JOHANNES KEPLER
UNIVERSITÄT LINZ
Netzwerk für Forschung, Lehre und Praxis

Estimating Perceived Tempo of Music Audio Files

Diplomarbeit

zur Erlangung des akademischen Grades

Diplom-Ingenieur

in der Studienrichtung

Informatik

Eingereicht von:
Richard Vogl, 0255290

Angefertigt am:
Department for Computational Perception

Betreuung:
Univ.-Prof. Dr. Widmer
Univ.Ass. Dipl.-Ing. Seyerlehner

Linz, Oktober 2009

Kurzfassung

Schlüsselworte: Musik Informationsgewinnung, Signalverarbeitung, Tempo Extrahierung, wahrgenommenes Tempo, Noten Onset Erkennung, Inter Onset Interval, maschinelles Lernen, Nearest-neighbor Klassifizierung, Supportvectormaschinen, Genetische Algorithmen.

Die vorliegende Arbeit stellt eine alternative Methode zur Tempobestimmung von Musik Dateien vor. In den meisten Arbeiten die sich mit diesem Thema beschäftigen, wird versucht das Tempo eines Musikstückes auf irgendeine Art und Weise direkt aus der digitalen Musikdatei zu *berechnen*. Diese Herangehensweise beinhaltet einige Schwierigkeiten; mit einer davon muss man sich bereits vor dem Berechnungsvorgang herumschlagen: Was ist überhaupt das richtige Tempo eines Musikstückes? Ist es das notierte Tempo im Notenblatt oder eher das Tempo mit dem ein Musiker mit dem Fuß wippt während er spielt? In dieser Arbeit wird versucht den Berechnungsprozess von Vorne herein zu vermeiden, indem Machine-Learning Algorithmen verwendet werden. Diese werden mit verschiedenen tempo- und geschwindigkeitsrelevanten Features der Musikdateien aus einer Trainingsdatenbank trainiert. Das Tempo von unbekannten Musikstücken wird dann mithilfe der trainierten Machine-Learning Algorithmen geschätzt.

Diese Technik wurde bereits von [SWS07] vorgestellt. In der vorliegenden Arbeit wird diese Technik weiter vorangetrieben um bessere Klassifikationsergebnisse zu erzielen. Es kommen andere Machine-Learning Algorithmen, mehr Features und ein zweistufiger Klassifizierungsalgorithmus zum Einsatz. Die so gewonnenen Ergebnisse werden dann unter Zuhilfenahme einer Benutzerumfrage, in der Testpersonen die Geschwindigkeit zweier Musikstücke vergleichen, evaluiert. Dieser

Schritt ermöglicht es das fragwürdige ‘Beats per Minute’ (BPM) Tempo, mit dem die Dateien in der Trainingsdatenbank annotiert wurden, zu umgehen.

Die Ergebnisse sind vielversprechend, obwohl sie auch klar zeigen, dass im Bereich ‘Perceptual Tempo Estimation’ noch viel Forschung und Arbeit investiert werden muss.

Abstract

Keywords: Music information retrieval, signal processing, tempo extraction, perceptual tempo, note onset detection, inter onset interval, machine learning, nearest neighbor classification, support vector machine, genetic algorithm.

This work is about an alternative method for tempo extraction of music audio files. Most works on this topic try to *compute* the tempo of a piece of music in some way using the waveform information stored in digital files. This method bears some big problems; one of these raises already before the calculation process starts: What is the correct tempo of a piece of music? Is it the score tempo notated in the sheet music? Is it the tapping tempo, with which a musician would tap along with his foot while playing?

In this work we try to omit the process of calculation by using machine learning algorithms. They are trained with different tempo and speed related features from audio files which's tempi are known. The tempi of unknown music pieces are then estimated using the trained machine learning algorithm.

While this method was already introduced by [SWS07], we push the work further by trying to improve the estimation results using different machine learning algorithms, more features and a two level estimation process. The results are furthermore evaluated using a survey where probands compare the speed of different music pieces. We omitted in this way the questionable annotated 'beats per minute' (BPM) tempo of the files in our training databases.

The results are promising, although they also show that much research and work has to be done in the field of perceptual tempo estimation.

Contents

1 Introduction

1.1 Problem

Estimating the tempo of music is one of the fundamental problems in music information retrieval. Although it is quite simple for most humans to tap along with the rhythm of a music piece, the problem can not yet bee seen as satisfyingly solved for all kinds of music. The tempo of a piece of music is a very basic property and therefore a very important measure for music similarity. Tempo information can be used with any application based on music similarity, such as automated music/artist recommendation, playlist generation and music library arrangement. Exact tempo information could also be useful for music transcription and score following. In art and entertainment tempo and rhythm information can be used for visualization purposes such as automated light shows, as support tool for VJ's (video jockeys), other visual effects, etc ... There are programs and tools which can estimate the pace of electronic dance music quite accurate. They usually use low pass filtering and a very simple onset detection mechanism (often realized in hardware).

Given the very specific limitations of this kind of music (strong, monotonous and tempo defining beat), such tools can't be seen as a solution of the tempo estimation problem. It is already quite hard to estimate tempo of non-electronic music with strong rhythms (pop/rock), the problem gets even harder with classical music and other kinds of music with even less defined rhythmic patterns such as 'ambient'.

1.1.1 Types of Tempo

What is the tempo of a piece? This question is very fundamental but still can not be answered globally. We could claim that a piece of music has several tempi depending on the level of observation and/or application (compare [CL07]).

- **Score tempo** If existent, the score tempo would be the tempo notated in the score sheets (in classical music e.g.). However, the real tempo of the piece depends on how it is performed. Even if we synthesize the score by means of midi reproduction, the perceived tempo could still differ from the score tempo. Usually the perceived tempo would then be a integer fraction/multiple of the score tempo.

- **(Foot) tapping tempo** The tapping tempo would be the frequency with which a human being would tap subconsciously to the music. Usually this would represent the perceived tempo. However, tapping experiments like in [DGP99] revealed that the tapping tempo depends on multiple factors like age and mood and is not consistent when comparing results of different subjects.

- **Fastest event tempo** When looking at an onset-detection algorithm output (see Section 4.3) we could try estimating the tempo calculating the smallest common divisor of the inter onset intervals (times between the onsets, see Section 4.5). The result would be the fastest regular beat in that specific piece of music. If we found the right multiplier, we could obtain something like the perceived tempo through the fastest event tempo. However, this multiplier is likely to be different for every piece of music.

- **Rate of measure** The rate of measure depends on the score tempo and on the meter of the piece. The length of one measure can be calculated from the score tempo and the meter of the piece, therefore the rate of measure r_m is defined as $r_m = 1/t_m$, where t_m is the length of one measure.

1.2 Goal

The goal of this work is to find a method to sort music according to their perceived tempo. This has not inevitably to be the score tempo nor tapping tempo. What we are looking for is rather a mechanism which assigns music pieces a label which allows a measure of similarity and order. Human probands should then confirm the correctness of the assigned tempo in a survey. This is the only way to evaluate the quality of the calculation of perceived tempo, since ground truth data in beats per minute rather correspond to score tempo then to perceived tempo.

We try to achieve this goal by omitting any tempo calculation process/algorithm and rather use machine learning to estimate the tempo. A possible problem we might encounter lies in the training and test databases, which consist of files with annotated BPM tempo, which might or might not be consistent with the perceptual tempo. A better training and test database would consist of music files which are sorted according to their perceived tempo.

2 Related Work

There are several works on tempo induction. We just want to present some few works which are relevant for this work. In the course of MIREX'06, for example, a tempo induction contest took place with several contestants. The algorithms presented at this contest performed quite well and take use of interesting methods.

[ADR06] estimates tempo and tempo phase by using onset detection in eight subbands. The periodicity of this onset functions is calculated using autocorrelation, spectral product and spectral sum (see also [ADR04]). Then dynamic programming is used to determine and track the optimum paths of tempo candidates. The best paths are then time-averaged. For the MIREX'06 contest a second tempo was calculated by using a single fusion method using majority rules for all paths resulting from the single periodicity methods (autocorrelation, spectral product and spectral sum).

[DB06] uses also an onset function combined with autocorrelation to find the tempo. He uses a shift invariant comb-filter to extract the primary tempo. A secondary tempo is then extracted using the primary tempo. The tempo phase is calculated using a weighted impulse train which is correlated with the onset function.

[Eck06] uses autocorrelation and additional phase information to enhance the results. This method was originally design to gather meter information of music.

[Pee06] uses also an onset function and their autocorrelation function (ACF). Additionally he uses the discrete fourier transform of the onset function which provides a similar result as the autocorrelation function. The two signals are then combined

to find the tempo using a combination of a Viterbi decoding algorithm (to find the most probable 'tempo state' path) and a beat marking algorithm.

In [Set06] the so-called 'Periodicity Transform' (compare [SS01]) is used on features extracted from the audio material.

[Uhl06] uses again spectrogram subbands to compute an onset function like 'accent signal'. Here the subbands are wighted by the amount of rhythmical information contained in the band (to get the amount of rhythmicity, the ACF is used again). The tatum[1] values are then calculated using two methods, namely: ACF and a two-way mismatch error function. Another ACF is calculated using the accent signal. To obtain the tempo values some kind of expert system and a voting mechanism is used.

[GD06] proposes several algorithms working on the same principle: Extracting different tempo relevant features from the audio material and then calculating the autocorrelation function of these features over time. The tempo phase is, again, computed using a impulse train which is correlated with the feature functions.

Meter analysis could help to find the rate of measure and would provide additional rhythmical information that could lead to the perceived tempo. [KEA06] presents a method which takes use of hidden markov models to gather information about the rhythmical structure of music data.

[PT07] uses a so called 'energy envelope' which is quite similar to an onset function. They than compute a self-similarity matrix using a distance function (similar to the AMDF — see section 4.4). The tempo is selected using 'rhythmical signature' clustering.

[SWS07] introduced at ISMIR'07 a very genuine and new approach on tempo induction. In this work machine learning is used in a second stage to omit the often

[1] The Tatum grid is the 'lowest regular pulse train that a listener intuitively infers from the timing of perceived musical events'. The grid can be computed by using a histogram of inter-onset intervals. The term was coined by J. A. Bilmes in an MIT Master's thesis Timing is of essence published in 1993 and named after the jazz musician Art Tatum. Compare `http://en.wikipedia.org/wiki/Tatum_grid`

empirically designed tempo calculation processes. In a first stage tempo related features are calculated from the audio signal. This features are the autocorrelation function of a simple onset function and a modified, compressed version of the fluctuation pattern (which was introduced in [Pam01]). The features with the corresponding beat per minute (BPM) values form the ground truth for a k-nearest-neighbor (KNN) machine learning algorithm. The feature vectors of new songs which's tempo should be determined are then classified using the KNN classifier. The results obtained with this approach are at least comparable to the results in the tempo estimation contest of MIREX'06.

[Ahm07] is an interesting work on how tempo perception influences the human. They measured how runners respond to music with different tempo in terms of step frequency and motivation.

3 Algorithm Overview

The algorithm implemented in this work is based on the idea of [SWS07] to use features and a machine learning algorithm to estimate the tempo of a given music file. Where [SWS07] only uses two different features—namely ACF (see section 4.4) and FP (see section 4.6)—and solely KNN-classification (see section 5.1) this work tries to improve the results by engaging at following points:

- **Using more features:** As described in section 4, a broad variety of features was tested in this work to find a good feature set for tempo estimation using machine learning algorithms.
- **Comparing different machine learning algorithms:** Aside from nearest neighbor classification (NN, KNN, see section 5.1) we evaluated the capabilities of support vector machines (SVMs, see section 5.2), a very powerful machine learning algorithm.
- **Using a two stage classification algorithm to engage the double/half tempo problem**

As already stated, we rather use tempo relevant features (like e.g. [GD06]) than directly audio/spectral data or an onset function calculated from this data—although we use the onset function to calculate some features. We furthermore use just small chunks of audio material instead of the whole file. This reduces calculation duration of the time consuming feature calculation process, but also brings some problems as described in section 9. To further reduce calculation time for evaluation runs, we implemented a caching system for features. We store the extracted feature information for every single audio file. As long as the features do not change we do not have to recalculate them. The proposed algorithm uses

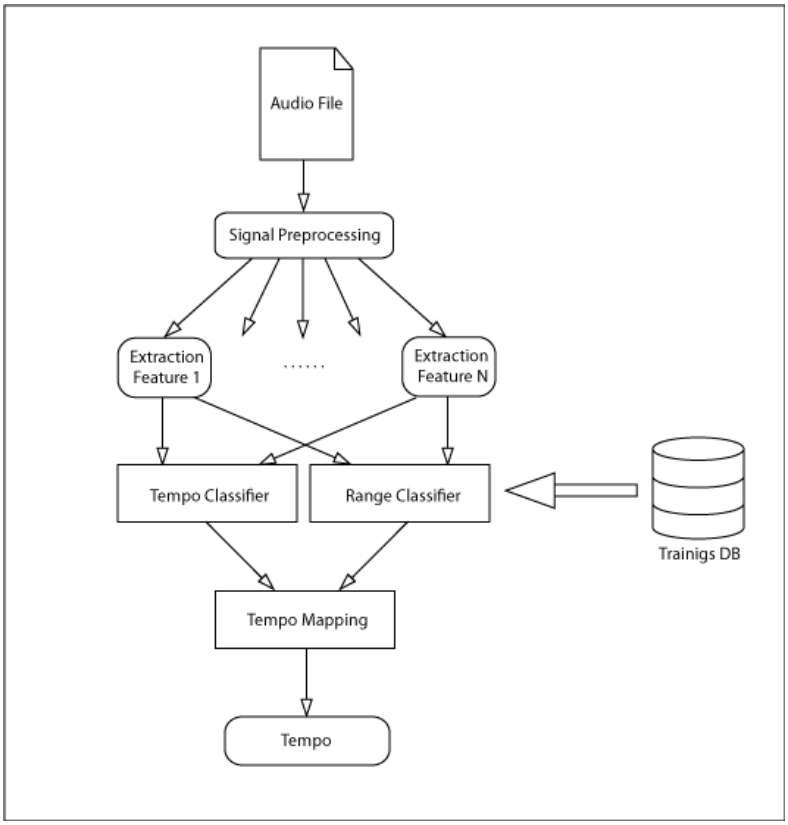

Figure 3.1: The single steps of the tempo extraction algorithm.

two stages in the classification process. As stated in [SWS07], the tempo estimation algorithm often estimates the tempo double or half of the real tempo. This double/half errors could be avoided using the following two-staged classification process: In the first stage a tempo-range is classified. Meaning that the music file is assigned to one of N bins which correspond to a certain range of tempo. In the second stage the tempo is estimated similarly as it is done in [SWS07]. Then it is tried to combine the results of the two stages. In this work two methods are proposed to do so: A kind of mapping into the estimated range and secondly the estimation of the tempo just within the estimated tempo range. Figure 3.1 shows an overview of all stages of the algorithm.

4 Feature Extraction

In this chapter all the features used for tempo estimated are introduced and their calculation schemes and algorithms are explained. Furthermore, the preprocessing steps for feature calculation or important algorithms and their parameters used by feature calculation methods are explained.

4.1 Signal Preprocessing

To have a common point to start, every audio file is preprocessed and converted to a certain wave format. Since we are not going to use the whole song—in some training sets there are already just excerpts (see Section 7.1)—a section from the middle of the file is used. Considering the silence at the beginning and ending of music files and a possible intro/outro this seems to be a better choice than starting at the very beginning of a file. If the contained audio signal is long enough, up to 12 seconds are used. If the file contains less than 12 seconds of audio material, the whole content is used.

4.1.1 Implementation

The first step is to analyze the audio files. For further processing we want the data in 8kHz sampled one-channel (mono) audio. The down-sampling is done using the Matlab® built in function `resample`, which performs also a low pass filtering to meet the Nyquist-Shannon sampling theorem.

Preprocessing of the files happens in the first few lines of `r_extract_features.m`. First the files sample rate and format is read, then the data (up to 12 seconds from the middle of the file) is read. Afterwards the data is converted to the already mentioned 8kHz mono format.

4.2 Frequency Transform

In signal processing, transforming the source signal into a frequency domain is a very common step. It spreads out the sum of all the single frequency-components and makes it possible to extract single events and information about this events. The most common way is to use the fourier transform ([Wik09b]) where the components of the signal are calculated by means of sines and their phase. The formal definition for a continuous signal $x(t)$ is:

$$\widehat{x}(\omega) = \frac{1}{2\pi} \int_{-\infty}^{\infty} x(t) \cdot e^{i\omega t} dt. \tag{4.1}$$

Where $\widehat{x}(\omega)$ are the complex frequency components of frequency ω. To calculate the single values of a discrete signal x_n the discrete fourier transform (DFT) is used. The single frequency values $\widehat{x}_\omega$ are given by the following equation:

$$\widehat{x}_\omega = \frac{1}{2\pi} \sum_{n=0}^{N-1} x_n \cdot e^{i2\pi \frac{kn}{N}}. \tag{4.2}$$

If we want to perform a DFT for signals with sizes N which are a power of two number, we can use the so-called 'fast fourier transform' (FFT) which is a very effective way of calculating the DFT. The FFT has a runtime complexity of only $O(NlogN))$ while the traditional DFT has an $O(N^2)$.

If we divide the whole signal into small windows and perform a DCT (or FFT respectively) for every window, we get a so-called 'short-time-fourier-transform' (STFT) spectrogram. We have now a two dimensional matrix with the single frequency components as complex values (which can be interpreted as phase and magnitude pairs) dependent of time (usually x-axis) and frequency (y-axis). Inherent for this kind of transform is the usage of 'windows' and 'window overlap'.

The spectral information is always calculated for a certain window (number of digital values of audio signal). Whereas the window overlap indicates the amount of samples reused in the next window. Since we are mainly interested in the rhythmical information a relative small window size, which results in a low frequency resolution, will do. On the other hand a bigger window-overlap is desired since we want to locate the acoustical events exactly. Most features which are used in this work are based on some kind of spectrogram calculated from the raw audio data. There are some possibilities to alter the spectrogram and/or calculate it in a slightly different way. The following chapter is a short discussion of different possibilities for altering a STFT spectrogram.

Frequency Resolution Since the human auditory system does not work linearly in frequency space, the interpretation of the spectrogram in hertz (Hz) as we receive it from any fourier transform may be inadequate. There have been introduced some alternative frequency scales which are more conform with the human auditory system.

1. **Chromatic Scale:** The idea is to use only frequency bands which correspond to tones in an equally spaced scale. If we use A4 as our base tone ($n = 0$) for note frequency calculation, the single note frequencies for a chromatic scale can be calculated using:

$$f(n) = 440 \cdot \left(\sqrt[b]{2}\right)^n \tag{4.3}$$

   Where $f(n)$ is the frequency of the n^{th} tone starting with $n = 0$ at A4 (440Hz). b is the number of tones in an octave which is usually 12. If we want to divide the octave into cents, we can use $b = 12 \cdot 100$. Naturally a transformation from Hz to a chromatic scale adds additional calculation time. Experimentation showed that algorithms like the 'goertzel algorithm' ([Wik09c]) or comb filters are much slower than performing a full range FFT. Empirical evaluation showed that there were no significant improvements for onset detection.

2. **Mel Scale**: The Mel scale (based on melody) is a frequency scale proposed by [SVN37] based on pitch comparison experiments. The conversion from

Hz to Mel can be done using this equation:

$$m = 1127.01048 \cdot \log_e (1 + f/700) \tag{4.4}$$

Since this scale represents properties of the human perception of tone heights its often used for feature calculation which are used in music-similarity applications.

3. **Bark Scale**: The Bark scale is somewhat related to the Mel scale, but based on different experiments. The scale is divided in 24 'critical bands' , based on the human auditory system (see [Zwi61]). The formula to calculate the Bark frequency band is given by:

$$b = 13 \cdot \arctan(0.00076 \cdot f) + 3.5 \cdot \arctan\left((f/7500)^2\right), \tag{4.5}$$

or can be approximated—if Mel frequency is given—by 1 Bark $\approx$ 100 Mel.

Absolute Magnitudes of Spectrogram If we take the absolute values of the complex number of the STFT spectrogram, we get the magnitudes of the sines which are part of the original signal. This values are proportional to the sound pressure (which is converted by a microphone and digitalized by a analog digital converted). If the complex value C is given, we obtain the magnitude using:

$$r = |C| = \sqrt{\Re(C)^2 + \Im(C)^2} \tag{4.6}$$

Again, the human auditory system does not perceive volume linearly or proportionally to sound pressure, therefore there exist some alternative scales.

1. **Decibel Scale**: Bel is a logarithmic unit of measurement that expresses the magnitude of a certain sound pressure relative to a reference level (usually the threshold of human hearing 0B, $p_{ref} = 20\mu Pa$. Usually the values are given in decibel (dB = $1/10Bel$). Given the sound pressure p the dB value can be calculated as

$$L_p = 20 \cdot \log_{10} \left(\frac{p}{p_{ref}} \right) [dB] \tag{4.7}$$

The sound pressure values provided are usually the RMS (root mean square) values. The multiplication with 20 results from the factor 10 to get the result in *deci* Bel. The factor 2 can be explained with the definition of 1 Bel, which is the logarithmic proportion of two power/energy values. Since the voltage induced by the microphone is (in a certain range) proportional to the sound pressure (therefore $U = p \cdot k$, where k is a constant provided by the microphone) and the power of the voltage is given by $P = U^2 \cdot R$, the initial equation for L_p could be written as:

$$L_p = \log_{10}\left(\frac{(p \cdot k)^2 \cdot R}{(p_{ref} \cdot k)^2 \cdot R}\right) = \log_{10}\left(\frac{p^2}{p_{ref}^2}\right) = 2 \cdot \log_{10}\left(\frac{p}{p_{ref}}\right)[B] \quad (4.8)$$

In most cases we do not really know how loud the given wave signal is compared to the threshold sound pressure. We can therefore reformulate the formula for calculating the dB value to:

$$L_p = 20 \cdot \log_{10}(p) - 20 \cdot \log_{10}(p_{ref})\,[dB] \quad (4.9)$$

Since p_{ref} is constant the whole expression on the right side of the minus sign is a constant, resulting in

$$L_p = 20 \cdot \log_{10}(p) + L_{p_{ref}}[dB]. \quad (4.10)$$

The reference value is therefore just a additive expression which can be omitted if we normalize the calculated results anyway. Furthermore this additive expression is changed if the signal is amplified while playing (which is in fact done every time music is played since the reproduction with the exact same volume is practically not possible) and therefore not important, if we do not want to compare the absolute volume of two different music pieces.

2. **Energy Spectrogram**: Using the above defined value for the power induced by the microphone (which is proportional to energy of the signal) we get:

$$E = p^2 \cdot k \quad (4.11)$$

Omitting again the multiplicative constant we simply get $E = p^2$.

3. **Phone/Sone Scale**: Phone was proposed as perceived loudness level by S. S. Stevens. The scale is designed to compensate the difference on perceived loudness with same sound pressure level (SPL) over different frequencies. One phon is defined as a 1kHz tone with 1dB ($p_{ref} = 20\mu P$) SPL. Sone was also proposed by S. S. Stevens in 1937 and is a measure for perceived loudness too. One Sone is defined as 40 Phon which is a 1Khz tone with 40dB ($p_{ref} = 20\mu P$). The scale is then designed so that a doubling of the Sone value is perceived as doubling of the loudness. The Sone value of a signal also depends on the frequency of the signal. See also [Wik09e] and [FZ99].

Phase Spectrogram Naturally we can also use the phase information encoded in the complex values. If C the complex value is provided, we receive the phase values using:

$$\varphi = \angle(C) = \begin{cases} \arccos\left(\frac{\Re(C)}{|C|}\right) & \text{if } \Im(C) \geq 0 \\ -\arccos\left(\frac{\Re(C)}{|C|}\right) & \text{if } \Im(C) < 0 \\ \text{undefined} & \text{if } |C| = 0 \end{cases} \tag{4.12}$$

The problem with phase values is, that a stationary sine has a constant value in the STFT spectrogram, but his phase will constantly change at a given rate proportional to the frequency. Furthermore the mapping into the range $[-\pi...\pi]$ causes discontinuities in the phase spectrogram. This problem can partially be solved using the first differential of the phase spectrogram, resulting in almost constant values for stationary sine signals. This procedure is illustrated in Figure 4.1 where phase values are used to calculate an onset function.

