{\sigma_{Bi}^2} + \frac{1}{\sigma_{Ai}^2})(\mu_{Ai}-\mu_{Bi})^2)$,

where dim is the dimension of the feature space. Putting everything together leads to the KL distance for models with diagonal covariance matrices:

$$D_{\mathrm{KL}}^{\mathrm{diag}}(\Phi_{\mathcal{A}},\Phi_{\mathcal{B}}) = D(\Phi_{\mathcal{A}}\|\Phi_{\mathcal{B}}) + D(\Phi_{\mathcal{B}}\|\Phi_{\mathcal{A}}) =$$

$$\frac{1}{2}\sum_{i=1}^{\dim}(\frac{\sigma_{Ai}^2}{\sigma_{Bi}^2} + \frac{\sigma_{Bi}^2}{\sigma_{Ai}^2} - 2 + \frac{(\mu_{Ai}-\mu_{Bi})^2}{\sigma_{Bi}^2} + \frac{(\mu_{Ai}-\mu_{Bi})^2}{\sigma_{Ai}^2}) =$$

$$\frac{1}{2}\sum_{i=1}^{\dim}(\frac{\sigma_{Ai}^2}{\sigma_{Bi}^2} + \frac{\sigma_{Bi}^2}{\sigma_{Ai}^2} - 2 + (\frac{1}{\sigma_{Bi}^2} + \frac{1}{\sigma_{Ai}^2})(\mu_{Ai}-\mu_{Bi})^2),$$

the symmetric Kullback-Leibler distance between two multivariate normal distributions of $\mathcal{A}$ and $\mathcal{B}$, where μ_{Ai}, μ_{Bi} are the mean values of dimension i and $\sigma_{Ai}^2, \sigma_{Bi}^2$ are the variances of dimension i.

4.2.3 Likelihood ratio hypothesis test

[GSR91] used a likelihood ratio hypothesis test (LRHT) to segregate speech from different speakers. The whole audio file is segregated in small segments which are then re-grouped, according to the distances between two neighbouring segments. Using the likelihood ration hypothesis test did not require any prior knowledge of the speakers.
Let $L_A = L(x; \Phi_{\mathcal{A}})$ be the likelihood that feature vector sequence x was generated by $\Phi_{\mathcal{A}}$, and $L_B = L(y; \Phi_{\mathcal{B}})$ be the likelihood that feature vector sequence y was generated by $\Phi_{\mathcal{B}}$. Then the null hypothesis is $L_0 = L(x \cup y; \Phi)$, the likelihood that both feature vector sequences were generated by the same model Φ and the alternate hypothesis is $L_1 = L_A L_B = L(x; \Phi_{\mathcal{A}})L(y; \Phi_{\mathcal{B}})$, the likelihood that both feature vector sequences were generated by different models. In this scenario, where $x = \mathcal{F}_{\mathcal{A}}$, and $y = \mathcal{F}_{\mathcal{B}}$:

$$\lambda = \frac{L_0}{L_1} = \frac{L(\mathcal{F}_{\mathcal{A}} \cup \mathcal{F}_{\mathcal{B}}; \Phi_{\mathcal{A}\cup\mathcal{B}})}{L(\mathcal{F}_{\mathcal{A}}, \Phi_{\mathcal{A}})L(\mathcal{F}_{\mathcal{B}}, \Phi_{\mathcal{B}})}$$

is the likelihood ratio, where $\Phi_{\mathcal{A}\cup\mathcal{B}}$ is the maximum likelihood estimate given the samples $\mathcal{F}_{\mathcal{A}} \cup \mathcal{F}_{\mathcal{B}}$. If we use the probability density functions of the distributions, we obtain that λ can be written as:

$$\lambda = \lambda_\mu \lambda_\Sigma.$$

In this equation, λ_μ is the likelihood ratio that tests the hypothesis that the two segments were generated by the same Gaussian model with the same covariance matrix with no assumption being made about the equality of the means and λ_Σ is the likelihood ratio that tests the hypothesis that the two segments are from the same Gaussian model with the same means disregarding the covariance matrices. [GSR91] only to used λ_Σ which can be expressed as:

$$D_{\text{LR}}(\Phi_{\mathcal{A}}, \Phi_{\mathcal{B}}) = \lambda_\Sigma = \left(\frac{|\Sigma_A|^\alpha |\Sigma_B|^{1-\alpha}}{|\alpha\Sigma_A + (1-\alpha)\Sigma_B|} \right)^N,$$

where N_A and N_B are the number of feature frames used for estimating $\Phi_{\mathcal{A}}$ and $\Phi_{\mathcal{B}}$ respectively, $N = N_A + N_B$, and $\alpha = \frac{N_A}{N}$.

Remark: In this thesis, only λ_Σ was used, according to what [GSR91] suggests, where similar results where obtained for λ_μ and λ_Σ, but λ_Σ is preferred since it is invariant to the time-invariant filter of the speech. It is possible that

$$\lambda_\mu = \left(1 + \frac{N_A N_B}{N^2}(\mu_B - \mu_B)' W^{-1} (\mu_A - \mu_B)\right)^{-\frac{N}{2}}$$

or a combination of λ_Σ or $\lambda\mu$ work better for this music model similarity task. This requires further investigation.

4.3 Distances on Gaussian mixture models

The distances operating on GMMs are introduced in this section. They can be computed using the parameters of a GMM from e.g. $\mathcal{A}$, $\Phi_\mathcal{A}$ which are the means $mu_{\mathcal{A}\rangle}$, the covariance matrices $\Sigma_{\mathcal{A}\rangle}$ and the distribution weights $c_{\mathcal{A}\rangle}$.

4.3.1 Log-likelihood distance

The probability of a given feature frame x generated by $\Theta_\mathcal{B}$ is $L(x;\Theta_\mathcal{B})$, which is equal to $N_\mathcal{B}(x)$, since $\Theta_\mathcal{B}$ is a Gaussian mixture model. The probability of a feature frame sequence $X = (x_1, x_2, \ldots, x_n)$ generated by $\Theta_\mathcal{B}$ is $\prod_{k=1}^n L(x_k;\Theta_\mathcal{B})$. Using the log-likelihood instead leads to:

$$\mathrm{LL}(X;\Theta_\mathcal{B}) = \log\prod_{k=1}^{n} L(x_k;\Theta_\mathcal{B}) = \sum_{k=1}^{n}\log L(x_k;\Theta_\mathcal{B}) = \sum_{k=1}^{n}\log N_\mathcal{B}(x)$$

Having two songs $\mathcal{A}$ and $\mathcal{B}$, feature frames from both songs $\mathcal{F}_\mathcal{A}$ and $\mathcal{F}_\mathcal{B}$, and maximum likelihood estimates $\Theta_\mathcal{A}$ and $\Theta_\mathcal{B}$, a symmetric log-likelihood distance [AP02] between both probability density functions can be expressed with:

$$D_{\mathrm{LL}}(\Theta_\mathcal{A},\Theta_\mathcal{B}) = \mathrm{LL}(\mathcal{F}_\mathcal{A};\Theta_\mathcal{A}) + \mathrm{LL}(\mathcal{F}_\mathcal{B};\Theta_\mathcal{B}) - \mathrm{LL}(\mathcal{F}_\mathcal{A};\Theta_\mathcal{B}) - \mathrm{LL}(\mathcal{F}_\mathcal{B};\Theta_\mathcal{A})$$

The self similarities $\mathrm{LL}(\mathcal{F}_\mathcal{A};\Theta_\mathcal{A})$ and $\mathrm{LL}(\mathcal{F}_\mathcal{B};\Theta_\mathcal{B})$ have been added for normalisation purpose. [AP02] is saying, that this is the most precise and logical way to compute a distance. A major drawback on the other hand is that the feature frames from both songs are needed every time a distance is computed. This requires either an extraction or storing of the frames, which is either very time-consuming or requires a lot more storage compared to only storing the model parameters of the probability density functions. They suggest not using the original frames, but randomly sampling generic frames from the appropriate model.

4.3.2 kNN distances

The distances used for single Gaussians cannot be easily adapted to work with GMMs. This means there is no closed form for the relative entropy between GMMs. The distance between single Gaussian components from $\Theta_\mathcal{A}$

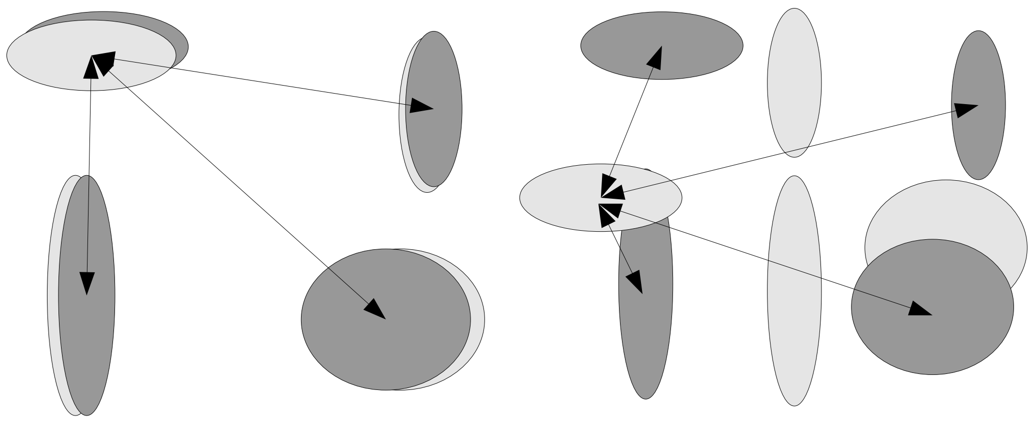

Figure 4.1: The sum of all paired distances between the mean of one light Gaussian component to all the means of the dark Gaussian components is almost the same for both cases with two close distributions on the left and two distant distributions on the right.

and $\Theta_{\mathcal{B}}$ can be computed, using e.g. Euclidean Distance or Kullback-Leibler Distance, what is missing is a mapping between components from $\Theta_{\mathcal{A}}$ to $\Theta_{\mathcal{B}}$.

Let $\Gamma(\Theta_{\mathcal{A}})$ be the set of Gaussians of $\Theta_{\mathcal{A}}$, and $\Phi_x \in \Gamma(\Theta_{\mathcal{B}})$ a certain Gaussian from $\Theta_{\mathcal{B}}$. Then $\text{kNN}(k, \Phi_x, \Gamma(\Theta_{\mathcal{B}}), D)$ denotes the set of the k nearest Gaussians to Φ_x in $\Theta_{\mathcal{B}}$, using distance D, which can be either the Euclidean distance or Kullback-Leibler distance.

A very simple distance is to sum over all paired distances between all $\Phi_x \in \Gamma(\Theta_{\mathcal{A}})$ and $\Phi_y \in \Gamma(\Theta_{\mathcal{B}})$, but cases where the distance between two very close models and two models more apart from each other would yield to comparable distances are easy to generate (Figure 4.1).

Another idea is adding up the distances between each $\Phi_x \in \Gamma(\Theta_{\mathcal{A}})$ to its nearest Gaussian $\Phi_y \in \Gamma(\Theta_{\mathcal{B}})$. A drawback of this approach is, that outlying Gaussians from the destination distribution are unlikely to be considered (Figure 4.2, left). Using a symmetric approach, this could be solved (Figure 4.2, left and right).

$D^{\text{KL}}_{\text{kNN}}(\Theta_{\mathcal{A}}, \Theta_{\mathcal{B}})$ is the kNN distance with Kullback-Leibler distance between

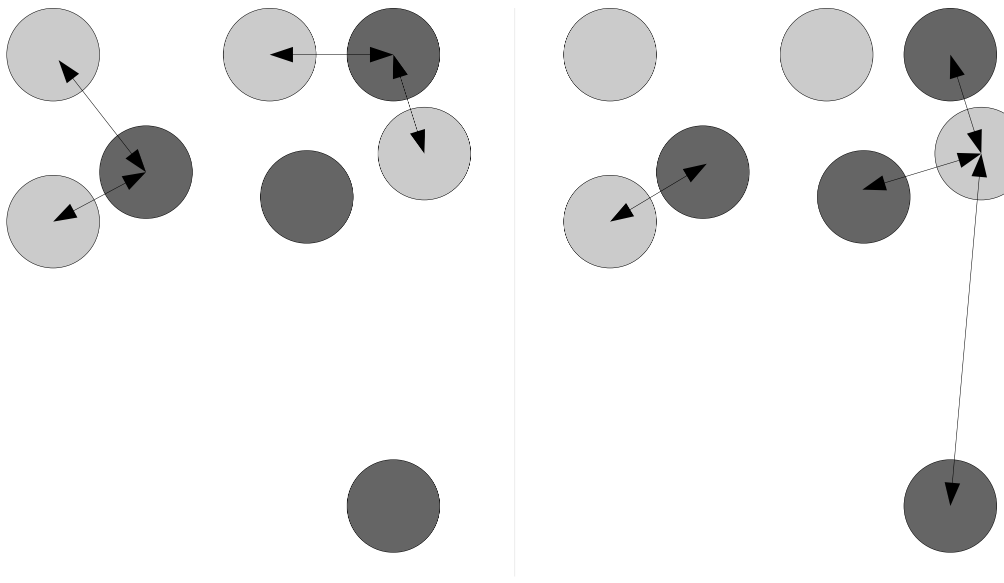

Figure 4.2: The left side shows the distances to the closest neighbour of each light Gaussian, the right side shows the distances to the closest neighbour of each dark Gaussian. Using both, each Gaussian is taken into account at least once.

Gaussian components:

$$D_{\mathrm{kNN}}^{\mathrm{KL}}(\Theta_{\mathcal{A}}, \Theta_{\mathcal{B}}) = \sum_{\Phi_x \in X} \sum_{\Phi_y \in Y^K} D_{\mathrm{KL}}(\Phi_x, \Phi_y) + \sum_{\Phi_y \in Y} \sum_{\Phi_x \in X^K} D_{\mathrm{KL}}(\Phi_y, \Phi_x)$$

and $D_{\mathrm{kNN}}^{\mathrm{EUCL}}(\Theta_{\mathcal{A}}, \Theta_{\mathcal{B}})$ is the kNN distance with Euclidean distance between Gaussian components:

$$D_{\mathrm{kNN}}^{\mathrm{EUCL}}(\Theta_{\mathcal{A}}, \Theta_{\mathcal{B}}) = \sum_{\Phi_x \in X} \sum_{\Phi_y \in Y^K} D_{\mathrm{EUCL}}(\Phi_x, \Phi_y) + \sum_{\Phi_y \in Y} \sum_{\Phi_x \in X^K} D_{\mathrm{EUCL}}(\Phi_y, \Phi_x)$$

where $X = \Gamma(\Theta_{\mathcal{A}})$ are the Gaussians of $\Theta_{\mathcal{A}}$, $Y^K = \mathrm{kNN}(k, \Phi_x, \Gamma(\Theta_{\mathcal{B}}), D)$ are the k nearest Gaussians to Φ_x in $\Theta_{\mathcal{B}}$, $Y = \Gamma(\Theta_{\mathcal{B}})$ are the Gaussians of $\Theta_{\mathcal{B}}$, $X^K = \mathrm{kNN}(k, \Phi_y, \Gamma(\Theta_{\mathcal{A}}), D)$ are the k nearest Gaussians to Φ_y in $\Theta_{\mathcal{A}}$, and D is D_{KL} and D_{EUCL} respectively.

