A BIC-based approach to singer identification

Xavier Janer Mestres

Master thesis subitted in partial fulfillment of the requirments for the degree:

Màster en Tecnologies de la Informació, la Comunicació i els Mitjans Audiovisuals

UNIVERSITAT POMPEU FABRA

Supervisor: Xavier Serra and Pedro Cano

Department of Information and Communication Technologies
Universitat Pompeu Fabra
Spain
September 2007

Contents

Abstract

A new singer identification system is presented in this thesis. The system is based on the idea of using only the vocal segments of a song to build the model of a particular singer. The most important contribution of the technique is the way these vocal segments are located. The borders between vocal and instrumental parts are first detected with the Bayesian Information Criterion(BIC), which is fed with our new panning coefficients. Then, each segment is classified as vocal or instrumental by a decision tree based on MFCCs. Having vocal segments located, our method works like most speaker identification systems do, that is, by training a GMM for each singer through the Expectation-Maximization algorithm. The performance of the singer identification system has not been as successful as expected, however many other applications can be thought of based on the proposed segmentation technique.

Chapter 1

Introduction

In this chapter the main goals of the project are stated. We first explain the motivation for working on the field, and we describe briefly the different parts of the project.

The state of the art is also described in this chapter. Since the project involves two main tasks (speaker identification and singer identification), the overview of the state of the art is explained according to this classification.

1.1 Goal of the project

1.1.1 Context

With the rapid progress of computer performance and world-wide networking capacities, and the great popularity of P2P software, digital music is one the most transferred data types through the Internet. All these facts lead to a situation where music collections are very large and extremely varied in terms of music styles, and they keep growing from day to day. However, users demand easy and efficient ways to access these collections. Here is where Music Information Retrieval (MIR) appears.

MIR includes two complementart tasks, on one hand, content-based MIR tries to extract information from the audio data itself, on the other hand context-based MIR describes music files in terms of its contextual information, e.g. artist and song name, genre, lyrics... Because we deal with aspects of the audio data, this project falls into the category of content-based MIR.

From the point of view of content-based MIR, a lot of information can be extracted directly from the audio data. This information can result into low-level or high-level descriptors.

Low-level descriptors are closely related to the signal itself. Depending on the granularity from the signal, we can divide them into three categories [2]:

- **Instantaneous descriptors:** These can be obtained at any point of the signal, and describe the amplitude or the energy of the signal.

- **Descriptors from small regions:** These are normally spectrum-related descriptors, such as spectral centroid, spectral tilt or harmonicity, and they are called *from small regions* because the signal needs to be previously framed and windowed.

- **Descriptors from longer segments:** The most common goal of this segmentation is to obtain notes and silences separately. Then, a low-level analysis can be performed on the notes segments and features such as fundamental frequency, spectral shape or depth of vibrato can be extracted.

With combinations and transformations of low-level descriptors, a new set of features can be obtained: the high-level descriptors. This set contains information extracted from the audio that can be understood by an end-user with no previous knowledge in signal processing. Some examples are pitch (or melody), tempo and instrument.

Within MIR, one of the most challenging and user demanded problems is music recommendation and playlist generation. It is commonly solved by applying Machine Learning (ML) techniques provided with low- and high-level descriptors. As a result of these techniques, different kinds of similarity between songs can be obtained, which will be the basis to generate recommendation playlists.

1.1.2 Motivation

The singing voice is one of the most important elements in music [27], especially in popular music. Listeners can easily use the lead vocals of a song to identify the artist or band that plays a particular song. Therefore, an automatically-extracted model of the singer's voice can be a very useful musical descriptor in the field of content-based MIR. Applications are numerous, taking into account the large music libraries that are currently available through the Internet:

Find songs from the same artist: in popular music, songs from the same artist are almost always sung by the same singer. Thus, a model of the singer's voice could be used to automatically find songs from a particular artist within a large music collection, avoiding the tedious task of manually tagging each song.

Find songs by artists with similar voices: By defining a measure of comparison of two voice models, a distance between two models can be determined. Since these models are based on statistical measures, it is usually an easy task. Therefore, it could be used to find music from artists with similar voices.

Find cover songs: Together with other descriptors, cover songs can be detected. If two songs share a lot of features (such as tempo, structure, chords. . .) but the singer's models are very different one from each other, it can be hypothesized that they are cover songs.

Find guest artists' cameos: Having models for hundreds of singers, small participations of some artists in other band's recordings can also be easily found.

Regarding artist identification, the best systems are still far from perfect or even excellent results. In Music Information Retrieval Evaluation eXchange (MIREX) 2005 [1], an Audio Artist Identification contest was made and the best performance had a 72.45% of correct identifications in a large database. These results show that there is still some work to do in this field, and that's the main goal of this project. However, facing this problem is not easy, and several steps have to be done. They are described in the next section, the description of the project.

1.1.3 Description of the project

The first draft of the project was to investigate on new features for text-independent speaker identification. The starting point were some descriptors developed by the singing voice research team at the Music Technology Group (MTG), so my specific task was to analyse these descriptors and decide whether they were useful for the particular task of speaker identification, under clean or noisy conditions.

After some months of analysis, comparisons and tests, the results were quite discouraging. Neither the whole set of descriptors of any subset of them gave good identification results. In all the tests done, these new descriptors were outperformed by the descriptors commonly used in this task, the Mel-scale Frequency Cepstral Coefficients.

Then I came across a segmentation algorithm, the Bayesian Information Criterion (BIC), which applied to TV broadcast audio was used to segmentate the different speakers. I found it interesting so I tested it with music extracting the MFCCs, and after tunning some parameters, the results were surprising. It segmented the structure of songs almost perfectly, and it had good performance even when applied to different musical genres. Now, this music segmentation algorithm is part of Essentia, a content-based music analysis software library developed and mantained at the MTG.

After this, I investigated how BIC could be used to segmentate the audio into instrumental segments and vocal segments. I first tried to apply the BIC algorithm using the MFCCs to reach this purpose. I had to tune the parameters so that the algorithm returned smaller segments, but the results were not convincing. The segments didn't represent any homogeneity in terms of vocal/instrumental segmentation.

I asked the other researchers at the MTG what features could be used to reach my goal, and I was told about a new set of panning coefficients that had been recently developed and that they could help me get good results. So I fed the BIC algorithm with the panning coefficients and the results were promising.

I had all the pieces to build a singer identification system, so the rest of the project was just implementation and testing.

[1]MIREX 2005: http://www.music-ir.org/mirex2005

1.2 State of the art

As explained above, this project deals with two different tasks: text-independent speaker identification and singer identification. The second could seem to be a subset of the first one, but since singing voice differs from speech (especially in terms of spectral analysis) and taking into account the fact that speaker identification is usually performed over unnoisy data while singer identification has to deal with background music, they turn out to be two different tasks.

1.2.1 Text-Independent Speaker Identification

Speaker Recognition involves any task that deals with the identity of the speaker. It is commonly divided into two subtasks: Speaker Verification and Speaker Identification (SID) [18]. The former consists of determining whether the identify of the speaker of a given sample actually is the person that claims it is, whereas the latter consists of associating a given sample's speaker with one of a group of known voices.

In this project we only deal with SID, so in this section the state of the art of this field is reviewed. Current SID systems differ mainly on two points: the features extracted from the sound and the statistical method used to model the speaker. The state of the art is reviewed according to this classification.

Feature Extraction

The first automatic Text-Independent Speaker Identification systems were developed in the 70s. Sambur[22] took advantage of the new hardware available at that time to compute Linear Predictive Coefficients from large speech databases, and used them in SID tasks. His results were promising but the efficency of the LPC extraction needed to be improved.

Cepstrum-based coefficients became popular in the 80s and they were also investigated for its use in SID systems [11]. Some years later, Reynolds [19] analyzed several cepstral features implementations filtered with different filterbanks over clean speech data and after observing that Mel-Frequency Cepstral Coefficients(MFCCs) and Linear-Frequency Cepstral Coefficients (LFCCs) performed about the same in his experiments, he concluded that the filterbank spacing didn't affect the overall system performance.

However, other studies demonstrated that LPCCs could be seriously affected by noise [25], so currently the most widely used cepstral features are MFCCs, and they have become the *de facto* standard in SID systems.

Up to this point, all the features investigated were those that carried vocal tract information (e.g. cepstral features), whereas other speaker-dependent information regarding the speaker's source was left away. Plumpe et al.[15] studied how the variations in the movement of the vocal folds from one individual to another could help improve the performance of SID systems. Their method automatically parametrised the glottal flow derivative, that is, the differences in the time-varying volume velocity air flow through the glottis (the slitlike opening between the vocal folds), and the performance results showed about a 5% reduction in SID scores when these new features were added to traditional mel-cepstral measures.

Chandran et al[28] proposed the usage of the high-order spectrum (HOS) instead of the traditionally used power spectrum in SID tasks. HOS differ from power spectrum on the fact that it retains both the phase and amplitude information from the Fourier transform. Their results showed that HOS phase parameters perform at the same level as MFCCs on the same data, so they questioned the traditional notion that Fourier phase information is unimportant in speech processing.

High-level features have also been studied in terms of their application in SID systems. This was the purpose of a summer workshop called SuperSID, that took place at the Center for Language and Speech Processing (CLSP) of the John's Hopkins University in Baltimore, MD, with the presence of the best experts in the field. They put together and compared the new features that described high-levels of information such as prosody, phonemes and lexicon. Within prosodic features, pitch and energy distributions and dynamics and prosodic statistics (concerning pause durations and the fundamental frequency) were extracted; phone features concern information at phoneme level and taking advantage of time series of phonemes; lexical features are extracted in the form of word unigrams and bigrams which have been observed to be function of target/model sex and age difference. Using two kinds of classifiers (GMM and a perceptron), all these new high-level features were analysed and compared with the traditional MFCCs features. They showed that using all their new descriptors together with the MFCCs, a new record for accuracy on this task with astonishingly low errors rates was achieved.

Another interesting reseach topic has been channel compensation which was also investigated by Reynolds [19]. The idea behind it is that environmental conditions can affect SID systems performance, so this irrelevant information should be removed. The information we are dealing with is background noise and channel mismatch (speech recorded by different microphones in the training and testing phase). There are several methods for achieving channel compensation: Cepstral Mean Removal, RASTA processing and Quadratic Trend Removal. In spite of being the simplest method, Cepstral Mean Removal was proven to outperform the other techniques.

Speaker Model

Regarding the statistical technique used to model the speaker, traditional methods can be divided into two main categories: parametric methods and non-parametric methods [12]. Parametric methods are those whose structure is characterized by a collection of parameters that normally represent probability distributions. They have the advantage that it is easier to understand changes in the data through changes in the parameters. Non-parametric methods are those in which minimal assumptions regarding the probability distribution are made.

Within non-parametric methods, there are two main techniques: k-Nearest Neighbour (kNN) and Vector Quantization (VQ). k-Nearest Neighbour is one of the simplest machine learning algorithms. Higgins et al.[1] used a modified Nearest Neighbour method, in which input data was examined and discarded if the frame was already represented, in order to reduce memory and computational requirements. In the classification phase, a new sample is assigned to the most important class of its k-neighbourhood.

The idea behind VQ is very similar to k-NN: all the input data is reduced to K non-overlapping clusters, and wach cluster is represented by a code vector c_i. The resulting set of code vectors forms a codebook $c = c_1, c_2, \cdots, c_K$, which reduces considerably the amount of input data and is thus used as a model of the speaker. To classify a new speaker, the distance between its codebook and all the known-speakers codebooks is calculated using any of the available methods, and the one at less distance is assigned. There are two main points of investigation in VQ: the size of the codebook and the method used to measure the distance between two codebooks, in which Soong et al.[10] has been investigating.

Parametric methods are normally based on Gaussian distribution, since it can be easily parametrised with powerful results. The first SID systems used unimodal Gaussian Models [3], but some years later it was observed that more Guassian distributions should be used to obtain better performance, so Gaussian Mixture Models (GMM) were applied [18]. In GMM, each speaker is modeled by a mixture of Gaussian distributions, which can be understood as each one representing a phoneme. To obtain the parameters for each distribution, the Expectation-Maximization algorithm is used, which is an iterative method to obtain the parameters with maximum likelihood. Because its good performance and its great adaptation capacity, GMM has become the standard technique used in SID tasks.

It is worth mentioning that some improvements have been made to GMMs. One of the most significatives was also done by Reynolds [8], when he developed the Gaussian Mixture Model-Universal Background Model (GMM-UBM). The idea is not only to build a model for each speaker, but build a background model as well. This model represents the alternative speakers, that is, the speakers which don't appear in the known set.