Another way to address this problem would be to calculate the expected phase change rate for a stationary sine in frequency bin n and subtract the expected values from the actual values. This would result also in almost constant values for stationary sines which are exactly in bin n and linear raising values for sines close to the bin. The expected values can be calculated using:

$$\phi_{expected} = t \cdot f_n \cdot 2\pi \tag{4.13}$$

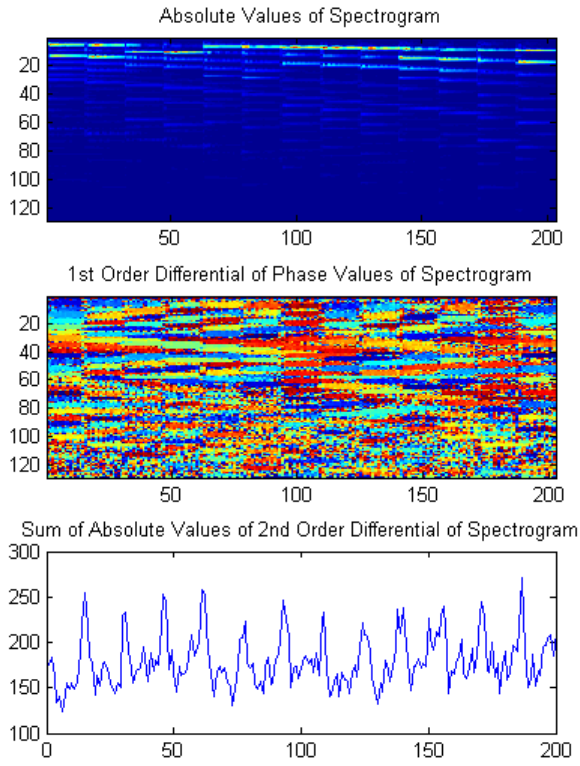

Figure 4.1: Using phase information to extract onset function

Where t is the time of the current frame of the STFT spectrogram and f_n is the frequency of the current bin.

The final feature extraction algorithms use multiple and different kinds of spectra since every spectrogram has its own advantages and disadvantage.

implementation For the frequency transform process implemented in Matlab® the `spectrogram` function is used. The frequency scale and magnitude scale transformation is performed after the spectrogram calculation using helper functions like `melscale.m`. The frequency transformation is performed in the `onset.m` function at the very beginning. Parameters which influence the frequency transform are also passed to `onset.m` and are:

- `usemel` if true, spectrogram will be transfered into melscale before onset function calculation.
- `usephicorr` if true, phase values will be corrected by means of removing phase growth for stationary signals.
- `lpfrequ` lowpass frequency for spectrogram. if not equal zero, all signal parts below lpfrequ will be canceled out before calculation.
- `p` parameter struct with FFT parameters.
  - `p.fs` sample frequency in Hertz (8000 Hz)
  - `p.fft_time` FFT window in milliseconds (32 ms)
  - `p.fft_hoptime` FFT hop-time in milliseconds (4 ms)
  - `p.mel_filters` number of Mel filter banks (40)

4.3 Onset Detection

Music can be interpreted as a list of discrete acoustic events—similar as music is stored in midi files. Such acoustic events can be broken down into several parts: 'onset,' 'attack,''transient' and 'decay'. In Figure 4.2 the single parts are illustrated in context of a played note. Most acoustic events can be described by this model although the instances can assume very different shapes. E.g. a single violin note can extend over several seconds which means it can have a very long transient, talking in terms of our model. On the other hand percussive signals such as finger snaps for example, can be very short with an overall duration of a split second. The onset is therefor a certain point in the described model and in time—namely the very beginning of such a discrete acoustic event. Onset detection is the search for these beginnings. The acoustic events are mainly notes played by instruments including/especially percussive events (see [Kla99]). Whenever working with rhythm, tempo or notation in terms of audio signals, finding the discrete events is a crucial step. Since we want to know the exact time an event

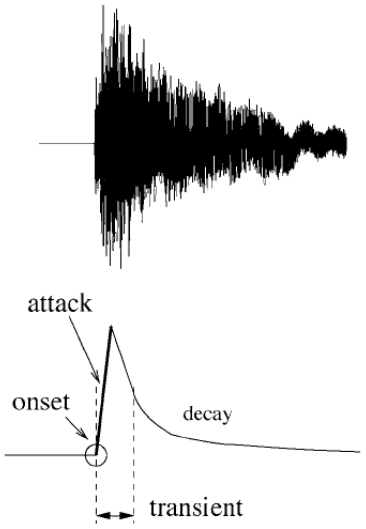

Figure 4.2: 'Onset,' 'attack,' 'transient' and 'decay' in the case of a single note (in [BDA+05]).

starts, it is the onset we are looking for. Onset detection usually takes place in two stages (see [BDA+05] and [Dix06]):

1. Calculation of onset function
2. Selection and labeling of prominent peaks of the onset function

4.3.1 Onset Function Calculation

In the first stage a so-called onset function is calculated. In an onset function every value corresponds to a certain point of time in the audio signal. Peaks in the onset function indicate onsets in the audio signal. [BDA+05] does not use the term onset function but rather calls the calculation of his onset functions 'reduction'. The onset of a new note played usually results in some kind of change in the audio signal. Onset function calculation therefore aims at reflecting the amount of 'change' occurring in the signal. The most simple way to do this might be to use the envelope of the audio signal (see [BDA+05]). To calculate a simple onset function one could calculate the derivative of the envelope. The envelope can be

calculated via squaring and low-pass filtering the signal. This could be done using following equation:

$$e(t) = \frac{\sum_{i=-w/2}^{w/2} s(t+i)^2}{w} \tag{4.14}$$

Where $s(t)$ is the source audio signal, $e(t)$ the envelope and w the window size for smoothing (low-pass filtering). This window size strongly depends on the sample rate. For envelope calculation windows with a length around 25ms perform well. Using the above (Section 4.1) mentioned sample rate of 8kHz this would result in a window of 200 samples. Once the envelope is calculated it can be downsampled to a rate of 100Hz. This is about the threshold of the rate at which the human ear can distinguish between two acoustical events (see [Moo97], [Dix06] uses the same f_s for his onset functions). Now we can calculate an onset function using the discrete first differential of the envelope $e'(t)$.

$$e'(t) = e(t) - e(t-1) \tag{4.15}$$

This procedure is illustrated in Figure 4.3. However, the envelope is not a very robust way to calculate an onset function. A very common strategy to obtain onset functions is via the spectrogram. In case of [DSD02], [Kla99], [NTI04] and [BDA+05] the audio signal is first split into a number of frequency subbands using filters. For these subbands the onset function is separately calculated. This single onset functions are then combined again in different ways. The method of splitting up the signal into different frequency ranges follows the human auditory system which also treats frequency bands separately like stated in [Sch98]. [Dix06] presents several algorithms to extract robust onset functions from spectral information omitting splitting up the signal into subbands. In this work the exact onsets are not required but rather a tradeoff between onset labeling precision and calculation time had to be found. For this reason splitting up the signal into subbands is not used since that multiplies the calculation time by the number of subbands used. Most of the algorithms which are implemented in this work are taken from [Dix06]—some have been altered slightly. The methods which have been analyzed further are listed below:

- Spectral Flux:
  If we take the absolute values of the STFT spectrogram, one can observe dis-

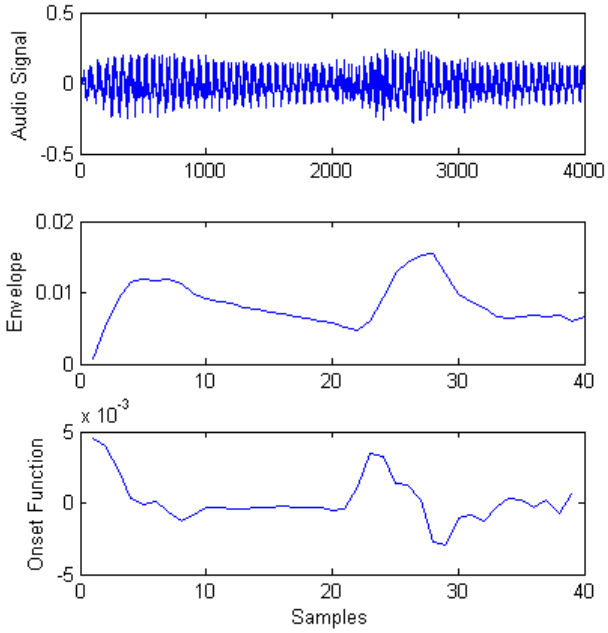

Figure 4.3: Calculating onset function via signal envelope.

continuities along the time axis whenever a note onset occurs. The idea is to extract this discontinuities by means of calculating the first order derivative over time of the absolute values of the spectrogram. If we now take only the positive values of the derivative of the spectrogram (half-wave rectified) and sum over all frequency bins, we get a one-dimensional onset function. This onset function is then defined as:

$$SF(n) = \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} H\left(|X(n,k)| - |X(n-1,k)|\right). \tag{4.16}$$

Where $H(x) = \frac{x+|x|}{2}$ is the half-wave rectifier function. The procedure is similar to using the derivative of the envelope but with some advantages: Calculating the derivative in frequency domain is more robust in case of multiple or close onsets in different frequency bins. Furthermore low-pass filtering is not needed since the vibrant nature of the audio signal is eliminated due to transformation to frequency domain. Figure 4.4 illustrates the

single steps to obtain an onset function using the equation shown above. [SWS07] uses a similar approach. The only difference is the conversion of

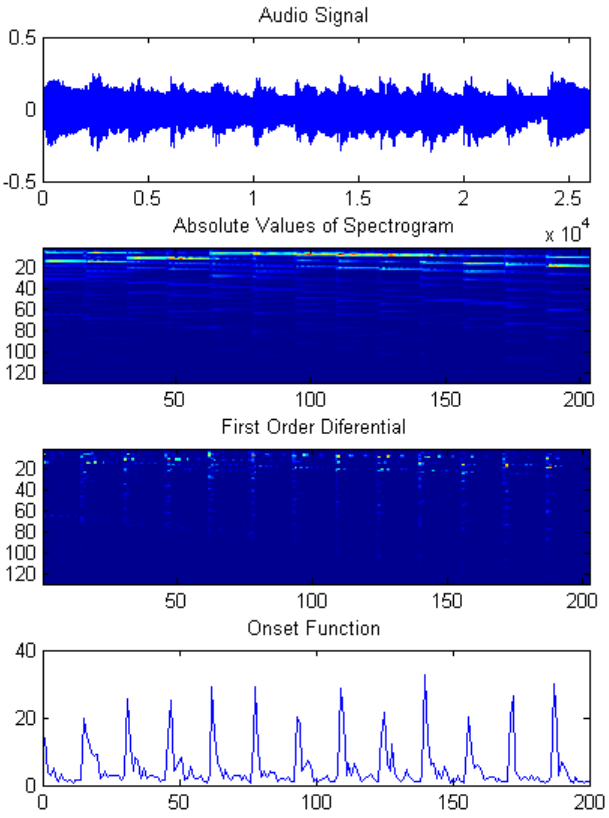

Figure 4.4: Using spectral fluctuation to calculate an onset function.

the spectrogram to a 40-bin mel-frequency spectrogram and using dB for scaling the values (as described in [Ell07]). In this work one can choose by parameter setting if onset function calculation should use a mel-frequency scale or not. Conversion to dB is not implemented since empirical testing did not show any improvement when using dB.

- Phase Deviation:
  Contrary to the method presented before this detection algorithm solely uses the phase information of the STFT spectrogram. In this case the second

derivative is used. This is because the phase in the STFT of a stationary sinusoid is more or less a linear function. Calculating the first derivative produces therefore values proportional to the frequency of the sinusoid. The second derivative then produces a similar image to the first derivative of the absolute values which can be used again to calculate the onset function. To do this the sum over the frequency bands is calculated. The onset function can be written as:

$$PD(n) = \frac{1}{N} \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} |\psi''(n,k)| \,. \tag{4.17}$$

Another very important point that has to be considered is the range of the phase-values which reaches from $-\pi$ to π. After every calculation step (excluding the sum) values which fell out of this range have to be mapped back into it. Otherwise the resulting artifacts might corrupt the onset function quality. The single images from signal via phase diagram to the onset function are shown in Figure 4.1.

- Complex Domain:
  Since we already used the magnitudes and the phase to calculate an onset function the question arises if we could just use the whole complex values to get an onset function which combines both information of phase and value. So what is done here is in fact just the first derivative of the complex values with a corrective term for the phase. Onset function and phase correction values are defined as:

$$CD(n) = \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} |X(n,k) - X_T(n-1,k)| \tag{4.18}$$

  where

$$X_T(n,k) = |X(n,k)| \cdot e^{\psi(i,k)+\psi'(i,k)}. \tag{4.19}$$

  Since the phase of a stationary sine in the spectrum is already changing constantly at a given rate, the expected value for the phase value has to be calculated with $\phi_{expected}(i,k) = \phi(i-1,k) + (\phi(i-1,k) - \phi(i-2,k))$ where $\phi'(i,k) = \phi(i-1) - \phi(i-2)$ describes the rate of change of the sine—which

is proportional to the frequency of the sine. To obtain a real-valued onset function we then sum over the absolute values. Figure 4.5 shows a diagram with two complex values, the expected third value and a real third value which is not part of the stationary signal. One can also observe how the phase difference is used to calculate the next expected value

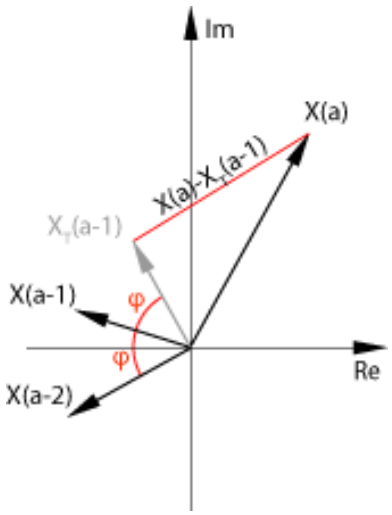

Figure 4.5: Complex values $(X(a)..X(a-2))$ of a frequency bin, expected value for $X(a)$ and difference.

- Weighted Phase Deviation:
  A Problem when using solely phase values to create an onset function is that frequency bands which contain no significant signals have a random phase. This random values now add much noise to the onset function if they are equally treated. The idea here is now to multiply the phase values (their second derivative in fact) with their corresponding absolute values. So that the above mentioned random phase values of nonexistent frequency are canceled out. We define the onset function as:

$$WPD(n) = \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} |X(n,k) \cdot \psi''(n,k)| \,. \tag{4.20}$$

  This is quite similar to the CD function where both phase and value are used for onset function calculation but with a different kind of combination.

- Rectified Complex Domain:
  The onset function is defined as

$$RCD(n) = \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} RCD(n,k) \tag{4.21}$$

  where

$$RCD(n,k) = \begin{cases} |X(n,k) - X_T(n-1,k)|, & \text{if } |X(n,k)| \geq |X(n-1,k)| \\ 0, & \text{otherwise} \end{cases}. \tag{4.22}$$

  This equation is an improved version of the CD procedure. Similar as in the equation for the SF onset function the values are half-wave rectified. This reduces the risk that hard offsets as in staccato played notes and percussive sounds result in peaks in the onset function. To do a half-wave rectification of the complex values, we only use the difference of complex values whenever the difference of the absolute values is positive. Although this is not really half-wave rectification, we call this procedure rectified complex domain onset function because of the similarity to the process of half-wave rectification.

- Selected Phase Deviation:
  Empirical experimentation showed that the WPD function does not really show improvement compared to the results of the PD equation.

$$SPD(n) = \sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} |\psi''(n,k)| \cdot SF(n). \tag{4.23}$$

  This equation presents another way to combine phase and value of the spectrogram. Here the SF onset function (which is calculated solely using absolute values of the spectrogram) is used to weight all bins of the phase spectrogram. Doing this, only onset candidates are considered and amplified if the phase shows also a prominent change. On the other hand false onset function peaks are dampened if the phase is continuous.

As always the source material on which the calculation is based has much influence on the resulting onset function. In case of onset function calculation the source

material is the STFT spectrogram. As already described in Section 4.2 there are two parameters which have strong influence on the resulting spectrogram. This parameters are the window size for which the fourier-transformation is done, and the overlap which defines how many samples are used again by the next window. Instead of overlap many authors use the term 'hopsize' which is defined as $hs = ws - ol$, where hs is the hopsize, ws is the used windowsize and ol is the overlap. Using hopsize instead of overlap has the advantage that when keeping the hopsize constant while changing the window size, the sample rate of the resulting signal stays unchanged. To get a resulting sample rate of more then 100Hz for our onset function, and keeping our audio sample rate of 8kHz we have only a very small range to vary the hopsize: $hs = [1..8000/100] = [1..80]$. Choosing a hopsize of only a few samples results in a very spiky onset signal and is therefor not adequate. [SWS07] amongst others uses a hopsize of 32 samples which proved to work well in this work too. Empirical experiments using 64 samples (the next and last power of two number in that range) as hopsize showed no significant change. Trying to get a resulting sample rate of exactly 100Hz would lead to using a hopsize which is not an integer divisor of the window sizes (which ought to be a n^{th} power of 2 to be able to use the FFT algorithm). This would further lead to complex calculations for chunk overlap when splitting the source audio signal into smaller parts (see Section 6). Furthermore the mapping of the single samples of the onset function back to the original audio signal, which is already quite complex (also see Section 6) because of the nonlinear influence of the window size on the relative position of onset peaks in the onset function, would be further complicated.

The window size has a direct influence on the frequency resolution of the spectrogram. Choosing a very small window size brings the resulting onset functions very close to the envelope signal we discussed before. The separation of the single frequency bands is a very important step towards robustness of the resulting onset function. The larger the window size the better this separation comes out. However, the length of the used window has a second effect on the resulting onset function. The longer the window is, the more it acts as some kind of smoothing operator. The smoothing can be desirable to some point, but at a certain window length onset peaks start to merged. [Dix06] uses a window size of 2048 at a signal

sample rate of 44.1 kHz which results in a window length of 46ms. Keeping the same window length at a sample rate of 8kHz we get a window size of about 372 samples which would suggest to use 256 or 512 samples as window size. Most of the experiments were conducted with window sizes of 256, 512, 1024 and 2048 samples since a window size of 256 samples is still quite spiky. Figure 4.6 illustrates the influence of window size on the onset function.

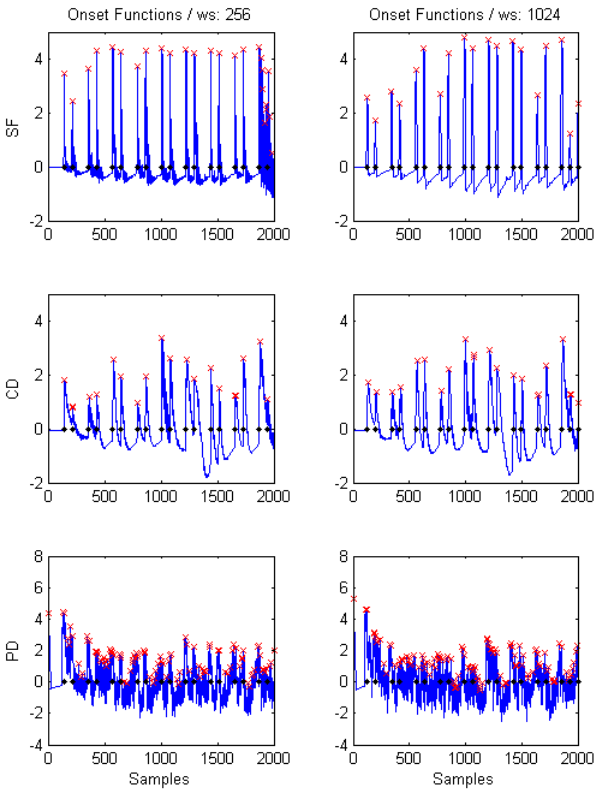

Figure 4.6: Two different window sizes for SF, CD and PD onset function.

A rather unwanted effect of the window used for spectrogram calculation is the fact that the longer the window is, the farther the calculation process at a certain sample n of the audio signal 'looks' into the future. Namely, the sample $n_l = n + ws - 1$ has already an impact to the onset function when we are calculating at the point n (ws be the window size and n_l the last sample of the window used).

The impact of samples far away from the current point of calculation grows as the window size grows. The question is now, how to obtain the exact point in time respectively the sample of the original audio signal to which a certain point (e.g. a peak) of the onset function corresponds. The most obvious solution to this problem is probably following idea: If a peak occurs in the spectrogram, the resulting onset function will have a maximum exactly when the window spans over the whole onset and the onset is exactly in the center of the window. This would result in the mapping function:

$$s_a = (s_{os} - 1) * hs + \frac{ws}{2} + 1 \tag{4.24}$$

$$t(s_{os}) = \frac{s_a}{f_s} = \frac{(s_{os} - 1) * hs + \frac{ws}{2} + 1}{f_s}, \tag{4.25}$$

where $t(s_{os})$ is the time and s_a the sample of the audio signal to which the point s_{os} (number of sample) of the onset function will be mapped, hs is the hopsize ws is the window size and f_s the sample frequency of the audio signal. Unfortunately empirical testing on this problem has shown that the fraction of the window length which has to be added is not constant and the relation to the window size not even linear. The phenomenon which can be observed is that the larger the window size is, the more of the onset in the spectrogram has to be in the window to produce a peak in the resulting onset function. Furthermore the delay depends also on the type of onset function calculation used. For a small range of window sizes (128-512 samples) this can be more or less successfully modeled with a linear correction factor. To get the precise delay, which is needed for accurate evaluation over different window sizes, we have to formulate a nonlinear function based on empirical data obtained from experiments with different window sizes with every onset function. Figure 4.7 shows an excerpt of a table with different onset-delays resulting from different window sizes. The expected onsets where taken from corresponding midi files and given in seconds. To get the expected sample point we used equation 4.25 avoiding any correction and transforming it to get:

$$s_{os\ \text{expected}} = \frac{t_{os\ \text{real}} \cdot f_s - 1}{hs} + 1. \tag{4.26}$$

The estimated peaks were selected by the in Section 4.3.2 described peak selection algorithm. The differences in column five are calculated with $d = s_{os\ \text{expected}} -$

	STFT		s_{os}				
	Hopsize	Winsize	exp.	calc.	diff	% wl	%wl/Log(wl)
os1	32	128	794,3	793	1,3	32,50%	4,64
	32	256	794,3	790	4,3	53,75%	6,72
	32	512	794,3	786	8,3	51,87%	5,76
	32	1024	794,3	775	19,3	60,31%	6,03
	32	2048	794,3	752	42,3	66,09%	6,01
	32	4096	794,3	705	89,3	69,77%	5,81
os2	32	128	1227	1225	2,0	50,00%	7,14
	32	256	1227	1222	5,0	62,50%	7,81
	32	512	1227	1216	11,0	68,75%	7,64
	32	1024	1227	1204	23,0	71,88%	7,19
	32	2048	1227	1182	45,0	70,31%	6,39
	32	4096	1227	1135	92,0	71,88%	5,99
os1	64	128	614	614	0,0	0,00%	0,00
	64	256	614	612	2,0	50,00%	6,25
	64	512	614	609	5,0	62,50%	6,94
	64	1024	614	603	11,0	68,75%	6,88
	64	2048	614	591	23,0	71,88%	6,53
os2	64	128	830,3	830	0,3	15,00%	2,14
	64	256	830,3	829	1,3	32,50%	4,06
	64	512	830,3	826	4,3	53,75%	5,97
	64	1024	830,3	821	9,3	58,12%	5,81
	64	2048	830,3	810	20,3	63,44%	5,77
						Mean	**5,80**
						Std	**1,79**

Figure 4.7: Samples of onsets (SF function) using different window sizes resulting in different delays.