4.3.3 Earth mover's distance

In [RTG98] the Earth Mover's Distance (EMD) is introduced to compute distances between signatures. Applied to our task, computing distances be-

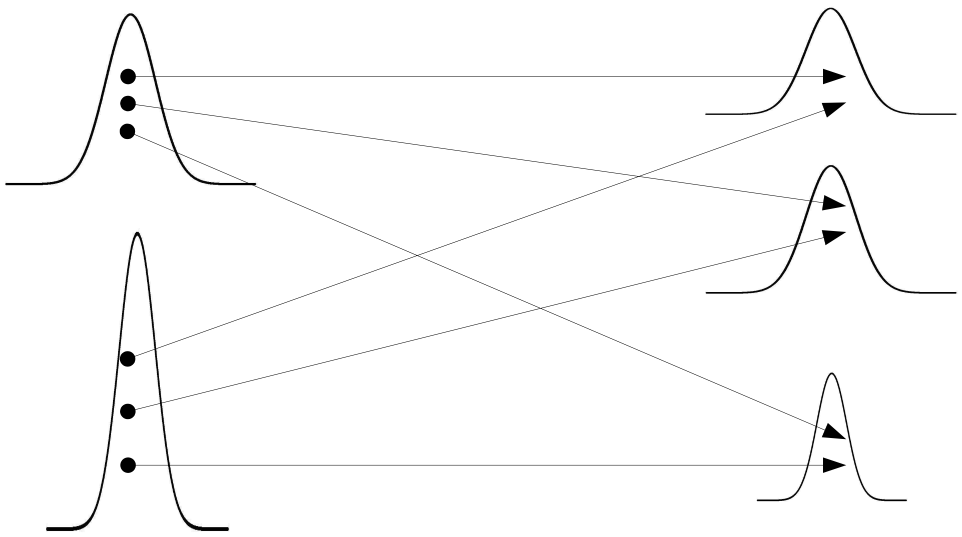

Figure 4.3: Earth mover's distance: earth (probability mass) from the supplying Gaussians has to be moved to the consuming Gaussians.

tween GMMs, the earth mover's distance can be defined the following way:

$$D^p_{\mathrm{EM}}(\Theta_{\mathcal{A}}, \Theta_{\mathcal{B}}) = \sum_{\Phi_x \in X} \sum_{\Phi_y \in Y} D_p(\Phi_x, \Phi_y) F(\Phi_x, \Phi_y),$$

where $X = \Gamma(\Theta_{\mathcal{A}})$, $Y = \Gamma(\Theta_{\mathcal{B}})$, D_p is D_{KL} or D_{EUCL}, and F is the flow from Φ_A to Φ_B. The flows can be gained from the solution of a transportation problem [AMO93], where the suppliers are the Gaussians of $\Theta_{\mathcal{A}}$ with their distribution weights c_{Ai} as supply, the consumers are the Gaussians of $\Theta_{\mathcal{B}}$ with the capacity of their distribution weights c_{Bi}, and the cost of shipping supply from Φ_x to Φ_y is $D_p(\Phi_x, \Phi_y)$.

Remark: In the original description of the earth mover's distance, the distance is normalised by the sum of all flows which is equal to the sum of the capacities of the consumers. Using earth mover's distance for distributions, this sum equals 1. Furthermore, since the overall supply and the overall capacity is 1, the feasibility condition that the total demand never exceeds the total supply is always met.

4.4 Summary

In this chapter, several ways to compute distances between probability density functions from $\mathcal{A}$ and $\mathcal{B}$ are introduced.

- Based on single Gaussians
  - Euclidean distance $D_{\mathrm{EUCL}}(\Phi_{\mathcal{A}}, \Phi_{\mathcal{B}})$
  - Kullback-leibler distance $D_{\mathrm{KL}}(\Phi_{\mathcal{A}}, \Phi_{\mathcal{B}})$
  - Likelihood ratio hypothesis test $D_{\mathrm{LR}}(\Phi_{\mathcal{A}}, \Phi_{\mathcal{B}})$
- Based on Gaussian mixture models
  - Log-likelihood distance $D_{\mathrm{LL}}(\Theta_{\mathcal{A}}\Theta_{\mathcal{B}})$
  - kNN distance $D^{p}_{\mathrm{kNN}}(\Theta_{\mathcal{A}}\Theta_{\mathcal{B}})$
  - Earth mover's distance $D^{p}_{\mathrm{EMD}}(\Theta_{\mathcal{A}}\Theta_{\mathcal{B}})$,

  where p is the distance on the single Gaussians, which can be the Euclidean distance or the Kullback-Leibler distance.

Chapter 5

Song selection and automatic feature selection

In the previous chapters it has been shown how features can be extracted, statistical models can be estimated, and distances on statistical models can be computed. To create a playlist, songs have to be selected. This chapter is about creating playlist, using song selection strategies and a distance function, and about adapting the distance function to the user.

At a certain point during the playlist generation process, the songs in the dataset can be divided into 3 groups.

- Positive rated songs (+rated, $\mathcal{P}$). This group contains all the songs that have been already proposed by the system and the user agreed with the choice by listening to the songs.
- Negative rated songs (-rated, $\mathcal{N}$). This group contains all the songs that have been already proposed by the system and the user disagreed with the choice by skipping the songs.
- Candidate songs $\mathcal{C}$. This group contains all the songs that have not yet been proposed by the system.

The task of the song selection is to choose a song from $\mathcal{C}$. Ideally, the chosen song is accepted by the user, and he enjoys listening to it. So basically, the goal of the song selection is to choose a song that is predicted to be accepted by the user. Predictions can be made by incorporating $\mathcal{P}$ and $\mathcal{N}$.

The described song selection is basically a classification task. The songs in $\mathcal{C}$ have to be divided into 2 classes. Songs the user is supposed to accept and songs the user is supposed to reject. For a classification task, a distance function is needed. Depending on the classification approach, this distance function is explicitly or implicitly used. Besides the song selection and the distance function used for song selection, this chapter is about adapting the distance function to the user. An automatic feature selection algorithm is used to customise the distance function to only work on a feature subset, the one that is supposed to work best for the user.

This chapter is divided into the following parts. In the introduction section, work related to classification in music information retrieval is introduced. The second section is about the used distance function, where distances on different feature spaces like they were introduced in the previous chapter are combined. The used song selection approach is introduced in the third section, the final section is about automatic feature selection.

5.1 Introduction to classification

This chapter offers an introduction to classification approaches used in music information retrieval. For each approach, an example is given of how this machine learning method could be used to perform song selection. Furthermore, the used approach is introduced in more detail.

As already mentioned, $\mathcal{P}$ and $\mathcal{N}$ are used to divide $\mathcal{C}$ into two classes, those songs that are predicted to be accepted by the user and those songs that are predicted to be rejected by the user. From the class of songs that are predicted to be accepted by the user, one song has to be chosen. Since it cannot be assumed that the classification approach perfectly separates the songs into those two classes, it is useful to choose a song with a high prediction to be accepted by the user.

5.1.1 Closest neighbour

In [LO04], the most similar song to a given song is the song which has the smallest distance to the given song. For playlist generation, one could use the closest song to any of the songs from $\mathcal{P}$ to be the next recommendation.

5.1.2 K-nearest neighbours

Having different classes, k-nearest neighbours (kNN) assigns any given $c \in \mathcal{C}$ to the class which most of its k nearest neighbours belong to. In [TC02] kNN is used to perform genre classification. Playlists could be generated by defining two classes $\mathcal{P}$ and $\mathcal{N}$ and then assigning each candidate song to the class which has more nearest neighbours of the k neighbours. One of the candidate songs with the most nearest neighbours in $\mathcal{P}$ could then be recommended.

5.1.3 Distance function and song selection strategies

[PPW05a] uses a distance function and song selection strategies to generate playlists. Knowing $\mathcal{P}$ and $\mathcal{N}$, the song selection strategies compute distances between candidate songs and already classified songs and then recommend one of the candidate songs.

5.1.4 Statistical classifiers

Statistical classifiers perform classification in assigning a given observation x to the class with the highest likelihood. Likelihoods that a given observation x belongs to class j can be gained from the prior probability of the observation $p(x)$, the probability of the observation given the class model $p(x|\Theta_j)$, and the probability of the class $p(\Theta_j)$ with Bayes rule:

$$p(\Theta_j|x) = \frac{p(x|\Theta_j)p(\Theta_j)}{p(x)},$$

where Θ_j is the maximum likelihood estimate for the observations of class j. The class with the highest likelihood is then chosen.

$$\operatorname*{argmax}_j p(\Theta_j|x) = \operatorname*{argmax}_j \frac{p(x|\Theta_j)p(\Theta_j)}{p(x)} = \operatorname*{argmax}_j p(x|\Theta_j)p(\Theta_j)$$

[Pee02] classifies sounds in that way. Playlist generation could be performed by building maximum likelihood estimations $\Theta_{\mathcal{P}}$ and $\Theta_{\mathcal{N}}$, and then choosing a song that has a high likelihood for class $\mathcal{P}$ and a low likelihood for class $\mathcal{N}$.

5.1.5 Linear discriminant analysis

Linear discriminant analysis (LDA) is used to perform a feature space transformation having information about the class affiliation of the feature samples. In the transformed feature space, features from different classes can be better discriminated. This is used in [LT03] for genre classification. Using two classes for playlist generation, the feature space would be reduced to one dimension, and, for instance, the song would be chosen that is close to the songs of $\mathcal{P}$ and far away from the songs of $\mathcal{N}$.

5.1.6 Artificial neural networks

Artificial neural networks (ANN) are able to approximate any non-linear function. [SSWW98] represents each song by one feature vector. An ANN is trained with the vectors from the training data. This ANN is then used to perform genre classification. Having an ANN with a binary output trained with the samples from $\mathcal{P}$ returning 1 and with the samples from $\mathcal{N}$ returning 0, the ANN could be used as a predictor given a song $c \in \mathcal{C}$ returning 1 if the user is supposed to like it or 0 if not.

5.1.7 Support vector machines

Support vector machines (SVM) try to learn the course of a border between two classes in an implicitly transformed feature space while also maximising the distance from all observations to the border. In [SM05], SVMs are used to classify audio files into genres. Playlist generation could be done by training a SVM with the already classified samples from $\mathcal{P}$ and $\mathcal{N}$ and then picking a $c \in \mathcal{C}$ from the right side of the border.

5.1.8 Discussion of the classifiers

The "distance function and song selection strategies" approach is used in this thesis.

Based on a distance function, which will be further explained in 5.2, the song selection strategies compute distances from candidate songs to already classified songs to select one of the candidate songs and add it to the playlist. The song selection strategies are explained in 5.3 in more detail.

The used song selection strategies combine abilities of the closest neighbour classifier and the k-nearest neighbours classifier. It is a rather simple approach, but of course the simplicity strongly depends on the chosen selection strategy. A drawback of some of the other proposed classifiers is that the training of these classifiers is computationally more intensive, and, since user feedback should be incorporated, the training cannot be done in advance. An advantage of the used approach is that this classifier is able to perform reasonable classification with only little training data. If, for example, ANNs or SVMs have been trained once, the classification process is quicker than computing distances between models. But the distances between songs on a certain feature space can be pre-computed and stored somewhere, and then easily accessed when required.

Remark: It is not claimed that the chosen classifier is the best way to perform playlist generation. A comparative study on classifiers for playlist generation would be an interesting investigation. Furthermore, classifiers could be combined. An LDA transformation trained on 2 classes ($\mathcal{P}$ and $\mathcal{N}$) could be used to transform the feature space to a one-dimensional space on which another classifier could then be used.

5.2 The global distance function

Distances between observations are the basic tool in a classification task. For some classifiers like kNN, a distance function is explicitly used, classifiers like ANNs oder SVMs use distances implicitly for their classification task.

The distance function used in this thesis combines the distances between two songs on different feature spaces. Therefore, and in analogy to [Pam06], a weighted distance function has been chosen to combine different features.

Feature extraction, statistical modelling and the computation of all paired distances can be done in advance. Using m features and n songs, this results in m symmetrical $n \times n$ matrices M, each storing all the paired distances on a certain feature space. Since all the distances are known before the actual playlist generation starts, each of the distance matrices can be variance normalised. In variance normalisation, all paired distances on a certain feature space and a certain distance are normalised so that the distribution of all paired distances has zero mean and a variance of 1. Variance normalisation is required since the distances on models operating on different features have different ranges. This is due to the features having different ranges. While flatness is always between 0 and 1, other features, e.g. the used unnormalised Shannon entropy, can reach much larger values. This could be also rectified by normalising the feature ranges, but then the resulting distances would not be so robust for outliers caused by the modelling stage. To summarise, variance normalisation rescales the distributions of the distances in such a way that their mean is 0 and variance is 1, which means that most of the observations should be between -1 and 1.

Using n songs and a $n \times n$ matrix M_q, containing all paired distances, this normalisation is performed in the following way:

$$M'_q(i,j) = \frac{M_q(i,j) - \mu_q}{\sigma_q^2},$$

where q is the current feature space (MFCC, CENT, BAND, SKEW, KURT, FLAT, CREST, SHAN, REN, FLUX), μ_q is the mean of all matrix entries and σ_q^2 is the variance of all matrix entries in this feature space. Let M'_q be the variance normalised distance matrix of feature space q using the best distance for that feature space. All the different distance matrices are now combined by adding them onto each other with binary weights, the final matrix is again variance normalised:

$$M_\omega^G = f(\sum_{q \in Q} \omega(q) M'_q),$$

where $\omega(q)$ is a binary weight that denotes whether feature q is used in the global distance function or not, f is the variance normalisation function, and $\omega \in \Omega$, where Ω is the set of all possible combinations of feature spaces. Using binary weights reduces the weighted distance function to a distance function combining distances of a subset of the complete feature space. Using more than just 0 and 1 as values for the weights would be preferable but could not be realised since the approach to find appropriate weights (see section 5.4) is a brute force approach which would require too much time. Without the second variance normalisation stage, distances on different feature subsets would not be comparable. Having 10 different feature spaces, $|\Omega|$ equals $2^{10} - 1$. The choice of the best ω is performed by the automatic feature selection that will be explained later in 5.4.

5.3 Song selection strategies

The task of the song selection is to select a song r from the candidate songs $\mathcal{C}$. r is the song that is presented next to the user, and is then rated by the user by either being listened to or skipped. The best choice of r is the song with the highest prediction to be accepted by the user.

All the experiments start with a given seed-song s, a song defined by the user whereby the system is supposed to play songs similar to the seed-song.

[LS01a], recommends playing the n closest songs to a given seed-song. This strategy will be used in this thesis and will be denoted by **S0**.

[Log02] uses automatic relevance feedback. The m closest songs to each of the n closest songs to a given seed-song are computed. Each song is scored and then all the scores are combined to create a final playlist. Compared to just playing the n closest songs (S0), it is reported that the performance is slightly worse.

In [Log04] several approaches for finding a song similar to a given song set are compared. They either choose the song that has the lowest average distance to the songs in the song set, the song that has the lowest median distance to the songs in the song set, or the song that has the lowest distance to its closest song in the song set. They report that the average distance performs worst and the choice of the candidate song with its closest song set neighbour performs best. The choice of the candidate song with its closest song set neighbour (where in this thesis the song set will be $\mathcal{P}$) will be investigated in this thesis and be denoted selection strategy **S2**.

[PPW05a] suggests two more song selection strategies. One is to play the song that is the closest neighbour to the song that was accepted last. This one will be referred as **S1** in this thesis. The one that will be referred as **S3** selects the candidate song that has the closest +rated neighbour from a subset of the candidate songs. This subset includes all candidate songs that have a closer +rated neighbour than their closest -rated neighbour. Their closest neighbour among the already classified songs was accepted by the user. If none of the candidate songs satisfies that requirement, the candidate song with the best ratio of the distances to the closest +rated and -rated neighbour is chosen. In [PPW05a], S1 and S3 are further compared with S0 and S2, the order is S0, S1, S2, S3, where S3 performs best.

Besides the selection strategies S0 to S3, two additional types of selection strategies have been developed and evaluated, S4 and the kNN selection strategies.

Several definitions are needed for the explanation of the song selection strategies:

Let $\mathcal{P}$ be the accepted songs and $\mathcal{N}$ be the rejected songs, $\mathcal{C}$ the candidate songs set, and p, n and c be representatives respectively.

s is the given seed-song, r the song that is recommended by the system. $\mathcal{U}(+)$ are the songs the user listens to and $\mathcal{U}(-)$ are the songs the user skips. $\mathcal{U}(+)$ and $\mathcal{U}(-)$ are only used for the evaluation and not known to the system. STOP is the stop condition, that will be $\mathcal{P}$ reaching 20 songs in the experiments later.