Another great improvement was done by Chaudhari et al. [16], when they implemented a multi-grained Gaussian Mixture Model. The levels of granularity represent different levels of information: the coarser levels are used to model each phone class, while the finest levels model all the samples within a phone class. This new model has been proven very effective in large speaker populations, e.g. utterances with 10.000 speakers.

Apart from GMM, several other techniques have been used in SID tasks. Arificial Neural Networks (ANN), in most methods implemented as Multi-Layer Perceptron, have been used because of their weak hypothesis about statistical distributions and their discriminant training power. However, they also have the disadvantages that the optimal structure has to be tested by a trial-and-error procedure and that the training dataset has to be split into training data and cross-validation set. More recently, Support Vector Machines (SVMs) have been tested to improve GMMs, not only using SVMs alone but also merging them with GMMs. Several tests have been done, but the improvement in performance is not very important. Furthermore, it is hard to attribute this improvement just to the speaker model used because many other factors also should be taken into account (e.g. the training method, the number of mixtures. . .) [9].

1.2.2 Singer Identification

In this section we'll analyse both artist and singer identification. It can be seen that the latter is a subset of the former, but since artist identification is almost always based on the singer, we can compare both tasks as solving the same problem.

One of the first artist identification systems was developed by Whitman et al.[29], where they used ANN and SVN classifiers applied to spectral features from short music clips. It's important to note that they didn't make any difference between vocal and instrumental segments. The performance of the resulting system was not impressive, since it just achieved a 70% of accuracy in a 10-artist database, and 50% of accuracy in a 20-artist database.

Working on the same database as the previous paper, Berenzweig et al.[4] improved the overall performance up to a 65% of accuracy in the 20-artist database. The reason for this improvement is that during the pre-processing, a vocal vs. instrumental frame detector was run, and only the vocal segments were added to the training data. Therefore, it can be concluded that using the spectrum of vocal segments as training data improves the overall system performance.

Zhang [30] applied the standard text-independent speaker identification tecnhiques for a singer identification task. He manually collected vocal segments from several music recordings, extracted MFCCs and modeled each singer with a GMM. He got a 82% of accuracy, but considering that the databse was little and that the vocal detection was manually performed, this performance result is not very successful.

Li et al. [14] developed some new acoustic features for singer identification, that extracted information about the singer's vibrato. Applying several banks of filters (triangular, parabolic and cascaded), and transforming the resulting energies into the cepstral domain, they extracted the Octave Frequency Cepstral Coefficients (OFCC). Their experiments on a 12-singer database showed that OFCCs outperformed MFCCs and LPCCs.

All the techniques reviewed up to this point didn't take into account the fact that vocal segments used to train GMM (or any other model) have background music and it has a lot of influence over the voice when a spectral analysis is done. Tsai et al.[26] tried to remove background music from the audio by taking advantage of the similarities between instrumental-only regions and the accompaniment of the singing regions. Then they built a model for each singer and a model for the background non-vocal regions. Comparing their new technique with traditional MFCCs + GMMs showed an important improvement in performance.

Chapter 2

Tools

In this chapter we will introduce the tools we have used in this project. The main topics studied in this thesis are audio processing and machine learning, so here we will explain the libraries we have used to deal with both fields. We first describe the programming languages in which our programs have been written, then we will discuss MTG's audio processing library Essentia, and finally we will explain the machine learning libraries used.

2.1 Programming Languages

This project has added modules to the libraries Essentia and Gaia. Therefore, we didn't choose our implementation's languanges but we used the ones already used by these libraries. Both Essentia and Gaia are libraries with a Python interface but their modules are written in C++, because of the best performance that it offers. Although these languages weren't chosen by us, we found this selection very appropiate.

C++ is a general-purpose programming language that is used in the implementations of the modules of Gaia and Essentia. Since it is a compiled language (the source needs to be compiled), the performance is much higher than for interpreted languages (such as Python); that's the reason why the modules for both libraries are written in C++. It is an object oriented language, so it allows virtual functions, operator overloading, multiple inheritance, templates, exception handling, ...

Python is an interpreted language, so its performance is quite slow. However, its simplicity makes it an excellent choice for using it to provide an interface to a library, as it is the case in Essentia and Gaia. One of its main advantages are the packages, that are collections of utilities unified inside one module that help a lot the user.

I have also used Python and bash to write some scripts to process data: tranform Weka files into Torch files, tranform Torch files into human readable files, run any other script hundreds of times... An example of a very useful script can be found at the Appendix (B.1).

2.2 Audio processing

At MTG, the research group in which this project took place, there are several tools for audio processing. Our research team, which focuses on content-based Music Information Retrieval, has been developing for some years a library Essentia, which has been extremely useful in our work.

2.2.1 Essentia

Essentia is an audio processing library developed at MTG. It is organized in modules, which are written in C++. To interact with the library, however, it is much better to use its Python interface, so it's more clearly and easily accessible, and equally efficient.

It has many useful modules that can solve any problem regarding audio processing, from lower levels (e.g. filtering, spectral processing) to higher levels of information (e.g. key and tempo extraction, dissonance, danceability. . .).

We have not only used Essentia, but in this project we have added some modules to it. Specifically, we have implemented the BIC segmentation algorithm and the extractor of panning coefficients.

Essentia is commonly used together with the machine-learning library Gaia, which is also developed at MTG and has the same modular structure.

2.3 Machine Learning

In this project we used two tools for what is called machine learning: Torch and Weka. However, we used them for different tasks. Torch was used to build the models of each singer and Weka was used to find a decision tree to do the classification between vocal and instrumental segments.

2.3.1 Torch

Torch[1] is an open-source machine learning library developed at IDIAP Research Institute, at Switzerland. It is written in C++ and has many efficient implementations of machine learning tools that help us extract knowledge from large amount of data, so it is an excellent library to use when implementing machine-learning-related programs.

The most important tools implemented in Torch are:

- Gradient machines: Multi-layer perceptrons, Radial basis functions, mixture of experts, time-delay neural networks. . .
- Support Vector Machines, both for classification and regression
- Non-parametric models

[1]Torch website: http://www.torch.ch

- Parametric models such as KMeans, Gaussian mixture models, Hidden Markov models, Bayes classifier...

We have used Torch's GMM implementation, more specifically, we have used the class called DiagonalGMM. Its name comes from the fact that the covariance matrix used is a diagonal one. And as will be explained later on, GMM needs to be initialized and trained. So we have used two other Torch classes: KMeans and EMTrainer. KMeans is used to initialize the data, that is, apply a clustering algorithm so that each datapoint is matched with one distribution (the initial guess). EMTrainer is used to estimate the parameters of the Gaussian distributions that maximize the likelihood, given the data.

2.3.2 Weka

Weka[2] is a set of machine learning tools for data mining tasks. It is well known and used all around the world because it is very easy and intuitive interface and because it is open source software. It contains many implementations of techniques for data pre-processing, classification, regression, clustering and visualization.

It is written in Java, so it is not very efficient. Thus, it is more common to use it just as an analysis tool than to use it as a programming library to implement machin-learning-related software.

Weka provides a beautiful interface that allows the user to introduce the data, to select an appropiate method and to apply it. Results appear in a text window with plenty of relevant details, including information about the dataset and the parameters of the method applied. Furthermore, Weka runs by default a 10-cross validation algorithm that provides a thorough analysis of the performance of the method used.

In our project, we have used Weka to build the decision tree that classifies each music segment as vocal or instrumental. Specifically, we have used Weka's J48 algorithm implementation.

[2]Weka website: http://www.cs.waikato.ac.nz/ml/weka

Chapter 3

Concepts

The goal of this chapter is to provide the reader with the underlying theorical concepts applied in this project. These concepts are divided into two main groups: digital audio processing and statistics.

The concepts within digital audio processing are necessary to understand what kind of information is extracted from the audio signal and how this information can be helpful for each specific task in which it is applied. The concepts explained are spectral processing topics, from the essential knowledge to MFCCs, and the extraction of panning coefficients.

Within statistics, the concepts explained are Gaussian Mixture Models, which are used to generate a model for each speaker and determine the most similar model for an unknown voice, and Bayesian Information Criterion, which is used in the audio segmentation.

3.1 Digital Audio Processing

Digital Audio Processing is an essential task for the extraction of features from audio files. In this section we will briefly describe basic concepts on digital audio processing, paying full attention to the process of extraction of the features used in this project: MFCCs and panning coefficients. However, since both descriptors are based on spectral analysis of the signal, we will begin this section by giving an overview of this field, and then go deeply into the descriptors.

3.1.1 Spectral processing

Digital audio spectral processing takes advantage of the Fourier Transform to change the domain of the signal, from temporal to frequency domain. The reason for this change is that in this domain much more information can be extracted from the signal and this information is much more useful, specially in the field of content-based Music Information Retrieval.

The basic Fourier Transform can only be applied to continuous-time signals. However, we know that the signals we are going to deal with are not continuous but sampled (in a process that's beyond the scope of this project, so it is skipped in this report), therefore we need to use the Discrete Fourier Transform (DFT), which is defined as:

$$X(\omega) = \sum_{n=0}^{N-1} x(n) e^{-j\omega_k n}$$

where $k = 0, 1, \cdots, N-1$ is the number of bins, n is the number of samples, N is the total number of samples, ω is the discrete radian frequency and $x(n)$ is the amplitude of the signal at the n-th sample.

If we apply the DFT to all the signal at once, we will obtain all the frequencies of the signal but with no temporal information. This information is crucial to get a good analysis, because we will be able to know the frequencies of the signal for each short lapse of time. Therefore, we need to use an adapted version of the DFT, the Short-Time Fourier Transform (STFT).

The STFT segmentates the signal into small overlapped chunks and applies a window on every chunk. By applying a window we obtain a removal of (almost all) the artificial frequencies that appear into the signal when it is segmented, basically at the edges of each chunk. There are several types of windows that can be useful depending on the application of the analysis. Then, each windowed chunk is individually analysed by applying the STFT, defined as:

$$X_l(k) = \sum_{n=0}^{N-1} w(n) x(n + lH) e^{-j\omega_k n}$$

where $l = 0, 1, \cdots, L$ is the number of frame, L is the total number of frames, w is the window and H is the time advanced at each chunk (also known as hop-size).

Commonly, zero-padding is applied to the windowed signal in order to get more frequency resolution. In consists on adding a sequence of zeros at the end of the chunk, and it is very usefule while it has no drawbacks.

There are some parameters that have to be set depending on the application of the analysis. These parameters are the length of the window, the number of samples advanced at every new chunk (hop-size), the amount of zero-padding applied and the type of window. In our case we have used window of size 2048 samples, a hop-size of 1024 samples, no zero-padding and a window of type Blackman-Harris 62dB.

3.1.2 Mel-Frequency Cepstral Coefficients

MFCCs are one of the most useful descriptors in the field of speech processing. They were developed primarily for its usage in speech recognition tasks, because they extract a lot of speaker-independent information. However, MFCCs have been shown to also extract a lot of speaker-dependent information, and therefore they are the state-of-the-art descriptors for text-independent SID tasks. We used them when modeling each singer's voice.

MFCCs are obtained as follows: the signal is segmented and each chunk is windowed and analysed with the STFT. Then, each spectrum is multiplied with a bank of filters, because we are more interested in the envelope of the shape of the spectrum that in the details of the little fluctuations in spectrum. This bank of filters is a set of band-pass filters that are multiplied by the spectrum to obtain an average value of each subband. There are a lot of different filters (triangular, linear or logarithmic). For our purpose, we will use a Mel filter bank.

The Mel filter bank is based on the Mel scale, which is an auditory scale that simulates the frequency scale of the humar ear. In this filterbank, the bands are linearly spaced from 0 to 1000 Hz and then they are logarithmically scaled, as can be observed at figure 3.1.

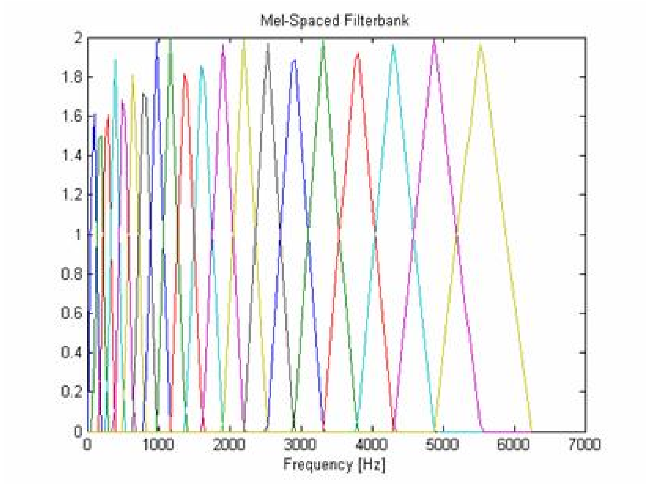


Figure 3.1: Representation of a Mel filter bank.