$s_{os\ \mathrm{real}}$. The percentage of this difference in relation to the window size is then given as $p_{ws} = \frac{d \cdot hs}{ws}$. As expected we can also observe that the hopsize has no influence on the delay. Figure 4.8 shows the nonlinear relation between the percentage of window size added as delay and the window size itself. In booth images one can observe that this relation seems to be of logarithmic nature. The last row of the table in Figure 4.7 shows the results if we calculate

$$k = \frac{ws}{\log_2(p_{ws})}. \tag{4.27}$$

k stays almost constant for most onsets and window sizes. If we take in account the little variation of calculated peaks, the standard deviation of k seems acceptable.

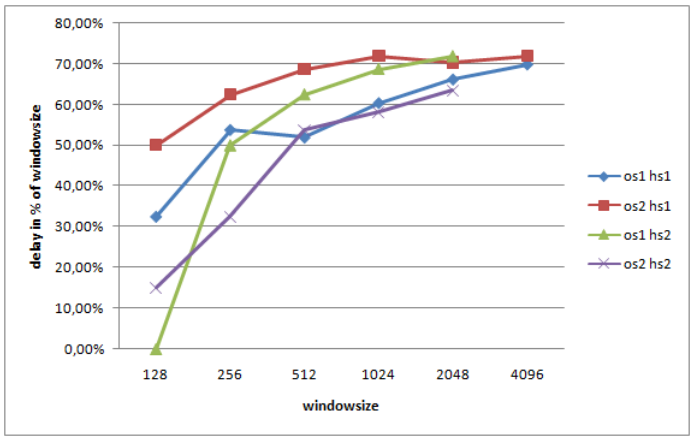

Figure 4.8: Nonlinear relation between window sizes and delay as percents of window size.

If we now create an equation which expresses this logarithmic relation, we get

$$c(ws) = \log_2(ws) * ws * \overline{k} \tag{4.28}$$

as corrective factor. Where $\overline{k}$ is the mean of the in Figure 4.7 calculated values for k. To get this correction value a table with 8 different onsets for every onset functions was created. The resulting equation for $t(s_{os})$ is then given by

$$t(s_{os}) = \frac{(s_{os} - 1) * hs + c(ws) + 1}{f_s}. \tag{4.29}$$

With this approximation formula for $c(ws)$, the results for mapping from onset function samples back to time values is significantly improved for a wide range of window sizes. This is important especially for evaluating the quality of onset detection in Section 4.3.3.

The function which uses the results of onset function calculation should be provided with uniform data no matter which onset function calculation algorithm was used. To normalize the onset function we use mean and standard deviation of the onset function to eliminate DC offset and scale the y-values to a certain level. This procedure can be mathematically written as

$$f_n(x) = \frac{f(x) - \mu_f}{\sigma_f} \tag{4.30}$$

where

$$\mu_f = \frac{1}{N} \sum_{x=1}^{N} f(x) \tag{4.31}$$

and

$$\sigma_f^2 = \frac{1}{N} \sum_{x=1}^{N} (f(x) - \mu_f)^2. \tag{4.32}$$

4.3.2 Onset Peak Selection

After calculating the onset function we have to find the prominent peaks in the onset function. These peaks represent the onsets. Bello et. al. ([BDDS04], [BDA+05]) solely uses an adaptive thresholding to separate significant peaks from noise. The peaks are then identified using every local maximum exceeding the threshold. This algorithm only works satisfyingly if the onset function is very robust containing very little noise and onsets cause distinct peaks. [Dix06] uses a set of constraints which have to be satisfied for detecting a peak. Since this method is more robust when dealing with noisy onset functions it seemed the better choice when working with more simple onset function generation algorithms. The constraints used and their parameters have a big impact on the quality of the peak-picking algorithm. A tradeoff has to be made between false positives and false negatives returned. If we take for example the threshold, less peaks are return when increasing the threshold. Assuming that the false positives are less prominent peaks, the threshold is directly proportional to the ratio of false positives to false negatives. In this work it is more important to receive significant peaks from the peak picking algorithm rather then receiving all onsets. Required are only the onsets which define the tempo.

The peak selection algorithm implemented in this work follows the implementation in [Dix06]. A peak in the onset function has to meet several conditions to be

recognized. This conditions are:

$$f(n) \geq f(k) \text{ for all } k \text{ such that } n - w \leq k \leq n + w \tag{4.33}$$

$$f(n) \geq \frac{\sum_{k=n-m\cdot w}^{n+w} f(k)}{m \cdot w + w + 1} + \delta \tag{4.34}$$

$$f(n) \geq g_\alpha(n-1) \tag{4.35}$$

where $2{\cdot}w{+}1$ is the size of the window in which the peak has to be a local maximum. $m \cdot w + w + 1$ is the size of the window for which the mean is calculated. A peak has to be higher than this mean plus the threshold δ. g_α is a dynamic threshold function which follows the signal slowly, given by:

$$g_\alpha(n) = \max\left(f(n), \alpha \cdot g_\alpha(n-1) + (1-\alpha) \cdot f(n)\right) \tag{4.36}$$

The parameter α regulates the pace at which the function follows the signal and has to be in the range $0 < \alpha < 1$. Figure 4.9 illustrates the effects of every constraint on the peak picking algorithm. As shown in Figure 4.9, the function

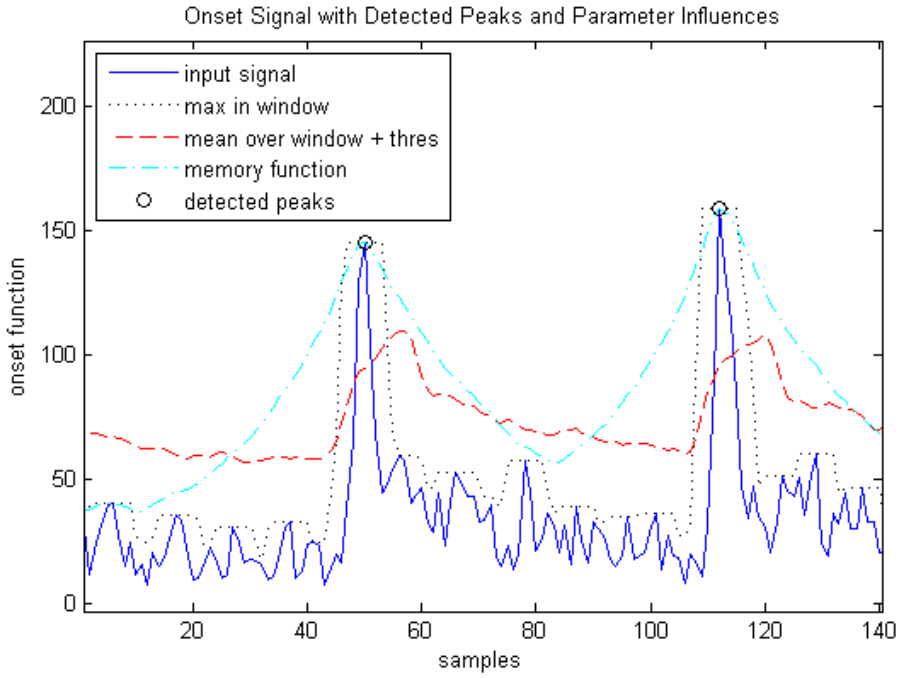

Figure 4.9: Onset function, peak picking criteria and the resulting peaks selected.

g_α has been enhanced in this work. It does not only takes in account peaks before but also peaks after the current sample. The idea was to enhance peak detection for noisy and spiky onset function signals where some smaller peaks could appear before the real onset peak. The new formula for g_α is then given by:

$$g_{\alpha_1,\alpha_2}(n) = \max\left(g_{b\alpha_1}(n), g_{f\alpha_2}(n)\right) \tag{4.37}$$

where

$$g_{b\alpha_1}(n) = \max\left(f(n), \alpha_1 \cdot g_{b\alpha_1}(n-1) + (1-\alpha_1)f(n)\right) \tag{4.38}$$

and

$$g_{f\alpha_2}(n) = \max\left(f(n), \alpha_2 \cdot g_{b\alpha_2}(n+1) + (1-\alpha_2)f(n)\right). \tag{4.39}$$

The function now has two parameters α_1 and α_2 which can be adjusted separately. α_1 regulates the 'memory' and α_2 the 'anticipation' of the dynamic threshold. The parameters for peak picking are summarized in the following list:

1. w: The half-window size for local maximum calculation. Overall window size is given by: $w + 1 + w$.
2. m: Factor for window which is used for floating threshold calculation. Overall window size for floating threshold is given by: $m \cdot w + 1 + w$.
3. δ: The threshold value for floating thresholding. Will be added to the mean of the floating threshold window.
4. α_1: 'Memory' of dynamic threshold function.
5. α_2: 'Anticipation' of dynamic threshold function.

4.3.3 Evaluation

To evaluate the onset functions a test set containing about 2,5 hours of piano music was used. Although piano music is not the only target application for the framework developed in this work, its a good source for onset detection since the single events contain a certain amount of percussive sounds and therefor the onsets

can be localized very exactly. The true onset values were extracted from midi files which have been recorded simultaneous to the audio files on a piano with sensors to capture information about the played notes and pedal states. A small amount of the test files are synthesized midi files of music played on a midi keyboard. The synthesizer used was 'The Grand 2'[1] by Steinberg. Table 4.1 lists all files used for evaluation.

Filename	Recorded	Duration
Batik1990_Mozart_kv279.wav	✓	00:16:28
Batik1990_Mozart_kv280.wav	✓	00:15:06
Batik1990_Mozart_kv281.wav	✓	00:14:38
Batik1990_Mozart_kv282.wav	✓	00:15:00
Batik1990_Mozart_kv283.wav	✓	00:15:00
Mozart_kv284.wav	✗	00:26:16
Batik1990_Mozart_kv330.wav	✓	00:18:48
Batik1990_Mozart_kv332.wav	✓	00:18:04
beethoven_moonlight1.wav	✗	00:06:06
beethoven_moonlight2.wav	✗	00:02:04

Table 4.1: Files used for test set

To evaluate the quality of onset detection, the onsets of the test set are calculated using one of the above presented onset function algorithms. Then the peaks are selected using our peak selection algorithm. The onsets extracted in this manner are then compared to the true onsets provided by the corresponding midi files. Multiple onsets in the midi files (chords) are merged if they are closer then 16ms and if there were no separate onsets detected. This tolerance time was chosen because 10ms is the threshold of distinguishable onsets (see [Moo97]) and 16ms is the next power of two number which results in a tolerance field of 4 samples at a wave file sample frequency of 8kHz and an STFT spectrogram hopsize of 32 samples. The merging was implemented because multiple onsets do not have to be (and can not by these algorithms!) detected in this application since they do not change the rhythmical information. On the other hand when onsets which are closer than 16ms are detected, it should not be penalized. The time of 16ms is also used as tolerance for detecting onset matches to a real onset from the midi

[1] http://www.steinberg.net/de/products/legacyproducts/legacyproducts_thegrand2.html

file. [Dix06] uses a tolerance of 50ms which seems quite large and explains the good results while not using correction for onset delay caused by STFT windows.

Since there are many parameters which influence the quality and characteristic of onset detection the question arises how to report and compare the results and quality of the introduced onset detection algorithms. The most common way is using a receiver operating characteristic (ROC) curve. To be able to calculate and construct this curve we need to define some statistical values before. First we define the set of correct estimated onsets (points in time) which correspond to real onsets (true positives or correct classified onsets) as:

$$c = \{x \in t_{est}, \exists t_{real} : |t_{real} - t_{est}| \leq T_t\} . \tag{4.40}$$

where t_{est} is the set of estimated onset times, t_{real} the set of real onset times and T_t the tolerance time. The number of correct classified onsets is then calculated as

$$N_c = |c| . \tag{4.41}$$

The number of estimated onsets which are no real onsets (false positives) and the number of not detected real onsets (false negatives) can be calculated with

$$N_{f^+} = |t_{est}| - N_c \tag{4.42}$$

and

$$N_{f^-} = |t_{real}| - N_c. \tag{4.43}$$

With this statistical values of our detected onsets we can now calculate some error statistics following [Wik09d] which was also used in [Dix06]. We use the term 'precision' as a measure for exactness or fidelity which tells how many of the estimated onsets are really onsets. High precision does not allow conclusion on how many of the real onsets have been found. Precision is defined as:

$$P = \frac{N_c}{|t_{est}|} \tag{4.44}$$

On the other hand 'recall' defines a measure which allows conclusion on how many of the real onsets have been found by the onset calculation algorithm. Recall is defined as:

$$R = \frac{N_c}{|t_{real}|} \tag{4.45}$$

Often a combination of precision and recall is used to find the best parameter settings. One measure often used is the so-called 'f-measure ' which can be written as:

$$F = \frac{2 \cdot P \cdot R}{P + R} \tag{4.46}$$

However, since the wanted onsets/onset function depends on the features which should be extracted we are not always interested in the best combination of precision an recall. For inter onset interval calculation (Section 4.5) this may be the case, but for autocorrelation calculation (Section 4.4) a high precision is more important than recall. Using precision and recall we can now draw the mentioned ROC curve like shown in Figure 4.10.

The resulting values for precision, recall and f-measure for the different onset functions are displayed in Table 4.2. For this table only the values with highest f-measure were chosen. ROC curves for different parameter settings for peak picking are shown in Figure 4.10 and Figure 4.11. Here the two main parameters (δ: threshold, α_1/α_2: dynamic threshold) of peak picking were varied.

OS Function	Precision	Recall	F-Measure	E (ms)
SF	92.82	82.56	87.39	4.96
PD	19.14	74.15	30.43	5.97
WPD	26.73	60.36	37.05	6.47
CD	35.66	56.35	43.68	8.78
RCD	51.85	71.22	60.01	7.30
SPD	71.41	83.13	76.83	5.35

Table 4.2: Results for onset detection.

OS Function	Precision	Recall	F-Measure	E (ms)
SF	95.60	94.12	94.86	6.01
PD	54.45	36.25	43.52	6.21
WPD	47.23	75.52	58.12	14.41
CD	61.17	70.20	65.38	13.84
RCD	69.28	83.95	75.91	10.05
SPD	74.36	92.13	82.30	6.90

Table 4.3: Results for onset detection, using 50ms tolerance.

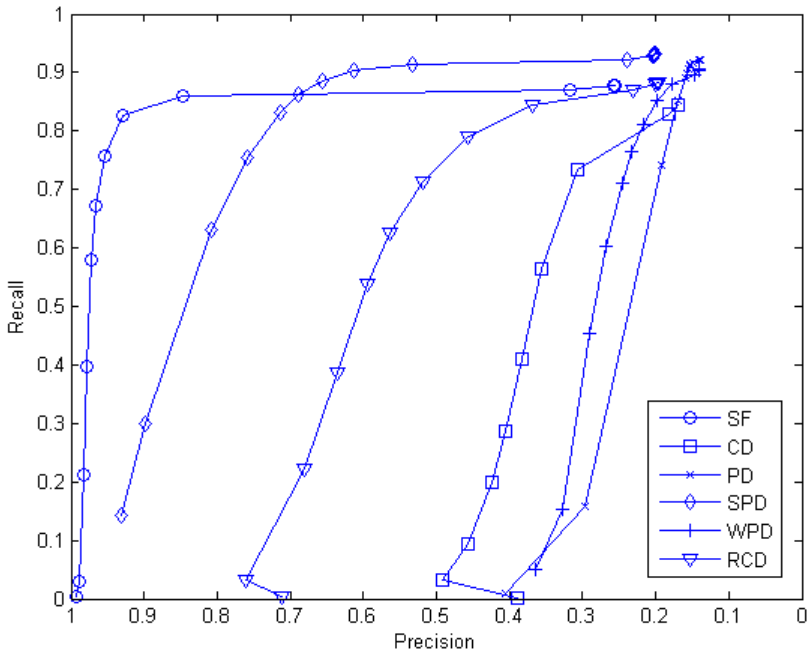

Figure 4.10: ROC curve using different threshold levels (δ) and no dynamic threshold for peak picking.

The results for SF function are comparable to them presented in [Dix06] the other functions perform not as well. This may be due to the fact that [Dix06] uses a larger tolerance field for correct onset matching (50ms vs. 16ms) and audio material with higher resolution (44.1kHz vs 8kHz) which result in more available samples for the same window length. Table 4.3, Figure 4.12 and Figure 4.13 show results using 50ms tolerance as in [Dix06]. The results presented in this work are more precise by means of the absolute mean error in ms shown in Table 4.2. This may be caused by the correction term used for onset delay caused by STFT windowing. This higher precision also allows us to use a larger window size for STFT (128ms@8kHz=1024 Samples vs. 46ms@44.1kHz=2048) which also allows us to use a lower sample frequency for source audio files without needing zero padding for STFT.

In this work the onsets are used as basis for features which allow a very exact guess on the tempo. [JR01] also mentions works which use wavelet transformation to ex-

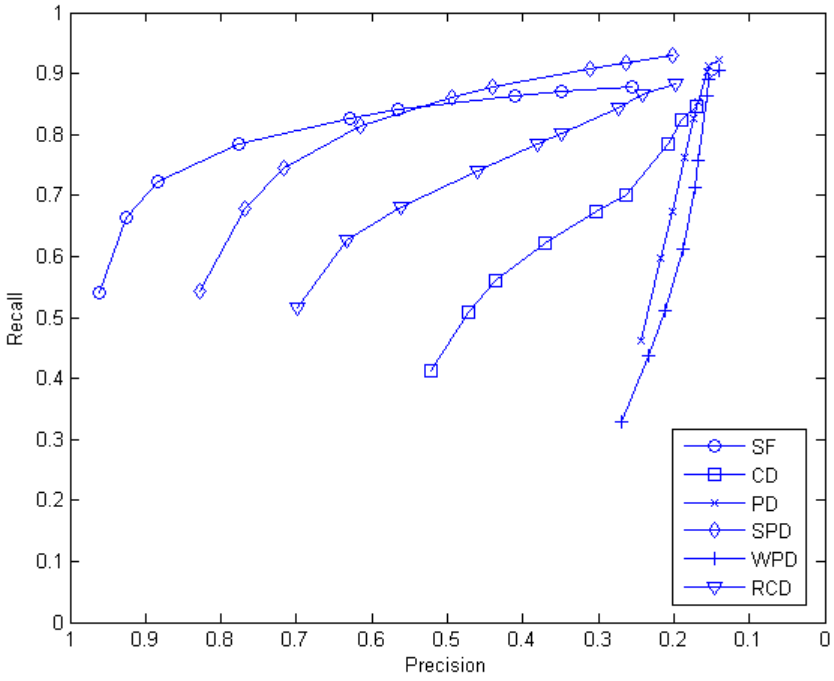

Figure 4.11: ROC curve using different dynamic threshold levels (α_1/α_2) at a fixed threshold for peak picking.

tract an onset function. [NTI04] uses convolutional-kernel operations on the STFT spectrogram to extract onset information. The methods used in this paper are base on image processing algorithms where the use of convolutional-kernels is a common way to localize shapes. Since convolution is always a very time-consuming operation this method was not considered further to be implemented in this work. Other genuine and interesting ways to obtain the note onsets are introduced in [DG02], [KP04] and [YD07], where amongst others Support-Vector-Machines (SVM's) are used to detect onsets. Since the complexity of these algorithms would go beyond the scope of this work, they are just mentioned here.

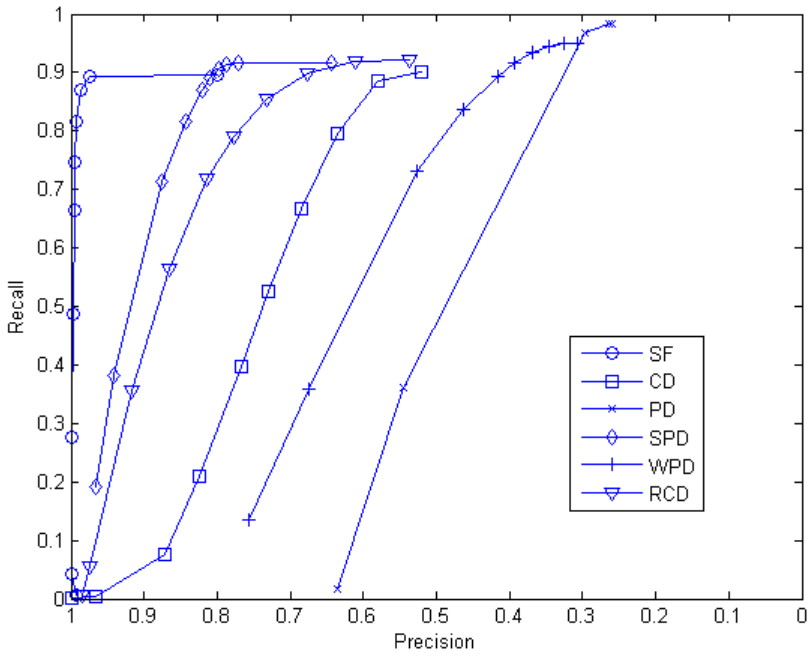

Figure 4.12: ROC curve using 50ms tolerance with dynamic threshold.

4.3.4 Implementation

Onset Function Calculation Onset detection is split up into the two main stages, onset function calculation and onset peak picking, like explained in Section 4.3. Onset function calculation is done in the function `onset.m`. Input parameters for `onset.m` are:

- `source:` Source signal. Can be a vector with signal or a filename of a wave file.
- `method:` Method/algorithm to use for onset function calculation. Implemented methods are: spectral flux (SF), phase deviation (PD), complex domain (CD), rectified complex domain (RCD), weighted phase deviation (WPD) and selected phase deviation (SPD). See Section 4.3 for details.
- `bounds:` Vector with two values which represent bounds for source signal. Only values between the provided indices will be used. If `bounds` is empty,

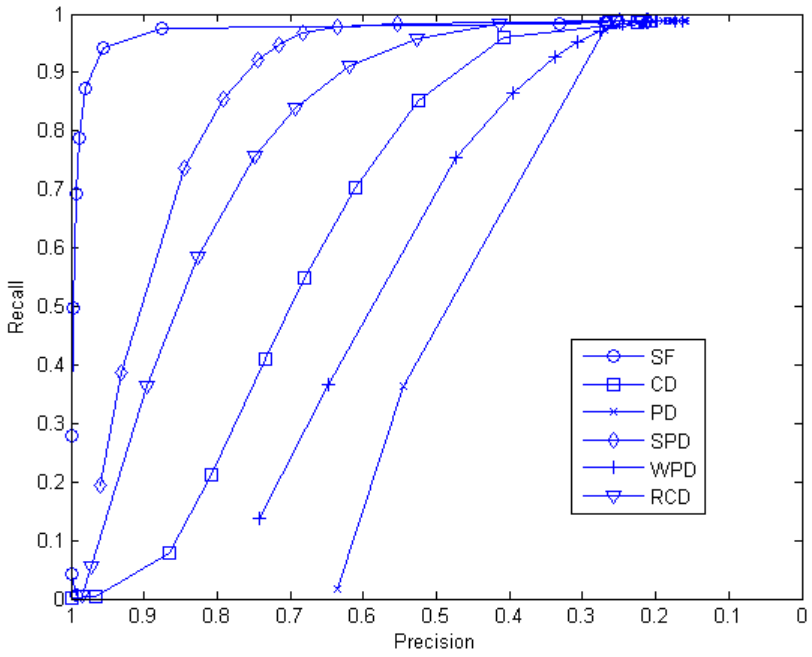

Figure 4.13: ROC curve using 50ms tolerance without dynamic threshold.

all values of source signal will be used.