1NN$(p, \mathcal{C})$ returns the nearest neighbour in the current feature space (depending on the global distance function) to p in $\mathcal{C}$ and 1NN($\mathcal{P}$,$\mathcal{C}$) returns c from $\mathcal{C}$ that has the nearest neighbour in $\mathcal{P}$.

$d(p, c)$ denotes the distance between p and c. score(p) denotes the score of song p. The distance will always be the global distance function M^G_ω, where ω is either the initial feature weights configuration or the last feature weights configuration determined by the feature selection.

In the initial situation, $\mathcal{P}$ contains only the seed-song s, $\mathcal{N}$ is empty, and $\mathcal{C}$ contains all songs from the database except s. The following song selection strategies are used:

- S0: The nearest neighbours to the seed-song are played.
- S1: The nearest neighbour to the last song that has been accepted is played.
- S2: The nearest neighbour to any of the accepted songs is played.
- S3: The nearest neighbour to any of the songs that have a closer +rated neighbour than their closest -rated neighbour is played.
- S4: The nearest neighbour to the songs with the best score is played.
- kNN: The candidate song with the best ranklist score is played.

5.3.1 Using only the seed-song: S0

Song selection S0 (see Algorithm 3) only uses the global distance function to generate a playlist, and does not involve the already-classified songs in the decision. S0 simply plays songs sorted according to their similarity to the seed-song. When a song is skipped, the system continues playing songs similar to the initial seed-song.

5.3.2 Using the last accepted song: S1

Song selection S1 (see Algorithm 4) for selection of the next song from all the candidate songs in C is easy, too. It proposes the closest neighbour to the last song, that has been classified positive. This selection approach is very vulnerable to songs which are accepted by the user although they are similar to many songs the user dislikes. This songs are called "bad positives".

Algorithm 3 Selection 0

```
play $s$
while !STOP do
    $r \leftarrow$ 1NN$(s, \mathcal{C})$
    play $r$
    if $r \in \mathcal{U}(+)$ then
        $\mathcal{P} \leftarrow \mathcal{P} \cup r$
    else
        $\mathcal{N} \leftarrow \mathcal{N} \cup r$
    end if
    $\mathcal{C} \leftarrow \mathcal{C} \setminus r$
end while
```

Algorithm 4 Selection 1

```
play $s$
$r \leftarrow s$
while !STOP do
    $r \leftarrow$ 1NN$(r, \mathcal{C})$
    play $r$
    if $r \in \mathcal{U}(+)$ then
        $\mathcal{P} \leftarrow \mathcal{P} \cup r$
    else
        $\mathcal{N} \leftarrow \mathcal{N} \cup r$
    end if
    $\mathcal{C} \leftarrow \mathcal{C} \setminus r$
end while
```

5.3.3 Using information from all accepted songs: S2

Selection S2 (see Algorithm 5) is similar to selection S1. The only difference is it does not return the closest neighbour of the last positive example but the closest neighbour to any of the positive examples. This selection is still vulnerable to "bad positives", since $\mathcal{N}$ is not regarded while choosing r. Nevertheless, better performance is expected as not only the last positive song is regarded while choosing r, but $\mathcal{P}$ in complete.

Remark: In later experiments, it is hypothesised that the user does not change his musical preference during a playlist generation process. For users that change their musical preference while they are listening to music recommended by the system, selection S1 could work better than S2.

Algorithm 5 Selection 2

```
play $s$
$r \leftarrow s$
**while** !STOP **do**
  $r \leftarrow$ 1NN($\mathcal{P}, \mathcal{C}$)
  play $r$
  **if** $r \in \mathcal{U}(+)$ **then**
    $\mathcal{P} \leftarrow \mathcal{P} \cup r$
  **else**
    $\mathcal{N} \leftarrow \mathcal{N} \cup r$
  **end if**
  $\mathcal{C} \leftarrow \mathcal{C} \setminus r$
**end while**
```

5.3.4 Using information from all accepted and rejected songs: S3

Selections S1 and S2 only regard positive examples. Selection 3 (see Algorithm 6) regards $\mathcal{P}$ as well as $\mathcal{N}$ while selecting r. It is a two-stage process. In the first stage, those candidate songs with their nearest neighbour in $\mathcal{N}$ are disregarded. From all remaining candidate songs $\mathcal{C}'$ the one with the nearest neighbour in $\mathcal{P}$ is chosen. If $\mathcal{C}' = \emptyset$, from all candidate songs $\mathcal{C}$ the one is chosen that has the best ratio of distance to the nearest positive neighbour and distance to the nearest negative neighbour. This happens more frequently than expected, which is an indication of the deficiency of the distance measurement. Since our distances are variance normalised, negative distances are likely to occur. Before computing the ratios, an offset is added to all distances that are used while ratios are computed. The value of the offset is chosen so that the smallest distance used for ratio computation is 1.

5.3.5 Using the most promising accepted song: S4

In the selection strategies S1 - S3, a song from $\mathcal{P}$ is chosen to lead to the next song which the system then plays. None of those strategies considers the information as to whether a song has led to more positive or negative songs. But this is additional information which the immediate user feedback provides and it might be useful. Selection strategy S4 (see Algorithm 7) uses solely this information. As soon as a song is put into $\mathcal{P}$, an initial score of 0 assigned to it. This score is increased by 1 every time the song leads to another positive song. Every time the songs leads to a negative song the

Algorithm 6 Selection 3

```
play $s$
$r \leftarrow s$
while !STOP do
  $\mathcal{C}' \leftarrow \emptyset$
  for all $c \in \mathcal{C}$ do
    if $d(c, \mathrm{1NN}(\mathcal{P}, \mathcal{C})) < d(c, \mathrm{1NN}(\mathcal{N}, \mathcal{C}))$ then
      $\mathcal{C}' \leftarrow \mathcal{C}' \cup c$
    end if
  end for
  if $\mathcal{C}' \neq \emptyset$ then
    $r \leftarrow \mathrm{1NN}(\mathcal{P}, \mathcal{C}')$
  else
    $r \leftarrow \mathrm{argmin}_{c \in \mathcal{C}} \frac{d(c, \mathrm{1NN}(\mathcal{P}, \mathcal{C}))}{d(c, \mathrm{1NN}(\mathcal{N}, \mathcal{C}))}$
  end if
  play $r$
  if $r \in \mathcal{U}(+)$ then
    $\mathcal{P} \leftarrow \mathcal{P} \cup r$
  else
    $\mathcal{N} \leftarrow \mathcal{N} \cup r$
  end if
  $\mathcal{C} \leftarrow \mathcal{C} \setminus r$
end while
```

score is decreased by 1. In each selection step the nearest candidate song to the song with the highest score is played.

Just regarding how many accepted and rejected suggestions a song from $\mathcal{P}$ leads to, the song with the highest score is the most promising to lead to another accepted song.

5.3.6 Using ranklists with accepted and rejected songs: kNN selections

Unlike strategies S1 to S3, the kNN selections (algorithm 8) not only consider the nearest +/- neighbour for the decision about which song to propose next. The kNN selections take a look at the ranking positions of their k nearest positive or negative neighbours depending on the scoring function sc(x). Another difference between kNN selection and selections S0 to S4 is that S0 to S4 run completely deterministically. In contrast kNN selection chooses one

Algorithm 7 Selection 4

```
play $s$
score($s$) = 0
while !STOP do
    $x = \text{argmax}_{p \in \mathcal{P}}\, \text{score}(p)$
    $r \leftarrow \text{1NN}(x, \mathcal{C})$
    play $r$
    if $r \in \mathcal{U}(+)$ then
        $\mathcal{P} \leftarrow \mathcal{P} \cup r$
        score($x$) = score($x$) + 1
    else
        $\mathcal{N} \leftarrow \mathcal{N} \cup r$
        score($x$) = score($x$) − 1
    end if
    score($r$) = 0
    $\mathcal{C} \leftarrow \mathcal{C} \setminus r$
end while
```

song from a class of songs by random from the class of candidate songs that have the best score which is likely to contain more than one song.

Ranklist$(c, \mathcal{P})$ is a function that returns a list L containing the elements of $\mathcal{P}$ ordered by their distance to c and sc(x) is the score assigned to a given ranklist x. random$(\mathcal{C})$ is a function that picks a random song from the song set $\mathcal{C}$.

The algorithm uses two parameters. One is the function that gives the score of a candidate song, the other one is k which is a parameter of the scoring function. 4 different scoring functions are used:

- knpn : the score function returns the position of the kth positive classified element in L, the k nearest positive neighbour (knpn). An example list $L = (p, n, p, p, n, n, n, n, p, p, n)$ would be scored 4 for $k = 3$ and 9 for $k = 4$.
- knpnS : the only difference between knpnS and knpn is that the sum is added over the positions of the positive neighbours. For instance, list $L = (p, n, p, p, n, n, n, n, p, p, n)$ would be scored 8 for $k = 3$ and 17 for $k = 4$.
- knnn : the score function returns the position of the kth negative classified element in L, the k-nearest negative neighbour (knnn). For the

Algorithm 8 kNN Selections

```
play $s$
$r \leftarrow s$
while !STOP do
  for all $c \in \mathcal{C}$ do
    score($c$) = sc(ranklist($c, \mathcal{P}$))
  end for
  $x \leftarrow \text{argmax}_{c \in \mathcal{C}}\, \text{score}(c)$
  $y = \text{score}(x)$
  $\mathcal{C}' = \emptyset$
  for all $c \in \mathcal{C}$ do
    if score($c$) $= y$ then
      $\mathcal{C}' \leftarrow \mathcal{C}' \cup c$
    end if
  end for
  $r \leftarrow \text{random}(\mathcal{C}')$
  play $r$
  if $r \in \mathcal{U}(+)$ then
    $\mathcal{P} \leftarrow \mathcal{P} \cup r$
  else
    $\mathcal{N} \leftarrow \mathcal{N} \cup r$
  end if
  $\mathcal{C} \leftarrow \mathcal{C} \setminus r$
end while
```

example list $L = (p, n, p, p, n, n, n, n, p, p, n)$ the score would be 6 for $k = 3$ and 7 for $k = 4$.

- knnnS : like for knpn and knpnS, the only difference between knnn and knnnS is that the score from knnnS is the sum over the positions of the k nearest negative neighbours of our candidate song. The scores of $L = (p, n, p, p, n, n, n, n, p, p, n)$ would be 13 for $k = 3$ and 20 for $k = 4$.

For both knpn and knpnS, lower scores are better. For knnn and knnnS, higher scores are better. Therefore, knnn and knnnS scores are inverted. As a result, the best choice is always from the class with the candidate songs that get the lowest score assigned.

Table 5.1 shows an example, where each of the four strategies leads to a different result for $k = 3$. 3npn would choose c_1 or c_2 as next song, 3npnS would chose c_1 as next song, 3nnn would chose c_4 as next song and 3nnnS would chose c_1 or c_4 as next song. In contrast to the sum-strategies, knpn

candidate	$L(c)$	3npn	3npnS	3nnn	3nnnS
c_1	(p,p,n,p,n,n,p)	**4**	**7**	6	**14**
c_2	(p,n,p,p,n,n,p)	**4**	8	6	13
c_3	(p,n,p,n,p,n,p)	5	9	6	12
c_4	(p,p,n,n,p,p,n)	5	8	**7**	**14**

Table 5.1: Comparison of of the 4 strategies for 4 different lists, the best scores are marked bold.

and knnn only regard the position of the kth element, whereas knpnS and knnnS favour those lists where the closer neighbours of the searched kind (positive or negative) are closer, too. knpnS and knnnS return k times the arithmetic mean of the first k positions, knpn and knnn return the median position of the $2k-1$ - closest elements.

knpn and knpnS lead to suggestions r which are close to those songs that have already been classified positive, whereas knnn and knnnS return candidates which are far away from those songs that have already been classified negative.

5.4 Automatic feature selection

So far, distances are computed using the global distance function M^G_ω. This section is about the choice of ω which is assumed to make the global distance function incorporate the subset of the complete feature space that leads to playlists generated with least skips for a user with certain demands to the playlist. ω is chosen automatically. This is done in a brute force approach, where each $\omega \in \Omega$ is scored, incorporating the songs already rated and the ω with the best score is further used.

[LST04] hypothesises "that there might exist certain subsets from the original feature vector that could be more salient for a certain individual as he valuates the perceived similarity of two music pieces". Using musical surface features and the extracted tempo of a song, resulting in a 43 dimensions, M different radial basis function networks (RBFN) are trained, each using a different subset of the feature set. Each RBFN then returns the most similar song to a given song, and the user rates the returned songs. According to the user rating, the RBFN parameters are updated. At the end of the training stage, the RBFN and the corresponding feature subset that exhibited the most effective performance is chosen. It is reported, that this implementation verifies the initial hypothesis and exhibits significant improvement in perceived music similarity. This consolidates the assumtion that the distance function used for playlist generation should be user-adaptive.

[MF04] extracts 109 different features from midi files to perform genre classification. In a multiple stage process, genetic algorithms are used amongst others to reduce the feature space and to find feature weights. Genetic algorithms could be used in this thesis to considerably speed up the automatic feature selection. The resulting ω will not necessarily be the best, but still a good approximation.

[Pee02] uses discriminant analysis to reduce the 81 initial features to 27 and mutual information to reduce the 81 initial features to 20. Applied in a sound classification task, mutual information worked slightly better than discriminant analysis, but they both work slightly worse than using all 81 descriptors. That on the other hand is compensated by a 75% reduction of space dimensionality and an equivalent gain of computation-time. This essentially indicates that it is possible to reduce the feature space dimensionality a lot while the quality of the feature space concerning a certain task stays almost the same. It is assumed that adding features to the feature set which describe song characteristics that are not covered by the other features would lead to better playlists for those of the users who use those characteristics for separating accepted songs from rejected songs. Adding more and more features would make the dimensionality of the feature space eventually too large to be processed in a reasonable amount of time. A reduction of the dimensionality without loosing much of the quality of the features for a certain user therefore might be very useful for playlist generation.

As already mentioned, in this thesis a brute force approach is used to find a subset of features on which the +rated songs and the -rated songs can be separated best. Each feature combination described by $\omega \in \Omega$ is scored, using the already classified songs and the best feature combination is chosen. Three aspects are investigated in an automatic feature selection step, the compactness of $\mathcal{P}$, the compactness of $\mathcal{N}$ and the space between $\mathcal{P}$ and $\mathcal{N}$. The detailed procedure of automatic feature selection can be seen in Algorithm 9 which uses the following variables:

s^+ is the sum over all distances from each positive song p to its closest neighbour in $\mathcal{P}$. Low values indicate that class $\mathcal{P}$ is compact under the distance measurement from the subset which is described by ω.

Algorithm 9 Automatic feature selection

```
for all $\omega \in \Omega$ do
  set global distance function to $M^G_\omega$
  $s^+ \leftarrow 0$
  $s^\pm \leftarrow 0$
  for all $p \in \mathcal{P}$ do
    $s^+ \leftarrow s^+ + d(p, 1\text{NN}(p, \mathcal{P} \setminus p))$
    $s^\pm \leftarrow s^\pm + d(p, 1\text{NN}(p, \mathcal{N}))$
  end for
  $s^- \leftarrow 0$
  $s^\mp \leftarrow 0$
  for all $n \in \mathcal{N}$ do
    $s^- \leftarrow s^- + d(n, 1\text{NN}(n, \mathcal{N} \setminus n))$
    $s^\mp \leftarrow s^\mp + d(n, 1\text{NN}(n, \mathcal{P}))$
  end for
  $\text{score}(\omega) = w_p s^+ + w_n s^- - \frac{w_{pn}}{2}(s^\pm + s^\mp)$
end for
return $\omega$ with smallest score($\omega$)
```

s^- is the sum over all distances from each negative song n to its closest neighbour in $\mathcal{N}$. Low values indicate that class $\mathcal{N}$ is compact under the distance measurement from the subset which is described by ω.

$s^\pm$ is the sum over all distances from each positive song p to its closest neighbour in $\mathcal{N}$, $s^\mp$ is the sum over all distances from each negative song n to its closest neighbour in $\mathcal{P}$. High values indicate that class $\mathcal{N}$ and $\mathcal{P}$ are far apart from each other.