Once this filterbank is applied, the resulting coefficients are transformed by applying a Discrete Cosinus Transformation (DCT), and the new coefficients represent the cepstrum of the signal. This DCT is defined as:

$$c_n = \sum_{k=0}^{K-1} S_k \cos\left[n(k-\frac{1}{2})\frac{\pi}{K}\right]$$

where k is the number of spectral coefficient, K is the total number of spectral coefficients, n is the number of cepstral coefficient and L is the total number of cepstral coefficients to calculate, being necessarily $L \leq K$.

There are some parameters to be defined in order to obtain useful MFCCs. These parameters are the number of bands of the Mel filterbank, the number of cepstral coefficients to be extracted and the frequency range of the Mel filterbank. In our case, we have used a bank of 20 filters, taking just 13 of the resulting cepstral coefficients and the range of frequencies of the Mel filterbank was between 0 and 8000 Hz.

3.1.3 Panning coefficients

Panning coefficients are a set of descriptors that model the distribution of spatial information in a stereo mixed audio signal, distributed along the left-right axis[13]. They can have a lot of applications in the field of content-based Music Information Retrieval; we use them to detect changes in the instrumentation of a song.

The first step to obtain the panning coefficients is to calculate a ratio between the energy of the left and right channels. The result is a value within $[-45, +45]$ that represents the spatial localization of each frequency bin. It is defined as:

$$R(k) = \frac{2}{\pi} \arctan \left[\left| \frac{S_L[k]}{S_R[k]} \right| \right]$$

where k is each frequency bin.

Then a transformation is applied to make it more similar to human perception. According to Mills[17], humans have higher perception on the center of the azimuth, and a non-linear transformation is applied to simulate this behaviour. A graphical representation of this transformation can be observed at figure 3.2, and it is mathematically defined as:

$$R_W[k] = \begin{cases} -0.5 + 2.5x - x^2 & x \geq 0.5 \\ 1 - (-0.5 + 2.5(1-x) - (1-x)^2 & x < 0.5 \end{cases}$$

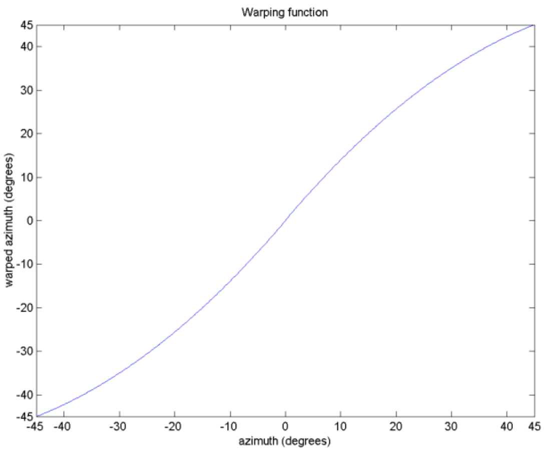


Figure 3.2: Transformation in the angle according to human perception.

Once this transformation is applied, an histogram of the spatial distribution of the signal is calculated, that is, the amount of energy at each spatial bin is calculated. More formally, it is defined as:

$$i_k = floor(MR_W[k])H_w(i_k) = \sum_{k=0}^{N} \Big| S_L[k] + S_R[k] \Big|$$

Then, this histogram is normalized:

$$H'(n) = \frac{H(n)}{\sum_{n=1}^{N} H(n)}$$

Once we have obtained the normalized histogram, it is transformed into the final panning coefficients which are shown to be a compact representation of the histogram. This last transformation is a cepstral analysis: the logarithm of the panning histogram is computed and then an IFFT (inverse Fast Fourier Transform) is applied to the result. Is has been empirically shown that a good tradeoff between resolution and size is brought by using 20 panning coefficients.

3.2 Statistics

In this section, the statistical techiques used in speaker modeling and audio segmentation will be briefly reviewed.

3.2.1 Gaussian Mixture Models

Before going deeply into the description of Gaussian Mixture Models, we'll review some os the underlying concepts so that GMM explanation can be perfectly understood.

One-dimensional Gaussian distribution

Gaussian Mixture Models(GMM) base their power on Gaussian (or Normal) distribution. This distribution is widely used in machine learning because of its great power and its ease of use: it can be parametrised with only two elements, the mean (representing the average) and the standard deviation (representing the variability).

The probability density function (PDF) of the one-dimensional Gaussian distribution is defined by:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp - \frac{(x-\mu)^2}{2\sigma^2}$$

where μ refers to the mean and σ to the standard deviation.

Figure 3.3 (extracted from the Wikipedia[1]) shows the graph of the PDF for Gaussian distributions with different parameter values.

[1]Normal distribution: http://en.wikipedia.org/wiki/Normal_distribution

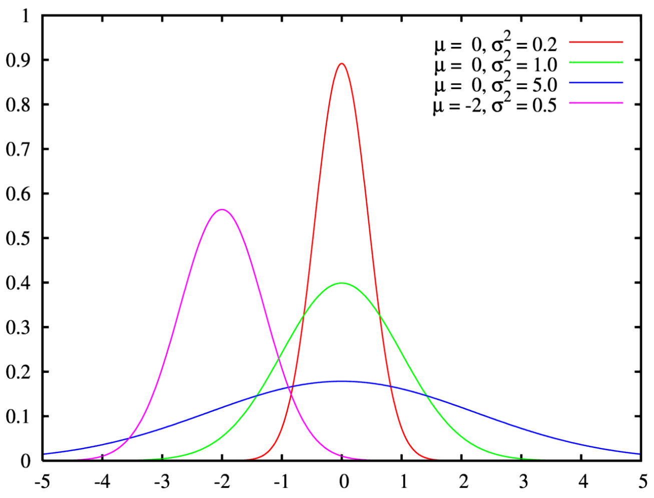


Figure 3.3: Graph of the probability density function Gaussian distributions with different parameter values.

Multivariate Gaussian distribution

When GMM are used in real-world applications, the data used to train the system is almost always multi-dimensional. Therefore, the distribution used by GMM is not the one-dimensioanl Gaussian but the multivariate Gaussian.

The PDF of the multivariate Gaussian distribution is very similar to the one-dimensional one. The main difference is that the mean value is now a vector and that the standard deviation is now the covariance matrix. Here we can see the PDF for an N-dimensional multivariate Gaussian distribution:

$$f_X(x_1, \cdots, x_N) = \frac{1}{(2\pi)^{N/2}|\Sigma|^{1/2}} \exp -\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)$$

where μ is the vector of means and Σ is the covariance matrix.

Now we have the key concepts needed to easily understand how GMM operates.

Gaussian Mixture Models

GMM is an unsupervised clustering method, and has the goal of recognising multiple categories in a collection of objects[21]. It is an unsipervised method because category labels are not given.

When clustering, GMM assumes that the data are generated from a mixture distribution P. P is formed by k components, having each an independent distribution. The overall distribution of P is given by:

$$p(\vec{x}) = \sum_{i=1}^{K} p(C=i)p(\vec{x}|C=i)$$

where $p(C = i)$ is the weight of the i-th component and $p(\vec{x}|C = i)$ is distribution of each component. In GMM, this distribution is a multivariate Gaussian one. Therefore, GMM is a weighted sum of k component densities with parameters Θ:

$$p(\vec{x}|\Theta) = \sum_{i=1}^{k} w_i \frac{1}{(2\pi)^{D/2}|\Sigma_i|^{1/2}} \exp -\frac{1}{2}(\vec{x} - \vec{\mu_i})^T \Sigma_i^{-1} (\vec{x} - \vec{\mu_i})$$

where w_i is the weight of the i-th component, $\vec{\mu_i}$ is the vector of means of the i-th component and Σ_i is the covariance matrix of the i-th component.

Now we can see one of the powers of GMM: the simplicity. The complete GMM can be parametrised only by:

$$\Theta = w_i, \vec{\mu_i}, \Sigma_i, i = 1, \cdots, k$$

Figure 3.4 (extracted from Russel and Norvig's book [21]) shows an example of a 2-dimensional GMM. In (a) we can see the raw datapoints, in (b) we can see the Gaussian components that generated the points and in (c) we can see the model reconstructed from the datapoints using a GMM.

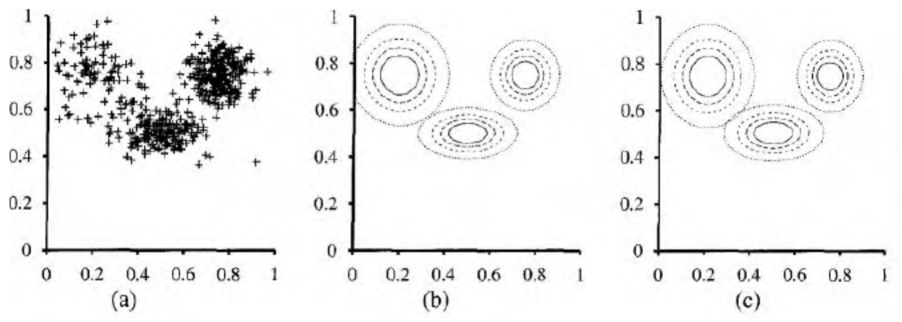


Figure 3.4: An example of a 2-dimensional GMM.

The task of training the GMM consists of estimating the parameters of the mixture density Θ that maximize the likelihood of the parameters given the data $\mathcal{X} = \vec{x_1}, \cdots, \vec{x_N}$:

$$\Theta^* = \max_{\Theta} L(\Theta|\mathcal{X})$$

To do the estimation of these parameters, GMM almost always take advantage of the Expectation-Maximization algorithm.

The Expectation-Maximization algorithm

The EM algorithm is an iterative optimization method that is used by GMM to estimate the unknown parameters Θ of a mixture distribution that maximize the likelihood given certain data $\mathcal{X}$. The most interesting characteristic of EM is that it uses some hidden variables which are not observable in the available data.

It can seem surprising that the usage of these hidden variables can lead EM algorithm to a good result. However, if we observe figure 3.5, we can understand why it happens. It shows two networks used to model a simple diagnostic, one using hidden variables and the other without hidden variables. The former requires 78 parameters to define the network whereas the latter requires 708! Therefore, the usage of hidden variables can lead to a simplified problem.

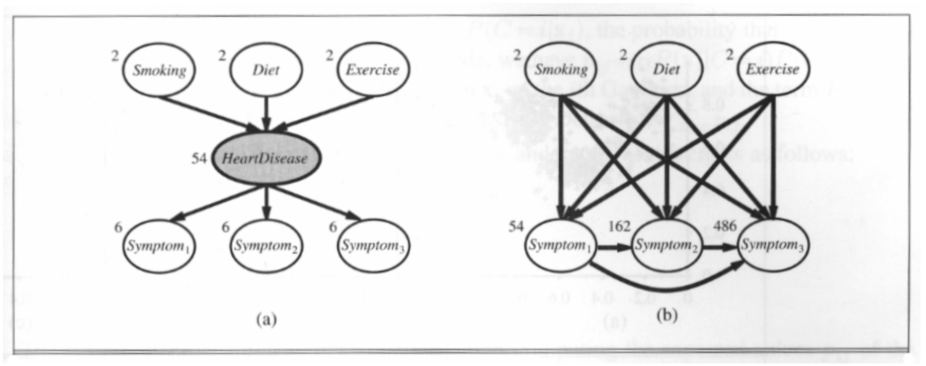


Figure 3.5: Comparison of two simple diagnostic networks. In (a), a hidden variable is used, whereas in (b) no hidden variables are used.

When applying the EM algorithm to the problem of estimating the parameters of a mixture of Gaussian distributions, the hidden variables are the component that generated each data point. Then, what EM will maximize is the log-likelihood of the parameters Θ given the observable data $\mathcal{X}$ and the hidden variables $\mathcal{Y}$:

$$\Theta^* = \max_{\Theta} L(\Theta|\mathcal{X}, \mathcal{Y})$$

As explained above, the EM algorithm is an iterative method. At each step, two tasks are done: expectation and maximization.