- `usemel:` If true, the spectrogram frequency resolution will be transfered into melscale before onset function calculation.
- `usephicorr:` If true, the phase values of spectrogram will be corrected by means of removing phase growth for stationary signals. See Section 4.2 for details.
- `lpfrequ:` Lowpass cut-off-frequency for spectrogram. If not equal zero, all signal parts below `lpfrequ` will be canceled out before onset function calculation.
- `p:` Parameter structure with STFT parameters.

The function returns a vector which represents the onset function calculated. A second parameter returned is the calculated STFT spectrogram. Since it is used

for some other feature extraction it is returned here so it has not to be calculated again later. The onset detection function m-file `onset.m` is designed to be modular so it is quite easy to add new onset detection functions. `onset.m` makes use of the `onsetDelayCorr.m` to remove onset delay caused by STFT windowing. If new onset function are added, `onsetDelayCorr.m` should also be modified to support the new onset function. To get the empirical estimated delay factor the `osfncdelay.wav` and `osfncdelay.mid` files should be used.

Peak Picking Peak picking is implemented in the `getpeaks.m` m-file. Input parameters are:

- `signal:` The onset function signal in which the peaks are to be located.
- `win:` The window size divided by two (full window size $= 2 * win + 1$).
- `wf:` A factor for the first part of the window for thresholding (full window size $= win * wf + 1 + win$).
- `thres:` The threshold value for peak picking.
- `memory:` A scalar or vector with two elements which provide the coefficients for the dynamic threshold. First value of the vector provides 'memory' and second provides 'anticipation'.
- `visu:` If true, a figure will be plotted which visualizes the onset function with selected peaks and criteria.

The function returns two values. The first vector represents the amplitudes of the selected peaks (which can be useful for later onset analysis). The second vector holds the indices from the onset function where prominent peaks where found.

4.4 Autocorrelation

The autocorrelation function (ACF) is a way to find repetitive patterns in signals. Since the rhythmical structure of music is usually some kind of repetitive pattern,

autocorrelation can be used to find rhythmical structures and their periodicity. In this work the signals generated by our onset detection algorithms is used as basis. We look for repetitive patterns in this onset-functions using autocorrelation algorithms. The autocorrelation compares the signal s to itself shifted by a certain lag τ and is defined as:

$$r(\tau) = \frac{1}{N} \sum_{x=1}^{N} s(x) \cdot s(x + \tau) \tag{4.47}$$

Where $r(\tau)$ is the autocorrelation value and N is the number of samples which should be compared. In signal processing one usually has to deal with data of limited size. Since we need to shift the signal to calculate the autocorrelation the question is how to deal here with the limited data amount. If S is the number of samples of the signal and the lag τ is given, we could now choose N so that $N = S - 2 \cdot \tau$. This would provide us with the maximal number of samples to use for correlation calculation. An alternative would be to choose $N = S$ and fill the missing samples (τ samples) with zeros (zero padding).

If we take a range $[0..\tau_{max}]$ for lag's we get the ACF which displays the autocorrelation values in dependence of lag. Figure 4.14 (b) shows a typical ACF of one of the piano samples from onset detection test set. When calculating the ACF we have three possibilities for choosing N as mentioned above. If we use zero padding, the autocorrelation values will get smaller as the lag grows since more and more zeros will be involved. If we choose $N = S - 2 \cdot \tau$, the number of samples which are compared (N) gets smaller as the current lag (l) grows. For ACF calculation we have now a third possibility which is to choose $N = S - 2 \cdot \tau_{max}$. This results in a constant N while not involving zero padding. The disadvantage of this method is the relative small size for N. If we do not process data in real time, the third method is often the better one since we can omit the problem of small N by choosing S big enough to have a sufficient big N (method used in Figure 4.14).

Although using it for a different application (fundamental frequency estimation) [CK02] gives an overview on the properties of the ACF and compares it to an adapted version of a very kin function: the average magnitude distance function (AMDF). The AMDF calculates the difference between shifted signal and source

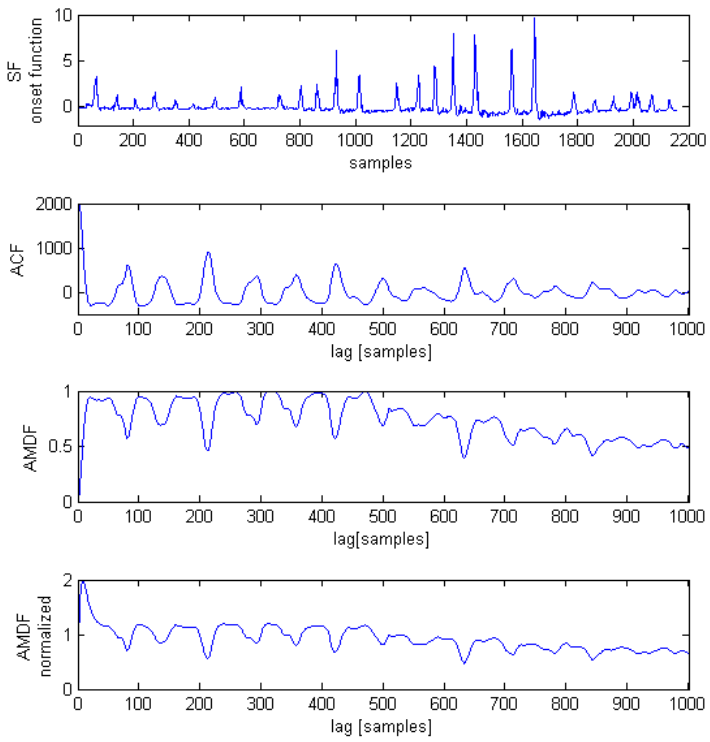

Figure 4.14: ACF (b), AMDF (c) and normalized AMDF (d) for lags [0..1000] of SF onset function (a) for excerpt of `moonlight1.wav`.

signal rather then calculation a correlation value. It is defined as:

$$d(\tau) = \sum_{x=1}^{N} \left(s(x) - s(x+\tau)\right)^2 . \tag{4.48}$$

The main difference between the AMDF and ACF function is that similar signals produce a high ACF value but a AMDF value close to zero. A big advantage of the AMDF is that a value of zero means that the two signals are equal. The ACF function output always depends on the amplitude of the source signal. To normalize the AMDF values one could simply divide the values by the maximum of all values:

$$d(\tau) = \frac{\sum_{x=1}^{N} \left(s(x) - s(x+\tau)\right)^2}{max(s)}. \tag{4.49}$$

A more sophisticated method of normalizing is presented in [CK02], where every value is divided through the average over shorter-lag values.

$$\overline{d}(\tau) = \begin{cases} 1 & \text{if } \tau = 0, \\ d(\tau) / \left[\frac{1}{\tau} \sum_{i=1}^{\tau} d(i)\right] & \text{otherwise} \end{cases} . \tag{4.50}$$

The resulting function does not have a zero value at lag $\tau = 0$ but one and slowly drops to zero at the first period of the signal. Another advantage is that there are no multiplications involved which makes it quicker to compute compared to the ACF function.

Figure 4.15 shows booth AMDF and ACF vectors of 545 songs arranged in an matrix. The single AMDF respectively ACF vectors are the columns of the matrix. The songs are sorted along the x-axis by their notated tempo. The music samples used in this diagram are taken from the 'songs' test set presented in Section 7.

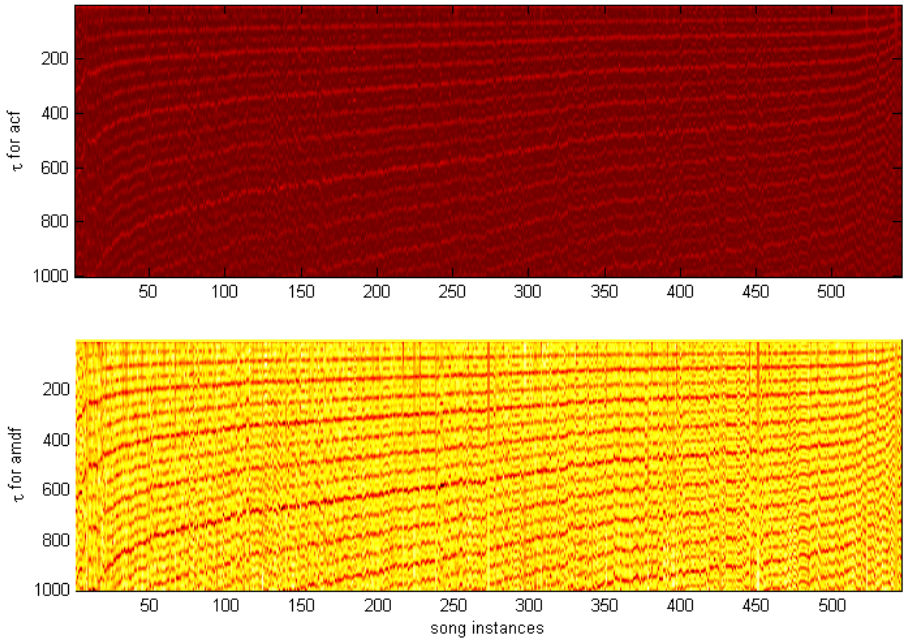

Figure 4.15: AMDF and ACF vectors for set of songs, ordered by their notated tempo.

In [SWS07], [DB06], [Pee06], [Uhl06], [MM04], [ADR04] and [DPW03]—which are all works on tempo extraction or rhythmic similarity—the ACF function is

used at some point. [Eck06] uses the ACF on STFT phase values creating an AC phase matrix. In this work both ACF and AMDF for feature calculation are implemented.

4.5 Inter-Onset-Intervals

If we take a look at an onset function output and its detected peaks (e.g. Figure 4.9), we can see the inter-onset-interval (IOI) as the interval between two detected onsets (peaks). IOIs are related to note lengths but are not necessarily the length of the played note which is indicated by the onset. The calculation of IOI can be formalized as:

$$IOI(i) = \frac{s_{os}(i) - s_{os}(i+1)}{f_s} \tag{4.51}$$

where $IOI(i)$ represents the inter onset interval of onset function peak i, p_osi the sample number of the onset function peak and f_s the sample frequency of the onset function. The division by f_s is only necessary if we want the IOI in seconds. To use the IOIs of an audio file as feature for tempo classification one could calculate different statistical values like mean, standard deviation and medians. The most common representation (as in [DPW03]) is to use a so-called IOI-histogram. In an IOI-histogram we measure the number of samples an IOI consists and put all IOIs of same length in one bin to create a histogram. Figure 4.16 shows an example for an IOI-histogram. The single bins can be calculated by means of

$$H_{IOI}(i) = |\{\forall j : i = IOI(j)\}| \tag{4.52}$$

In [DPW03] and [MM04] booth the consecutive as well as non-consecutive onsets (all possible combinations of detected onset peaks) are used for IOI and IOI-histrogram calculation. In this work one can choose if non-consecutive onsets should be considered for IOI calculation. Non-consecutive IOI calculation could be written as:

$$IOI = \{\forall i, j : i < j, s_{os}(i) - s_{os}(j)\} \tag{4.53}$$

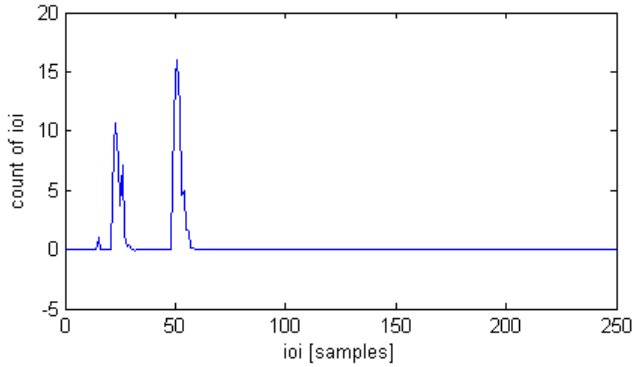

Figure 4.16: IOI-histogramm for audio excerpt (12s).

Where IOI in this equation represents the set of all IOI. It would be more complicated to express this relation as function so the way of a set has been chosen.

Furthermore it is possible to use the height of the onset peaks to weight the amount an IOI adds to the histogramm. The idea is that prominent onsets should have more impact in the IOI-histogram. This can be realized by calculating the histogram in this way:

$$H_{IOI}(i) = \sum_{j=1}^{N} \begin{cases} p_{os}(j) & \text{if} IOI(j) = i \\ 0 & \text{otherwise} \end{cases} \tag{4.54}$$

In Figure 4.17 the results for four different possibilities for IOI-histogram calculation are shown. It shows that weighting does not make much difference, while adding non-consecutive onsets produces a very similar image as calculation of ACF or AMDF functions. Maximum IOI lengths which were chosen in the diagrams (250 and 1000 samples, respectively) result of observations how long IOIs in an average file (250 samples) are, or for compatibility to AMDF and ACF calculation (1000 samples) when it comes to non-consecutive IOI which can theoretically be as long as the audio excerpt.

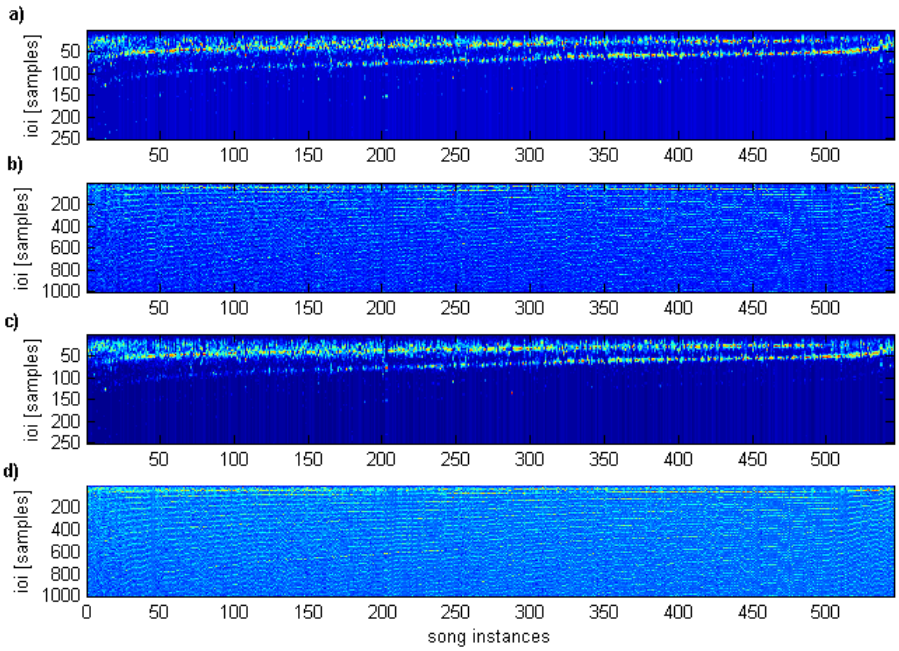

Figure 4.17: IOI-histograms of various files sorted by their notated tempo in BPM. (b) and (d) computed adding non-consecutive IOIs, (c) and (d) using IOI weighting.

4.6 Fluctuation Patterns

Fluctuation patterns (FPs) have been introduced by [Pam04] based on the idea of the periodicity histograms presented in [Sch98]. Its an STFT spectrum based feature which describes the amplitude modulation of the loudness of 20 frequency bands. [SWS07] implements a compressed version where the frequency bands are reduced to just one by summing across the frequency bands. The implementation in this work follows [SWS07] and takes use of the in [Pam04] presented Matlab® toolbox functions. Figure 4.18 shows booth the original fluctuation pattern and the compressed version. The first step in calculation is a so-called sonogram (an STFT spectrogram with a bark-frequency scale and a sone amplitude scale) then the modulation frequencies of the single bands are calculated through a fourier transform again. The details, formulas and implementation are found in [Pam01]

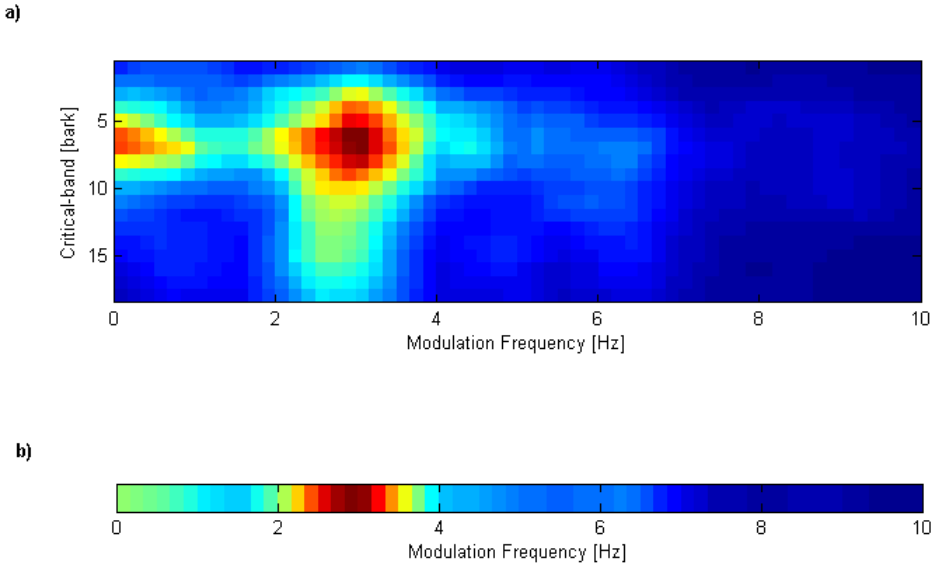

Figure 4.18: FP for song excerpt and compressed FP after summing across frequency bands.

and [Pam04]. Figure 4.19 shows the compressed FP feature of our 545 songs of the test set.

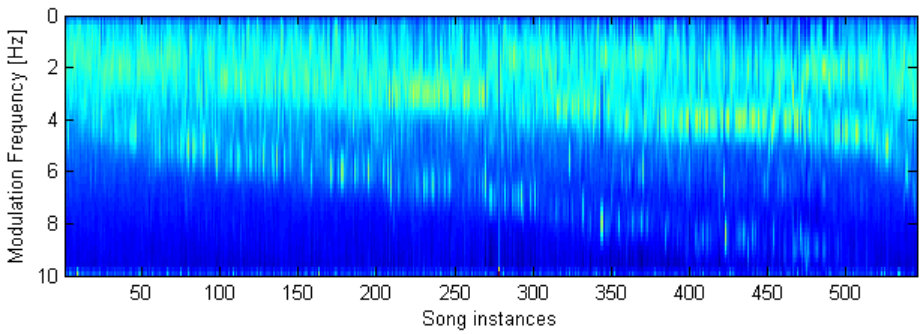

Figure 4.19: Compressed FPs for 545 songs ordered by their tempo.

4.7 Onset Density

The idea behind this feature is that music with more events in a certain amount of time generates the impression of fast music. To calculate an onset density feature we simply count the detected onsets and divide the amount by the number of samples.

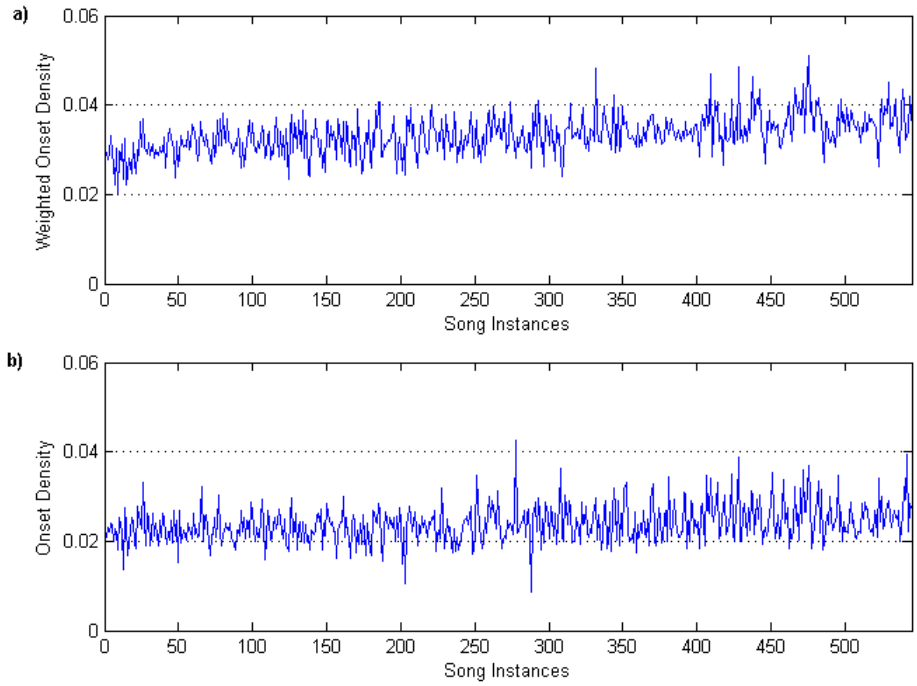

Figure 4.20: Onset density (b) and weighted OD output for 545 songs ordered by tempo. A very weak tendency of growing OD and WOD for higher tempo can be observed.

4.8 Activation Time

Activation time is a pseudonym for transient in the definition of Figure 4.2. Music with long lasting notes suggest the impression of slow music. We try to extract some kind of feature which reflects the activation times of the single notes. One

Idea to calculate such activation time was to use onset/offset combinations presuming the existence of an offset detection function. however, this is even harder to implement than an onset detection function. Another idea was to use the so-called spectral fingerprint (see [**?**] and Figure 4.21) of audio signals to extract note transient lengths. The spectral fingerprint is just a smoothed Mel spectrogram where the values which exceed a certain threshold are set to one, and values below this threshold are set to zero. This produced a fingerprint like image—thus the name.

We use this spectral fingerprint to get note on- and offset pairs and compute the average length of active notes. The lower 6 bands hold most of the energy and are almost always one, therefore they are for no use for this application and are ignored when calculating the activation time (see Figure 4.21). Then the first differential of every band is used to find matching onset/offset pairs and the average duration is calculated using this information.

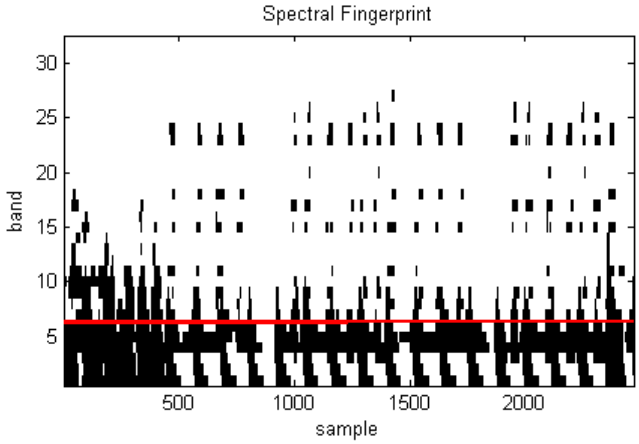

Figure 4.21: Spectral Fingerprint of an audio excerpt. The red line separates the first six bands, which are ignored for activation time calculation.

4.9 Rhythm Pattern

The Rhythm Pattern feature is motivated by the idea presented in [FU01]—the so called Beat Spectrum. [FU01] uses timbre features such as mel-frequency-

cepstrum-coefficients (MFCCs) and a distance function between these feature vectors to create a self-similarity matrix as shown in figure 4.22. In this work solely

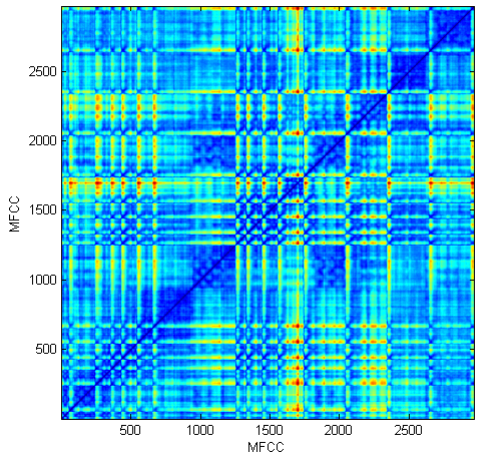

Figure 4.22: Self similarity matrix of MFCC feature vectors for a 30 second excerpt of an audio file. The distance function used for this matrix was a normal euclidean distance.

the MFCCs of the potential onsets are calculated and compared. The distances between the single onset points (their MFCCs) are then used to cluster the onsets into ten groups. The expected result would be that the onsets of distinct instruments/instrument combination are grouped together. What we receive is a pattern representing the distinct onsets for every instrument. Such a Rhythm Pattern is visualized in figure 4.23.