The feature selection is characterised by the feature selection vector

$$< w_p, w_n, w_{pn}, v >,$$

where w_p, w_n and w_{pn} are the adaptation weights and v denotes the number of skips that are needed until the next automatic feature selection step is performed.

The adaptation weights w_p, w_n and w_{pn} are used to weight each of the former mentioned three aspects in the score. For example, only regarding that all songs $\mathcal{P}$ should be close to each other would require $w_p = 1, w_n = 0, w_{pn} = 0$. An intuitive approach would be to ignore the scatter of $\mathcal{N}$. Furthermore, it should then be important that $\mathcal{P}$ has a small scatter. The distance between $\mathcal{P}$ and $\mathcal{N}$ should be large, but with less preference than the small scatter of $\mathcal{P}$. This configuration could be realised with $w_p = 2, w_n = 0, w_{pn} = 1$. The configuration weights are not adapted during playlist generation. They are defined at the beginning of the generation process.

Remark: There are a lot of other possibilities to score a feature combination ω. The one used is related to S3 and it is not hard to think of a scoring approach related to, for example, the kNN selections.

5.5 Summary

This chapter introduces a number of classification methods used in music information retrieval and how each of them could be used for playlist generation. The global distance function that is used to combine the distances on the several feature spaces is introduced. The song selection strategies which are used for selection the song from $\mathcal{C}$ that is played next are explained. Furthermore, automatic feature selection is introduced. In a brute force approach each feature combination is scored and the one with the best score is chosen. Using certain configurations, automatic feature selection is related to S2 ($< 1, 0, 0, v >$) or S3 ($< 1, 0, 1, v >$).

Chapter 6

Experiments

This section provides answers to the following questions:

- 1. What is the influence of different distances on the quality of the playlist? Is it reasonable to use computationally more intensive GMMs and an appropriate distance instead of using computationally less intensive single Gaussians and an appropriate distance?

- 2. Is it possible to increase the performance of the playlist generation system by using more features than just MFCCs or MFLCs, although they basically all describe the spectral shape?

- 3. Can playlist generation solely based on the distance measure be outperformed by incorporating user feedback? Which is the best way to incorporate user feedback?

- 4. Is it reasonable to use a distance measurement that is user-adaptive? Is enough training data for an adaptation provided during a playlist generation process?

To answer point 1, the distances are compared to each other, based on their performance on the basic features. After that, the performance for one basic feature and an additional side feature respectively is evaluated. According to their performance on certain genres, feature subsets are defined. The performance of the feature subsets is compared to the performance of a single basic feature to answer point 2. Thereafter, the different selection strategies are investigated to answer point 3. Automatic feature selection for adapting the distance function to the user is analysed at the end.

6.1 Introduction

In this introductory section, the remaining information that is needed to perform experiments is provided. This is ground truth, what the user's musical taste is expected to be like, the used evaluation scheme and the used measures to rate playlists, the database and the used framework.

6.1.1 Ground truth

For an evaluation of a playlist generation system, an appropriate ground truth has to be defined. User tests were out of the scope of this thesis since their setup and realisation would have been too time-consuming. Therefore, a ground truth has to be created, based on what users are assumed to like or dislike.

The question for an appropriate ground truth is not easy to answer. Music similarity is a very subjective sensation. For instance, songs could be rated as similar because they have the same instrumentation, on the other hand two songs could be rated similar because they both are about love. Ground truth suggestions from other music similarity publications hypothesise

- songs from the same album
- songs from the same artist
- songs from the same genre

or combinations to be similar [LO04, AP04a, LS01a]. In user tests according to music similarity, genre has been shown to be a reasonable ground truth [Pam06, BLEW03]. In [PPW05a], where a similar system is described, also genre information is used. A different ground truth scheme is used in [AHH$^+$03], where the known counterpart of several pieces have to be found respectively in a large set amongst many other pieces. The known counterpart to a piece is known to be similar according to timbre. In [BLEW03], different subjective similarity measures are evaluated. Music similarity information is gained from experts (artist similarity information, based on analysing artists by experts is taken from allmusic[1]), surveys and playlist co-occurrence. For playlists generated under given constraints, like they were described in the introduction chapter (1.3.1), the evaluation criteria is simple - how good were the constraints be realised.

[1]http://www.allmusic.com

Genre information is used in this work, too. It is hypothesised that songs sound similar exactly if they belong to the same genre, although this is not without controversy. In [AP03] it is stated that timbre is a poor genre classifier. They say that "many pop and rock songs use the same instrumentation. This suggests that timbre is not necessarily a good criterion to re-build an arbitrary genre taxonomy". A poor correlation between genre and their timbre similarity measure is reported. So even though a genre ground truth is not perfect for the task of this thesis, using it is nevertheless reasonable.

The task the system of this thesis has to perform is to find songs similar according to their genre information. Furthermore, it is assumed that users do not change the genre they want to listen to while a playlist is created.

6.1.2 Dataset for testing

The used dataset was presented as the training dataset for the genre classification contest of the 5^{th} International Conference on Music Information Retrieval (ISMIR)[2]. It consists of 729 songs taken from magnatune[3], licensed as creative commons. The included genres are distributed the following way (where, for example, CL denotes the set of songs with genre classical):

- classical: 320 songs, cl_1 to $\mathrm{cl}_{320} \in \mathrm{CL}$
- electronic: 115 songs, el_1 to $\mathrm{el}_{115} \in \mathrm{EL}$
- jazz_blues: 26 songs, jb_1 to $\mathrm{jb}_{26} \in \mathrm{JB}$
- metal_punk: 45 songs, mp_1 to $\mathrm{mp}_{45} \in \mathrm{MP}$
- rock_pop: 101 songs, rp_1 to $\mathrm{rp}_{101} \in \mathrm{RP}$
- world: 122 songs, wo_1 to $\mathrm{wo}_{122} \in \mathrm{WO}$

The genres are not equally distributed. A database with equally distributed genres would be preferable, since the a priori probabilities of picking a song of a certain genre would not be genre-dependent. Conclusions about how the used features perform separating songs from a certain genre from all the other songs could be directly drawn. But experiments with a smaller dataset, consisting of 25 randomly chosen songs of each genre did not bring meaningful results with the evaluation measures that will be explained later, since

[2]http://ismir2004.ismir.net/genre_contest/index.htm
[3]http://www.magnatune.com

the evaluation task requires more than 25 songs of a genre to make the evaluation sufficiently insusceptible to noise which is caused by outliers. Since the dataset contains only 26 songs from the jazz_blues genre, the number of songs of each genre in an equally distributed dataset could not be substantially increased. Outliers, which are songs that are dissimilar to other songs from their genre according to the distance measurement, could be caused by, for example, problems during the modelling stage or a timbre dissimilarity.

Only one dataset is used. Even though it would be preferable to evaluate the found parameters on another set with unseen data, this is not necessarily required. Unlike genre classification, no genre models are pre-estimated. The songs rated by the user build the training data. Each evaluation run contains its own training stage implicitly.

6.1.3 Evaluation setup and criteria

To generate a playlist, a seed-song has to be given. Then another 19 songs having the same genre than the seed-songs have to be found, which leads to a playlist with 20 songs which is about the number of songs that fit onto a CD. The system starts recommending songs. A virtual user, having knowledge about the genre of each song either is rating a song positive (which means that a real user would listen to a proposed song) or negative (skipping it), depending on whether the genre of the proposed song matches the genre of the seed-song or not. A single run is finished as soon as the playlist reaches 20 songs that were rated positive by the user (including the seed-song). Meanwhile the number of skips is counted. The performance of generating a playlist given a certain seed-song can be measured with the number of skips that were required to generate it.

Evaluation run

In a complete evaluation run, all the songs from the genres CL, EL, MP, RP and WO are taken as seed-song once. Songs from JB were not taken as seed-songs (but remained in the database), which is justified in 6.2.1. For each seed-song s, a playlist $\mathcal{P}_s$ can be obtained. $\mathcal{S}_s$ denotes the number of skips that are needed to complete $\mathcal{P}_s$ so that it contains 20 +rated songs. For each configuration of the system, two evaluation measures which have been developed for this task can be computed. A complete evaluation run includes the creation of 703 playlists. The two evaluation measures are called average skip measure and median skip measure.

Average skip measure: For each genre, the arithmetic mean of the skips of all playlists generated by taking each song of a genre as seed-song once is computed. The average skip measure (ASM) is the arithmetic mean of those arithmetic means.

$$\text{ASM} := \frac{1}{|\text{genres}|} \sum_{G \in \text{genres}} \left(\frac{1}{|G|} \sum_{k=1}^{|G|} \mathcal{S}_{G_k} \right),$$

where $\mathcal{S}_{G_k}$ denotes the number of skips of the playlist that was generated with the kth song from genre G. This two-stage arithmetic mean is used to make the genres equally important. The genres are not equally distributed and improvements for a highly represented genre should not have a larger influence in the score of a configuration than improvements by the same factor in a smaller represented genre.

Median skip measure: For each genre, the median of the skips of all playlists generated by taking each song of a genre as seed-songs once is computed. The median skip measure (MSM) is the arithmetic mean of those medians.

$$\text{MSM} := \frac{1}{|\text{genres}|} \sum_{G \in \text{genres}} \text{median}(\mathcal{S}_g, g \in G)$$

MSM will be stated sometimes to give an impression how much ASM is influenced by outliers. Decisions are always made as a result of ASM.

ASM specifies how often the skip button has to be pushed on average to reach a playlist including the seed-song and 19 additional desired songs. MSM specifies how often the skip button has to be pushed at most to reach a playlist including those 20 desired songs in half of all attempts. Picking a seed-song and then skipping every second song would lead to 18 or 19 skips, depending on whether the first proposed song is accepted or rejected. An ASM of 16 means that the probability that the user will like the next proposed song is 54%, $\frac{19}{19+16} = 0.54$. A lower ASM (MSM) always indicates a better playlist.

Giving a seed-song and using a song selection strategy that randomly picks songs from the candidate songs would lead to an average ASM of 144. This has been empirically determined by doing this eperiment 100 times for each of the seed-songs.

Remark: The question of how the seed-song is chosen is beyond of the scope of this thesis. To find an eligible seed-song, the approach of [PPW05b]

could be used where the whole music collection is ordered by timbre similarity using a travelling salesman algorithm. Songs can then quickly be retrieved with the help of an input wheel.
The ASM of randomly picking songs from the dataset until the song has the favoured genre is 7.36, which was empirically determined by doing 10,000 runs for each genre. Adding 7.36 to the ASMs that will be reported later in this chapter would be an upper limit for a task where instead of a seed-song only a genre is given. It is assumed that the effective ASM for a task like that will be lower. When a "seed-song" is randomly found, and the actual playlist generation task is starting, there are on average already 6.36 (7.36 minus the one that is rated positive) songs rated negative, which is additional information that can be used by the selection strategies.

6.1.4 Janus Recognition Toolkit

All the work has been done with the help of the Janus Recognition Toolkit (JRTk), an automatic speech recognition toolkit developed at the Interactive System Labs in Karlsruhe, Germany (Universität Karlsruhe) and Pittsburgh, PA, USA (Carnegie Mellon University). It is implemented in C code, with an interface in Tcl/Tk, having an object-oriented look and feel. MFCCs and statistical modelling were already included. The side features and some of the distances (earth mover's distance, log-likelihood distance) have been implemented on C level. The other distances, the song selection strategies and the automatic feature selection have been implemented in Tcl.

6.2 Preliminary experiments

Besides the experiments that are required to answer the questions posed in the introduction of this chapter, there are two more experiments that need to be investigated. Firstly, work is done to analyse whether all genres can be used to generate playlists. Furthermore, an investigation is made to determine whether kMEANS or MAS training should be used to estimate the GMMs.

6.2.1 Choice of genres

As already mentioned, the database contains 6 different genres. No playlists have been generated with seed-songs from the jazz_blues genre. Every genre can contain some songs that are dissimilar to the other songs from the genre. Thus, having only 26 songs from a genre, picking one as a seed-song and then allowing the system find the additional 19 of the remaining 25 is a very demanding task if, for example, there are more than 6 outliers among those 25 remaining tracks. It is likely that the system will recommend many songs of the 703 songs from other genres before picking one of the outliers "by chance".

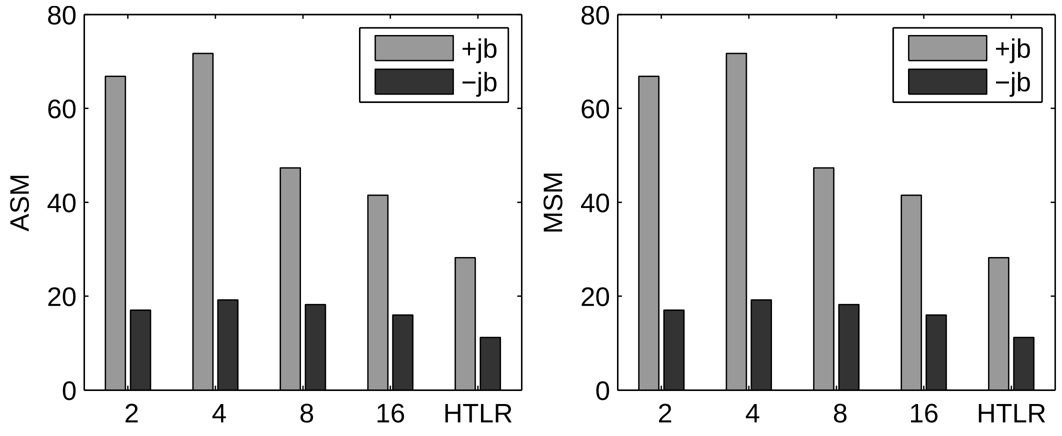

Figure 6.1: ASM and MSM for the dataset, using the MFCC feature, modeled in GMMs with 2, 4, 8 and 16 Gaussians using earth mover's distance and a single Gaussian using LRHT distance, comparing results for all seed-songs with results for all seed-songs without those from jazz_blues genre.

In Figure 6.1, the results of some runs are shown comparing the performance on the set generating playlist from all 729 seed-songs to the performance on the set generating playlist from only 703 seed-songs, omitting seed-songs from the jazz_blues genre. Using earth mover's distance on GMMs with 4 Gaussians to GMMs with 8 Gaussians, an improvement from 66.0 skips to 48.7 skips in ASM is shown which is mostly caused by an improvement from average 290.5 skips to 189.1 skips for playlists with jazz_blues seed-songs which is absolutely unreasonable to that great extent. As seen, the completely unstable jazz_blues results can affect the overall ASM and MSM very much, and the only possibility to draw reasonable conclusions from ASM and MSM for different system configurations was to omit seed-songs from jazz_blues genre. However, jazz_blues songs remain in the database and can be proposed during a playlist generation process.

6.2.2 MAS vs. kMEANS

It is asumed that MAS training leads to better models. The quality of a model trained with kMEANS depends strongly on the initialisation, for which k-means clustering is used in this thesis, while the initial single Gaussian needed for MAS training can be analytically estimated. Using a bad initialisation, kMEANS finds the model with the local maximum of the likelihood next to the initialisation. MAS training and kMEANS training have been compared to see whether MAS-trained models are a better estimate of given data then kMEANS-trained models. It can be seen in Figure 6.2 that the log-likelihood for the MAS-trained models is higher than the log-likelihood for the kMEANS-trained models. However, for playlist generation both training methods lead to comparable ASMs, no advantage for one of the two techniques in comparison to the other can be reported. In all of the following experiments, MAS training was used to train GMMs.