At the **expectation step**, the probability that datapoint $\mathbf{x}_j$ was generated by component i is computed:

$$p_{ij} = P(C = i|\mathbf{x}_j)$$

If we apply Bayes' rule, we have the following transformation:

$$p_{ij} = P(C = i|\mathbf{x}_j) = \alpha P(\mathbf{x}_j|C = i)P(C = i)$$

where the term $P(\mathbf{x}_j|C = i)$ is the probability at the point $\mathbf{x}_j$ of the i-th Gaussian distribution and the term $P(C = i)$ is the weight of the i-th Gaussian distribution.

We can observe that defining the probability p_{ij} (the expectation step) can be seen as computing the expected values of the hidden variables Y_{ij}, where Y_{ij} is 1 if the datapoint $\mathbf{x}_j$ was generated by the i-th Gaussian component and 0 otherwise.

We can also define the overall probability of each Gaussian p_i, which will be required at the next step:

$$p_i = \sum_j p_{ij}$$

At the **maximization step**, the new mean, covariance matrices and weights for each component are computed. For the sake of simplicity, the mathematical transformation details will be skipped in this report. Anyway, they are available at the excellent EM tutorial written by Bilmes[5]. These new values are updated as follows:

$$\begin{aligned}
\mu_i &\leftarrow \sum_j \frac{p_{ij}\mathbf{x}_j}{p_i} \\
\Sigma_i &\leftarrow \frac{p_{ij}(\mathbf{x}_j - \mu_i)(\mathbf{x}_j - \mu_i)^T}{p_i} \\
w_i &\leftarrow p_i
\end{aligned}$$

The values that are computed at this step are the estimations of the parameters that maximise the log-likelihood of the data given the expected values of the hidden variables.

Because EM algorithm is an iterative method, its good performance and convergence depend on the initialization. In our case, the initialization refers to the values of the hidden values, that is, assign the Gaussian component that generated each datapoint. Therefore, we use a simple yet powerful method to do this task: the k-nearest neighbour method.

k-nearest neighbour is also an iterative method. At the first step, each datapoint is assigned randomly to one of the categories. Then, the iterative process goes on as follows:

1. For each category, calculate the mean.
2. For each datapoint, calculate the nearest category mean, and reassign its class to this category.
3. Stop when no difference is made between two steps.

Using this simple method we can have a quite accurate initialization that allows EM algorithm reach a good clustering.

Testing the models

Up to this point we have explained how a model of a speaker is build. Now we will explain how each model is tested given new unseen data.

This new data is a set of points in a multi-dimensional space. Each GMM determines a probability for each data point to be generated by the current model, by adding the probabilities of each data point for all Gaussians. Taking the averaged sum of the probabilities of all the points of the new unseen data, we know the probability that this model generated the data. The model that returns a highest probability is taken as the generator of the analysed dataset.

3.2.2 Bayesian Information Criterion

The Bayesian Information Criterion (BIC) is a model identification method and was developed by Schwarz[23] (that's why it's also referred to as Schwarz information criterion). Model identification is a task that consists of selecting a statistical model from a set of potential models, given some data.

More formally, given the dataset $\mathcal{X} = x_i \in R^d$, $i = 1, 2, \cdots, N$ which we wish to model and the model candidates $\mathcal{M} = M_i$, $i = 1, 2, \cdots, K$, the BIC is defined by:

$$BIC(M) = logL(\mathcal{X}, M) - \lambda \frac{1}{2} \sharp(M) \log(N)$$

where $logL(\mathcal{X}, M)$ is the maximum log-likelihood of the model M given the data $\mathcal{X}$, $\sharp(M)$ is the dimension of the model M and λ is the penalization weight.

Then we assume that the data set $\mathcal{X}$ is generated by an independent multivariate Gaussian process[7]:

$$x_i \sim N(\mu_i, \Sigma_i)$$

where μ_i is the mean vector and Σ_i is the covariance matrix of the distribution.

In our audio segmentation case, we apply BIC to a simplified problem: we take a (small) window of data and assume that there's at most one change point. Therefore, we two hypothesis, either all the windowed data is generated by the same Gaussian distribution (H_0) or the windowed data is generated by two different Gaussian distributions, having a change at time i:

$$\begin{aligned} H_0 &: x_1, x_2, \cdots, x_N \sim N(\mu, \Sigma) \\ H_1 &: x_1, \cdots, x_i \sim N(\mu_1, \Sigma_1); x_{i+1}, \cdots, x_N \sim N(\mu_2, \Sigma_2); \end{aligned}$$

To compare these two hypothesis, we use the maximum likelihood ratio test $R(i)$:

$$R(i) = N \log |\Sigma| - N_1 \log |\Sigma_1| - N_2 \log |\Sigma_2|$$

where Σ, Σ_1 and Σ_2 are the covariance matrices from all the data, from $x_1, \cdots, x_i$ and from $x_{i+1}, \cdots, x_N$, respectively and N, N_1 and N_2 are the lengths of the full data, and the first and second subwindows, respectively.

The difference between the values of BIC for these two models is defined by:

$$BIC(i) = R(i) - \lambda P$$

where P is the penalty, defined by:

$$P = \frac{1}{2}(d + \frac{1}{2}d(d+1)) \log N$$

being λ the penalty weight and d the number of dimensions of the data.

According to the BIC rule, if the value $BIC(i) > 0$ then it means that the window is better modeled two different Gaussian distributions and therefore the window should be segmented at time i. On the other hand, if $BIC(i) <= 0$ then the window should not be segmentated. The optimal segmentation point can be found by:

$$\hat{i} = arg \max_i BIC(i)$$

Chapter 4

System analysis

In this chapter we will describe all the investigations we have made during the last year. We will begin giving an overview of the whole system, and then each block will be described individually.

4.1 Global vision of the system

The goal of ths project is to build a system that identifies the performer of a song, focusing the analysis on the identity of the singer. A more detailed explanation of the goal can be found at chapter 1.1, and there's a review of the existing literature in the field at chapter1.2.2.

Our project is based on the investigations made by Berenzweig[4], who showed that using only vocal segments of a song to build the model of a singer, the performance of the identification task improves, rather than using the whole song. So we thought a new way to segmentate an audio file into vocal or instrumental segments, and we investigated how this segmentation improved the performance.

The system can be divided into 2 main tasks: detecting voiced segments within a song and modeling the voice of the speaker from the segments detected as having voice. Each task is described with full detail in the following sections. A block diagram of the whole process of training the system can be observed at figure 4.1.

However, this was not the goal of the project at the beginning of the project. Our first idea was to develop new descriptors to improve the task of text-independent speaker identification (SID). In the following sections, a cronologically ordered description of the investigations carried out during the project is made. Section 4.2 explains the tests done trying to discover new descriptors for text-independent SID; section 4.3 describes the method used to segmentate the audio signals into voiced and instrumental segments and section 4.4 shows how the model of a singer is performed and how the artist of an unkown song is identified.

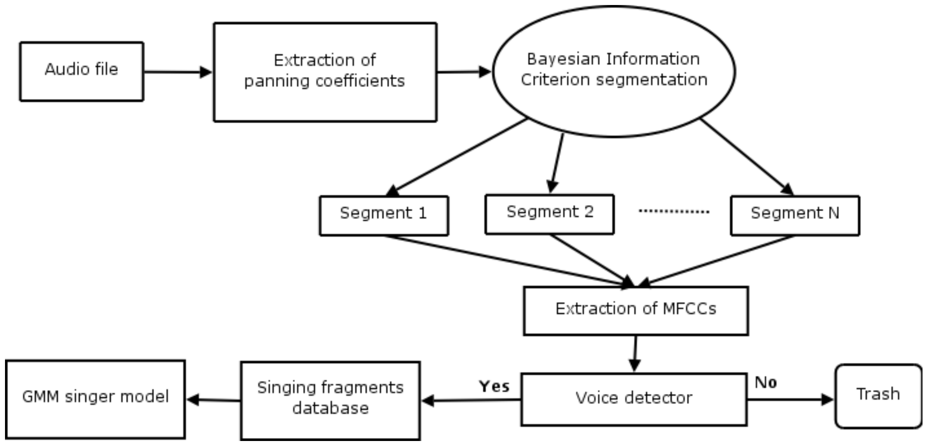


Figure 4.1: Block diagram of the overall training system.

4.2 Speaker identification

The first goal of this project was to improve the performance of text-independent SID systems. We started by reading the existing literature in the field and doing a review of the most important papers that are mentioned in section 1.2.1.

As explained in chapter 2, the performance of SID depends on two main tasks: the features extracted from the speech signal and the statistical technique used to build the model of each speaker. Given that this project took place at the Music Technology Group of the Universitat Pompeu Fabra, we focused our investigations on the digital signal part of the field.

Description of the speech database

The first step for a correct investigation of new descriptors for SID tasks was to get a reliable starting point. From the literature reviewed we already knew that the state-of-the-art speaker identification method was to extract Mel-Frequency Cepstral Coefficients (MFCCs) and use them to build a Gaussian Mixture Model (GMM) for each spaker, so we just implemented a system with this structure to have a suitable measure of comparison for the new descriptors to be investigated. We built this system using the GPL-lincensed Torch library (see 2.3.1), which allows the user to easily implement machine learning techniques, among them GMMs.

To train and test the system we used the TIMIT Database[1]. This database was developed by Texas Instruments together with the Massachusets of Technology to provide speech data for the acquisition of acoustic-phonetic knowledge and for the development and evaluation of automatic speech recognition systems. It contais speech from 630 speakers that represent 8 mkajor dialect divisions of American English, and for each speaker, there are 10 phonetically-

[1]TIMIT Database: http://www.mpi.nl/world/tg/corpora/timit/timit.html

rich sentences recorded.

All speakers in the database are adults, native speakers of American English and were judged by a professional speech pathologist to have no clinical speech pathologies. All recordings are made under clean conditions. The database is formed by 438 male speakers and 192 female speakers.

The fact that it is a database addressed to automatic speech recognition systems is not a handicap for its application in text-independent speaker identification systems, because a database containing small amounts of speech from a relatively diverse speaker population is very useful for our task. Having speaker from a range of dialects can even make our system more robust.

Furthermore, it is very important that there's something in this database that helps the performance of SID tasks: the lack of intersession interval. It means that, for each speaker, their sentences were all recorded during the same day. Therefore, there are no differences due to speaker's healthy conditions (e.g. having a cold), slight pitch changes. . . These phenomena are the cause of the bad performance of some SID systems.

For each one of the 630 speakers there are 10 files, where each file is the audio of a recorded sentence read by the speaker. There 2 two sentences that were read by all 630 speakers (*sa* sentences), while the other 8 sentences were different for every speaker. These 8 sentences are grouped into phonetically-compact (*sx* sentences) and phonetically-diverse (*si* sentences).

Since it is an automatic speech recognition database, the training data are the utterances from 420 speakers, while the remaining 210 speaker are used as testing data. However, because the nature of our project, these two subgroups changed in our case. We trained the system the same way as Reynolds[20] did using also the TIMIT database. For each speaker, the training data was formed by the two common sentences (*sa* sentences), three *si* sentences and three *sx* sentences. The remaining two *sx* sentences were used as testing data. In summary, we used 24 seconds of speech for the training, and 3 seconds of speech for each test, having a total amount of 1260 tests (one for each *sx* sentence).

As we said before, we wanted to have a good starting point so as to have a good way to measure the performance of the new descriptors to be investigated. Using the state-of-the-art tested over the clean-data TIMIT database, we got a 97.86% of correct identification (using 32 Gaussian for each model), which is pretty close to the 99.5% claimed by Reynolds[20]. It is obvious that there's no need to improve a performance of a 99.5%, but we have to take into account the fact that this excellent performance was achieved testing over clean data. So we added noise to the TIMIT databse to test the performance of our new descriptors under more realistic conditions.

We wrote a little Python script (that can be found at the Appendix B.1) that added some noise to the data, using the open-source software SoX[2]. The noise was generated with a SNR (signal-to-noise ratio) of 12 dB. Using this noisy

[2]SoX: http://sox.sourceforge.net/

database and MFCCs to build the models, we got a maximum performance of 93.58%, which is a pretty good performance statistic but still gives chance to improvements. Let's see if our new descriptors will.

The new descriptors

One of the most successful research groups within MTG is the one working on singing voice analysis and syntesis. During the last 10 years of investigations, they have developed a large number of tools and applications related to singing voice, and thus, their experience could be very helpful for our project. After some conversations, we concluded that it would be interesting for both groups to investigate the performance in SID tasks of some descriptors developed by them that modeled the timbre of the singing voice.

We were given a C++-written library for the Windows platform, and the first step was to adapt it to be available to run it on a Linux platform. After some days of work, the first task was successfully achieved.