To use this 'snapshot' of the rhythmic structure of the music excerpt as a feature, the relevant information is extracted by using solely the mean and standard deviation of the inter onset intervals per 'channel'. The calculation of the inter onset values is performed similar as presented in Section 4.5. Then the mean and standard deviation for every channel is calculated and stored in two feature vectors.

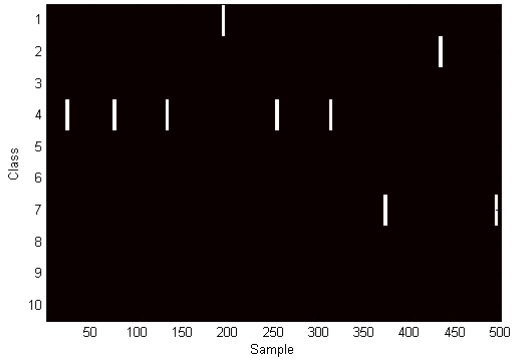

Figure 4.23: Rhythm Pattern for audio sample.

4.10 Other Tempo Algorithms

One could use classic tempo calculation algorithms to calculate potential tempo candidates which can be used as tempo related features. A motivation of using calculated tempo as a feature is following theory: Files on which the tempo calculation fails might have certain musical features which cause the problem. If different files have the same musical features, it is likely that the tempo calculation algorithm makes the same mistake on both files - thus producing similar results. Where the results of such files might be wrong in terms of BPM tempo values, the results can be useful as features. One disadvantage of using the complete algorithms is that most of the algorithms need much calculation time. For the use as feature the tempo in BPM is not better than any value proportional to the tempo. For that reason we can simplify most of the tempo calculation algorithms to a very rudimentary form and use it as a tempo related feature.

Most of the more simple tempo calculation algorithms (e.g.: [Uhl06], [Pee06], [MM04], [DB06], ...) use the ACF of the onset detection function to calculate the tempo of the piece of music. What we have done in this work is to use this simple method (ACF respectively AMDF of onset detection function) and use the position of the first peak (first valley respectively) as a value proportional to the tempo which would be calculated by such algorithms. The result of this very simple and straight forward method holds all the information we need, so we can

use it as a tempo related feature.

4.11 Feature Post Processing

The larger feature vectors like ACF and AMDF, as well as the IOI-histogram with non-consecutive onsets tend to be noisy and spiky which leads to bad results with machine learning algorithms like KNN. Therefore, this larger feature vectors are smoothed using a normal moving average filter. Furthermore, the vectors are normalized by means of subtracting the mean and dividing through the standard deviation.

4.12 Implementation

As described above, onset detection and frequency transform are performed in several other files. The implemented feature calculation process can be found in the `r_extract_features.m` file. Every feature mentioned above has its own block in this file where all necessary calculation for feature extraction are done. For this calculations Matlab® intern functions and function from *ma-toolbox* ([Pam04]) are heavily used. The calculation procedures follow the above described theoretical formulas. For more detailed information see the inline commentary in `r_extract_features.m`.

The feature caching is done in the files `isCached.m` and `readCached.m` which are booth used in `r_analyse_files.m`. `r_analyse_files.m` performs the block calculation of feature for a given filelist.

5 Machine Learning

In some works on tempo induction the tempo is calculated from extracted features in some way (see [APT+07], [GD06], [MM04], [DB06],...). Most of the other works use an onset (detection) function, accent function or fluctuation pattern as basis for their calculations (see [ADR06], [DB06]).

Other works using tempo related feature are e.g. [Tza02] where such features are used for genre classification or [PK07] where tempo relevant features are used for automated drum transcription.

[SWS07] uses a different approach where the tempo is not calculated but rather estimated through comparison of features and choosing the tempo of the most similar instance. Using this procedure we avoid several problems that occur when trying to calculate prominent peaks of ACF or IOI-histograms. Typically there are multiple prominent peaks which are integer multiples of the real tempo—which is one of the main problems of tempo estimation.

In the next sections machine learning algorithms which have been considered suitable for this work are introduced.

5.1 Nearest Neighbors

The nearest neighbor (NN) classifier is one of the simplest machine learning algorithms. In nearest neighbor classification, distances between a feature vector which's class is unknown and all the training vectors is calculated. Distance functions can be e.g. euclidean distance, cityblock distance or correlation. Then the

class of the closest vector is returned as the class for the new vector. Since the decision for the classification depends directly on the training data, training solely consists of optional preprocessing of the feature vectors. The following list provides the formulas and a brief description of some distance functions:

- **Euclidean distance:** The euclidean distance is the distance between two points a, b in a n dimensional euclidean space with coordinates $a_i..a_n$ and $b_i..b_n$ and is defined as:

$$d_{\text{euclidean}}(a,b) = \sqrt{\sum_{i=1}^{n} (b_i - a_i)^2} \tag{5.1}$$

  The single feature vectors are therefor interpreted as coordinates of points in an n-dimensional euclidean space.

- **Cityblock distance:** Cityblock distance (also taxicab geometry or Manhattan distance) is defined as the sum of the (absolute) differences of their coordinates. It is somewhat related to the idea of the AMDF. The cityblock distance of two points a, b in an n dimensional space with coordinates $a_1..a_n$ and $b_1..b_n$ can be calculated:

$$d_{\text{cityblock}}(a,b) = \sum_{i=1}^{n} |b_i - a_i| \tag{5.2}$$

- **Correlation:** Here, the correlation of the two vectors is used as a measure of distance. The use of correlation implies the interpretation of the feature vectors as signals—which is in this case totally legitimately. If a and b are vectors of length n with the discrete values $a_1..a_n$ and $b_1..b_n$, the correlation is defined as:

$$d_{\text{correlation}}(a,b) = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} (a_i)^2 \sum_{i=1}^{n} (b_i)^2}} \tag{5.3}$$

  Correlation is in fact a measure of how similar two compared signals are.

An enhancement of the NN classifier is the k-nearest neighbor (KNN) classifier. Here the first k nearest neighbors are used to calculate the class of the instance which should be classified. There are several possibilities to combine these k classes of the nearest neighbors:

Consensus: All neighbors must have the same class, which is also assigned to the classified instance. If there is no consensus, there is no decision and no class is assigned. This method is suitable for decisions where only very secure decisions can be accepted (e.g. in medicine). For this work this approach is not appropriate since we have many classes (tempi) with possibly few instances in every class.

Majority: The class most often occurred among the neighbors is assigned to the classified instance. There are basically two ways to treat situations where there are several classes with same occurrences:

Nearest neighbor: The class with the nearest neighbor among all candidates is chosen.

Random: The class is randomly chosen form the possible candidates.

Mean: The mean of the classes of all neighbors (tempi) is calculated and assigned to the classified instance. This alternative only makes sense in combination with numerical classes like in this work the tempo itself.

Means of clusters with majority: The first step is to cluster all neighbors to groups with similar tempo. Then the mean of the cluster with most instances is calculated and assigned to the classified instance.

Figure 5.1 illustrates the influence on the number of used neighbors (k) on the classification results. The consensus rule applied in Figure 5.1 would lead to a zone (width depends on how much neighbors would be used) between the two classes where new instances would not be assigned to any class.

In this work KNN classifiers with different values for k have been used. Due to the large sizes of ACF/AMDF and IOI-histogram feature vectors, the distance functions of choice were correlation and difference rather then the usually used euclidean distance. The problem is that, when using euclidean distance, very similar vectors are quite 'far' away because of the high dimensionality of our feature vectors ($\approx$1000).

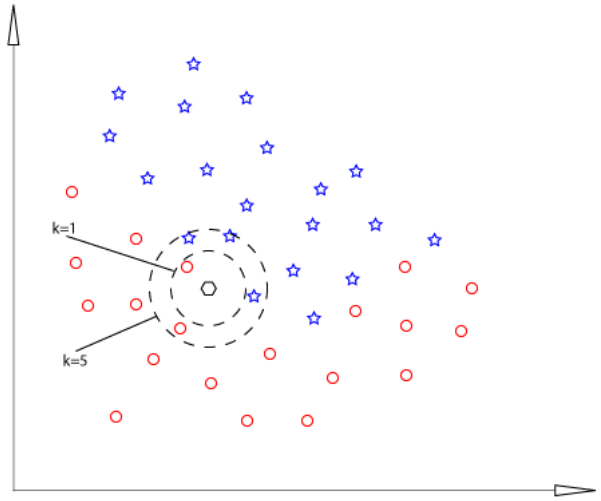

Figure 5.1: Feature space with training instances assign to two classes (stars and circles) and an instance which should be classified (hexagon). Different ks result in different classes assigned to the new instance.

5.1.1 Implementation

For the first experiments WEKA[1] (see [WF05]) was used. For the implementation in Matlab® the KNN implementation of the bioinformatics toolbox was used.

5.2 Support Vector Machine

Support vector machines (SVMs) are a class of machine learning algorithms which are famous for their versatileness. They are used in different areas of engineering as well as in medicine.

SVMs interpret the feature vectors as coordinates of a point in an n dimensional space. The goal is to find an $n-1$ dimensional hyperplane in that space that splits the training data according to their classes. In fact not just any hyperplane is wanted but the hyperplane that provides the largest distance to the closest instances of each class. This plane is automatically the one which provides the

[1] http://www.cs.waikato.ac.nz/ml/weka/

minimal generalization error. Because of this, this concept is also known as maximum margin classifier. We can formalize this decision process if we define our input feature vector $x = (x_1, ...x_n)$ and a decision function $f(x)$. We assign x to one class if $f(x) \geq 0$ and to the other if not. We could now define

$$f(x) = \langle w \cdot x \rangle + b = \sum_{i=1}^{n} w_i x_i + b, \tag{5.4}$$

where w is the normal vector of the separating hyperplane and b is the bias—the normal distance to the coordinate origin—of the hyperplane. The hyperplane itself is therefore defined as:

$$\langle w \cdot x \rangle + b = 0. \tag{5.5}$$

Figure 5.2 shows a two dimensional feature space with samples separated by an one dimensional hyperplane (a straight line). The parameters b and w of the plane are also illustrated. The goal of training the SVM is therefore finding an optimal

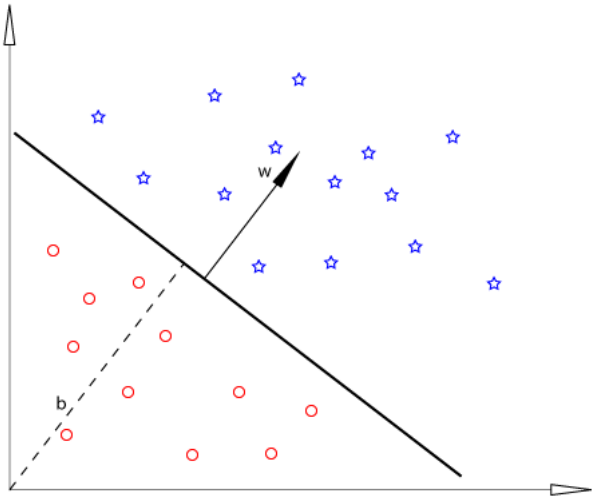

Figure 5.2: A hyperplane separating two classes of a dataset with maximal margin.

w and b where the margin of the separating hyperplane is maximal.

The problem of the search for the optimal plane is solved using quadratic programming (QP) solving algorithms like the sequential minimal optimization (SMO) algorithm. This basic type of SVMs is called linear, hard-margin SVM. 'Linear'

because it works only on linearly separable data and 'hard-margin' because the margin is chosen so that no data points are allowed on the wrong side of it.

One extension to linear-SVMs is to use so-called soft-margin constraints, which allow instances of the training data on the 'wrong' side of the separating hyperplane. The optimization problem for choosing the hyperplane becomes now a trade-off between margin size and penalty through wrong sided instances. Whereas with hard-margin SVMs no solution could be found for training sets with continuous transition, soft-margin SVMs are able to find the optimal separating hyperplane within this data sets.

For handling non-linear separable training data another variant of the classic SVM is used. The idea for non-linear SVM is to transform the training data into a higher dimensional space where the single points are linearly separable. The transformation is usually avoided by using the so-called 'kernel-trick'. The kernel-trick is a way of using higher dimensional (up to infinite dimensional) spaces without knowing how many dimensions are used nor defining the transformation function. The trick is to use a so-called non-linear kernel function which replaces all dot-products for vector calculations in the normal space. A kernel function has to fulfill following criteria:

Continuity: The kernel function has to be continuous which means that very small changes in the input result in very small changes in the output. This can formalized by the Cauchy definition of continuous functions. Let f be a function $\mathbb{R} \mapsto \mathbb{R}$ and suppose c to be an element of fs domain. The function f is continuous at the point c if for any ϵ

$$\forall x, \exists \delta : \epsilon > 0, \delta > 0, |x - c| < \epsilon \Rightarrow |f(x) - f(c)| < \delta. \tag{5.6}$$

Which means: We should be able to find a δ for any ϵ however small so that, for any x, if $|x - c| < \epsilon$ also $|f(x) - f(c)| < \delta$. In the case of kernel function that means that if we make small changes at one vector the resulting mapped vector and therefore the scalar product should change only little.

Symmetry: The property of symmetry represents the commutativity of the scalar product—which is replaced by the kernel—in the higher dimensional feature

space. Meaning: If K is our kernel function and x, y are vectors of our feature space, K needs to fulfill

$$K(x, y) = K(y, x) \tag{5.7}$$

Positive semidefinite form: The kernel function K has to be a bilinear form which fulfills

$$\forall x : K(x, x) \geq 0. \tag{5.8}$$

To qualify as bilinear form, K has furthermore to satisfy following three conditions:

$$K(u + u', v) = K(u, v) + K(u', v) \tag{5.9}$$

$$K(u, v + v') = K(u, v) + K(u, v') \tag{5.10}$$

$$K(\lambda u, v) = K(u, \lambda v) = \lambda K(u, v) \tag{5.11}$$

With this properties one can define a variety of kernel functions. Some well known kernel functions which fulfill the criteria defined above and have proven to work well, are listed below:

Polynomial (homogeneous):

$$K(u, v) = (g \langle u \cdot v \rangle)^d \tag{5.12}$$

Where $d \in \mathbb{R}/0$ is the degree of the polynomial and $g \in \mathbb{R}$ is a factor for the scalar product of the two vectors.

Polynomial (inhomogeneous):

$$K(u, v) = (g \langle u \cdot v \rangle + c)^d \tag{5.13}$$

Inhomogeneous variant of the polynomial, $c \in \mathbb{R}$.

Radial Basis Function:

$$K(u, v) = \exp\left(-g \cdot |\cdot v + u|^2\right) \tag{5.14}$$

Gaussian Radial Basis Function:

$$K(u, v) = \exp\left(-\frac{|\cdot v + u|^2}{2\sigma^2}\right) \tag{5.15}$$

Sigmoid:

$$K(u, v) = \tanh\left(g\left\langle u \cdot v\right\rangle + c\right) \tag{5.16}$$

Every kernel function has different properties regarding the mapping of the feature vectors into the higher dimensional space. Therefore, the choice of the kernel function depends on the distribution of the instances of the single classes.

If the machine learning algorithm should be trained on more then just one (respectively two) classes, the normal SVM can be extended to do so. Multi class SVMs are usually built using multiple SVMs. There are two very common methods to so. We can either (a) use one SVM to separate one class from all others or (b) one SVM for every pair of classes. When we want to classify a new instance using variant (a), we use the class of the SVM which produced the higher output i.e. the higher distance from its separating hyperplane. Using variant (b), we count how often every SVM 'wins' against others and assign the class of the SVM which won most often.

Another application for SVM is to use it for regression calculation. The idea of support vector regression (SVR) is not to use the hyperplane to separate the instances of two classes but rather to find a hyperplane which covers all data points to approximate a function which generates the instances. Although equations for defining the optimal plane differ from the original SVM the learning process is quite similar (compare [SO98]).

For more detailed information about SVMs see [CST05], [CL09] or [Wik09f].

In this work SVM are used for tempo range classification. SVM can handle the high dimensionality of our feature vector quite well (see Section 7. We also tried to use SVR directly for tempo estimation. The results were much worse than with KNN classification, most likely because of the high number of points in the very high dimensional feature space.

5.2.1 Implementation

We used three different SVM implementations in this work. First we tried to experiment with the SVM learning algorithm in WEKA[2] (see [WF05]). After some time a Matlab® algorithm was written to perform class estimation using Matlab® intern SVM functions. Since the Matlab® intern SVM implementation did not offer enough parametrization possibilities, libsvn[3] (see [CL09]) was used later on as SVM implementation—also for GA parameter optimization.

5.3 Parameter Optimization

Since machine learning algorithms typically have a variety of parameters which influence the quality of the resulting classifier, it is important to trim the parameters to get an optimal classification result. Parameter for machine learning algorithms like the KNN classifier are more or less intuitive to choose—results can easily evaluated because of a countable amount of possibilities. In case of a KNN classifier we would have to optimize the amount of neighbors, the distance metric and the decision algorithm for choosing one class of all neighbors. Certain types of feature vectors may imply a distance function which seems to be appropriate. Which kind of decision algorithm makes sense can also be reasoned by looking at the model of our problem. The optimal amount of neighbors can then be determined by evaluation with different values for k (neighbor count). Thus parameterizing a KNN classifier is a very structured process.

When it comes to SVMs the task is more difficult. First we have to choose a type of SVM. The decisions between soft/hard-margin SVMs, linear/nonlinear SVMs and regression SVMs may be easy if the model of the problem is well known. The choice of the kernel function and parameter settings for the kernel as well as for nonlinear SVMs is a much harder problem. Lets start with the SVM kernels: These are functions, usually $\mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}$ (where n is the dimension of the feature vectors) which cover three processes:

[2] http://www.cs.waikato.ac.nz/ml/weka/
[3] http://www.csie.ntu.edu.tw/~cjlin/libsvm/

- The mapping of a feature vector into a higher dimensional (even infinite dimensional) feature space.
- The calculation of the scalar or dot-product of our feature vectors in this feature space.
- The inverse transformation of the result into the original feature space.

Usually the structure of the distribution of the sample points in the feature space is not obvious (especially when the feature vectors are high dimensional). Therefore, the choice of the kernel can usually not be made by analyzing the instances in our feature space. Often it is a decision based on experience or experimentation. If we do not have any experience with kernel functions in our feature space, we have several possibilities to evaluate different kernel functions with experimentation. Usually one would use numerical optimization to solve such types of problems. The problem with parameters for kernels and kernel types is that we can not make predictions of the resulting classification accuracy when making changes in the parameter settings or choose a different kernel. The resulting accuracy function (with respect to parameter settings and kernels) is therefore very chaotic with many local minima. Therefore, we can neither use normal *greedy* numerical optimization since it is likely to fail because such algorithms are prone to get stuck in local minima. Nor can we try *all* possible combinations since we have an infinite amount of kernel functions and parameter settings. There are some algorithms which are known to be able to find the global optimum within such problem spaces. Such algorithms are for example grid search, genetic algorithm or simulated annealing.

Grid search: This method works good on flat fitness functions (solution quality depending on parameter setting) with multiple local minima but can be very time consuming if we have many parameters and/or large ranges for parameters. Using grid search, we calculate the fitness function for every point in a grid in our feature space. To obtain the grid we divide the ranges for every feature dimension into equidistant pieces (depending on the feature we can also use logarithmic, exponential,... etc. division). We then build all possible combination of the obtained feature values and calculate the fitness

function. The smaller the pieces for feature division, the better the results but the more time consuming is the calculation. If the feature space is high dimensional, the number of fitness calculations necessary raises very fast. A small example should demonstrate that:

□ *We have a feature space with dimension* $D = 10$ *and want to perform a grid search. How many calculations are necessary if the range of every dimension is* $r = [-100..100]$ *and the grid distance is* $d = 10$*? How long would the calculation take if one fitness evaluation run takes* $t_e = 1$ *second?*

Every feature dimension would be divided into $N_f = \frac{r_{max} - r_{min}}{d} = \frac{100+100}{10} = 20$ points. If we have a dimensionality of $D = 10$, we get $N_{total} = {N_f}^D = 20^{10} = 2 \cdot 10^{11}$ grid points. The total calculation time would therefor be $t = N_{total} t_e = 2 \cdot 10^{11}$ seconds which is more then 6000 years.

Because of the high dimensionality of some of the feature vectors used in this work, grid search is not appropriate for SVM parameter optimization.

Genetic algorithm: Genetic algorithms are a class of algorithms which simulate evolutionary properties of live like combination, mutation and survival of the fittest on the level of genes. Usually the algorithm starts with some (e.g. 20) parameter settings for the function which's output should be optimized. At first the parameter settings are created randomly. Then the algorithm starts to combine single values of two parameter sets (combination) and/or changes values randomly (mutation). The parameter sets which 'survive' (are kept for the next round) are chosen by their fitness value.

Simulated annealing: In simulated annealing the behavior of metal atoms while finding a state of least energy when cooling liquid metal is simulated. As long as the temperature is high, the probability that an atom moves into a position where its energy is higher, is relatively high, and becomes less and less as the temperature falls. This behavior is transformed to our parameter space and fitness function. We have a parameter which tells us the temperature of the current state. If we are in a state with high temperature,

the possibility of choosing a path in our solution room which results in a worse fitnesses is relatively high and falls with the temperature. Simulated annealing is therefore a greedy search with the possibility to go 'uphill' as long as the temperature is not to low. The advantage is that its possible for the algorithm to leave local minima. The whole algorithm can be done with just one instance to optimize, or multiple, starting for example with something similar as we use in grid search.

In this work genetic algorithms where used to find optimal parameter settings for SVM tempo range classification and SVR tempo estimation.

6 Framework Implementation

The framework was developed using MathWorks™ Matlab®. For some features external toolboxes as e.g. the *ma-toolbox* from [Pam04] were used. Matlab® offers many function as for example machine learning methods such as the KNN classifier and is therefore an ideal environment to develop frameworks for experimentation purposes. Matlab® comes also with a simple implementation of support vector machines. The first experiments with SVMs where realised with Matlab® intern function. Later the libsvm[1] (see [CL09]) framework was used due to performance and parametrization reasons. For some experiments the Java implemented machine learning framework WEKA[2] (see [WF05]) was used.

6.1 Tempo Experimenter

The core element of the framework is a Matlab® GUI which unites all the functionalities to experiment with the functions provided by the framework. Figure 6.1 shows the GUI and its components. The two lists in the upper left and right area are file lists for training respectively test audio files. It is not necessary to add files to to the test set, since the framework also offers methods to use the training files for testing.