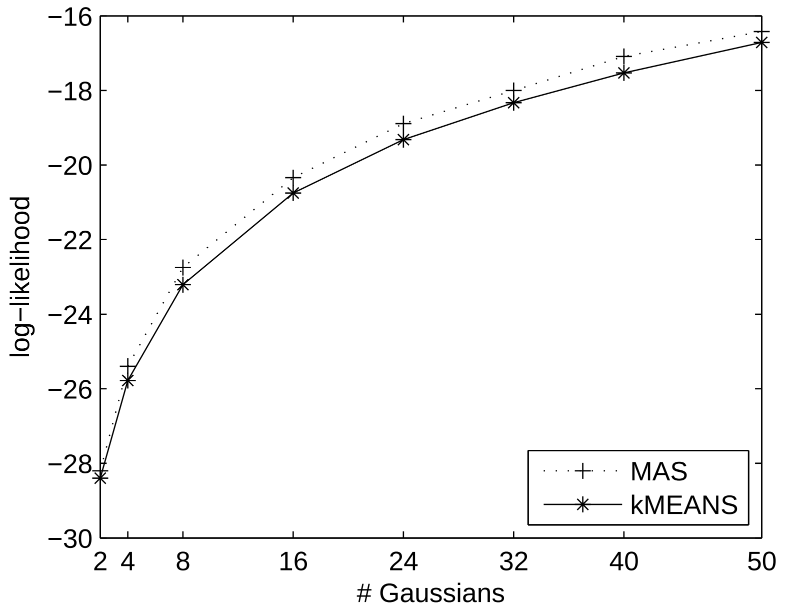

Figure 6.2: The log-likelihoods of GMMs with different numbers of Gaussians given the training samples. GMMs were trained either with kMEANS or MAS.

6.2.3 Basic configuration

The following applies for all the experiments: if not stated differently, they are done with selection S3 and without automatic feature selection. MAS training is used to train the GMMs. Songs from the jazz_blues genre are omitted when a seed-song is selected.

6.3 Distances

The influence of different distances on the quality of the playlists has been investigated, mainly to ascertain how distances operating on GMMs perform in comparisson to distances operating on single Gaussians. It is assumed that distances on GMMs lead to better playlists, since, for example, the number of parameters of the model, which has an influence on the accuracy of the estimation of the data, not only depends on the dimensionality of the feature but can be separately adjusted by the number of used Gaussians. Firstly, different distances for the basic features MFCC and MFLC are compared.

6.3.1 kNN distance: choice of k

Figure 6.3 shows how the choice of k affects the performance of the two kNN-Distances $D_{\text{kNN}}^{\text{KL}}$ and $D_{\text{kNN}}^{\text{EUCL}}$ on MFCC. k is set to all odd numbers from 1 to

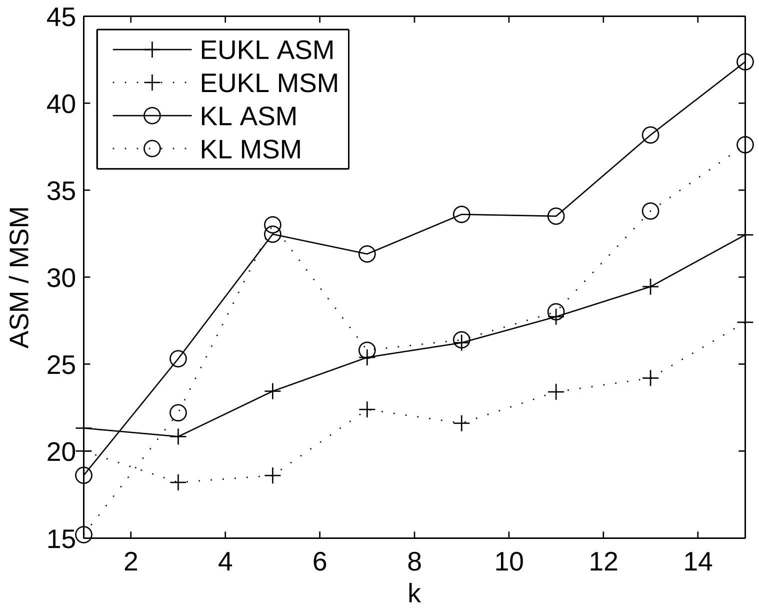

Figure 6.3: Performance of the kNN distances, applied to MAS-trained GMMs with 16 Gaussian components.

15 inclusive. A trend can be seen that the quality of the playlists is decreasing

when k is increased. This is by no means surprising. Having k larger than one, the distance of a GMM to itself would result in large distances for GMMs with a large scatter of Gaussian components. The distances would not only be computed between belonging Gaussians but also between additional pairs (that have large distances when the Gaussians are far away from each other). Furthermore, it would be possible that a GMM has a smaller distance to another GMM than to itself when using a k larger than one. Earth mover's distance solves this problem. In the future experiments, the kNN distances will be restricted to $k = 1$.

6.3.2 Comparison of distances

In Figure 6.4, the performance of different distances on MFCCs is illustrated. The different distances are $D^{\mathrm{KL}}_{\mathrm{EMD}}$ (EKL), $D^{\mathrm{EUCL}}_{\mathrm{EMD}}$ (EEC), $D^{\mathrm{KL}}_{\mathrm{1NN}}$ (KL1), $D^{\mathrm{EUCL}}_{\mathrm{1NN}}$ (EC1), D_{LL} (LLD), D_{LR} (LR) and D_{KL} (KL), where the tokens in brackets denote the labels in the illustrations.

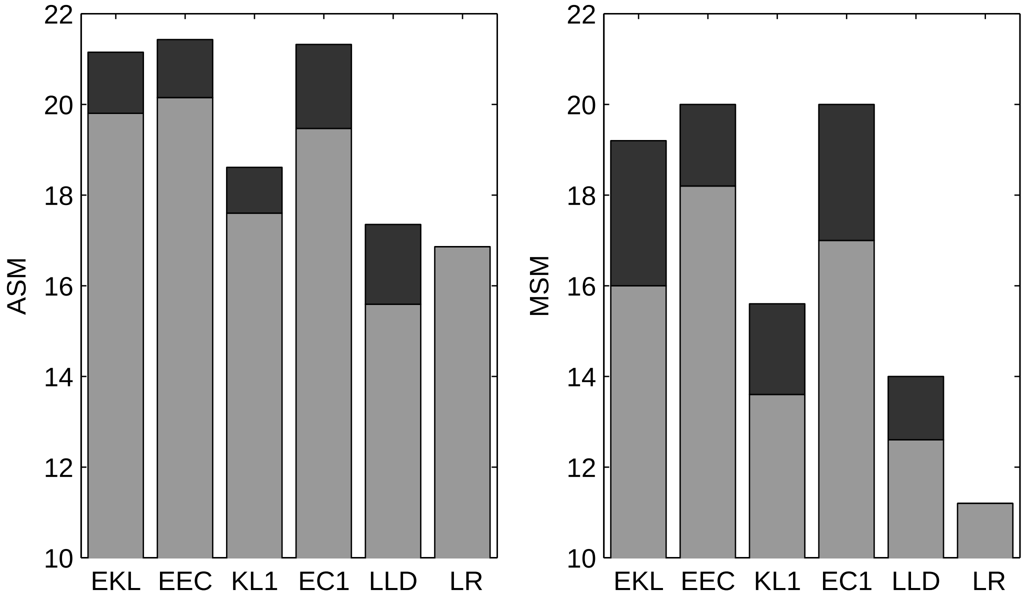

Figure 6.4: Performance of different distances on MFCCs.

The GMM-based distances have been calculated based on GMMs with 2, 4, 8 and 16 Gaussian components. The dark bars on top of the light bars denote the difference between the best performing GMM and the worst performing GMM.

It was assumed that each GMM distance has an optimum of Gaussian components to work on. Actually having insufficient Gaussian components makes the GMM not exact enough, while having too many Gaussian components leads to either overtraining or the GMMs cannot be sufficiently trained due to the lack of training frames, which are limited to 3,000 in this thesis. The optimum should be somewhere in-between. But a trend whether it is better to use more or less Gaussians or a minimum for each of the GMM-based distances operating on GMMs cannot be reported. In other music similarity tasks, very different information for the best number of Gaussian components is reported, ranging from 3 (e.g. [AP02]) up to 64 (e.g. [BLEW03]). In [AP04a] the best number of Gaussians in a GMM modelling 20 MFCCs was found to be 50 as a result of an exhaustive search. Although they are using a dataset that has been refined by hand to form timbre consistent clusters (same artist/same album ground truth), and the disregarding of non-homogeneous songs according to timbre, they are unable to report only one maximum for a fixed number of MFCCs and variable number of Gaussians. Hence it is not surprising that no conclusion can be drawn according to the best number of Gaussians in a task that from a timbre point of view has a lot more inconsistent clusters. The detailed results for the different GMMs can be seen in the Tables A.2, A.3, A.4, and A.5. In addition, in this thesis a scoring measure is used that is much more vulnerable to noise since other components besides the distance measurement are involved in creating the playlist.

A very good example for this vulnerability is the large difference (between ASM and MSM for the LRHT - distance) of 5.66 between 16.86 ASM and 11.20 MSM (details in Table A.1 in the appendix). While the other genres have differences between around 0.2 and 1.5 (ASM to MSM), there are two outlier genres, MP and RP. MP has a difference of 11.31 (19.31 ASM down to 8.00 MSM) and RP even has a difference of 15.69 (17.69 ASM down to 2.00 MSM). For RP, over the half of the playlist have 2 or less skips, but a few outliers produce playlists with extremely high skips of up to 219 that cause such a high ASM.

Several conclusions can be made based on MFCCs. For the earth mover's distance and the 1NN distance, the Kullback-Leibler versions worked better than the Euclidean versions (remember, both earth mover's distance and 1NN distance need an additional distance to compute distances between Gaussian components). This is reasonable since the Kullback-Leibler distance is more exact and also incorporates the covariance matrices, while two Gaussians would be close to each other if only their means are close using the Eu-

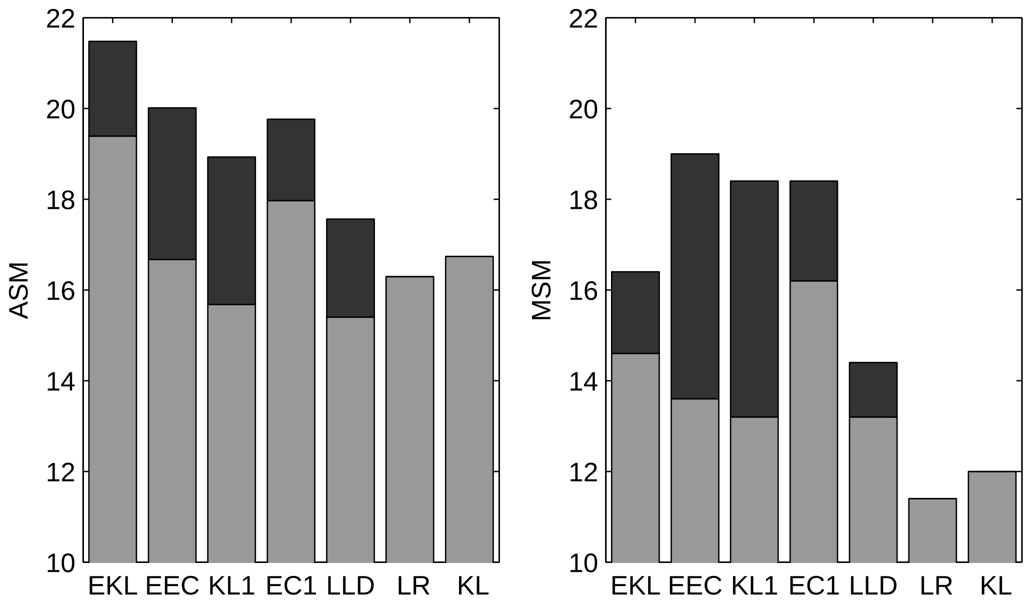

Figure 6.5: Performance of different distances on MFLCs.

clidean distance. The earth mover's distance works worse in comparison to the appropriate 1NN distance respectively. A reason for that could be that the used dataset in combination with the used ground truth requires the distances to be more robust for outliers, and the distances are not required to be too exact, which is actually the opposite of what was shown before and which was not expected to be observed. LL distance leads to the best results from those distances operating on GMMs. This is not a surprise, since computing a distance with the same criteria that was used to estimate the model is a very reasonable procedure. LL distance is followed by LRHT distance, which outperformed all the other distances operating on GMMs.

For MFLCs (Figure 6.5), again LL distance archieved the best results followed by 1NN with KL and LRHT distance. In opposite to MFCCs, earth mover's distance with Kullback-Leibler distance works worse than earth mover's distance with Euclidean distance on MFLCs. Altogether, there is no indication that one of the distances should be preferred to the other distances based on the results which are vulnerable to outliers.

Although LL distance returns the best ASM results, it is not used for further experiments as it has the large drawback that the feature frames are needed every time a distance is computed. This either costs a lot of time or a lot of space, and, since LRHT is only a little bit worse (which as seen is also caused by some outliers), LRHT is chosen. Since LRHT performs slightly better using MFLCs, MFLCs are favoured to MFCCs. A big advantage of LRHT is that it uses single Gaussians instead of GMMs which leads to lot of time savings during the training stage. Also, the computation of the distance is computationally less intensive than the computation of those distances operating on GMMs.

There is no result reported for KL distance on MFCCs. The computation of the KL distance leads to numerical problems in approximately 5% of all paired distances. For MFLCs, which are better de-correlated, no numerical problems occurred.

LRHT distance has also been chosen for all the side features as it turned out that the LRHT distance on single Gaussians leads to comparable results for MFCCs / MFLCs in comparisson to distances operating on GMMs. Nevertheless, it is possible that other distances might work better for certain side features. This requires further investigation.

6.4 Enlargement of the feature space

So far, only the basic features, MFCCs or MFLCs, have been used to generate playlists. The following section investigates whether adding more features can improve the performance of the automatic playlist generation system.

6.4.1 Basic feature + a single side feature

In the first step of this section, MFLCs as basic features are used and combined with one side feature respectively. Automatic feature selection with the adaptation vector $< 1, 0, 0, 5 >$ is used to let the system decide whether it wants only to use MFLCs, only the side feature, or both together. Figure 6.6 shows the ASM results for using MFLCs + 1 additional side feature. It can be seen that the only ASM improvements are obtained by adding centroid, centroid LDA, Renyi, or Renyi LDA. Actually, adding side features should not decrease the performance provided the adaptation is working properly. Performing an adaptation step after 5 skips for the first time could be one reason for a deterioration. Until the first 5 skips are reached, MFLCs and the side feature are used together which can cause the performance to go down if the side feature is really bad. Having bad adaptation weights could be another cause. How the adaptation weights are set reasonably is discussed

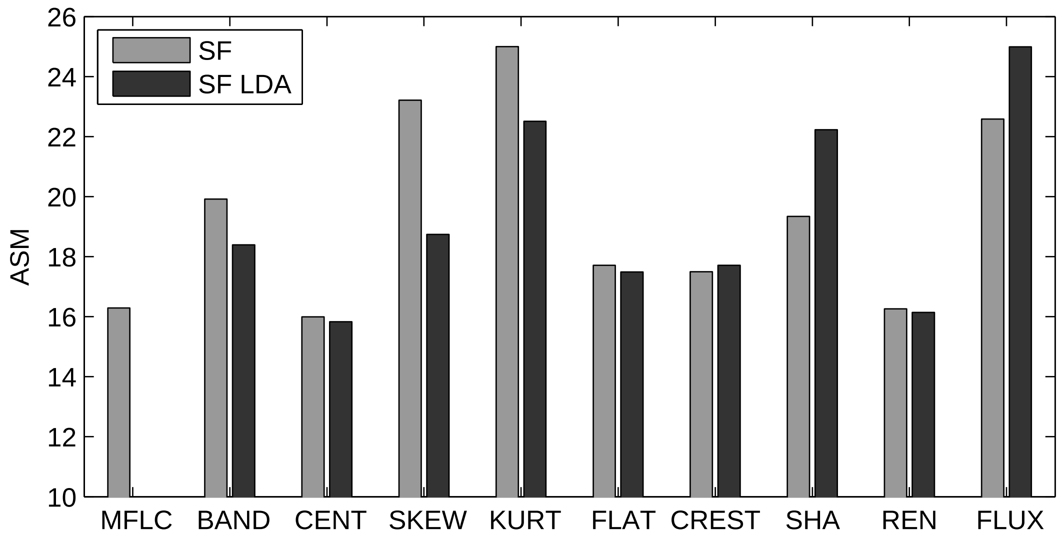

Figure 6.6: Adding one side feature to MFLCs, ASM.

later. Comparing LDA and non-LDA versions of the side features, the LDA versions seem to work better for most of the features. Although there are features that yield better results in their non-LDA versions, LDA is preferred since it is desirable that the side features are as uncorrelated as possible. Furthermore, even if it was not investigated, the dimensionality can be reduced with LDA. For all the following experiments, if not stated differently, LDA side features will be used. These results should not be used to compare the quality of the used side features in general since it is assumed, for example, that the number of coefficients each side feature is extracted in might be well estimated for a certain subset while it might be estimated worse for another subset.