The library contains several descriptors, some of them commonly extracted but most of them totally new, specially when applied to SID tasks. The common descriptors are energy, pitch, MFCCs and their deltas. The uncommon descriptors were related to some higher-levels of information, such as the existence of vibrato and other characteristics of the singing voice. Here we have a detailed explanation of the most important ones:

MFCCs: Already explained at section 3.1.1.

Energy: In a continuous signal, the energy is defined as:

$$E_s = \int_{-\infty}^{\infty} |x(t)|^2 dt$$

In our case, since our speech signals are sampled, the signal is discrete and the energy is defined as:

$$E_s = \sum_{n=0}^{N-1} |x(t)|^2$$

where N is the total number of samples of the signal or the part of the signal we are analyzing (e.g. a windowed frame).

Related to the energy of the signal, the most used feature is the averaged energy, whis is defined as:

$$E_s = \frac{1}{N} \sum_{n=0}^{N-1} |x(t)|^2$$

Pitch: The fundamental frequency of the signal is returned in Hz or in cents. The algorithm for this calculation was developed at MTG and is based on the Two-Wai Mismatch procedure. It is described in an article by Cano[6].

Vibrato Probability: It returs a value between 0 and 1 corresponding to the probability of that singing voice frame to contain a vibrato. It is based on computing the zero-crossing points of the delta pitch and compare the envelope of this feature with the envelope of the delta pitch of a chirp signal, which is constant. Subtracting the first envelope to the second, if the resulting value is high it means it has vibrato, while if the value is low it means that the signal has no vibrato.

Energy without Vibrato: It returns the same value of energy as explained before, but if the system detects vibrato in the signal, the amplitude modification caused by it will be removed from the measure.

Pitch without Vibrato: It returns the same value of pitch as explained before, but if the system detects vibrato in the signal, the frequency modification caused by it will be removed from the measure.

First to Second Harmonic Amplitude: It returns the ratio between the spectral energy of the second and the first formant. As pointed out by Sundberg[24], the strength of the fundamental frequency is related to the kind of phonation (leaky, flowy or neutral phonation when this ratio is low, and pressed phonation when this ratio is high).

Spectral Tilt: Difference in dBs between the first harmonic peak amplitude and the maximum harmonic peak amplitude in a frequency range from 3kHz and 8kHz.

Analysis of the performance

The first step for the analysis of the performance of these new parameters was to see their performance as they were, so we plugged all of them into our system and, as expected, the results were terribly bad. This was not very surprising because there were a lot of descriptors and it is obvious that some of them would not be useful for our task.

Then we began the deep analysis to the descriptors with the assistance of the machine-learning software WEKA (see 2.3.2). We first stated that some of the descriptors were very correlated to some others. Some correlations were obvious, for example Pitch and Pitch without Vibrato, their deltas, etc. Some others were not so obvious at first sight: Pitch and Spectral Tilt and Harmonic Phase Stability and Excitation Slope.

This correlations can be discovered by means of WEKA using two of its numerous features: visualizing the data or doing a Principal Component Analysis. With the former, WEKA shows a graph correlating data from two different descriptors, and is the result is almost a line, then they are highly correlated. If the result is something more unpredictable, or a regular cloud of points, then these two descriptors are uncorrelated. We can see an example of each kind at figure 4.2.

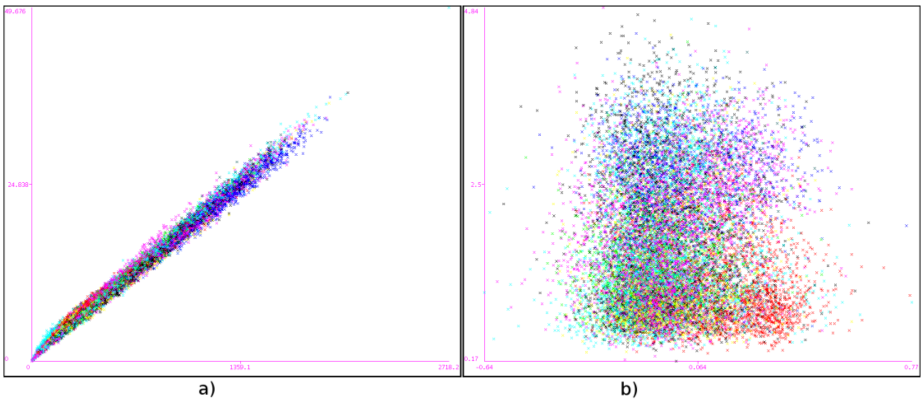


Figure 4.2: Graph of the representation of (a) two very correlated descriptors and (b) two very incorrelated descriptors.

The other way to detect correlated descriptors, running a Principal Component Analysis, returns a big correlation matrix. Each element (i, j) of this symmetric matrix shows the correlation between the descriptor i and the descriptor j. The range for this similarity is $[-1, 1]$, where values close to 1 mean that the two descriptors are very correlated, values close to 0 mean that the two descriptors are very incorrelated and values close to -1 mean that the two descriptors are inversely correlated.

Once the correlated descriptors were removed, we could investigate the performance of the remaining ones. To do so, we plugged them into an ARFF file (the text file type used by WEKA) and analysed them. We first tried to append each frame analysis to the file, but there were two problems:

1. **The file was immense:** Having 630 speakers, 30 seconds of speech for each speaker, and an analyzing hop-size of 10ms, leads to 1,890.000 analysed frames (each frame is represented in a line in the ARFF file), about 400 bytes per line, so the file size would be 756MB... That is too much for Weka.

2. **It is hard to identify at frame level:** if Weka is given each frame separately, it tries to classify the identity of the speaker of that single frame. Therefore, the advantage of knowing that all frames within a file belong to the same speaker is missed.

So we had to find a solution, and it turned out to be a very simple one: instead of appending each frame to the training data, just appending the mean values of each descriptor for each entire file would work much better. And it worked. The amount of data was drastically reduced and Weka could process it with no problem.

Then we could try to find the most important descriptors of our set. With Weka, we had two main possibilities: Principal Component Analysis and Decision Trees.

PCA is very commonly used in machine learning to reduce the number of dimensions of a dataset. It is based on the eigenvalues and the eigenvectors of the covariance matrix of the normalized dataset. Therefore, it makes a change of the coordinate system so the first coordinate is the one with greatest variance, and so on folloing the order of the eigenvalues. It is a very powerful tool because of its simplicity, but unfortunately it had drawback in our case: the result of this transformation is a matrix with lower dimensions but this information cannot be used to discover which are the less important descriptors, that is, those that should to be excluded.

This information is given by Decision Trees. This technique make a statistical analysis of the dataset and returns a tree that at each leave has a classification element. The descriptors found at the highest levels of the tree are the most selective ones, so these are the most important ones for our case. An example of a decision tree returned by the J48 algorithm of Weka can be found at the appendix, section A.1.

After a large number of tests, the features that were most successful were:

- MFCCs (1-12)
- Fundamental Frequency
- First to Second Harmonic Amplitude
- Delta Energy without Vibrato
- Energy without Vibrato

The first thing to point out is that the descriptors at the top of the tree are all the MFCCs. So, they are the most important descriptors in our set. It is a first proof that these descriptors work worse than MFCCs. But there was still a test left: try to see if these descriptors added to MFCCs could improve the performance or not.

So we tested all the descriptors together both under clean and noisy conditions and, as can be observed at figures 4.3 (clean conditions) and 4.4 (noisy conditions), the performance of the whole set of descriptors was worse than the performance of using just MFCCs.

Under clean conditions, the best performance is achieved using just MFCCs as descriptors and with models of 32 Gaussian distributions, with a 97.86% of correct identifications. Adding the our descriptors to MFCCs the best performance decreases to a 90.08% of correct identifications, using 64 Gaussian distributions.

Under noisy conditions, the tests show a similar behaviour as in clean data, so in both cases using only MFCCs seems to perform much better that adding new descriptors to MFCCs. Again, the best performance is achieved using just MFCCs, but now with 64 Gaussian distributions (although the difference of performance between using 32 and 64 Gaussians is very small). And adding the new descriptors to MFCCs, the best performance decreases a lot, it goes down to 75.65% using 128 Gaussian distributions to model each speaker.

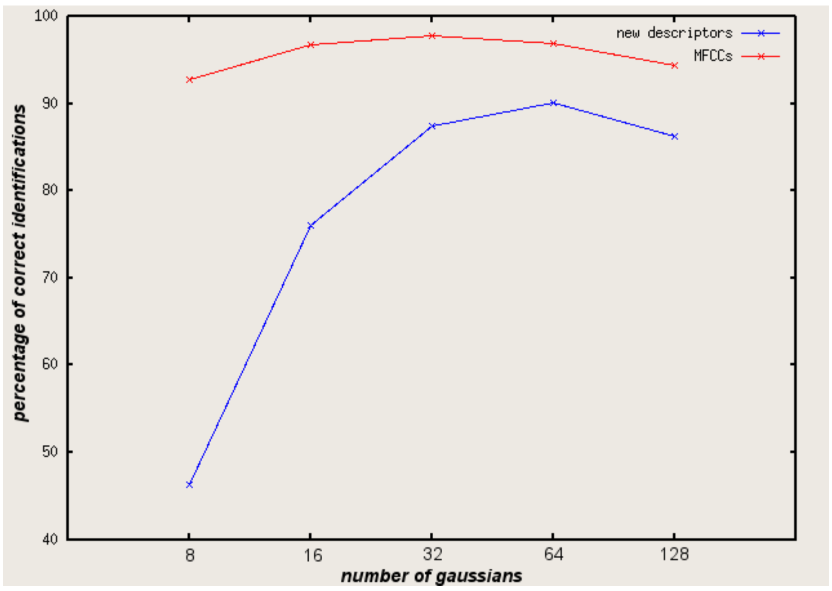


Figure 4.3: Comparison of the performance of both descriptor sets under clean conditions.

The peformance of these new descriptors is not only worse than using just MFCCs but it also needs more complex models to represent each speaker. Therefore, the computational cost of the overall process will be higher.

So, as a conclusion of the first part of the investigation, the best descriptors for the task of text-independent SID are MFCCs using 32 Gaussian distributions to build each model. That's the method we will use to build each model of speaker.

4.3 Segmentation block

4.3.1 BIC implementation

In this block, the input signal is segmented into some homogeneous segments, in terms of the input coefficients given to the algorithm. The method that allows us to do this segmentation is called Bayesian Information Criterion (BIC), and is already explained at section 3.2.2.

This algorithm is used to select a statistical model from a set of potential models, given some data. We use a simplified form of the problem by assuming

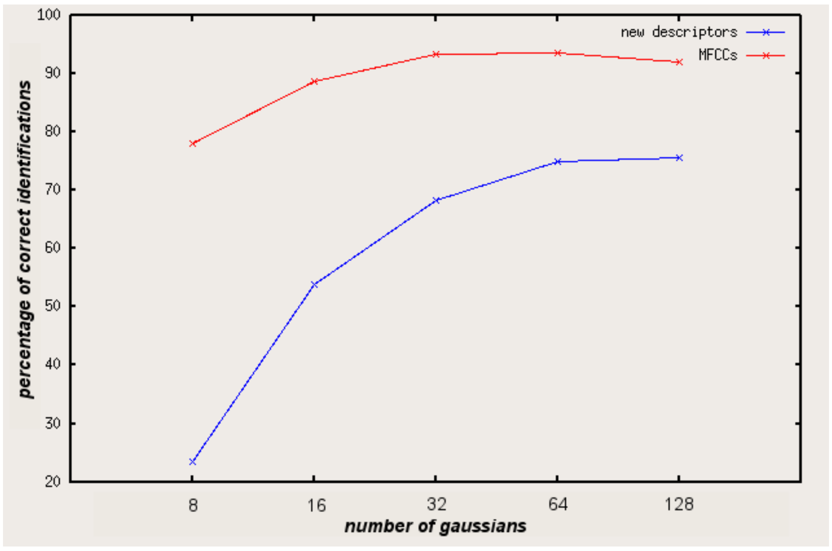


Figure 4.4: Comparison of the performance of both descriptor sets under noisy conditions.

that for each set of data there were only two possible models: either the dataset was generated by two different probability distributions (so we have to find the point of the change of distribution) or all the dataset was generated by the same distribution.

The execution of the BIC algorithm given a data chunk is:

1. Set $n = inc\_size$
2. Divide data chunk into two parts: from 1 to n and from n+1 to $window\_size$
3. Calculate the BIC value
4. If BIC returns true, return n
5. If BIC returns false, set $n+ = inc\_size$ and return to point 2

We first tested the performance of this segmentation by feeding the algorithm with MFCCs. For each frame we extracted 13 MFCCs and we put them together having as a result a matrix of size 13 x $\frac{signal\_size(sec)}{hop\_size(sec)}$. This is the data BIC uses to return a segmentation, by applying the following 3-pass algorithm:

First Pass

The first pass is thought to find coarse segmentation points. It is achieved by using relatively large windows and increment sizes.