On the left side beneath the file list for training files are two option fields for the classifiers used for tempo estimation. The left field configures the range classifier (first stage), the right one the 'tempo' classifier (second stage). One can choose

[1] http://www.csie.ntu.edu.tw/∼cjlin/libsvm/
[2] http://www.cs.waikato.ac.nz/ml/weka/

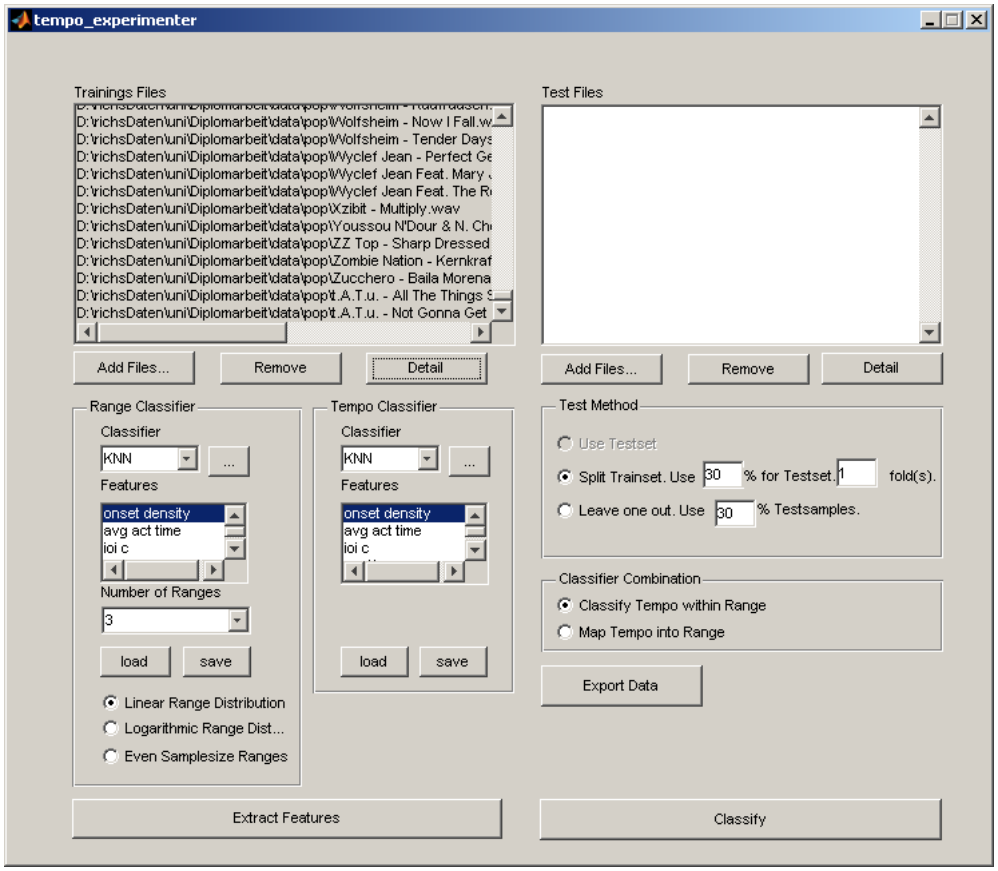

Figure 6.1: The GUI of the Matlab® framework which provides functionality to use the functions of the framework comfortably.

the type of machine learning algorithm (KNN, SVM), the features to use (see Section 4) and, in case of the range classifier, also the number of ranges and range distribution. There are three different range distributions implemented:

- **Linear Range Distribution:** The tempo ranges are equally distributed over the tempo range of the training files. Which means: If the slowest file has a tempo of 40BPM and the fastest a tempo of 200BPM, the range is 160BPM. If we use four tempo ranges the first one reaches from 40BPM to 80BPM, the second from 81BPM to 120BPM, the third from 121BPM to 160BPM and the fourth from 161BPM to 200BPM.
- **Logarithmic Range Distribution:** This method is similar to the 'Linear

Range Distribution', only difference is that the limits of the ranges are not distributed equidistantly but logarithmic.

- **Even Samplesized Distribution:** Using this method every tempo range contains more or less the same amount of files (samples). The limits of the ranges are chosen so that every range is equally filled with files. This method is a good choice if the distribution of the tempo of the files is very uneven.

On the right side beneath the file list for test files are the options how the testing samples should be generated. If the user provided test files in the file list, one can choose 'Use Test Set' which means the files provided should be used to test the classifiers. If no test files are given, one can choose between splitting the training set into training and test set at a certain percentage (the files are randomized before they are splitted) or using leave one out testing. The leave one out method is very time consuming since the classifiers have to be trained again for every file. Beneath the options for the test set we can configure how the two classifiers of the first and second stage should be combined. One can chose between:

- **Classify Tempo within Range:** The first stage provides the range for the tempo, the second stage then can only classify within this range.

- **Map Tempo into Range:** Here the second stage classifier first tries to estimate the tempo within the full range of tempi. Then the result of the first stage is used to map the tempo into the correct range by multiplying or dividing the tempo by/through two.

In the very bottom of the GUI are two big buttons, the 'Extract Features' and 'Classify' button. The 'Extract features' button starts the feature extraction process, but usually it is not necessary to manually extract the features, since this is also done before the classifiers are trained. The extracted features are cached and as long as the feature set has not changed or the cache is missing, the features will not be recalculated. The 'Classify' button starts the training and classification process. After the classification a window as shown in Figure 6.2 pops up and visualizes the results of the classification process. The first panel shows the real tempo in BPM (red) compared to the estimated tempo (blue), the estimated tempo

corrected using the estimated ranges (green) and the estimated tempo corrected using the real ranges (magenta). The last values is interesting since it indicates how good the results could get if we where able to estimated ranges with 100% accuracy.

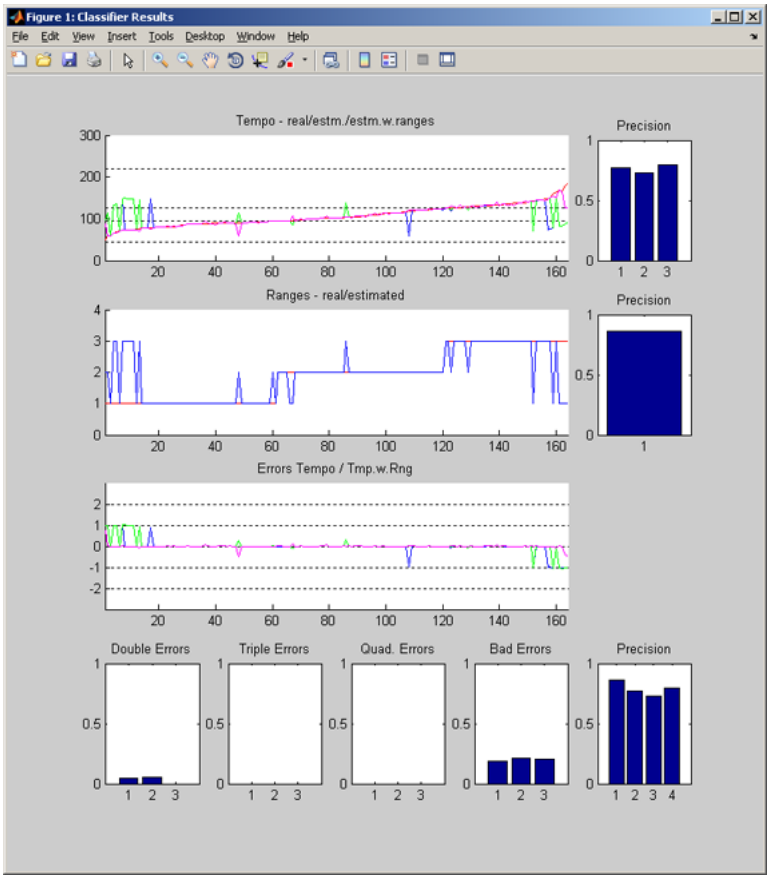

Figure 6.2: .

The second panel shows the real ranges (red) and the estimated ranges (blue). In the last panel the errors of the estimated tempi from the first panel are shown. The errors are calculated using the error measure described by formula 7.2. An error of 1 or -1 means the estimated tempo is twice or half the real tempo—2 or -2 then four times or a fourth of the real tempo and so on. The colors are the same as in the first panel:

- **blue:** Estimated tempo without use of first stage classifier (without ranges).

- **green:** Estimated tempo corrected with first stage results (using ranges).
- **magenta:** Estimated tempo corrected with real ranges.

Figure 6.3 shows the window which pops up when clicking on the 'Detail' button below the file list. One can see the feature vectors of the features for the selected files in the corresponding file list. The feature vectors are sorted by the notated tempo of the files—therefor allowing a visual check for tempo related patterns in the features.

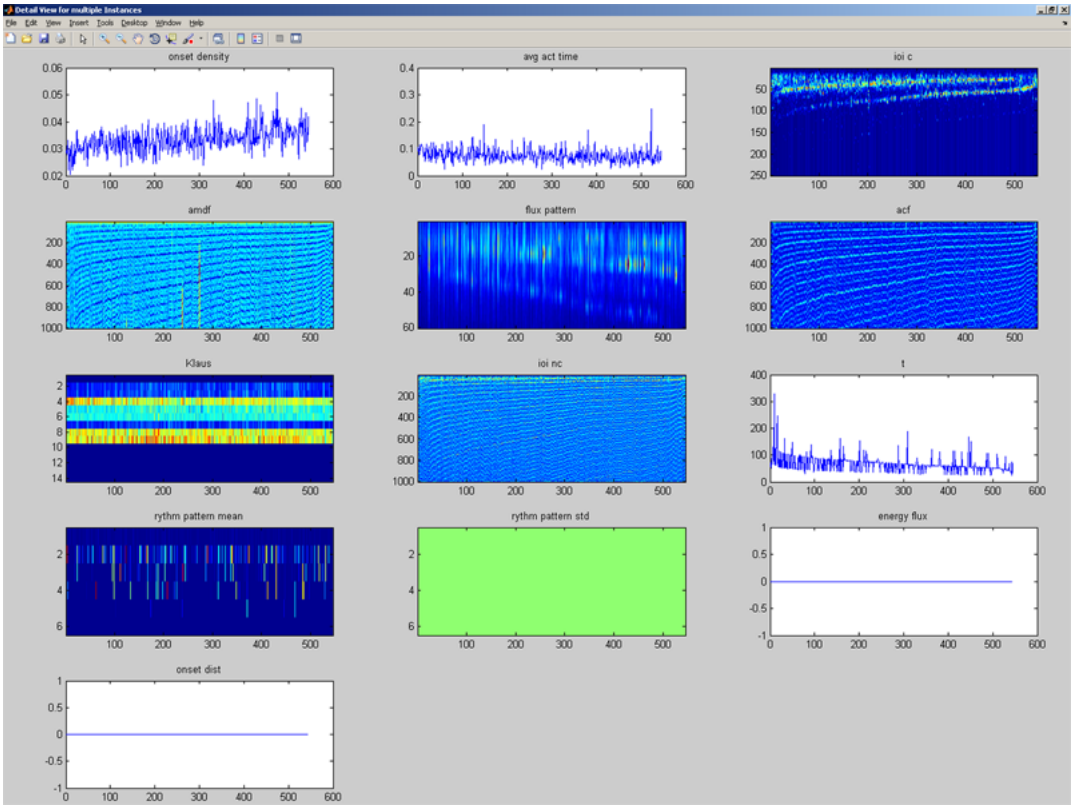

Figure 6.3: Feature visualization of the GUI.

6.2 Survey Database and Webform

For the survey presented in Section 8 a very simple webform and database was designed. The database consist of three tables called `data`, `users` and `files`. The `users` table stores information about the people participating in the survey: the name, and how often they participated. Since the specification of a name is voluntary this is solely for statistical purpose. The `files` table contains information about the songs used for the survey, namely: file ID, file name, and the tempo in BPM. The `data` table contains the information provided by the participants: time and date of the vote, user ID, which files had to be compared (their ID) and the vote provided by the participant (first file faster or second file faster).

The webform as shown in Figure 8.2 is a simple PHP[3] script storing the input in the database. As audioplayer a simple flash player taken from 'Audio Player Wordpress Plugin'[4] was used.

[3] http://www.php.net/
[4] http://www.1pixelout.net/code/audio-player-wordpress-plugin/

7 Evaluation

7.1 Data

The test data for evaluation of the single tempo estimation algorithms consists of three different data sets which origin from other works on tempo extraction. [SWS07] uses the same combination of test sets. The 'ballroom' dataset is a well known dance music collection from 'BallroomDancers.com' consisting of 698 audio excerpts of 30 seconds. Figure 7.1 shows the tempo distribution of this dataset. The tempo is given in beats per minute (BPM) which usually (but not necessarily) corresponds to the speed of quarter notes in the piece.

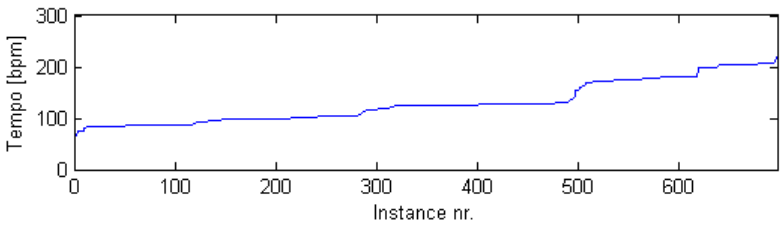

Figure 7.1: Tempo distribution of song excerpts in ballroom training dataset.

The 'Songs' dataset (publicly available at the ISMIR'04 tempo induction contest website[1]) consists of 465 audio excerpts of 20 seconds. It contains music from different genres including rock, classic, electronica, latin, samba, jazz, afrobeat, flamenco, balkan and greek. Figure 7.2 shows the tempo distribution of this dataset.

[1] http://www.iua.upf.es/mtg/ismir2004/contest/tempoContest/

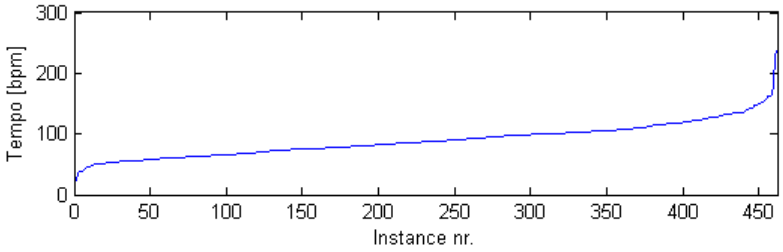

Figure 7.2: Tempo distribution of song excerpts in songs training dataset.

The third dataset called 'Pop' is a dataset created in [SWS07] by using music from their personal music collection and searching tempo annotation for the single songs on appropriate websites. It contains songs (full length) of popular music from the last two decades. Figure 7.3 shows the tempo distribution of this dataset. Figure 7.4 shows the tempo distribution of all three datasets merged together. The features of this merged set are used for cross-validation of the single machine learning algorithms.

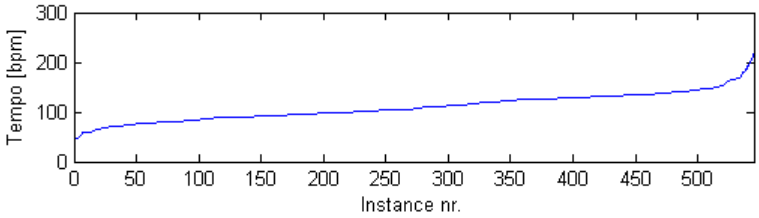

Figure 7.3: Tempo distribution of songs in pop training dataset.

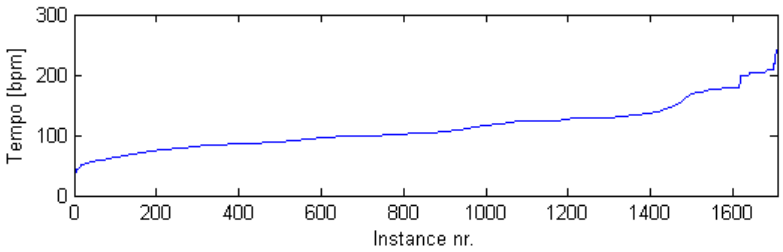

Figure 7.4: Tempo distribution of combined training set consisting of all three datasets.

7.2 Error Calculation

We are not always able to calculating the exact tempo of an audio track because of the nature of the proposed algorithm. If we are asked to classify a song with an tempo of 115.5 and there is no song in our training library with exact this tempo, the algorithm will never return this tempo. There would be possibilities to improve this drawback (like KNN with mean of n neighbors or SVR) but this improvements would not really solve this problem. For the applications this work aims on, this is not really a big problem. One can also control this behavior by choosing the right files for the training library.

Knowing about this, we have to consider this circumstance also for the evaluation process. Therefore, we define a window/a tempo range where an estimated tempo is considered correct if it lies within this window. For the evaluation procedures this tolerance window is always 4% of the ground truth tempo. This value has also been proven to be adequate in [SWS07].

While we used precision and recall when dealing with onset detection, here the accuracy measure is more useful. The calculation we use here follows [Wik09a] and is defined as:

$$acc = \frac{N_c}{N_{total}} \tag{7.1}$$

Where N_c is the number of correct estimated tempi (which lie within the 4% tolerance windwo) and N_{total} is the overal number of songs which had to be analysed.

A very interesting fact is, that often when the estimation was wrong, the estimated tempo is an integer multiple or fraction of the real tempo (the question stays if the annotated tempo is the real tempo). This is the kind of error we want to correct with the two stage classification process presented in this work, so this errors are of special interest to us. To see if the an error is of this kind, a special error measure is used (like in [SWS07]). The formula for this error measure is:

$$e = \begin{cases} \frac{\widehat{t}}{t} - 1 & \widehat{t} > t \\ -\left(\frac{t}{\widehat{t}} - 1\right) & \widehat{t} <= t \end{cases} \tag{7.2}$$

Where t is the ground truth tempo and $\widehat{t}$ is the estimated tempo. Using this error measure, double tempo errors result in values of 1 and half tempo errors result in values of -1.

7.3 Results and Discussion

In this section the results of the classifiers trained with different features (and feature combinations) are presented.

To evaluate the proposed algorithm we first analyze the performances of the single stages separately. To do this we first try to estimate the tempi of our songs in the training database with KNN classification (as proposed by [SWS07]). Then we test the first stage by trying to put the songs into the correct tempo classes using SVM classification. In the last step we take a look at the results if we combine first and second stage.

7.3.1 Tempo Classification Results using KNN

The first series of evaluation runs were tempo estimation runs using KNN classification. The evaluation should provide information about how the parameters of KNN classification influence the classification results. Details on the parameters and their meaning are provided in Section 5.1. Figures 7.5, 7.6 and 7.7 show classification results using different parameter settings and number of neighbors.

Figure 7.8 shows the classification results ignoring double and half tempo errors. This represents the optimal case when the first tempo estimating stage finds the right tempo classes for every song and the mapping is able to map the tempo into the right tempo range.

As mentioned in Section 5.1 the KNN classifier is not meant to work on data with continuous values. The problem that a certain value can never be the result of a

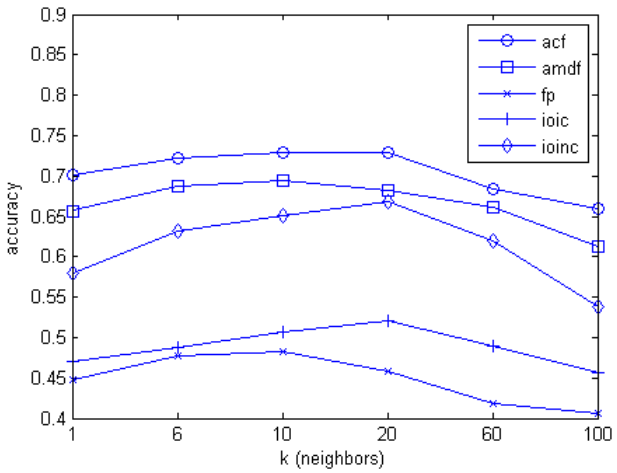

Figure 7.5: Accuracies for KNN classification using cityblock distance function.

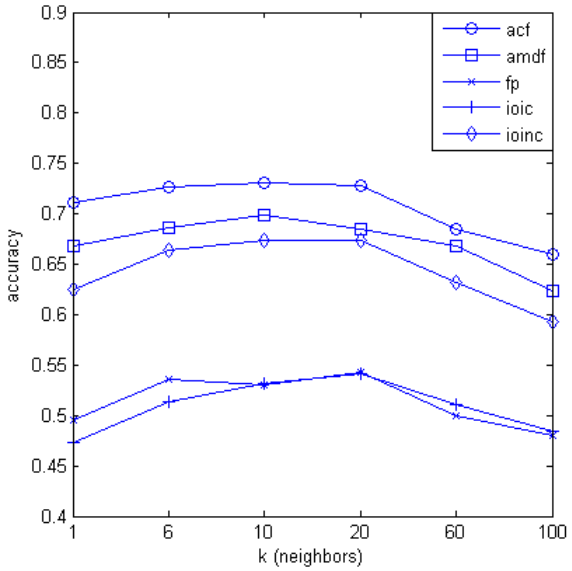

Figure 7.6: Accuracies for KNN classification using correlation distance function.

KNN classifier if its not in the samples is addressed by a 4% tolerance window in which the result is considered to be equal. The classifier perfomance itselfe is also vulnarable to continous values: Already slightly different values are not considered to vote for the same group. Therefore, the ground truth values should be rounded

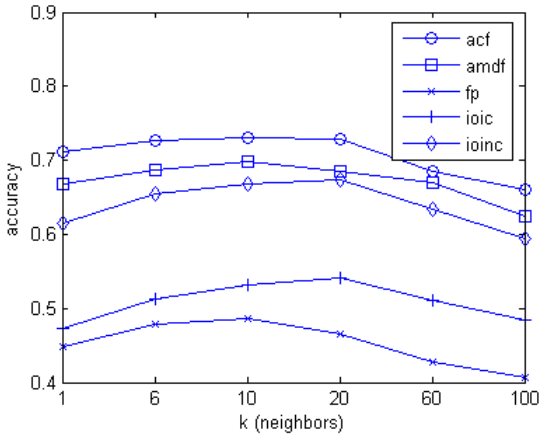

Figure 7.7: Accuracies for KNN classification using euclidean distance function.

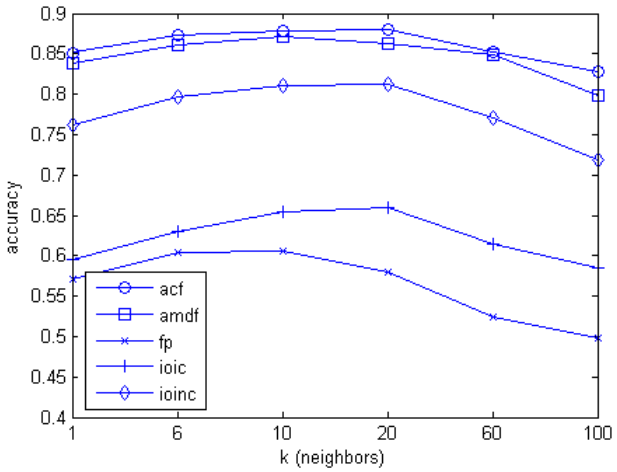

Figure 7.8: Accuracies for KNN classification using euclidean distance function. Double or half tempos are considered correct.

to integers to improve classification performance. The results for rounded ground truth values are shown in Figure 7.9, Figure 7.10 and Figure 7.11.

Table 7.1 shows best results of each parameter set. The best performing feature is always ACF, the best performing number of neighbors is always 10.

The results for KNN classification are quite similar as presented in [SWS07]. The

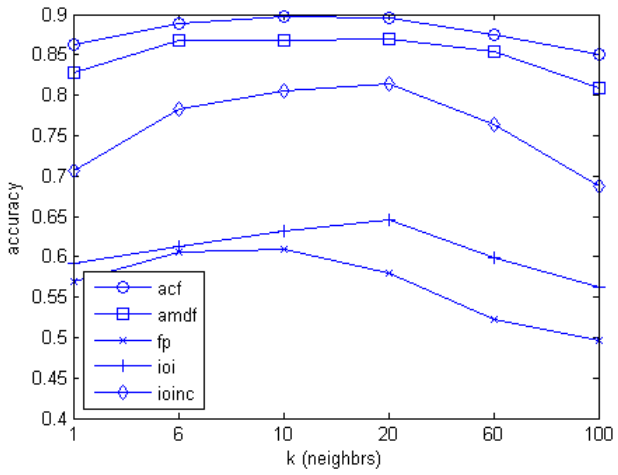

Figure 7.9: Accuracies for KNN classification using cityblock distance function. Double or half tempos are considered correct. Ground truth values (BPM) rounded to integers.