6.4.2 Manually chosen subsets

Adding one side feature only in a few cases leads to improvement of the overall ASM. The performance of best side features for a certain genre can be seen in Figure 6.7. Selecting a particular side feature, the performance for at least one genre can be improved. Renyi LDA performed best for classical and world music (from 1.27 to 0.43 and from 22.98 to 19.25 respectively), Renyi performed best for electronic (19.77 to 17.01), skewness performed best for metal_punk (23.58 to 15.16) and centroid improved rock_pop (from 13.88 to 11.86). The results for all side features and all LDA side features for all genres can be seen in Table A.6. This underlines the assumption that different

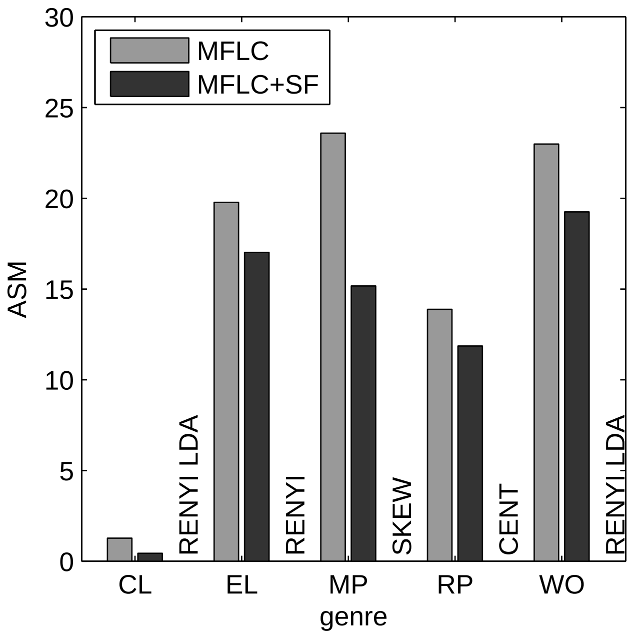

Figure 6.7: MFLCs + the best single side feature combination for each genre.

feature subsets should be used for different genres.

Having these results so far, 3 feature sets are manually chosen:

- MFLCs
- MFLCs+4SF: MFLCs, Renyi, Renyi-LDA, skewness and centroid
- MFLCs+8SF: MFLCs, centroid LDA, bandwidth LDA, skewness LDA, kurtosis LDA, flatness LDA, crest factor LDA, Shannon LDA, and Renyi LDA. Flux-LDA had to be disregarded for reasons of memory constraints. This set represents the the full feature set.

6.5 Selection Strategies

This section compares the different song selection strategies. They decide which song is proposed next, depending on the already-classified songs and the distance function.

6.5.1 Choice of k in the kNN selection

Before the different strategies can be compared to each other, an investigation is made as to which values k should be set and which of the four scoring functions leads to the best results. This experiment is done with the MFLCs+4SF set.

Figure 6.8 shows something rather interesting: while kNN selections perform worse for increasing k and 3 out of 4 scoring functions, the knnnS scoring function makes kNN selection maintain its performance with increasing k.

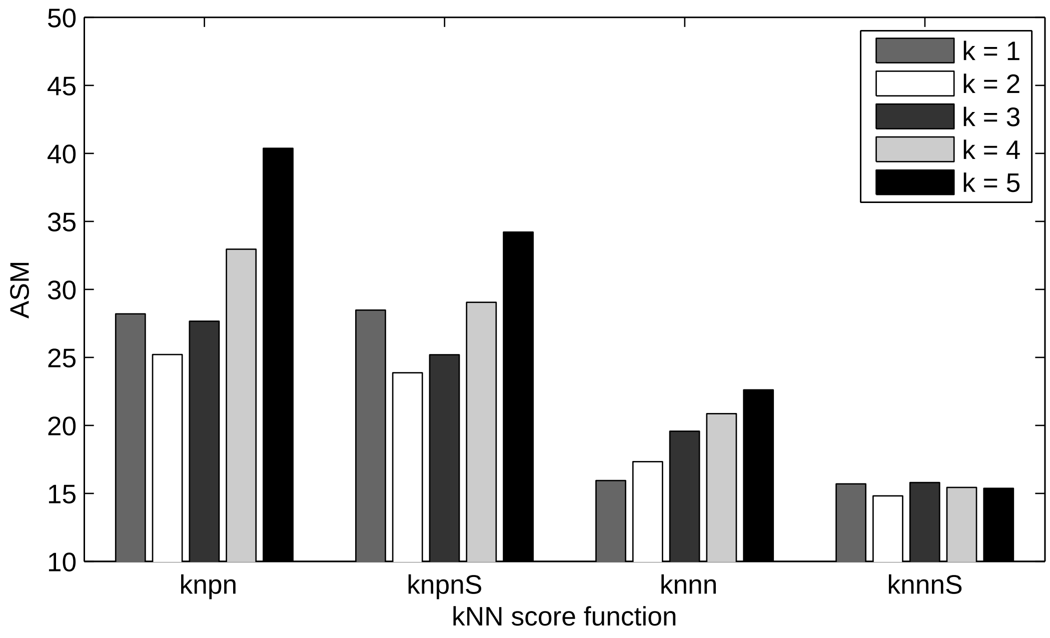

Figure 6.8: Comparison of kNN selection strategies on MFLCs+4SF.

For knnn and knpn this is easy to understand. Only regarding the position of the kth +rated or -rated neighbour, and therefore disregarding the first +rated or -rated neighbour is not meaningful. The fact that knnnS performs better than knpnS indicates that searching far away from -rated songs is better than searching close to +rated songs when regarding 2 or more positive/negative neighbours. This is interesting, since knnnS and knpnS are correlated. Knowing that the position of the first nearest negative neighbour of a candidate song is 4, implies also that the 3 nearest neighbours are positive. Nevertheless, cases can be constructed where each of the 4 different kNN selections for different values of k leads to a different subset of possible candidate songs, from which one is randomly chosen. 2nnnS leads to the best performance.

6.5.2 Comparison of selection strategies

Comparing 2nnnS with the other selection strategies introduced in the selection strategy chapter (5) illustrates that S3 selection works best for all 3 feature sets (Figure 6.9), followed by 2nnnS selection.

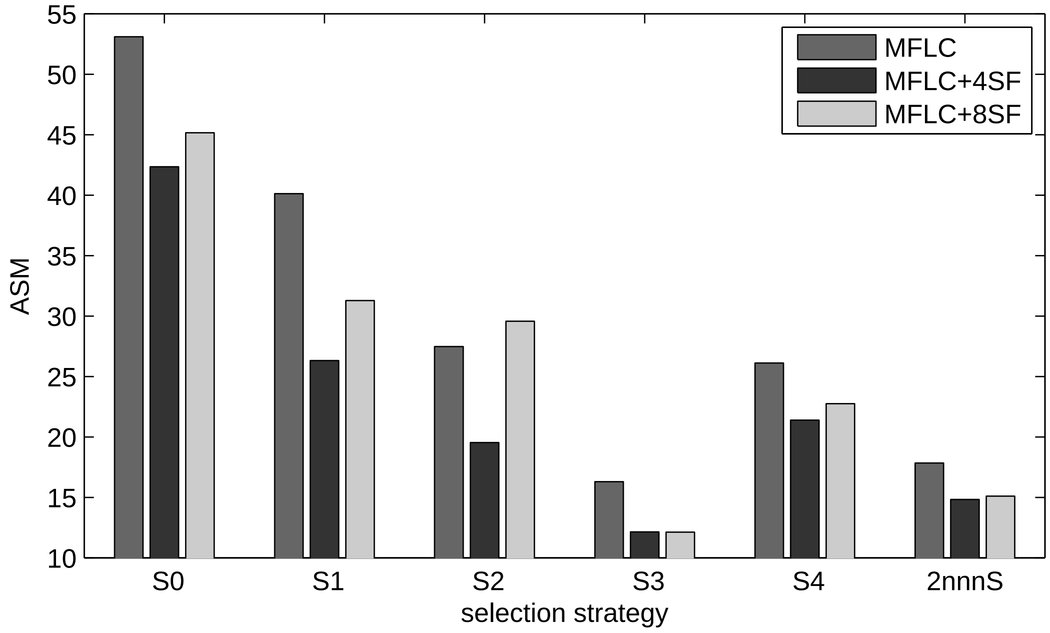

Figure 6.9: Comparison of different selection strategies on the 3 feature sets, ASM.

The fact that also considering the distance to -rated songs leads to increasing performance indicates that the used feature / distance combinations are not able to separate the two classes sufficiently. It is interesting that 2nnnS can almost compete with S3 although it needs at least 2 skips for each seed song until the selection strategy switches from random to 2nnnS (there are at least 2 skips needed to compute the position of the second nearest negative neighbour). Initially starting with another selection strategy until the requirements of 2nnnS are satisfied might be useful, but has not been investigated. Furthermore, it can be assumed that that playlists generated with 2nnnS are more interesting, since 2nnnS selection is not deterministic, unlike S0 - S4, that generate exactly the same playlist when the user feedback is the same for the proposed songs. Another point worthy of note is that 2nnnS gathers all its information from the ranklists. The real distance is only needed to compute the ranklist, and the candidate songs are chosen only from the positions of the examined neighbours, while, for eample, S3 knows by how much for example the nearest neighbour in $\mathcal{P}$ was closer than the nearest neighbour in $\mathcal{N}$.

S2 (choice of the candidate song with the nearest neighbour in $\mathcal{P}$) and S4 (choise of the most promising $p \in \mathcal{P}$ to lead to the next recommendation) lead to comparable results. While S2 achieves better results for MFLCs+4SF, S4 gets better results for MFLCs+8SF. This basically reflects what is already known from those two sets. MFLCs+4SF consists of manually chosen features that are assumed to well separate the two classes +rated and -rated and S2 can profit from that, since only the +rated songs are taken into account when the next song has to be chosen. MFLCs+8SF also includes features that could be counter-productive so the two classes have more overlap (see Table A.6 for which features perform really bad for a certain genre). Since S4 has the ability to learn which already classified songs lead to bad suggestions (since they are probably located in an overlapping region) it can better handle these circumstances. It is not surprising that S0 and S1 selection perform worse. Whilst S0 does not use information of the user rating at all, S1 only uses very little of it. Comparing S0 to all the other selection strategies, it can be seen that user feedback can considerably improve the quality of the generated playlists.

Selection S3 is able to improve the ASM on the MFLCs+8SF set in comparison to the MFLCs+4SF set. This states that S3 is robust enough to handle feature spaces that also include useless or even counter-productive features. Taking a look at the MSM (Figure 6.10), one finds that S4 and 2nnnS might have the same ability. Basically this is an eligible ability for scenarios where many different features are used and no manual or automatic feature selection can be performed.

6.6 Automatic feature selection

This last section from the experiments chapter further investigates the automatic feature selection (AFS) used for feature space adaptation to the user's needs. AFS has already been used in the experiments where the influence of the side features is investigated. An experiment is carried out according to how the AFS weights should be set. Another one is done according to the AFS interval v, denoting the number of times the skip button has to be pressed until the next AFS is performed.

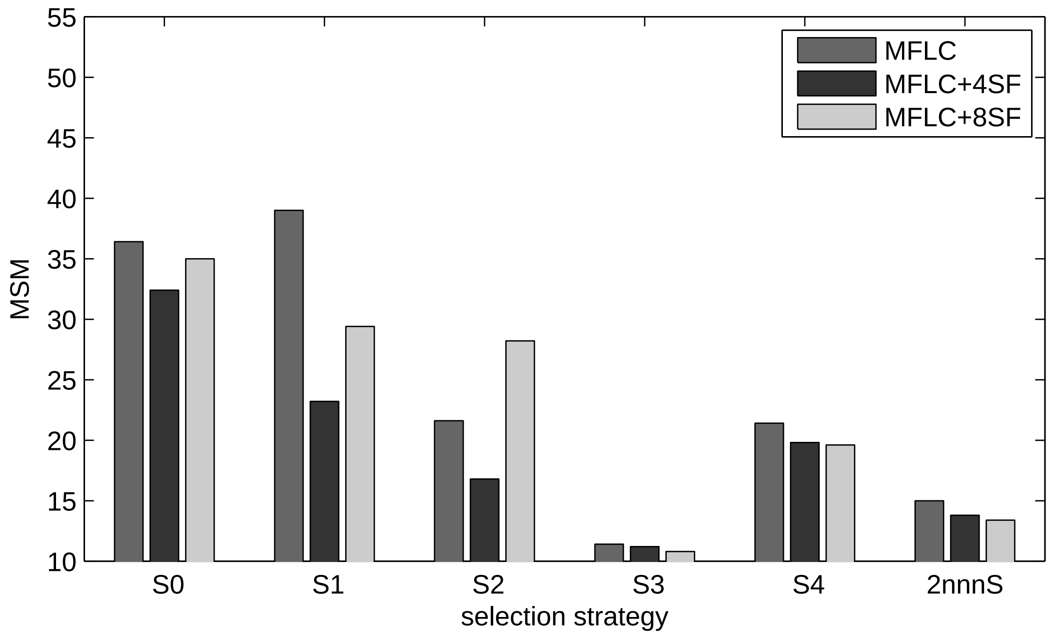

Figure 6.10: Comparison of different selection strategies on the 3 sets, MSM.

6.6.1 Different weighting configurations

Figure 6.11 (6.12) shows the ASM (MSM) for adapting with different AFS weights. The following weighting combinations are used:

- $< 1, 0, 0, v >$: it is important that each +rated song has a close +rated neighbour.
- $< 1, 0, 1, v >$: it is important that each +rated song has a close +rated neighbour and that it's closest -rated neighbour is far away.
- $< 1, 1, 1, v >$: it is important that each +rated song has a close +rated neighbour, that each -rated song has a close -rated neighbour, and that the closest neighbour from -rated to each +rated song is far away.
- $< 2, 0, 1, v >$: it is important that each +rated song's closest -rated neighbour is far away, but it is more important that each +rated song has a close +rated neighbour. This is the most intuitive approach.

For this experiment, v was set to 5 which is a trade-off between adapting very often (low values of v) and incorporating more training data (high values of v). An experiment where different values for v are investigated is discussed subsequently. It turns out that using the adaptation vector $< 1, 1, 1, 5 >$

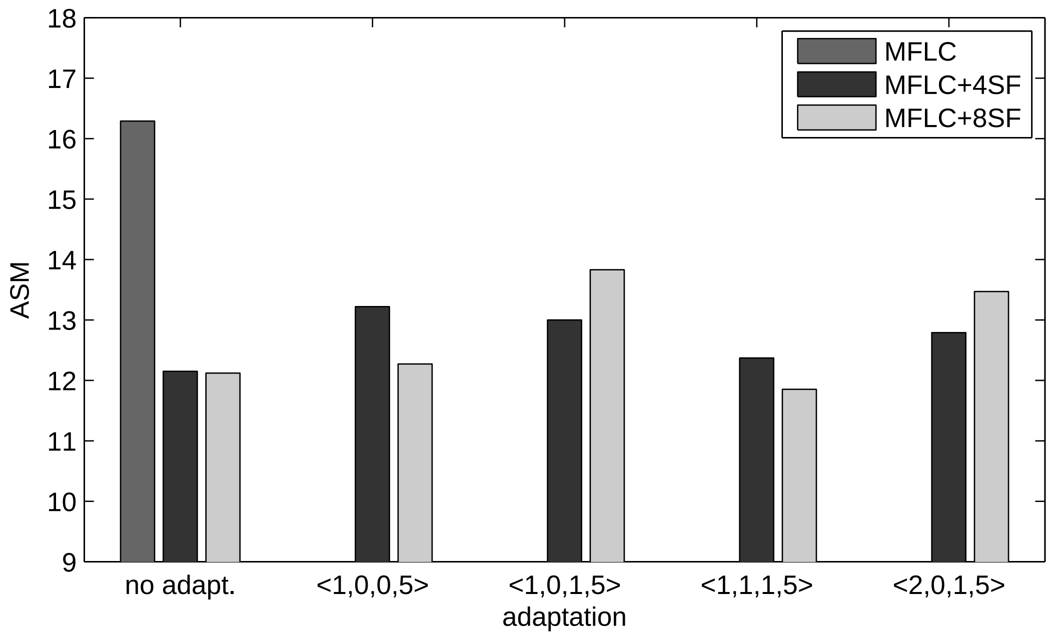

Figure 6.11: ASM for different AFS weights.