1. Set $n = 1$
2. Take a chunk of size $window\_size_1$ from n

3. Apply BIC algorithm to the chunk with inc_size_1

4. If BIC returns true, set a segmentation at the returning point, set $n+= inc\_size_1$ and return to point 2

5. If BIC returns false, set $n+= window\_size_1$ and return to point 2

Second pass

In this, the segmentation points found at the first pass are finely tuned:

1. For each segmentation point sp found at the first pass do

2. Take a data chunk of size $window\_size_2$ around sp: $(sp - window\_size2/2, sp + window\_size2/2)$

3. Apply BIC algorithm to the chunk with inc_size_2

4. If BIC returns true, replace sp with the returning point, and exit

5. If BIC returns false, remove sp

Third pass

The third pass checks that the segmentation point for each pair of adjacent segments is correct, that is, that there is no pair of adjacent segments sharing the same probability distribution.

1. For each pair of adjacent segments, determined by $sp1$, $sp2$ and $sp3$ do

2. Take a data chunk from $sp1$ to $sp3$

3. Calculate the BIC value for this chunk

4. If BIC returns false, remove $sp2$

5. If BIC returns true, exit

In this third pass the minimum length for each segment is also checked.

The parameters that must be set before running our BIC algorithm are:

- window size 1 (first pass)
- increment size 1 (first pass)
- window size 2 (second pass)
- increment size 2 (second pass)
- minimum segment size
- penalty weight (if this value is low then more segments are generated, and vice versa)

4.3.2 BIC tests

BIC algorithm is a popular technique for audio segmentation, commonly used to detect changes of speaker, channel or environment [7]. However, it was never applied to popuplar music segmentation, and we tested it.

We fed BIC algorithm with MFCCs, and using appropiate values for the parameters we got excellent and quite surprising results: BIC segmentation detected almost perfectly the structure of any song from several different music genres (e.g. pop-rock, jazz, r'n'b, reggae. . .). It was particularly surprising that such performance could be reached by a technique that required no training!

The parameter values that gave us the best results were:

$$\begin{aligned} windowsize1 &= 1000 \\ incrementsize1 &= 300 \\ windowsize2 &= 600 \\ incrementsize2 &= 50 \\ minimumsegmentsize &= 10 seconds \\ penaltyweight &= 5 \end{aligned}$$

Figure 4.5 shows an example of a successful BIC segmentation, applied to Coldplay's song *Yellow*.

This segmentation algorithm is now part of the Essentia audio-processing library (see section 2.2.1) and is one of the most useful segmentation algorithms within the library. Fed with these parameter values, this algorithm has many applications. For example, it can be used to find specific parts of a song (e.g. a solo) so each part can be analyzed according to its category, and it can also be used in a music-video player, as an intelligent skip button. . .

However, all these applications have nothing to do with the goal of this project. But we thought how we could take advantage of the BIC segmentation algorithm in the field of singer identification, and an idea came out: BIC algorithm could be applied over a song to find the vocal segments, that is, to find all the intersections between vocal and instrumental parts in a song.

We first tested the performance of BIC algorithm fed with MFCCs and tuning the parameters of the algorithm so that it returned more and shorter segments (by rising the penalty weight and reducing window and increment sizes). The results, however, didn't fulfill our expectations: segmentation points seemed were almost randomly set, regardless of vocal or instrumental sections. So we had to find a new set of music descriptors that could carry more information to differentiate vocal and instrumental parts.

And that's when we came across panning coefficients. These coefficients represent the distribution of spatial energy in a stereo mixed audio signal, and

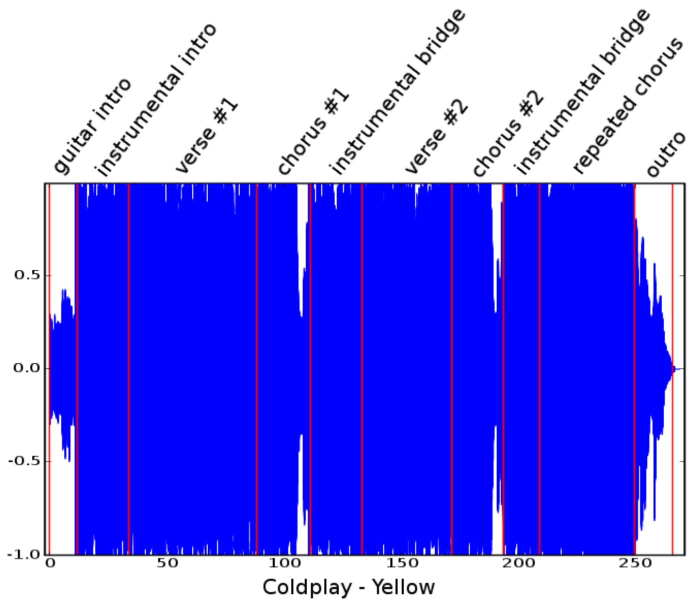


Figure 4.5: Plot of the signal of Coldplay's song Yellow, with the segments found using BIC algorithm represented and the song structure annotated.

were developed also at MTG.

In almost all recorded songs, instruments are not all placed at the center of the signal. Usually instruments tend to be more placed more at one side of the signal. However, lead voice is commonly placed in the center, so there is difference between vocal and instrumental parts. That's the basic idea behind the usage of BIC segmentation algorithm fed with panning coefficients.

The tests we did with panning coefficients were very successful: the algorithm detected very well almost all boundaries. However, there's something very important to point out: this analysis is not based on musical aspects of the signal but it is based on aspects of the way songs are mixed. Therefore, although results are many times successful, there will be some cases in which results will be terrible.

Up to this point, we had a segmentation algorithm that detected the intersections between vocal and instrumental parts. However, we still needed to establish a criterion to classify segments into vocal or instrumental. As explained above, panning coefficients don't represent aspects of the song itself but aspects of how it was mixed. Thus, the classification decision (vocal vs. instrumental) should not be based on this descriptors. And we thought that MFCCs are good sound descriptors, so we built a decision tree based on MFCCs that would classify each segment into vocal or instrumental.

We built a database to generate this decision tree. We restricted our domain

ARTIST	SONG	ALBUM
Coldplay	Don't Panic	Parachutes
Coldplay	Sparks	Parachutes
Lenny Kravitz	Are You Gonna Go My Way	Greatest Hits
Lenny Kravitz	Let Love Rule	Greatest Hits
Manu Chao	Desaparaecido	Clandestino
Manu Chao	El Viento	Clandestino
U2	Pride (In the Name of Love)	The Best Of
U2	With or Without You	The Best Of
Elton John	Your Song	The Very Best Of
Elton John	Crocodile Rock	The Very Best Of
Ricky Martin	She Bangs	Sound Loaded
Ricky Martin	Nobody Wants to Be Lonely	Sound Loaded
Christina Aguilera	So Emotional	Christina Aguilera
Christina Aguilera	Come On Over (All I Want Is You)	Christina Aguilera
Sheryl Crow	Run, Baby, Run	Tuesday Night Music Club
Sheryl Crow	All I Wanna Do	Tuesday Night Music Club
Melissa Etheridge	Chrome Plated Heart	Melissa Etheridge
Melissa Etheridge	Watching You	Melissa Etheridge
Vanessa Amorosi	Absolutely Everybody	The Power
Vanessa Amorosi	The Power	The Power
Texas	I Don't Want a Lover	The Greatest Hits
Texas	So in Love With You	The Greatest Hits
Mariah Carey	Can't Let Go	Emotions
Mariah Carey	Till the End of Time	Emotions

Table 4.1: Relation of songs used to train the vocal vs. instrumental classification tree

to pop-rock music. This database contained information about 24 songs from 12 different artists (2 songs per artist). These songs are thought to equally represent a wide range of singer voices and instrumentations within pop-rock. Hence, we have songs performed by 6 male and 6 female singers, and for each group 6 songs are more intense (the song has a high rhythm and drums are played loud) while the remaining 6 are smoother (soft or no drums, less instrumentation...). Table 4.1 shows the songs used.

For all these songs, we ran the BIC segmentation algorithm fed with panning coefficients to detect homogeneous segments in terms of instrumentation. Then, each segment was manually tagged (vocal or instrumental). For each segment, we extracted 13 MFCCs and we trained the system using the following statistical measures for each coefficient:

- MFCC mean
- MFCC variance
- MFCC delta mean

- MFCC delta variance
- MFCC delta-delta mean
- MFCC delta-delta variance

Thus, each segment was represented by 78 elements. We plugged all these data into WEKA and built with it a decision tree with the algorithm J48. We tuned the parameters of the algorithm so that a lot of prunning was applied to the tree and therefore we could obtain a rather small tree and have no overfitting problems. We got this tree:

```
var_mfcc6 <= 88.917252
|   var_mfcc13 <= 20.224783: instrumental (283.0/31.0)
|   var_mfcc13 > 20.224783
|   |   mfcc1 <= -437.93869
|   |   |   var_mfcc8 <= 47.16811: instrumental (99.0/8.0)
|   |   |   var_mfcc8 > 47.16811
|   |   |   |   dvar2_mfcc11 <= 59.083096: voice (68.0/31.0)
|   |   |   |   dvar2_mfcc11 > 59.083096: instrumental (40.0/7.0)
|   |   mfcc1 > -437.93869
|   |   |   var_mfcc11 <= 22.944561: instrumental (35.0/6.0)
|   |   |   var_mfcc11 > 22.944561
|   |   |   |   mfcc5 <= 3.581012: voice (113.0/23.0)
|   |   |   |   mfcc5 > 3.581012: instrumental (50.0/21.0)
var_mfcc6 > 88.917252
|   mfcc1 <= -482.016876: instrumental (53.0/21.0)
|   mfcc1 > -482.016876: voice (400.0/64.0)
```

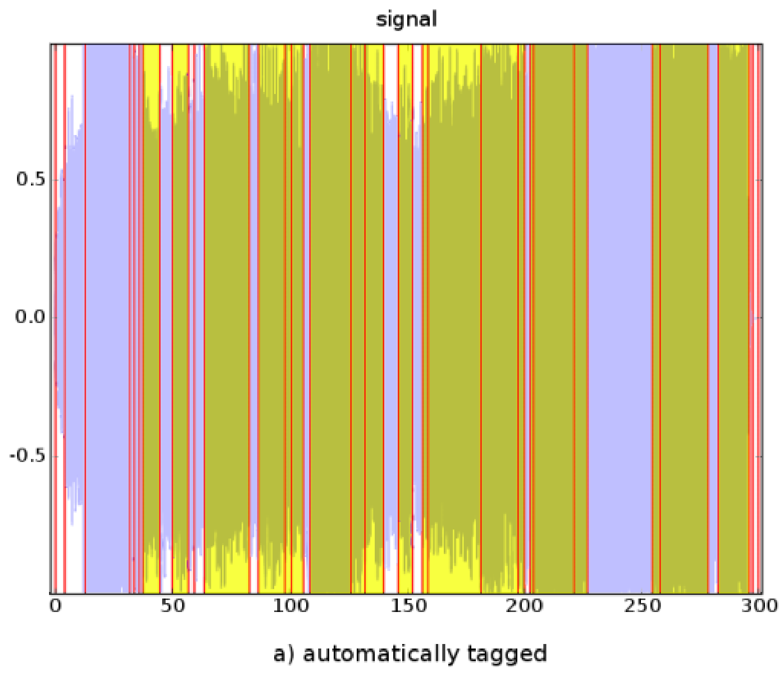
Weka runs some performance tests using a 10-fold cross-validation algorithm. The percentage of correctly classified for this tree was 77.5%, which is more than a correct performance taking into account the small size of the tree.

Figures 4.6 and 4.7 compare the vocal/instrumental tagging returned by our method and a manual tagging. In both examples we can see that both tags match in most cases, so our method works pretty well. The first song is *Shiver*, performed by Coldplay, and the second song is *Been Here Once Before*, performed by Eagle Eye Cherry. It is important to note that to train the vocal vs. instrumental decision tree, other songs performed by Coldplay were used, but none of Eagle Eye Cherry, so our method can generalize well even for new artists.

4.4 Singer modeling block

To model each singer, we use the MFCCs of only the segments tagged as vocal to train a 32-Gaussian GMM. This model is trained as explained in section 3.2.1, and the method is exactly the same as used in Speaker Identification tasks.