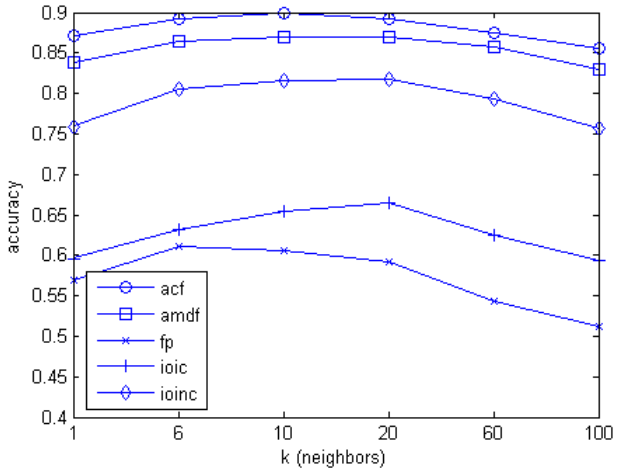

Figure 7.10: Accuracies for KNN classification using euclidean distance function. Double or half tempos are considered correct. Ground truth values (BPM) rounded to integers.

optimal feature seems to be ACF—all other features do perform much worse. For the final estimation algorithm a KNN classifier with 10 neighbors and correlation distance function was used. The feature used was ACF and the tempo values

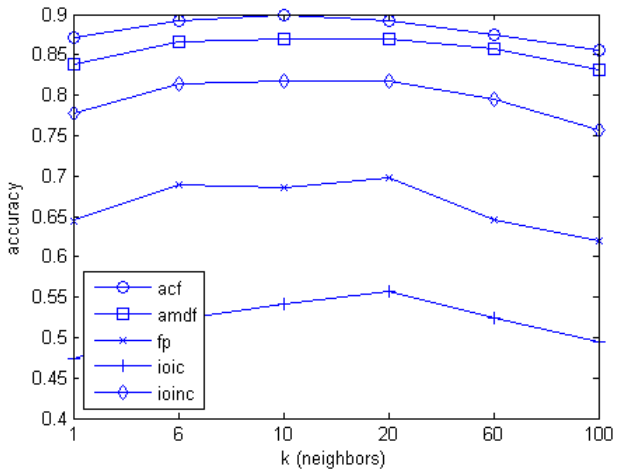

Figure 7.11: Accuracies for KNN classification using correlation distance function. Double or half tempos are considered correct. Ground truth values (BPM) rounded to integers.

Distance	Ground truth	Double errors	Accuracy
Cityblock	Rounded	No	89.70% (acf)
Correlation	Rounded	No	89.98% (acf)
Euclidean	Rounded	No	89.93% (acf)
Euclidean	Decimal	No	88.00% (acf)
Cityblock	Decimal	Yes	72.95% (acf)
Correlation	Decimal	Yes	73.01% (acf)
Euclidean	Decimal	Yes	73.01% (acf)

Table 7.1: Results for tempo classification using solely KNN classifiers and single features.

where rounded to integer values for KNN classification.

7.3.2 Tempo Class Classification Results using SVM

The SVM classification evaluation was done using the optimized parameters generated by the genetic algorithm. The SVM were trained with the single features on different test sets. The amount of tempo ranges was also varied to find an amount

of ranges where classification still works satisfyingly. Figure 7.18 shows all results at a glance. Figure 7.12 shows the results of a three class SVM trained on the songs data set. Figure 7.14 shows the results for the ballroom dataset, Figure 7.13 shows the results for the pop dataset and Figure 7.15 shows the results for all datasets combined using three classes.

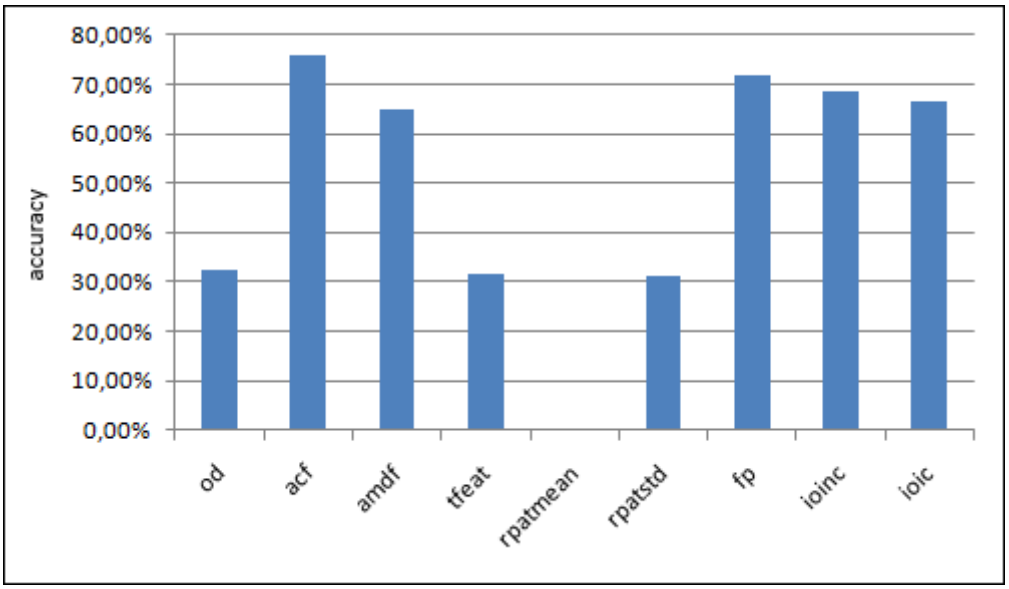

Figure 7.12: SVM classification results for songs dataset using three ranges.

One can easily see which features perform well in combination with SVM classification. ACF, AMDF, FP, non consecutive IOI and consecutive IOI are definitely the five best performing features. To see how the classification results changes when using more tempo ranges, we trained SVM classifiers with five and seven classes. We only used the pop dataset for figures with five respectively seven datasets in this section since the SVM performed best with the pop dataset. If the results get worse using more classes, one can see the change best when working with the best performing dataset. Figures 7.16 and 7.17 shows the classification results for five respectively seven datasets.

Figure 7.18 shows all results in one diagram. Table 7.2 shows SVM classification results using feature combinations. In the lines commented with 'majority vote' the SVMs were trained for the single features and the results were combined after

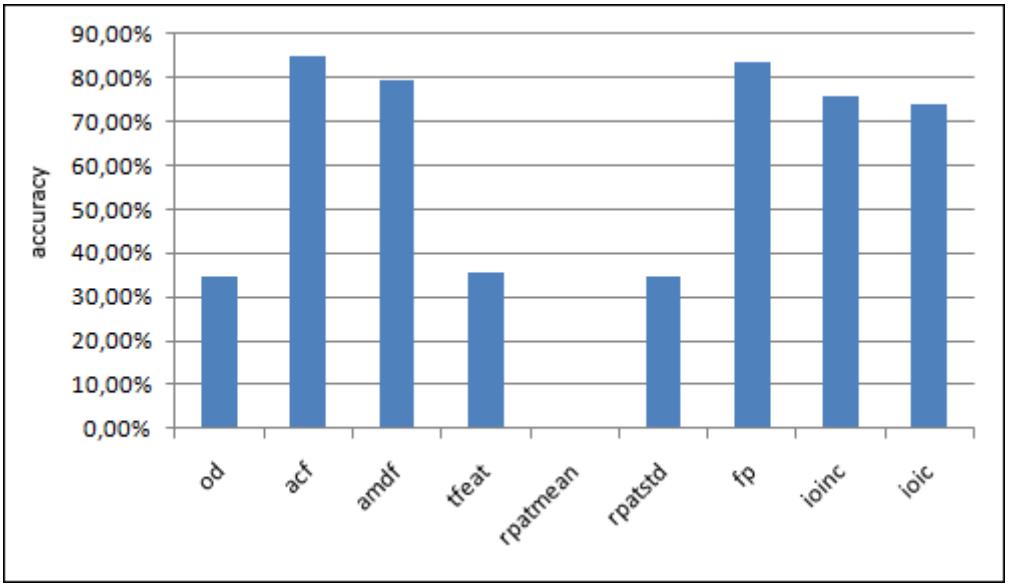

Figure 7.13: SVM classification results for pop dataset using three ranges.

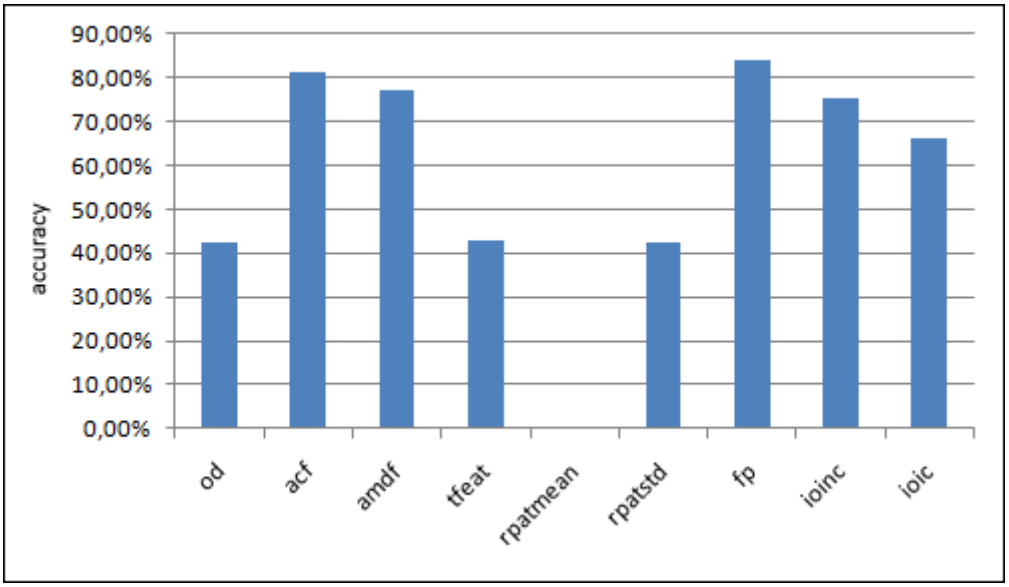

Figure 7.14: SVM classification results for ballroom dataset using three ranges.

classification using a majority rule voting.

As one can easily observe, using all features in a single vector does not work well. The resulting accuracy for the mix of all datasets with all features is just 35.78%.

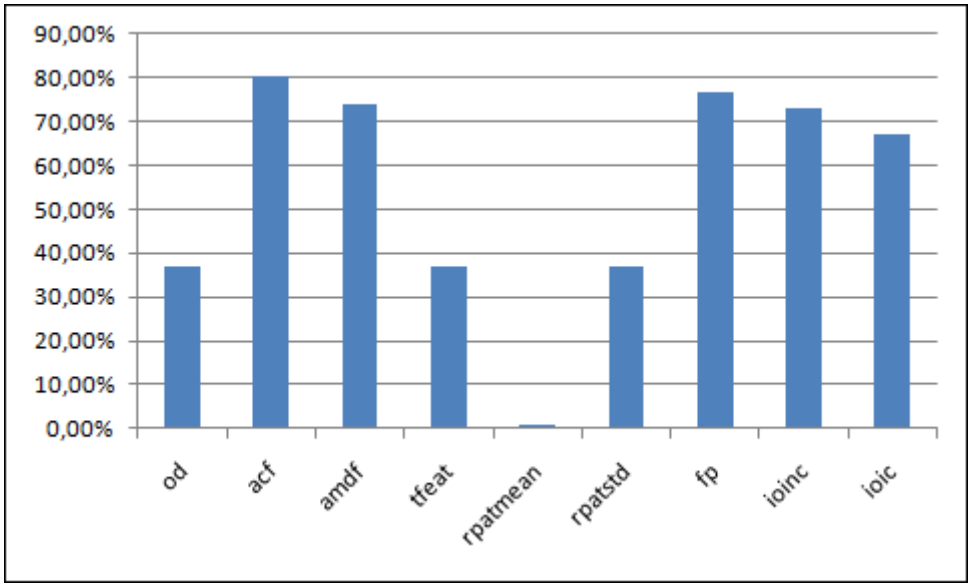

Figure 7.15: SVM classification results for all three dataset using three ranges.

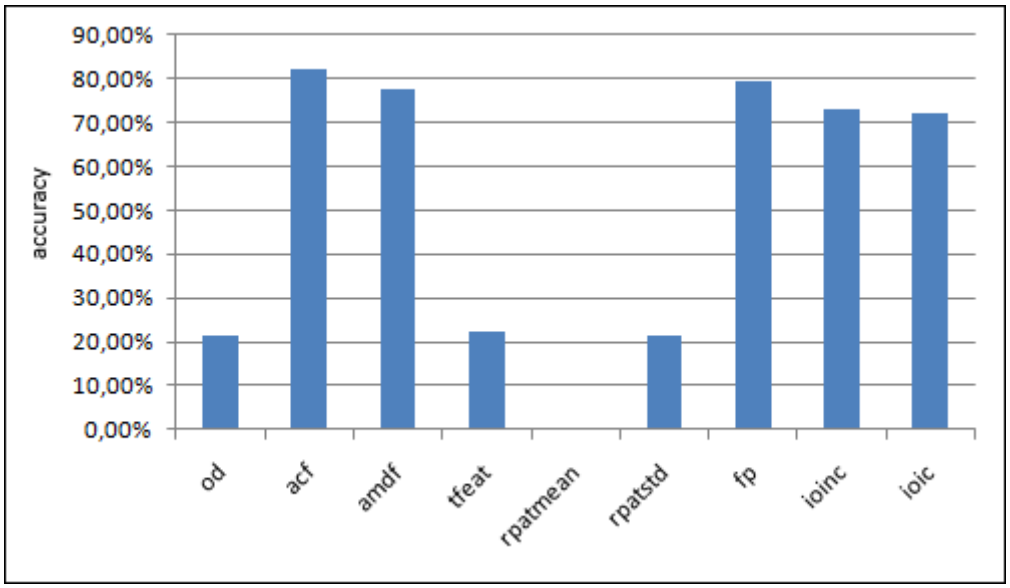

Figure 7.16: SVM classification results for pop dataset using five ranges.

Therefore, we introduced a majority voting algorithm. We trained different SVMs for every feature and combined the classification results afterwards using a simple majority vote algorithm. If no consensus can be obtained, the vote of the first

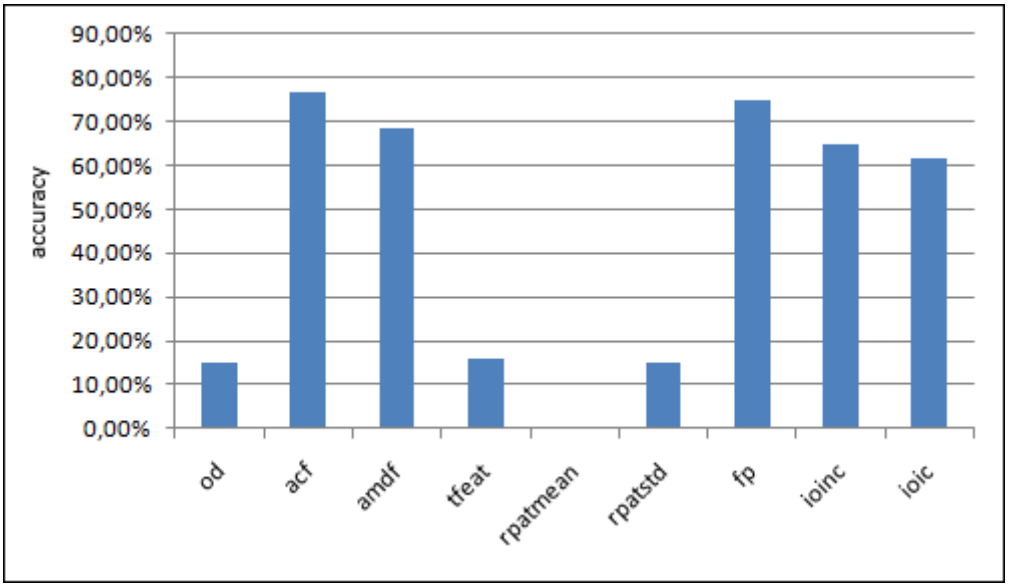

Figure 7.17: SVM classification results for pop dataset using seven ranges.

Dataset	Feature	Nr. of Ranges	Accuracy
pop	all	3	35.78%
all	all (majority vote)	3	77.93%
all	ACF, FP, IOINC, AMDF (majority vote)	3	75.12%
pop	all (majority vote)	5	79.63%
pop	ACF, FP, IOINC, AMDF (majority vote)	5	80.00%
pop	all (majority vote)	7	64.22%
pop	ACF, FP, IOINC, AMDF (majority vote)	7	74.45%

Table 7.2: SVM classification results using combination of features.

feature is taken—therefore, the features are sorted by their own accuracy (ACF first).

The evaluation with SVM were all done with rounded BPM values and even sample size distributed ranges. We used leave one out evaluation using the whole test set for this test.

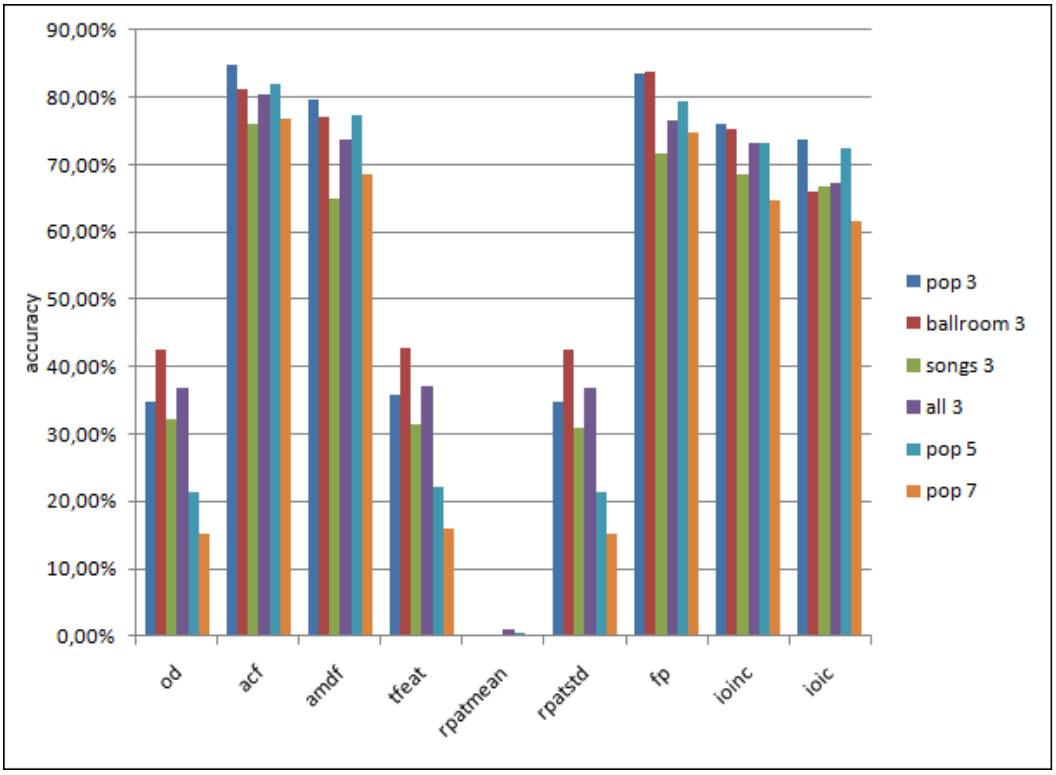

Figure 7.18: All SVM classification results compared in one diagram.

7.3.3 Tempo calculation using Tempo class and Tempo Classification

To combine the two classification stages, the best performing feature combinations and parameter settings of the single stages have been used. For the first stage we used five range SVM classifiers using five features, namely: ACF, AMDF, IOINC, IOIC and FP. The results of this five SVMs are combined using majority voting. The parameter settings for the SVM had been optimized by genetic algorithms. For the second stage we use KNN classifiers using ACF feature, 10 neighbors and correlation distance function.

We evaluated two possible combination methods for the first and second stage. These methods are:

1. **Mapping of estimated tempo into estimated range:** Here we assume

that whenever the estimated tempo is not within the estimated range, an error occurred where the estimated tempo is a multiple of two of the real tempo. Therefore, we divide/multiply the tempo so it is in the estimated range. If the tempo can not be corrected (i.e. mapped into the estimated range) this way, the original tempo is used.

2. **Estimating tempo within estimated range:** Here the range is first estimated. Then, only samples which are within this tempo range are used as training instances for the KNN classifier. This means the tempo of the second stage will always be within the estimated range.

Figure 7.19 shows how the combination method influences classification results. The results are quite similar for both methods. Estimating the tempo within the estimated tempo range performs slightly better for almost all datasets.

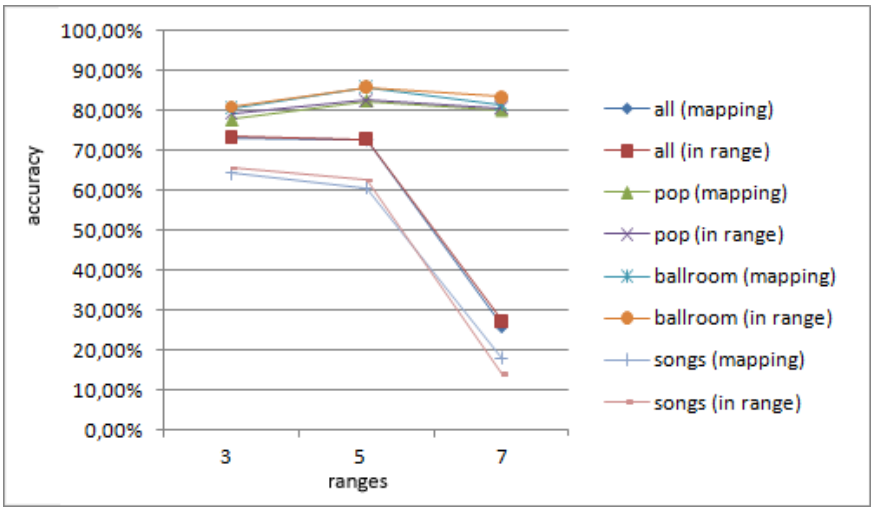

Figure 7.19: Results for two staged classification using different combination methods.

The second parameter which was evaluated in this section was the amount of ranges used for tempo range classification. Here Figure 7.20 shows that five ranges work best for all datasets except for the songs dataset.