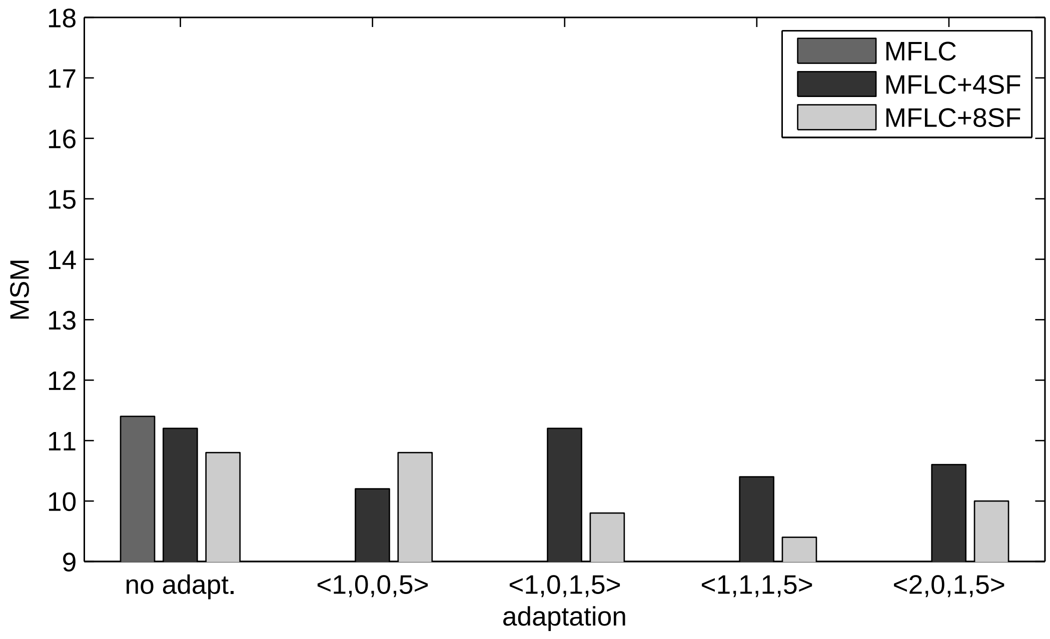

Figure 6.12: MSM for different AFS weights.

leads to the best results. Contrary to the assumption that the compactness of the -rated songs is unimportant, the AFS with all weights set equal performs best. If only s^+, and $s^\pm, s^\mp$ are regarded, songs from -rated can still be close to songs from +rated, since they can be compensated by -rated songs that are very far away from the +rated songs. This is more unlikely to happen if the -rated songs have to be close to each other, too. In actual fact, the gain from AFS was not as high as expected. The only improvements in ASM can be reported for an adaptation vector of $< 1, 1, 1, 5 >$ on MFLCs+8SF. Taking a look at the MSM shows that for almost every weight combination and used set the performance can be increased. For ASM this is not the case. What basically happens is that caused by the adaptation good playlists are improved a little bit, but bad ones become much worse. Since bad playlists can be caused by outliers, and there are usually noticeable less than 50% outliers in a complete evaluation run, MSM is affected positively by adaptation and ASM is affected negatively.

There is only a slight gain for using AFS in combination with S3 (MFLCs+8SF). Figure 6.13 shows the ASM results for AFS with the weighting vector $< 1, 1, 1, 5 >$ on the selection strategies S1 to 2nnnS. It can be seen that the

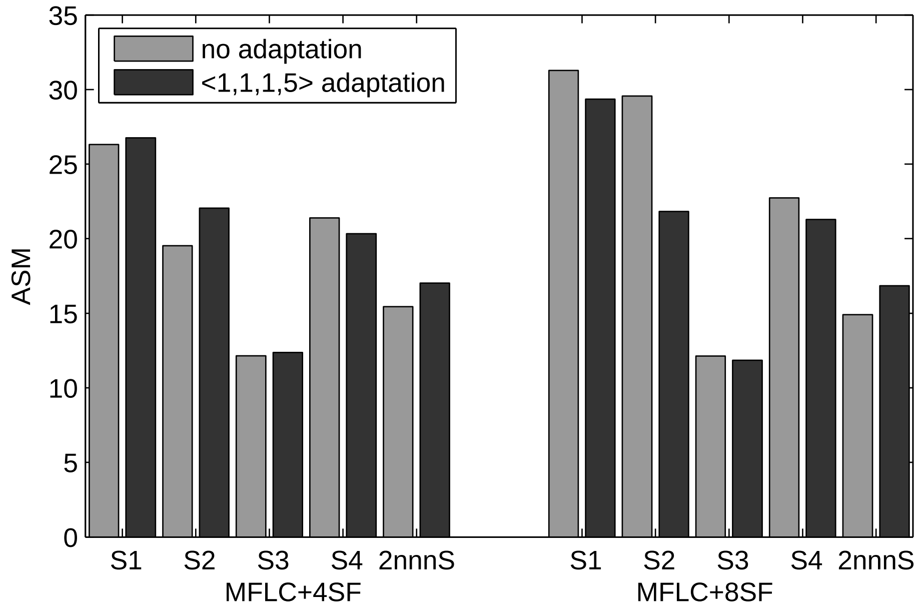

Figure 6.13: Automatic feature selection and different song selection strategies.

results can also get worse when doing AFS on the MFLC+4SF set. This is due to the limited number of feature combinations. Having only 5 features reduces the possible combinations to 31. Beyond that, the performance for MFLC+4SF is already a lot better than MFLC due to the manual preselection of features, where only side features are used that improve the performance on a certain genre most when using them with MFLCs. MFLC+8SF is performing worse without AFS, since also useless features might be included in MFLC+8SF. But when using automatic feature selection, performance is increased for each of the selection strategies except the kNN representative. The gain for S1 is small. AFS cannot fix the major drawback of S1, that only the last +rated song is taken as reference. The highest gain is observed for S2. This indicates that automatic feature selection is able to transform the feature space in a way that -rated songs are moved away from the +rated songs. S2 only regards the nearest positive neighbour of a candidate song and therefore can profit from that feature space transformation.

It is interesting to see that even for the MFLC+8SF set, the performance of 2nnnS decreases when using automatic feature selection while all the other selection strategies improve their performance. This is due to the fact that ASM uses the distance to the nearest +rated neighbour and the distance to the nearest -rated neighbour of each already rated song to score a feature combination, disregarding the k nearest neighbours for $k > 1$. One can assume that the performance of 2nnnS could be increased by applying an AFS where the scoring function is related to 2nnnS. AFS could be customised towards 2nnnS by returning the feature combination that performed best in the following task. For each +rated song, the ranklist of all already classified songs is computed. The scores, using 2nnnS on those ranklists are added. The feature combination with the largest sum is chosen.

A considerably higher increase in performance due to automatic feature selection is assumed to be observed when using more diverse features. All the used features essentially describe the same, they are almost all computed on single power spectrum frames. Using only those features, the system will never be able to separate slow songs from fast songs. If there is a tempo feature and a user who only wants to listen to slow songs, it is assumed that the automatic feature selection would detect that a feature combination only using the tempo feature would separate the +rated songs from the -rated songs best. Although the performance of AFS is limited due to the type and number of used features, the best result reported in this thesis uses AFS.

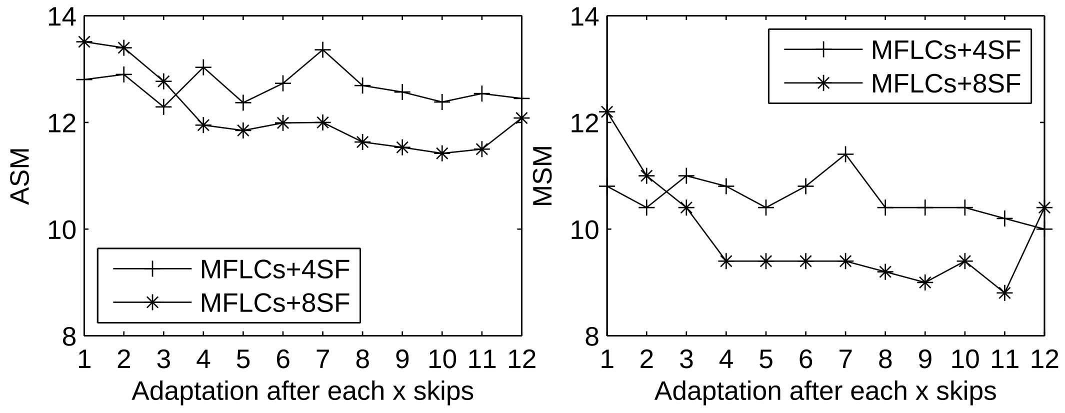

Figure 6.14: Comparison of different adaption intervals. An adaption step is performed after each x times pressing the skip button. S3 is used.

6.6.2 Choice of adaptation interval

The number of skips that need to be done by the user to perform an adaptation step should be an important issue. Doing adaptation too early and too frequently on one hand might lead to wrong adaptation results due to insufficient training data. Doing adaptation too seldom could lead to bad results since it could take too long till unimportant features are excluded. Figure 6.14 shows that no conclusion can be drawn from the experiments on set MFLCs+4SF, using selection strategy S3. This is because adaptation generally results only in negligible differences due to the reduced number of features, so that the choice of the adaption interval does not play an important role. The trend of the MLFCs+8SF feature set is however reasonable. For small values of v, the performance is bad, since there is not enough training data available to perform automatic feature selection. It is further assumed that the increase for $v > 10$ is caused by the fact that automatic feature selection and thus the exclusion of the useless or counter-productive features is performed too late. The best results are obtained for performing automatic feature selection after each $v = 10$ skips. An ASM of 11.4 is obtained, which is also the best result that can be reported in this whole thesis.

6.7 Summary

In this chapter, the evaluation criteria and the database are explained. Two measures to measure the quality of playlists are introduced, ASM and MSM. Information about the database and the framework is provided. Different experiments have shown that it is reasonable to use single Gaussians and LRHT distance instead of GMMs and GMM based distances, since the estimation of the models and the computation of the distances are computationally less intensive, and produce comparable results by all means. Adding side features can improve the performance, but the performance change depends very much on the selection strategy. S3 selection is the best, followed by 2nnnS. 2nnnS outperforms all the other kNN selections. When examining more than 1 neighbour, it is more important to search the new track far away from the -rated tracks than to search it close to the +rated tracks. Simple binary user feedback improves the performance a lot. Depending on the set and the selection strategie, adaptation of the distance function using AFS can result in improvements. The overall best performance reported in this thesis is an ASM of 11.4 for the MFLCs+8SF feature set, using song selection strategie S3 and AFS with the adaptation vector $< 1, 1, 1, 10 >$. Randomly selecting songs would obtain an ASM of 144.

Chapter 7

Conclusion and outlook

7.1 Conclusion

A system generating playlists consisting of songs similar to a given seed-song was developed and investigated. Similarity in the used evaluation meant that two songs are similar exactly if and only if they belong to the same genre. Immediate binary user feedback was gained during the playlist generation process, containing information of whether a song was accepted (same genre than the seed-song) or rejected (different genre than the seed-song) by the user. Besides the used basic features MFCCs (de-correlated with cosine transform) and MFLCs (de-correlated with LDA), different side features have been used to capture additional characteristics of the spectral frames. Song selection strategies incorporating a user-adaptive distance function have been used for the playlist generation process.

Different distances operating on single Gaussians or Gaussian mixture models (GMMs) have been evaluated in the context of playlist generation. Although Gaussian mixture models are better able to approximate the distribution of the feature frames, distances on GMMs did not lead to results that justified preferring distances working on GMMs to distances working on single Gaussians. For all the different distances that have been examined, only one (log-likelihood distance) outperformed the LRHT distance (likelihood ratio hypothesis test). For LRHT it further turned out that MFLCs work better than MFCCs. Based on that, single Gaussians and LRHT distance have been used for the side features, too.

Adding a single additional side feature to the basic feature, the performance over all genres could only seldomly and only slightly be improved (centroid,

centroid LDA, renyi, renyi LDA), but the performance for each genre could be increased a lot by adding a certain side feature. MFLCs and the four side features that performed best for at least one certain genre (renyi, renyi LDA, skewness, and centroid) form the MFLC+4SF feature subset. MFLCs and each of the LDA side features except flux LDA, which was excluded due to memory constraints form the MFLC+8SF feature subset. Except for S2 selection strategy on the MFLC+8SF set, the performance could be increased by adding side features to the basic MFLC feature. The additional characteristics of the spectral frames, captured by the side features hence are useful for playlist generation.

Different selection strategies have been evaluated in the context of playlist generation. The best selection strategy uses the distance to the nearest positive neighbour and the nearest negative neighbour of each candidate song for decision (S3), followed by 2nnnS which is the best kNN selection strategy. kNN selection strategies make their decision which song to propose next based on the ranklist of each candidate song. The ranklist of a candidate song consists of the already classified songs, sorted by their distance to the candidate song. It turned out that if more than only the first nearest neighbour is included in the decision, it is more likely that a user will like the next recommendation if it is far away from the negative rated songs than if it is close to the positive rated songs. All the other selection strategies S1 (using the last accepted song), S2 (using all the accepted songs), and S4 (using the most promising accepted song) lead to better results than S0, where playlist are generated only regarding the initial seed-song, without incorporating user feedback. This indicates that immediate user-feedback is very useful for playlist generation.

Results were further improved by adapting the feature space to the user, using automatic feature selection. It turned out that for selecting the best feature subset, the compactness of the +rated songs and the compactness of the -rated songs are important. It is further important that the two classes +rated and -rated are far away from each other. The gain of automatic feature selection depends strongly on the chosen feature set and the selection strategy. Allowing more features leads to larger improvements. On the complete feature set for the best selection strategy (S3), the gain was lowest. The overall best result was archived with the complete feature set, using selection strategy S3 and automatic feature selection. In this experiment, the evaluation measure ASM could be reduced from 144 (random) to 11.4 (which represents the average number of times the skip button has to be pressed to find 19 songs similar to a given seed-song, which represents the musical taste of a user). The results for 2nnnS selection could not be improved by adapting the distance function to the user, it is assumed that an automatic feature selection algorithm which is adjusted to the 2nnnS selection strategy could correct that.

For genre classification, spectral similarity is not sufficient since songs from different genres sometimes have similar sound textures. For certain genres (e.g. world), lots of outliers which are songs that have a large distance to the other songs from their genre, can cause bad playlists. Those large distances can be caused, for example, by timbre dissimilarity or problems during the modelling stage. The evaluation using the current evaluation criteria and the current distance measures is vulnerable to outliers. This is also due to the size of the dataset. Furthermore, a genre ground truth is not perfectly suitable for a playlist generation task. User tests for gaining a ground truth are needed.

7.2 Outlook

The song selection strategies could be further improved by adding characteristics of S4 (consider, whether already rated songs lead to good or bad proposals) and kNN selections (consider more than only 1 neighbour) to S3.

In addition there might be better (and primarily faster) ways than performing feature selection with a brute force approach (see e.g. 5.4), and there are lots of additional ways to score a feature combination.

Having shown that automatic feature selection can handle large feature spaces, more features are required to further improve the system. These could be features related to e.g. rhythm (e.g. [GD05]), tempo (e.g. [ARD04]), melody (e.g. [GKM03]), or keys (e.g. [ZZM04]). Mood-related features might be very useful. A more diverse feature space should on one hand improve the overall results for at least S3. On the other hand, it is expected that automatic feature selection will have a larger positive impact on the results.