We have used a database randomly chosen from a large music library. To model each artist, we use MFCCs of the vocal segments of two of his or her songs to train a GMM. Then, we use the rest of songs of each artist as testing data.



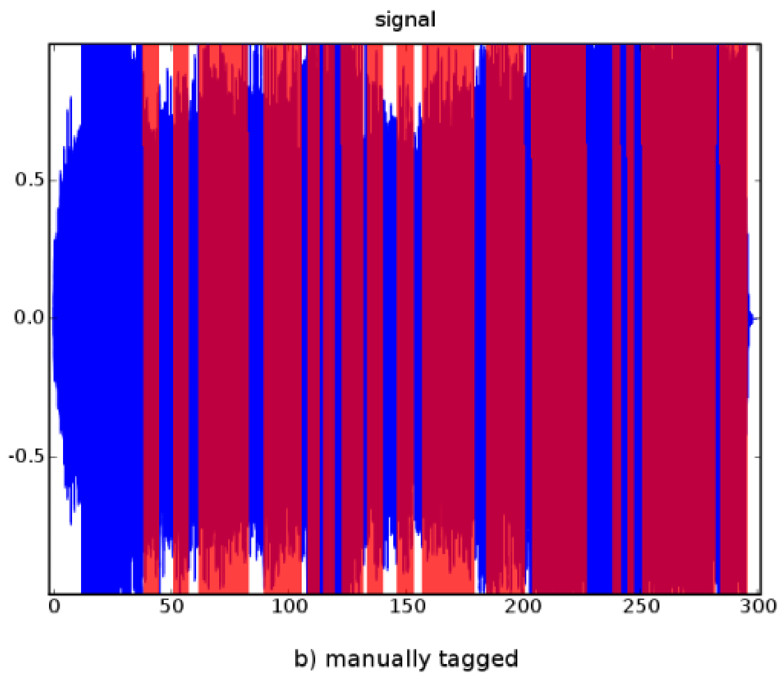


Figure 4.6: Comparison of manual and automatic vocal/instrumental tagging for the song *Shiver*, by Coldplay.

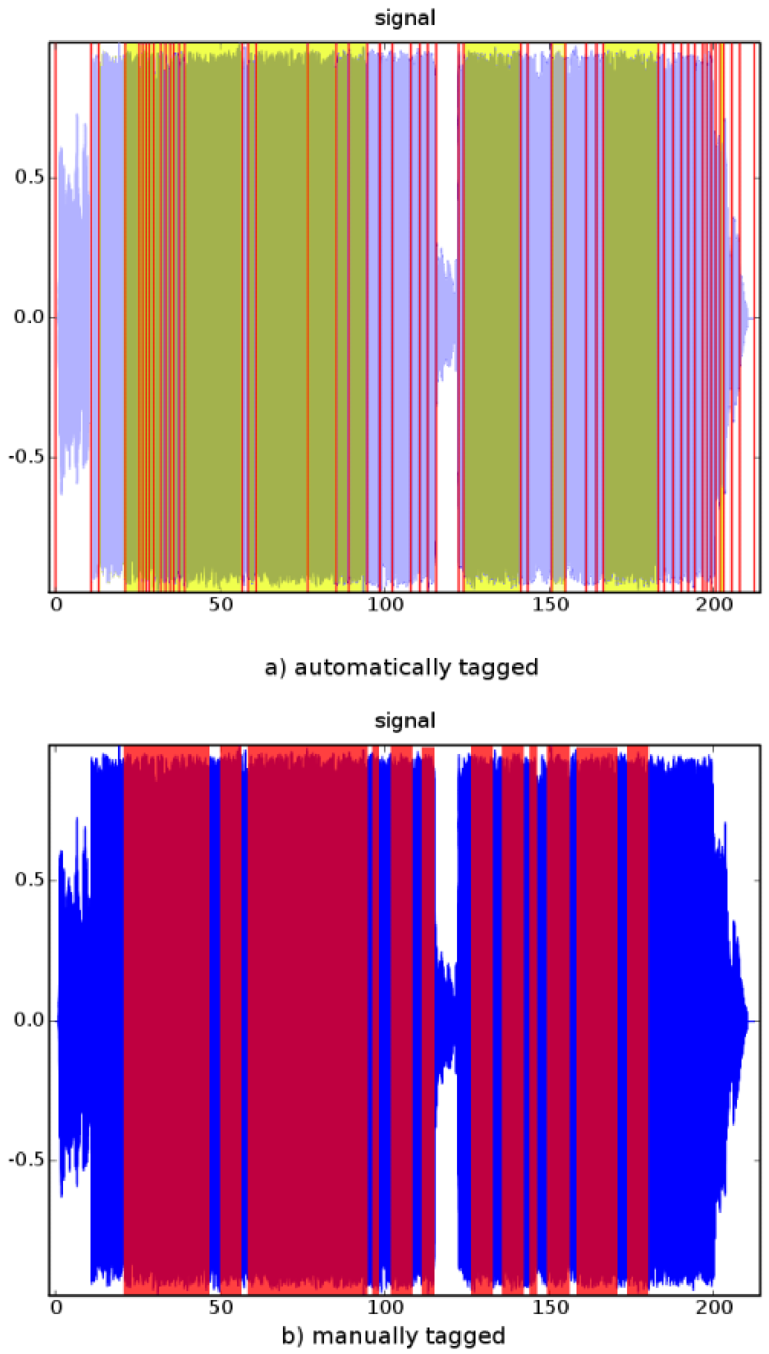


Figure 4.7: Comparison of manual and automatic vocal/instrumental tagging for the song *Been Here Once Before*, by Eagle Eye Cherry.

Chapter 5

Results

The results of the singer identification tests will be described in this section, as well as some comments about them.

5.1 Music database

We tested our singer identification system with a music database that contained the songs from 10 albums by 10 different artists. For each artist, 2 randomly chosen songs were used as training data, and the remaining songs of the album were used as testing data.

The artists in this database were almost randomly chosen: it contains music from 5 male artists and 5 female artists. However, the selection of a particular artist was made completely randon from a much larger database.

Table 5.1 details the artists and albums that contained the database.

In order to test the performance of our vocal vs. instrumental segmentation method, for each artist we trained two different models:

ARTIST	**ALBUM**
3 Doors Down	The Better Life
The Bloodhound Gang	Hoorary For Boobies
Britney Spears	Ooops! . . . I Did It Again
Celine Dion	Let's Talk About Love
Cher	The Greatest Hits
Enrique Iglesias	Enrique
Eric Clapton	Chronicles
Jessica Folker	Jessica
Phil Collins	Dance Into The Light
Shania Twain	Come On Over

Table 5.1: Relation of algums used to test the performance of our singer ID system.

ARTIST	**# correct**	**# songs**	**percentage**
3 Doors Down	5	8	63%
Bloodhound Gang	6	15	40%
Britney Spears	6	11	55%
Celine Dion	13	14	93%
Cher	2	10	20%
Enrique Iglesias	9	14	64%
Eric Clapton	3	13	23%
Jessica Folker	8	9	89%
Phil Collins	7	10	70%
Shania Twain	10	13	77%
TOTAL	**69**	**117**	**58.97%**

Table 5.2: Performance summary of the singer ID system using only vocal-detected frames.

ARTIST	**# correct**	**# songs**	**percentage**
3 Doors Down	8	8	100%
Bloodhound Gang	8	15	53%
Britney Spears	7	11	64%
Celine Dion	13	14	93%
Cher	3	10	30%
Enrique Iglesias	8	14	57%
Eric Clapton	3	13	23%
Jessica Folker	9	9	100%
Phil Collins	7	10	70%
Shania Twain	11	13	85%
TOTAL	**77**	**117**	**65.81%**

Table 5.3: Performance summary of the singer ID system using all frames.

- Using only vocal segments
- Using the whole song

Comparing the performance of the system with this two different data sources, we will be able to determine the importance of vocal segments over instrumental segments in the field of singer ID.

5.2 Experiments

Tables 5.2 and 5.3 show the results of the performance of our singer ID system using only the vocal-detected frames or using all the frames, respectively.

We can observe that the performance of the system when using only the segments detected as vocal is worse than using the whole song. It is qute surprising, especially taking into account the fact that Berenzweig proved that the usage of only vocal segments improved performance. However, we have some reasons to explain this fact:

Some of the artists of the database are hard to model: For example, Eric Clapton is mainly a guitar player, and in his songs guitar solos are often more important that the voice itself. Our vocal vs. instrumental detector is still not trained to distinguish between very harmonic sounds (such as voice and guitar), and therefore it is possible that the model of what should be the voice of Eric Clapton, is corrupted with guitar information.

Background vocals: In some of the songs of the database, there are a lot of background vocals, sung by a different singer than the leader of the band. Therefore, this other voices also corrupt the model. Furthermore, often the passages of the song with background vocals are longer than rest of the song.

Chapter 6

Conclusions

6.1 Conclusions

The main contribution of this project is the development of a new technique for locating vocal segments within a polyphonic song. This technique is based on two steps:

1. detect the boundaries between vocal and instrumental segments
2. classify each segment as vocal or instrumental

The segmentation is made with the BIC algorithm, using panning coefficients. The classification is performed by a decision tree that takes MFCCs as its parameters.

The results we got when testing the singer ID system were not the expected ones, but it is important to note that the most important block of this project is the segmentation more than the singer identification itself. We tested the segmentation in this application because it was an affordable one, but as will be explained in the next section, our new segmentation system can have many other interesting applications.

We can also say that the particular system we have built is thought to perform quite well only with pop-rock music. To apply it with other music genres, the segmentation part should not be modified, but the decision tree to classify each segment as vocal or instrumental should be rebuilt. Moreover, for certain music styles (such as jazz) it would be interesting to build a model for each soloist instrument. Therefore, jazz segmentation could be very easily made.

In conclusion, although the final results have not been completely successful, we can extract a lot of good feelings from the project. First of all, the experience of working in a research-group, how everyone helped each other in any field, and how reseach was getting new directions everyday has been an excellent experience for me.

6.2 Future work

As we have just said, the most important part of this project is the segmentation block. It can be applied in multiple tasks, and fed with different descriptors it can easily segmentate following any criteria. Therefore, many applications can arise.

A very interesting application would be to transform the system into a global-song vocal vs. instrumental descriptor. As the system is right now, it classifies too many non-vocal segments as vocal, and that's because only two models (vocal and instrumental) are trained. If there would be a model for many instruments that can be predominant in a song, this new descriptors could be easy to implement and very useful.

Furthermore, the application of panning coefficients can also lead to new applications. Knowing where the spatial energy lies, it would not be very hard to isolate the most prominent part of the spectrum, and build the models only with this subset of information. Then, models would really represent the voice of a singer and background music would be no problem at all.

Finally, another practical improvement would be to establish a similarity measure between models, that is, voices. Therefore, it could be possible to find, for example, songs performed by a singer with a similar voice to my preferred one.

Appendix A

Weka files

In this section of the appendix we will reproduce the output returned by Weka when several algorithms were run.