In Figure 7.21 all results are shown again in a table. The results for every data set including the combination of all sets, as well as the first and second stage

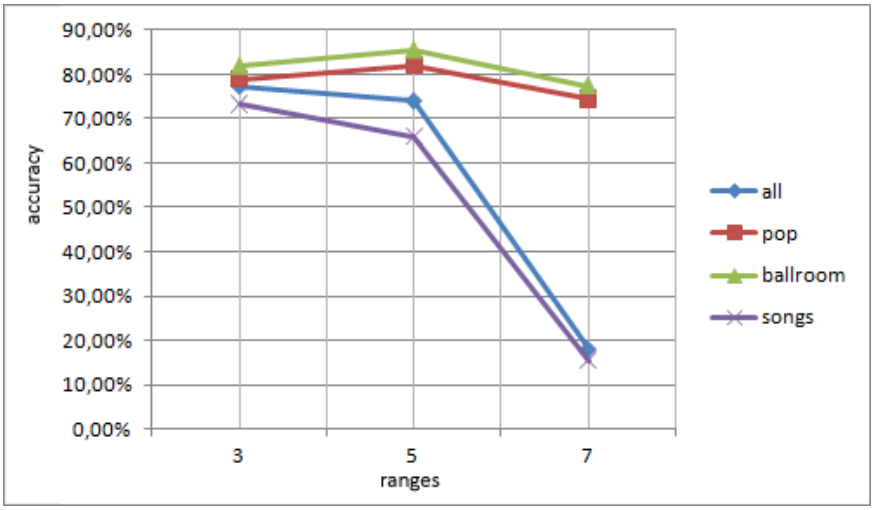

Figure 7.20: SVM classification results with different amount of ranges and data sets. All results obtained using ACF, AMDF, IOINC and FP features with majority vote combination.

accuracies are shown. The evaluation shows that the range classifier can improve the results of tempo estimation when the accuracy of the range classifier is high enough.

dataset	stage 1		stage 2	combinated	
	# ranges	acc	acc	method	acc
all	3	77,28%	75,12%	mapping	73,30%
all	3	77,28%	75,12%	in range	73,40%
all	5	73,95%	75,12%	mapping	72,60%
all	5	73,95%	75,12%	in range	72,89%
all	7	71,66%	75,12%	mapping	71,25%
all	7	71,66%	75,12%	in range	71,58%
pop	3	78,90%	80,00%	mapping	77,80%
pop	3	78,90%	80,00%	in range	79,45%
pop	5	82,02%	80,00%	mapping	82,39%
pop	5	82,02%	80,00%	in range	82,57%
pop	7	74,50%	80,00%	mapping	80,18%
pop	7	74,50%	80,00%	in range	80,73%
ballroom	3	81,81%	84,96%	mapping	80,66%
ballroom	3	81,81%	84,96%	in range	80,95%
ballroom	5	85,39%	84,96%	mapping	85,96%
ballroom	5	85,39%	84,96%	in range	85,96%
ballroom	7	77,36%	84,96%	mapping	81,52%
ballroom	7	77,36%	84,96%	in range	83,42%
songs	3	73,33%	66,67%	mapping	64,30%
songs	3	73,33%	66,67%	in range	65,59%
songs	5	66,02%	66,67%	mapping	60,65%
songs	5	66,02%	66,67%	in range	62,58%
songs	7	63,66%	66,67%	mapping	59,57%
songs	7	63,66%	66,67%	in range	62,15%
stage 1:	svm; feat.: majority(acf, fp, ioinc, ioic, amdf)				
stage 2:	knn; feat.: acf, N=10				

Figure 7.21: Results of combination of first and second stage.

8 Survey

Till now all we tried was to estimate the manually tagged tempo of our training set of music files. The final goal of this work was to estimate tempo which corresponds to the listeners perceptual tempo. We want to be able to sort music according to their perceived tempo respectively find music with the same perceptual tempo. To verify this a user survey was done which tries to evaluate the decisions made by our algorithm lists.

8.1 Data

As data for the survey the files of the pop dataset were sorted into seven bins according to their tempo. The boundaries for the tempo bins were: [0, 62.5, 87.5, 112.5, 137.5, 162.5, 187.5, 300]. To have a more or less linear distribution of tempos in the set, the numbers of files was chosen almost equally over the bins. A completely linear distribution was not possible because of the low number of files in some bins (especially in the high-tempo bin). The actually chosen numbers are: [14, 15, 17, 18, 16, 14, 6] - which still reflects the basic tempo distribution. Table 8.1 shows the files and their annotated BPM-tempo used in the survey.

8.2 System

To make the survey accessible to a broad amount of test persons it was published on a web server. The received data was stored in a database. The webform for

ID	Name	BPM	ID	Name	BPM
1	Seal - Kiss From A Rose	44	51	Falco - Tango The Night	126
2	Boyz II Men - I'll Make Love To You	47	52	Real McCoy - Another Night	126
3	Bryan Adams - Have You Ever Really Loved A Woman	48	53	Alien Ant Farm - Smooth Criminal	127
4	Boyz II Men - 4 Seasons Of Loneliness	50	54	Dr. Alban - It's My Life	128
5	Falco - Jeanny	53	55	Faithless - God Is A DJ	130
6	Backstreet Boys - I'll Never Break Your Heart	54	56	French Affair - My Heart Goes Boom	130
7	Prince - Purple Rain	58	57	Falco - America	131
8	Whitney Houston & Mariah Carey - When You Believe	59	58	Duran Duran - Girls On Film	133
9	Celine Dion - All By Myself	60	59	Ice MC - Think About The Way	133
10	Mariah Carey - Hero	60	60	Placebo - Every You Every Me	133
11	Mariah Carey - I Still Believe	60	61	Foo Fighters - Stacked Actors	135
12	R. Kelly - I Believe I Can Fly	60	62	Blondie - Atomic	136
13	Toni Braxton - I Don't Want To	60	63	Rank 1 - Airwave	136
14	Toni Braxton - Spanish Guitar	60	64	Vengaboys - We Like To Party	136
15	Busta Rhymes - Gimme Some More	68	65	Paul Van Dyk - For An Angel '98	138
16	Creed - With Arms Wide Open	70	66	Rank 1 - Breathing (Airwave)	138
17	Creed - Are you Ready	72	67	Elton John - Made In England	139
18	Eminem - Cleanin' Out My Closet	74	68	Nightwish - Astral Romance	141
19	Green Day - Brain Stew	76	69	Pharao - I Show You Secrets	142
20	No Doubt - Don't Speak	76	70	Dee Dee - Forever	144
21	Collective Soul - The World I Know	77	71	Propellerheads - History Repeating	144
22	Jay-Z - Hard Knock Life	77	72	Nightwish - Elvenpath	146
23	Massive Attack - Teardrop	77	73	Smashing Pumpkins - Tonight, Tonight	146
24	Fugees - No Woman, No Cry	78	74	Wolfsheim - A Look Into Your Heart	146
25	Lenny Kravitz - Again	79	75	The Offspring - Original Prankster	147
26	TLC - Diggin' On You	80	76	Blink 182 - All The Small Things	148
27	Cypress Hill - I Wanna Get High	82	77	Nightwish - Come Cover Me	148
28	Kid Rock - Cowboy	84	78	Nightwish - Know Why The Nightingale Sings	149
29	Eminem - Lose Yourself	86	79	Sex Pistols - God Save The Queen	149
30	Christina Aguilera - Genie In A Bottle	88	80	The Cure - Just Like Heaven	152
31	Smashing Pumpkins - Muzzle	88	81	Bomfunk MC's - Freestyler	164
32	The Cure - Strange Attraction	90	82	DJ Shadow - You Can't Go Home Again	164
33	50 Cent - In Da Club	91	83	Green Day - Hitchin' A Ride	164
34	Dr. Dre - Keep Their Heads Ringing	91	84	Nirvana - Stay Away	164
35	POD - BOOM	91	85	Nirvana - Breed	165
36	Rage Against The Machine - Microphone Fiend	91	86	Flying Blind - Smokescreen	168
37	The Clash - The Guns Of Brixton	93	87	Sum 41 - Motivation	168
38	Red Hot Chili Peppers - Californication	96	88	Bad Religion - A Walk	169
39	Enya - Anywhere Is	97	89	Stone Temple Pilots - Vasoline	169
40	Pink - Don't Let Me Get Me	98	90	The Offspring - The Meaning Of Life	172
41	Guano Apes - Lords Of The Boards	104	91	Metallica - Die, Die My Darling	179
42	Limp Bizkit - My Generation	105	92	Survivor - Eye Of The Tiger	180
43	Linkin Park - In The End	105	93	Las Ketchup - The Ketchup Song (Asereje)	185
44	Baltimora - Tarzan Boy	110	94	Green Day - Nice Guys Finish Last	187
45	Korn - Make Me Bad	110	95	Red Hot Chili Peppers - Around The World	192
46	U2 - Elevation	110	96	The White Stripes - Fell In Love With A Girl	192
47	Eels - MR E'S Beautiful Blues	114	97	Nightwish - Ghost Love Score	203
48	Anastacia - I'm Outta Love	125	98	Cheap Trick - I Want You To Want Me	206
49	Highlight - Can You Feel It	125	99	Pearl Jam - Spin The Black Circle	218
50	The White Stripes - 7 Nation Army	125	100	Metallica - Master Of Puppets	220

Figure 8.1: Files used for survey with annotated tempo.

user input was held very simple to be accessible to people without any musical nor computer-science background. The used interface is shown in Figure 8.2. To keep the evaluation process simple only two songs are compared at a time. This reduces the time needed for the probands and makes the evaluation more interesting since a complete evaluation can be done very quickly and is more likely to be repeated more often.

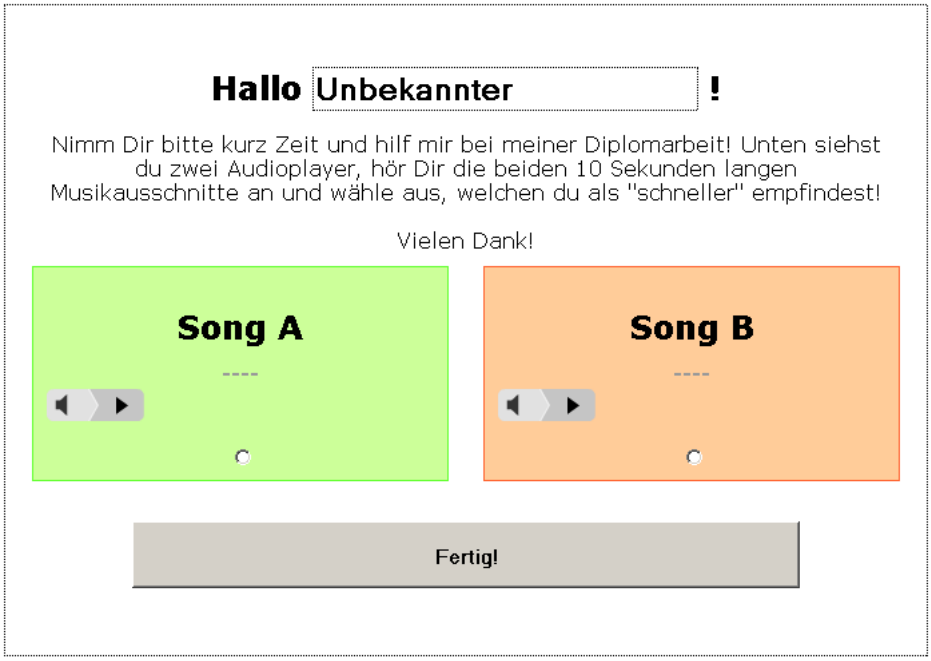

Figure 8.2: Webform for user-survey. The colored blocks hold the simple audio player and a radio button to choose the faster perceived song. The button below confirms the choice and generates a new pair to compare.

8.3 Results

The resulting data of the survey consists of a list of pairwise compared songs. The first thing we tried was to use this data as ordered-relation to create a directed acyclic graph (DAG). Unfortunately but foreseeable, the gained data from the survey was not acyclic. This means there where cycles in the relation like: Song A is slower than song B, song B slower than song C and song C slower than song A. This fact may also be a hint that humans do not have a absolute sense for tempo and double/half errors are likely to be made by humans too.

The next try was to evaluate how many of the relations can be confirmed by (a) the BPM tempo and (b) the estimated tempo. We did this by taking the real BPM tempo and the estimated tempo of the files and checked if the choice the user made is true regarding the BPM tempo and the estimated tempo. Here BPM tempo scores with an accuracy of 75.67% and the estimation algorithm with 65.73%. This

result has to be put into perspective regarding the fact that we are comparing a well ordered list with a partially cyclic list, so 100% could never be reached by any algorithm respectively tempo list. Another thing we have to take into account is that we trained the algorithms on the BPM data (which was certainly not 100% accurate—see Section 9) so it is not a surprise that the machine learning algorithm do not perform better than the direct comparison in this case.

Another interesting question was, since the machine learning algorithm do not concur as often with the user choice as the BPM tempo, how big are the mistakes (in terms of distance) of the machine learning algorithm and the BPM tempo respectively. To evaluate this we summed up the tempo difference of the two songs where the estimated tempo/BPM tempo was in the wrong order regarding the user choice. The absolute error for BPM tempo is then 35893 and the error for estimated tempo 20806. If one wants to weight the error by the absolute tempo values (we use the mean of both files here), so big absolute error at higher tempi are not weighted as heavily as big absolute errors at lower tempi, we get an absolute error of 301.34 for BPM tempo and 196.85 for estimated tempo. This leads to the conclusion that the BPM tempo is more often correct when it comes to ordering (which could be coincidence with this special data set) but the estimated tempo is much closer to the perceived tempo as the absolute error is smaller.

9 Conclusion

One of the main difficulties of this work resulted from the fact that the training files for the machine learning algorithms where labeled with BPM tempo. The first problem was that the BPM tempo was most probably measured by humans. Therefore, some of the songs had been labeled wrong or not accurate enough. On the other hand it is very likely that the training database already contains double/half tempo errors. A good example is the *songs* training database used in this work. It shows in most evaluation that the algorithms perform very poorly with this training database (see figures 7.19, 7.18). Reason for this might be that the songs are labeled poorly: Some songs are in the pop as well as in the songs database e.g. 'The Winner Takes it All' by *ABBA* but labeled differently in both datasets: 128BPM respectively 122BPM. When estimating the tempo with a click-along beat counter the 128BPM (the label in pop dataset) seemed to be correct. The bad labeled test files have also a huge influence on the global results when using all datasets—which can also be observed in Figure 7.19. For future experimenting it would be important to recheck the training data to avoid bad results caused by wrong ground truth data.

Another problem rose when putting together the song excerpts for the survey. Usually most songs, except for those of electronic dance music genres, do not have constant tempo over their whole length. This may reach from slight tempo variation, stronger tempo changes (like in classical music) or even parts with completely different tempo or meter. An extreme example for this is the last song of the survey song list (see Figure 8.1): 'Master of Puppets' by *Metallica*. It contains a long guitar solo in the very middle of the song. The song itself is labeled with 220BPM, the guitar solo is accompanied by slow drums and more soft guitars than the rest

of the song so a listener would be more likely to label it with 110BPM rather than 220BPM. The rest of the song is very fast, but as we just used the middle part of the song the 220BPM are not really appropriate for this sample. A solution for this problem would be to use 'tempo tracks' rather than one fixed tempo for a song in the training database. This tempo track would provide a tempo function over the whole length of the song. On the other hand we could try to estimate the tempo of a song at more than just one position of the track.

Another important conclusion was, that the range classifier has to work very accurate, to improve the global result. The range accuracy should be higher than 80% otherwise the combination of tempo and range classifiers lead to worse results than tempo classification alone. This can be observed in Figure 7.21.

Of course another crucial factor for good results is feature calculation. For that reason a big amount of work was invested into onset detection. The onset detection function is the basis for the most of the best performing features (ACF, AMDF, IOIC, IOINC) and therefore a important basis of the whole work. Some of the features meant to provide additional information for the 'perceived' tempo part (OD, RPAT) did not work very well, but finding such features would sure improve classification results.

The evaluation of the machine learning part showed that most of the machine learning algorithms perform equally well if configured properly. There may lie more potential in optimizing feature calculation than experimenting with more machine learning algorithms.

One of the interesting conclusions of the survey is that humans may also make double/half error when estimating tempo. So the perceived tempo problem should maybe be redefined in some way to reflect the very relative perception system of humans. Maybe for untrained human ears there exist nothing like a absolute order of songs concerning tempo. Maybe we should look for some kind of similarity measure for tempo and some kind of fuzzy system to explain tempo relations like: little slower, slower, somehow equally fast...

List of Tables

List of Figures

10 Bibliography

[ADR04] M. Alonso, B. David, and G. Richard. Tempo and beat estimation of musical signals. In *Proc. of the International Conference on Music Information Retreival (ISMIR'04)*, pages 158–163, 2004.

[ADR06] Miguel Alonso, Bertrand David, and Gaël Richard. Tempo extraction for audio recordings. In *MIREX Audio Tempo Extraction Contest*, 2006.

[Ahm07] Teemu Ahmaniemi. Influence of tempo and subjective rating of music in step frequency of running. In *Proc. of the International Conference on Music Information Retreival*, pages 183–184, 2007.

[APT+07] Iasonas Antonopoulos, Aggelos Pikrakis, Sergios Theodoridis, Olmo Cornelis, Dirk Moelants, and Marc Leman. Music retreival by rhytmic similarity applied on greek and african traditional music. In *Proc. of the International Conference on Music Information Retreival*, pages 297–300, 2007.

[BDA+05] Juan Pable Bello, Laurent Daudet, Samer Abdallah, Chris Duxbury, Mike Davies, and Mark B. Sandler. A tutorial on onset detection in music signals. *IEEE Transactions on Speech and Audio Processing*, 13(5):1035 – 1047, 2005.

[BDDS04] J. Bello, C. Duxbury, M. Davies, and M. Sandler. On the use of phase and energy for musical onset detection in the complex domain. *J. Bello and C. Duxbury and M. Davies and M. Sandler*, 11(6):553–556, June 2004.

[CK02] Alain de Cheveigne and Hideki Kawahara. Yin, a fundamental frequency estimator for speech and music. *J. Acoust. Soc. Am.*, 111(4):1917–1930, April 2002.

[CL07] Bee Yong Chua and Guojun Lu. Perceptual tempo determination from music signals. 2007.

[CL09] Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. Technical report, Department of Computer Science, National Taiwan University, Taipei 106, Taiwan (http://www.csie.ntu.edu.tw/ $\sim$ cjlin), 2009.

[CST05] Nello Cristianini and John Shawe-Taylor. *Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 9 edition, 2005.

[DB06] Matthew Davies and Paul Brossier. Fast implementation for perceptual tempo extraction. In *MIREX Audio Tempo Extraction Contest*, 2006.

[DG02] M. Davy and S. Godstill. Detection of abrubt spectral changes using support vector machines. an application to audio signal segmentation. In *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, pages 1313–1316, 2002.

[DGP99] C Drake, L. Gros, and A. Penel. How fast is that music? the relation between physical and perceived tempo. *Music Mind and Science*, 1999.

[Dix06] Simon Dixon. Onset detection revisited. In *Proc. of the Int. Conference on Digital Audio Effects (DAFx-06), Montreal, Canada*, pages 133–137, September 2006.

[DPW03] Simon Dixon, Elias Pampalk, and Gerhard Widmer. Classification of dance music by periodicity patterns. In *Proc. of the International Conference on Music Information Retreival*, 2003.

[DSD02] Chris Duxbury, Mark Sandler, and Mike Davies. A hybrid approach to musical note onset detection. In *Proc. of the Int. Conference on Digital Audio Effects (DAFx-02)*, volume 5, pages 33–38, 2002.

[Eck06] Douglas Eck. A tempo-extraction algorithm using an autocorrelation phase matrix and shennon entropy. In *MIREX Audio Tempo Extraction Contest*, 2006.

[Ell07] Daniel P.W. Ellis. Beat tracking by dynamic programming. Technical report, LabROSA, Columbia University, New York, July 2007.

[FU01] Jonathan Foote and Shingo Uchihashi. The beat spectrum: A new approach to rhythm analysis. In *Proc. of the International Conference on Multimedia and Expo*, 2001.

[FZ99] H. Fastl and E. Zwicker. *Psychoacoustics*. Springer, 1999.

[GD06] Fabien Gouyon and Simon Dixon. Influence of input features in perceptual tempo induction. In *MIREX Audio Tempo Extraction Contest*, 2006.

[JR01] F. Jaillet and X. Rodet. Detection and modeling of fast attack transients. In *Proc. of the Int. Computer Music Conference (ICMC'01)*, 2001.

[KEA06] A. P. Klapuri, A. J. Eronen, and J.T. Astola. Analysis of the meter of acoustic musical signals. *IEEE Transactions on Speech and Audio Processsing*, 14(1), 2006.

[Kla99] A. Klapuri. Sound onset detection by applying psychoacoustic knowledge. In *Proc. IEEE Conf. on Acoustics, Spech and Signal Processing (ICASSP'99)*, 1999.

[KP04] E. Kapanci and A. Pfeffer. A hierarchical approach to onset detection. In *Proc. of the Int. Computer Music Conference (ICMC'04)*, pages 438–441, 2004.

[MM04] M.F. McKinney and D. Moelants. Extracting the perceptual tempo from music. In *Proc. of the Int. Conference on Music Information Retrieval (ISMIR'04)*, 2004.

[Moo97] Brian C.J. Moore. *An Introduction to the Psychology of Hearing*. Academic Press, 5th edition, 1997.

[NTI04] Gabriel Pablo Nava, Hidehiko Tanaka, and Ichiro Ide. A convolutional-kernel based approach for note onset detection in piano-solo audio signals. In *Proc. of the Int. Symposium on Musical Acoustics*, pages 289–292, 2004.

[Pam01] E. Pampalk. Islands of music, analysis, organization, and visualization of music archives. Master's thesis, Institut f¨ur Softwaretechnik und Interaktive Systeme der Technischen Universit¨at Wien, 2001.

[Pam04] Elias Pampalk. A matlab toolbox to compute music similarity from audio. In *Proc. of the International Conference on Music Information Retreival, Barcelona, Spain*, 2004.

[Pee06] Geoffroy Peeters. Tempo detection and beat marking for perceptual tempo induction. In *MIREX Audio Tempo Extraction Contest*, 2006.

[PK07] Jouni Paulus and Anssi Klapuri. Combining temporal and spectral features in hmm-based drum transcription. In *Proc. of the International Conference on Music Information Retreival*, pages 225–228, 2007.

[PT07] Aggelos Pikrakis and Sergios Theodoridis. An application of emperical mode decomposition on tempo induction from music recordings. In *Proc. of the International Conference on Music Information Retreival*, pages 301–304, 2007.

[Sch98] E. D. Scheirer. Tempo and beat analysis of acoustic music signals. *J. Acoust. Soc. Am.*, 10(5):588–601, 1998.

[Set06] William A. Sethares. Tempo extraction via the periodicity transform. In *MIREX Audio Tempo Extraction Contest*, 2006.

[SO98] Alex J. Smola and Bernhard Sch Olkopf. A tutorial on support vector regression. Technical report, Statistics and Computing, 1998.

[SS01] W. A. Sethares and T. Staley. Meter and periodicity in musical performance. *J. NewMusic Research*, 30(2):149–158, 2001.

[SVN37] S. S. Stevens, J. Volkmann, and E. B. Newman. A scale for the measurement of the psychological magnitude pitch. *J. Acoust. Soc. Am.*, 8(3):185–190, Jannuary 1937.

[SWS07] Klaus Seyerlehner, Gerhard Widmer, and Dominik Schnitzer. From rhythm patterns to perceived tempo. In *Proc. of the International Conference on Music Information Retreival*, pages 519–524, 2007.

[Tza02] P. Tzanetakis, G. Cook. Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing*, 10(5), 2002.

[Uhl06] Christian Uhle. Tempo induction by investigating the metrical structure of music using a periodicity signal that relates to the tatum period. In *MIREX Audio Tempo Extraction Contest*, 2006.

[WF05] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, second edition, June 2005.

[Wik09a] Wikipedia. Accuracy, 2009.

[Wik09b] Wikipedia. Fourier transform, 2009.

[Wik09c] Wikipedia. Goertzel algorithm, 2009.

[Wik09d] Wikipedia. Precision and recall, 1 2009.

[Wik09e] Wikipedia. The sone scale, 2009.

[Wik09f] Wikipedia. Support vector machine, 2009.

[YD07] Wei You and Roger B. Dannenberger. Polyphonic music note onset detection using semi-supervised learning. In *Proc. of the Int. Conference on Music Information Retreival (ISMIR'07)*, 2007.

[Zwi61] E. Zwicker. Subdivision of the audible frequency range into critical bands (frequenzgruppen). *J. Acoust. Soc. Am.*, 33(2):248–248, February 1961.

Richard Karl Vogl

Staatsangehörigkeit: Österreich

Geburtsdatum: 25. Jänner 1982

Geburtsort: Linz

Ausbildung

- 1988 - 1992 Volksschule Steinbach an der Steyr
- 1992 - 1996 Hauptschule Grünburg
- 1996 - 2001 HTBLA Steyr, Höhere Abteilung für Elektronik, Ausbildungszweig Technische Informatik
- 2001 - 2002 Grundwehrdienst
- **Matura** mit **gutem Erfolg** bestanden
- Studium der **Informatik** seit September 2002

"Ich erkläre an Eides statt, dass ich die vorliegende Diplom- bzw. Magisterarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe."

Linz, am ....