User tests are required to generate a ground truth that could be used instead of genre. Even tough musical taste and genre are correlated, not every subjective playlist generation criteria can be modelled with genre affiliation. Furthermore a large database is needed, having labels that follow this ground truth to make the system more insusceptible to noise.

Several steps in the feature extraction process could be further improved:

- Selection of the most relevant segment of a musical piece (e.g. chorus) for the extraction of the feature frames (e.g. [ANS$^+$05]
- Use of variable window size and in particular window shift size related to the tempo of a song (e.g. [WC05])

The assumption that users do not change their musical preference at least slightly while they listen to music is likely to be wrong. The evaluation of different use cases, for example, those mentioned in [PPW05a], is required. In that context it could be useful to remove songs from the +rated and -rated sets after a certain time, or to incorporate songs from those two sets differently strong, depending on the point of time they were added to the sets respectively.

Nowadays portable MP3 players are not able to perform the proposed algorithms in a reasonable amount of time. But components like feature extraction, statistical modelling and distance computation as well as automatic feature selection could be performed offline or by a more powerful processor while, for example, synchronising music with the computer. So although the used similary measure is far from being perfect, the achieved results are much better than random selection, and portable MP3 players implementing the algorithms are already forseeable in the near future.

Appendix A

Tables

genre	mean	median
CL	2.58	3
EL	18.51	17
MP	19.31	8
RP	17.69	2
WO	26.20	26

Table A.1: Detailed results for LHRT distance on MFCCs.

distance	2	4	8	16	single Gaussian
D_{EMD}^{KL}(EKL)	19.97	21.15	20.67	19.80	
D_{EMD}^{EUCL}(EEC)	20.72	21.13	21.43	20.15	
D_{1NN}^{KL}(KL1)	17.60	18.23	17.86	18.61	
D_{1NN}^{EUCL}(EC1)	20.10	19.47	21.08	21.32	
D_{LL}(LLD)	17.35	**16.39**	**15.82**	**15.59**	
D_{LR}(LR)					16.86

Table A.2: Different distances on MFCCs, ASM, values lower than the LR value are marked bold.

distance	2	4	8	16	single Gaussian
D^{KL}_{EMD}(EKL)	17.00	19.20	18.20	16.00	
D^{EUCL}_{EMD}(EEC)	18.60	20.00	18.20	18.20	
D^{KL}_{1NN}(KL1)	15.60	15.20	13.60	15.20	
D^{EUCL}_{1NN}(EC1)	17.60	17.00	19.60	20.00	
D_{LL}(LLD)	12.60	14.00	13.00	12.60	
D_{LR}(LR)					11.20

Table A.3: Different distances on MFCCs, MSM, no values are lower than the LR value.

distance	2	4	8	16	single Gaussian
D^{KL}_{EMD}(EKL)	20.80	19.93	20.48	21.48	
D^{EUCL}_{EMD}(EEC)	20.01	18.54	17.22	16.67	
D^{KL}_{1NN}(KL1)	18.00	17.83	**15.68**	18.93	
D^{EUCL}_{1NN}(EC1)	18.37	19.32	19.76	17.97	
D_{LL}(LLD)	**15.40**	17.34	17.56	16.80	
D_{LR}(LR)					16.29
D_{KL}(KL)					16.74

Table A.4: Different distances on MFLCs, ASM, values lower than the LR value are marked bold.

distance	2	4	8	16	single Gaussian
D^{KL}_{EMD}(EKL)	16.40	15.00	14.60	15.60	
D^{EUCL}_{EMD}(EEC)	19.00	17.20	13.60	13.80	
D^{KL}_{1NN}(KL1)	15.60	14.00	13.20	18.40	
D^{EUCL}_{1NN}(EC1)	16.20	18.40	16.40	16.60	
D_{LL}(LLD)	13.20	13.40	14.40	13.80	
D_{LR}(LR)					11.40
D_{KL}(KL)					12.00

Table A.5: Different distances on MFLCs, MSM, no values are lower than the LR value.

	CL	EL	MP	RP	WO	ASM
MFLCs	1.27	19.77	23.58	13.88	22.98	16.29
+centroid	1.31	19.80	**22.49**	**11.86**	24.50	**15.99**
+centroid LDA	1.26	24.23	**15.22**	**12.56**	25.86	**15.83**
+bandwidth	1.30	21.61	30.98	17.01	28.70	19.92
+bandwidth LDA	1.45	23.89	24.56	15.07	27.01	18.39
+skewness	2.23	27.23	38.53	19.05	29.49	23.31
+skewness LDA	3.41	30.31	**15.16**	14.14	30.67	18.74
+kurtosis	1.27	27.97	44.84	15.41	35.53	25.01
+kurtosis LDA	1.51	27.09	35.69	16.24	32.02	22.51
+flatness	1.34	21.48	24.11	14.44	27.18	17.71
+flatness LDA	1.54	22.90	**21.38**	**13.70**	37.92	17.49
+crest factor	1.56	21.32	**22.82**	14.99	26.80	17.50
+crest factor LDA	1.42	22.03	**22.36**	**12.33**	30.41	17.71
+Shannon entropy	2.74	22.19	29.40	16.99	25.66	19.40
+Shannon entropy LDA	3.69	20.10	37.89	17.97	31.50	22.23
+Renyi entropy	**0.55**	**17.01**	**20.00**	20.50	23.25	**16.25**
+Renyi entropy LDA	**0.43**	23.87	19.96	17.17	**19.25**	**16.14**
+flux	1.31	28.57	28.00	**13.80**	41.20	22.58
+flux LDA	1.32	25.03	41.44	18.57	38.57	24.99

Table A.6: The performance of MFLCs + a single side feature. The bold marked values are lower than the MFLC baseline values.

List of Figures

List of Tables

Bibliography

[AHH$^+$01] E. Allamanche, J. Herre, O. Helmuth, B. Fröba, T. Kastner, and M. Cremer. Content-based identification of audio material using mpeg-7 low level description. In *Proceedings of the 2nd International Symposium of Music Information Retrieval (ISMIR)*, 2001.

[AHH$^+$03] E. Allamanche, J. Herre, O. Hellmuth, T. Kastner, and C. Ertel. A multiple feature model for musical similarity retrieval. In *Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR)*, 2003.

[AMO93] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows.* Prentice Hall, 1993.

[ANS$^+$05] S. A. Abdallah, K. Noland, M. Sandler, M. Casey, and C. Rhodes. Theory and evaluation of a bayesian music structure extractor. In *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR)*, pages 420–425, 2005.

[AP02] J.-J. Aucouturier and F. Pachet. Music similarity measures: What's the use? In *Proceedings of the 3rd International Conference on Music Information Retrieval (ISMIR)*, 2002.

[AP03] J.-J. Aucouturier and F. Pachet. Representing musical genre: A state of the art. *Journal of New Music Research*, 32(1), 2003.

[AP04a] J.-J. Aucouturier and F. Pachet. Improving timbre similarity: How high is the sky? *Journal of Negative Results in Speech and Audio Sciences*, 1(1), 2004.

[AP04b] J.-J. Aucouturier and F. Pachet. Tools and architecture for the evaluation of similarity measures : Case study of timbre similarity. In *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR)*, 2004.

[ARD04] M. Alonso, G. Richard, and B. David. Tempo and beat estimation of musical signals. In *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR)*, 2004.

[AT00] M. Alghoneimy and A. Tewfik. Personalized music distribution. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2000.

[AT01] M. Alghoniemy and A. Tewfik. A network flow model for playlist generation. In *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME)*, 2001.

[Auc02] J.-J. Aucouturier. Scaling up music playlist generation. In *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME)*, 2002.

[BL03] J. J. Burred and A. Lerch. A hierarchical approach to automatic musical genre classification. In *Proceedings of the 6th International Conference on Digital Audio Effects (DAFX)*, 2003.

[BLEW03] A. Berenzweig, B. Logan, D. Ellis, and B. Whitman. A large-scale evaluation of acoustic and subjective music similarity measures. In *Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR)*, 2003.

[DLR77] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the em algorithm. *J. Roy. Statistal Society B*, 39:1–38, 1977.

[EHUL96] T. Eisele, R. Haeb-Umbach, and D. Langmann. A comparative study of linear feature transformation techniques for automatic speech recognition. In *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, volume 1, pages 252–255, Philadelphia, PA, 1996.

[Foo97] J. Foote. Content-based retrieval of music and audio. In *Proceedings of SPIE Multimedia Storage and Archiving Systems II*, volume 3229, pages 138–147, 1997.

[Fuk90] K. Fukunaga. *Introduction to statistical pattern recognition.* Acad. Pr., 2. ed. edition, 1990.

[GD05] F. Gouyon and S. Dixon. A review of automatic rhythm description systems. *Computer Music Journal*, 29(1):34–54, 2005.

[GKM03] E. Gomez, A. Klapuri, and B. Meudic. Melody description and extraction in the context of music content processing. *Journal of New Music Research*, 32:23–40, 2003.

[GML03] S. Gao, N. C. Maddage, and C. H. Lee. A hidden markov model based approach to music segmentation and identification. In *Proceedings of the 4th IEEE Pacific-Rim Conference On Multimedia (PCM)*, 2003.

[GSR91] H. Gish, H-H Siu, and R. Rohlicek. Segregation of speakers for speech recognition and speaker identification. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 1991.

[HAE03] J. Herre, E. Allamanche, and C. Ertel. How similar do songs sound? towards modeling human perception of musical similarity. In *Proceedings of IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA), Mohonk, NY (USA)*, 2003.

[HAH01] J. Herre, E. Allamanche, and O. Helmuth. Robust matching of audio signals using spectral flatness features. In *Proceedings of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, 2001.

[HF01] D. B. Hauver and J. C. French. Flycasting: Using collaborative filtering to generate a playlist for online radio. In *Proceedings of the International Conference on Web Delivery of Music, Florence, Italy*, October 05 2001.

[HJ04] Y.-C. Huang and S.-K. Jenor. An audio recommendation system based on audio signature description scheme in MPEG-7 audio. *Proceedings of the IEEE International Conference. on. Multimedia and Expo (ICME)*, pages 639–642, 2004.

[HW79] J. A. Hartigan and M. A. Wong. A K-means clustering algorithm. *Applied Statistics*, 28:100–108, 1979.

[ISM] International conference on music information retrieval (ISMIR) http://www.ismir.net.

[JN84] N. S. Jayant and P. Noll. *Digital coding of waveforms.* Prentice-Hall Intern., 1984.

[KFN98] T. Kaukoranta, P. Fränti, and O. Nevalainen. Iterative split-and-merge algorithm for vq codebook generation. *Optical Engineering*, 37(10):2726–2732, 1998.

[Kul59] S. Kullback. *Information theory and statistics.* Wiley, 1959.

[LC00] B. Logan and S. Chu. Music summarization using key phrases. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, September 12 2000.

[LO04] T. Li and M. Ogihara. Content-based music similarity search and emotion detection. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2004.

[Log00] B. Logan. Mel frequency cepstral coefficients for music modeling. In *Proceedings of the International Symposium of Music Information Retrieval (ISMIR)*, 2000.

[Log02] B. Logan. Content-based playlist generation: Exploratory experiments. In *Proceedings of the 3rd International Conference on Music Information Retrieval (ISMIR)*, 2002.

[Log04] B. Logan. Music recommendation from song sets. In *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR)*, 2004.

[LS01a] B. Logan and A. Salomon. A content-based music similarity function. Technical report, Compaq Cambridge Research Laboratory, 2001.

[LS01b] B. Logan and A. Salomon. A music similarity function based on signal analysis. In *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME)*, 2001.

[LST04] A. S. Lampropoulos, D. N. Sotiropoulos, and G. A. Tsihrintzis. Individualization of music similarity perception via feature subset selection. In *Proceedings of the IEEE International Conference on Systems, Man & Cybernetics*, pages 552–556. IEEE, 2004.

[LT03] T. Li and G. Tzanetakis. Factors in automatic musical genre classification of audio signals. In *Proceedings of the IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, 2003.

[ME05] 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)*, pages 594–599, 2005.

[MF04] C. McKay and I. Fujinaga. Automatic genre classification using large high-level musical feature sets. In *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR)*, pages 525–530, 2004.

[mpe] Information technology - multimedia content description interface - part 4: Audio. ISO/IEC 15938-4:2002.

[MS91] T. Martinetz and K. Schulten. A "neural-gas" network learns topologies. In *Proceedings of the International Conference on Artificial Neural Networks*, 1991.

[Pam06] E. Pampalk. *Computational Models of Music Similarity and their Application in Music Information Retrieval.* PhD thesis, Technische Universität Wien, Fakultät für Informatik, 2006.

[PE02] S. Pauws and B. Eggen. Pats: Realization and user evaluation of an automatic playlist generator. In *Proceedings of the 3rd International Conference on Music Information Retrieval (ISMIR)*, 2002.

[Pee02] G. Peeters. Automatically selecting signal descriptors for sound classification. In *Proceedings of the 2002 International Computer Music Conference (ICMC)*, 2002.

[Pee04] G. Peeters. A large set of audio features for sound description (similarity and classification) in the cuidado project. Technical report, IRCAM, Paris, France, 2004.

[PM00] D. Pelleg and A. Moore. X-means: Extending k-means with efficient estimation of the number of clusters. In *Proceedings of the 17th International Conference on Machine Learning*, 2000.

[Poh05] T. Pohle. Extraction of audio descriptors and their evaluation in music classifiction tasks. Master's thesis, Technische Universität Kaiserslautern, Fachbereich Informatik, 2005.

[PPW05a] E. Pampalk, T. Pohle, and G. Widmer. Dynamic playlist generation based on skipping behavior. In *Proceedings of the 6th*

International Conference on Music Information Retrieval (IS-MIR), pages 634–637, 2005.

[PPW05b] T. Pohle, E. Pampalk, and G. Widmer. Generating similarity-based playlists using traveling salesman algorithms. In *Proceedings of the 8th Int. Conference on Digital Audio Effects (DAFx'05)*, September 2005.

[PvdW05] S. Pauws and S. van der Wijdeven. User evaluation of a new interactive playlist generation concept. In *Proceedings of the 6th International Conference on Music Information Retrieval (IS-MIR)*, pages 638–643, 2005.

[Pye00] D. Pye. Content-based methods for the management of digital music. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2000.

[RK] A. Ramalingam and S. Krishnan. Gaussian mixture modeling using short time fourier transform features for audio fingerprinting.

[RTG98] Y. Rubner, C. Tomasi, and L. J. Guibas. A metric for distributions with applications to image databases. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 59–66, 1998.

[SM05] N. Scaringella and D. Mlynek. A mixture of support vector machines for audio classification. In *1st music information retrieval evaluation exchange (MIREX)*, 2005.

[SSWW98] H. Soltau, T. Schultz, M. Westphal, and A. Waibel. Recognition of music types. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 1998.

[SW59] C. E. Shannon and W. Weaver. *The mathematical theory of communication.* Univ. of Illinois Press, 8. print. edition, 1959.

[TC99] G. Tzanetakis and P. Cook. Multifeature audio segmentation for browsing and annotation. *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pages 103–106, 1999.

[TC02] G. Tzanetakis and P. Cook. Musical genre classification of audio signals. In *Proceedings of IEEE Transactions on Speech and Audio Processing*, 2002.

[TEC01] G. Tzanetakis, G. Essl, and P. Cook. Automatic musical genre classification of audio signals. In *Proceedings of the International Symposium of Music Information Retrieval (ISMIR)*, November 09 2001.

[TTK05] M. Tolos, R. Tato, and T. Kemp. Mood-based navigation through large collections of musical data. In *IEEE Consumer Communications and Networking Conference (CCNC)*, January 2005.

[WC05] K. West and S. Cox. Finding an optimal segmentation for audio genre classification. In *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR)*, pages 680–685, 2005.

[ZZM04] G. Zoia, R. Zhou, and D. Mlynek. A multi-timbre chord/harmony analyzer based on signal processing and neural networks. In *Proceedings of the IEEE International Workshop on Multimedia Signal Processing*, 2004.