A.1 Decision tree returned by Weka's J48 algorithm

```
J48 pruned tree
------------------

timbreMel12 <= 13.959
|   timbreMel12 <= -5.5315
|   |   timbreMel10 <= 1.2556: 18 (584.0/124.0)
|   |   timbreMel10 > 1.2556: 4 (788.0/335.0)
|   timbreMel12 > -5.5315
|   |   timbreMel11 <= -43.53
|   |   |   timbreMel7 <= -76.028
|   |   |   |   timbreMel10 <= 7.0693: 29 (201.0/85.0)
|   |   |   |   timbreMel10 > 7.0693: 20 (268.0/149.0)
|   |   |   timbreMel7 > -76.028
|   |   |   |   timbreMel2 <= 67.098
|   |   |   |   |   timbreDeltaEnergyDBWithoutVibrato <= 0.020307
|   |   |   |   |   |   timbreMel6 <= -0.17689
|   |   |   |   |   |   |   timbreMel10 <= 10.412: 31 (259.0/128.0)
|   |   |   |   |   |   |   timbreMel10 > 10.412: 16 (248.0/170.0)
|   |   |   |   |   |   timbreMel6 > -0.17689
|   |   |   |   |   |   |   timbreMel5 <= -68.058
|   |   |   |   |   |   |   |   timbreMel10 <= 7.1573: 22 (249.0/29.0)
|   |   |   |   |   |   |   |   timbreMel10 > 7.1573: 20 (209.0/131.0)
|   |   |   |   |   |   |   timbreMel5 > -68.058
|   |   |   |   |   |   |   |   timbreNasality <= 0.4682: 25 (350.0/192.0)
|   |   |   |   |   |   |   |   timbreNasality > 0.4682: 7 (200.0/124.0)
|   |   |   |   |   timbreDeltaEnergyDBWithoutVibrato > 0.020307
|   |   |   |   |   |   timbreMel4 <= -12.795: 34 (502.0/145.0)
|   |   |   |   |   |   timbreMel4 > -12.795: 12 (218.0/178.0)
|   |   |   |   timbreMel2 > 67.098
|   |   |   |   |   timbreEnergyDBWithoutVibrato <= 9.2383: 33 (201.0/69.0)
|   |   |   |   |   timbreEnergyDBWithoutVibrato > 9.2383
|   |   |   |   |   |   timbreMel6 <= -11.128: 31 (336.0/195.0)
|   |   |   |   |   |   timbreMel6 > -11.128
|   |   |   |   |   |   |   timbreMel8 <= 12.915
|   |   |   |   |   |   |   |   timbreMel4 <= -7.9039: 21 (340.0/274.0)
|   |   |   |   |   |   |   |   timbreMel4 > -7.9039: 15 (280.0/107.0)
|   |   |   |   |   |   |   timbreMel8 > 12.915: 21 (342.0/86.0)
|   |   timbreMel11 > -43.53
|   |   |   timbreMel6 <= -3.8787
|   |   |   |   timbreMel7 <= -65.272: 29 (322.0/85.0)
|   |   |   |   timbreMel7 > -65.272: 12 (300.0/217.0)
|   |   |   timbreMel6 > -3.8787
|   |   |   |   timbreMel7 <= -53.37
|   |   |   |   |   timbreMel4 <= -1.1836
|   |   |   |   |   |   timbreMel11 <= -37.928: 7 (336.0/184.0)
|   |   |   |   |   |   timbreMel11 > -37.928: 23 (264.0/49.0)
|   |   |   |   |   timbreMel4 > -1.1836: 15 (202.0/147.0)
|   |   |   |   timbreMel7 > -53.37
|   |   |   |   |   timbreFormantToHarmArea <= 2.0616: 30 (200.0/114.0)
|   |   |   |   |   timbreFormantToHarmArea > 2.0616
|   |   |   |   |   |   timbreMel10 <= 12.676: 24 (336.0/38.0)
|   |   |   |   |   |   timbreMel10 > 12.676: 23 (206.0/59.0)
timbreMel12 > 13.959
|   timbreMel4 <= 14.705
|   |   timbreMel9 <= -42.149
|   |   |   timbreMel7 <= -66.717
|   |   |   |   timbreMel8 <= 31.32: 33 (359.0/275.0)
|   |   |   |   timbreMel8 > 31.32: 9 (447.0/27.0)
|   |   |   timbreMel7 > -66.717
|   |   |   |   timbrePitchHz <= 85.361
|   |   |   |   |   timbreMel10 <= 26
|   |   |   |   |   |   timbreMel8 <= 19.62
|   |   |   |   |   |   |   timbreMel5 <= -55.116: 33 (207.0/137.0)
|   |   |   |   |   |   |   timbreMel5 > -55.116: 6 (478.0/118.0)
|   |   |   |   |   |   timbreMel8 > 19.62
|   |   |   |   |   |   |   timbreMel6 <= -1.1824: 28 (308.0/114.0)
|   |   |   |   |   |   |   timbreMel6 > -1.1824: 14 (208.0/124.0)
|   |   |   |   |   timbreMel10 > 26
|   |   |   |   |   |   timbreMel12 <= 24.408: 14 (310.0/148.0)
|   |   |   |   |   |   timbreMel12 > 24.408
|   |   |   |   |   |   |   timbreMel9 <= -49.218
|   |   |   |   |   |   |   |   timbreMel11 <= -54.327: 32 (247.0/86.0)
|   |   |   |   |   |   |   |   timbreMel11 > -54.327: 5 (297.0/63.0)
|   |   |   |   |   |   |   timbreMel9 > -49.218: 3 (283.0/144.0)
|   |   |   |   timbrePitchHz > 85.361: 16 (224.0/119.0)
|   |   timbreMel9 > -42.149
|   |   |   timbreMel8 <= 7.6001
|   |   |   |   timbreMel6 <= -8.7593: 13 (302.0/107.0)
|   |   |   |   timbreMel6 > -8.7593: 19 (421.0/188.0)
|   |   |   timbreMel8 > 7.6001
|   |   |   |   timbreMel5 <= -48.744
|   |   |   |   |   timbreMel12 <= 23.879: 17 (610.0/307.0)
|   |   |   |   |   timbreMel12 > 23.879: 3 (358.0/178.0)
|   |   |   |   timbreMel5 > -48.744
|   |   |   |   |   timbreFirstToSecondHarmAmp <= 256980: 30 (509.0/315.0)
|   |   |   |   |   timbreFirstToSecondHarmAmp > 256980: 10 (404.0/58.0)
|   timbreMel4 > 14.705
|   |   timbreMel8 <= 8.6537
|   |   |   timbreMel7 <= -46.289: 13 (204.0/122.0)
|   |   |   timbreMel7 > -46.289: 19 (229.0/33.0)
|   |   timbreMel8 > 8.6537
|   |   |   timbreMel9 <= -33.882
|   |   |   |   timbreFirstToSecondHarmAmp <= 150690
|   |   |   |   |   timbreMel10 <= 21.318: 8 (340.0/72.0)
|   |   |   |   |   timbreMel10 > 21.318
|   |   |   |   |   |   timbreMel7 <= -63.327: 26 (200.0/34.0)
|   |   |   |   |   |   timbreMel7 > -63.327: 32 (244.0/145.0)
|   |   |   |   timbreFirstToSecondHarmAmp > 150690
|   |   |   |   |   timbreMel6 <= -3.4503: 12 (251.0/137.0)
|   |   |   |   |   timbreMel6 > -3.4503
|   |   |   |   |   |   timbreMel7 <= -59.582: 2 (213.0/151.0)
|   |   |   |   |   |   timbreMel7 > -59.582
|   |   |   |   |   |   |   timbreMel5 <= -59.061: 2 (437.0/103.0)
|   |   |   |   |   |   |   timbreMel5 > -59.061: 1 (484.0/108.0)
|   |   |   timbreMel9 > -33.882
|   |   |   |   timbreFirstToSecondHarmAmp <= 257140: 26 (200.0/124.0)
|   |   |   |   timbreFirstToSecondHarmAmp > 257140: 27 (485.0/62.0)


Number of Leaves  :  53

Size of the tree :  105
```

Appendix B

Scripts

B.1 addNoise.sh

```
#!/usr/bin/python
# Script per extraure els descriptors dels fitxers wav del directori amb essentia

import os
import sys
import commands
from string import split
from string import rfind

if len(sys.argv)!=2:
print "Usage: ./addNoise.sh SNR"
sys.exit(0)

fileString = commands.getoutput('find . -iname "*_44100.wav"')
files = split(fileString,'\n')

denom = pow(10,float(float(sys.argv[1])/20))

for file in files:
print file
stat = split(split(commands.getoutput("sox "+file+" -e stat"),'\n')[8])
ampSignal = float(stat[len(stat)-1])
snr = ampSignal / denom
noisedFile = "noise_"+sys.argv[1]+"_"+file[rfind(file,'/')+1:]
os.system("sox "+file+" /tmp/"+noisedFile+" synth whitenoise create vol "+str(snr))
os.system("soxmix "+file+" /tmp/"+noisedFile+" "+file[:rfind(file,'/')+1]+noisedFile)
```

Bibliography

[1] L. G. Bahler A. L. Higgins and J. E. Porter. Voice identification using nearest-neighbor distance measure. *IEEE Trans. Acoust., Speech, Signal Processing*, 2:375–378, April 1993.

[2] X. Amatriain. *An Object-Oriented Metamodel for Digital Signal Processing with a focus on Audio and Music*. PhD thesis, Universitat Pompeu Fabra, Barcelona, 2005.

[3] B. S. Atal. Automatic recognition of speakers from their voices. *Proceedings of the IEEE*, 64(4):460–475, 1976.

[4] A. Berenzweig, D. Ellis, and S. Lawrence. Using voice segments to improve artist classification of music. In *AES 22nd International Conference*, Espoo, Finland, June 15–17 2002.

[5] J. Bilmes. A gentle tutorial on the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models, 1997.

[6] P. Cano. Fundamental frequency estimation in the sms analysis, 1998.

[7] S. Chen and P. Gopalakrishnan. Speaker, environment and channel change detection and clustering via the bayesian information criterion, 1998.

[8] T. F. Quatieri D. A. Reynolds and R. B. Dunn. Speaker verification using adapted gaussian mixture models. *Digital Signal Processing*, 10(1-3):19–41, 2000.

[9] F. Bimbot; J. F. Bonastre; C. Fredouille; G. Gravier; M. I. Chagnolleau; S. Meignier; T. Merlin; J. Ortega Garcia; P. Delacretaz and D. A. Reynolds. A tutorial on text-independent speaker verification. *EURASIP Journal on Applied Signal Processing*, 4:430–451, 2004.

[10] L. Rabiner F. Soong A. Rosenberg and B. Juang. A vector quantization approach to speaker recognition. *IEEE Trans. Acoust., Speech, Signal Processing*, 10:387–390, April 1985.

[11] S. Furui. Cepstral analysis technique for automatic speaker verification. *IEEE Trans. Acoust., Speech, Signal Processing*, 29(2):254–272, April 1981.

[12] H.Gish and M. Schmidt. Text-independent speaker identification. *IEEE Signal Processing Mag.*, 11(4):18–32, October 1994.

[13] J. Janer. Panorama features for multichannel audio mixtures classification. Patent Pending, filed on Sept. 5, 2007.

[14] H. Li and T. Lay. Vibrato-motivated acoustic features for singer identification. *IEEE Trans. Acoust., Speech, Signal Processing*, 5, May 14–19 2006.

[15] T. F. Quatrieri M. D. Plumpe and D. A. Reynolds. Modeling of the glottal flow derivative waveform with application to speaker identification. *IEEE Trans. Speech Audio Processing*, 7(5):569–586, September 1999.

[16] U. V. Chaudhari; J. Navrratil; G. N. Ramaswamy; S. H. Maes. Very large population text-independent speaker identification usingtransformation enhanced multi-grained models. *IEEE Trans. Acoust., Speech, Signal Processing*, 1:461–464, 2001.

[17] W. Mills. On the minimum audible angle. *The Journal of the Acoustical Society of America*, 30(4):237–246, 1958.

[18] D. Reynolds and R. Rose. Robust text-independent speaker identification using gaussian mixture speaker models. *IEEE Trans. Speech Audio Processing*, 3:72–83, January 1995.

[19] D. A. Reynolds. Experimental evaluation of features for robust speaker identification. *IEEE Trans. Acoust., Speech, Signal Processing*, 2(4):639–643, October 1994.

[20] D. A. Reynolds. Large population speaker identification using clean and telephonespeech. *IEEE Signal Processing Lett.*, 2(3):46–48, 1995.

[21] S. Russell and P.Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, 2003.

[22] M. R. Sambur. Selection of acoustic features for speaker identification. *IEEE Trans. Acoust., Speech, Signal Processing*, 23(2):176–182, April 1975.

[23] G. Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978.

[24] J. Sundberg. *The Science of the Singing Voice*. Northern Illinois University Press, 1987.

[25] J. Tierney. A study of lpc analysis of speech in additive noise. *IEEE Trans. Acoust., Speech, Signal Processing*, 28(4):389–397, August 1980.

[26] W. H. Tsai and H. M. Wang. Automatic singer recognition of popular music recordings via estimation and modeling of solo vocal signals. *IEEE Transactions on Audio, Speech & Language Processing*, 14(1):330–341, 2006.

[27] G. Tzanetakis. Song-specific bootstrapping of singing voice structure. In *Proc. IEEE Int. Conference on Multimedia and Expo*, Taipei, Taiwan, 2004.

[28] D. Ning V. Chandran and S. Sridharan. Speaker identification using higher order spectral phase features and their effectiveness vis-a-vis mel-cepstral features. In *Biometric Authentication*, pages 614–622, 2004.

[29] B. Whitman, G. Flake, and S. Lawrence. Artist detection in music with minnowmatch. In *Proceedings of the 2001 IEEE Workshop on Neural Networks for Signal Processing*, pages 559–568. Falmouth, Massachusetts, September 10–12 2001.

[30] T. Zhang. Automatic singer identification. In *ICME'03*, July 6–9 2003.