Real-Time Speaker Identification

Evgeny Karpov

15.01.2003

University of Joensuu
Department of Computer Science
Master's Thesis

Table of Contents

Abstract

Nowadays it is obvious that speakers can be identified from their voices. In this work we look into the details of speaker identification from the real-time system point of view. Firstly, we review the well-known techniques used in speaker identification. We look into the details of every step in identification process and explain the ideas, which leaded to these techniques. We start from the basic definitions used in DSP, then we move to the feature extraction step and review two types of features, namely MFCC and LPCC, and finally we review two speaker modeling techniques, VQ and GMM. Secondly, we analyze described techniques from the time complexity point of view and propose several approaches to their optimization. Finally, we propose a novel approach to the feature matching step in the speaker identification and analyze it theoretically and practically. The main objective of this approach is an iterative pruning of speaker models, which are far away from the unknown speech sample, during the identification process. In order to analyze this method in practice we made appropriate software and using real data we ran several tests. Empirical results show that proposed approach greatly improves identification speed in feature matching step.

Acknowledgments

I would like to thank Tomi Kinnunen for his guidance and support during my work on this thesis.

Chapter 1

Introduction

In this chapter we make a brief introduction into the area of speaker identification and shortly describe the main parts of this thesis.

1.1 Basic definitions

The human speech conveys different types of information. The primary type is the meaning or words, which speaker tries to pass to the listener. But the other types that are also included in the speech are information about language being spoken, speaker emotions, gender and identity of the speaker. The goal of automatic *speaker recognition* is to extract, characterize and recognize the information about speaker identity [40]. Speaker recognition is usually divided into two different branches, *speaker verification* and *speaker identification*. Speaker verification task is to verify the claimed identity of person from his voice [6,35]. This process involves only binary decision about claimed identity. In speaker identification there is no identity claim and the system decides who the speaking person is [6].

Speaker identification can be further divided into two branches. *Open-set speaker identification* decides to whom of the registered speakers unknown speech sample belongs or makes a conclusion that the speech sample is unknown. In this work, we deal with the *closed-set speaker identification*, which is a decision making process of who of the registered speakers is most likely the author of the unknown speech sample. Depending on the algorithm used for the identification, the task can also be divided into *text-dependent* and *text-independent* identification. The difference is that in the first case the system knows the text spoken by the person while in the second case the system must be able to recognize the speaker from any text. This taxonomy is represented in Figure 1.1.

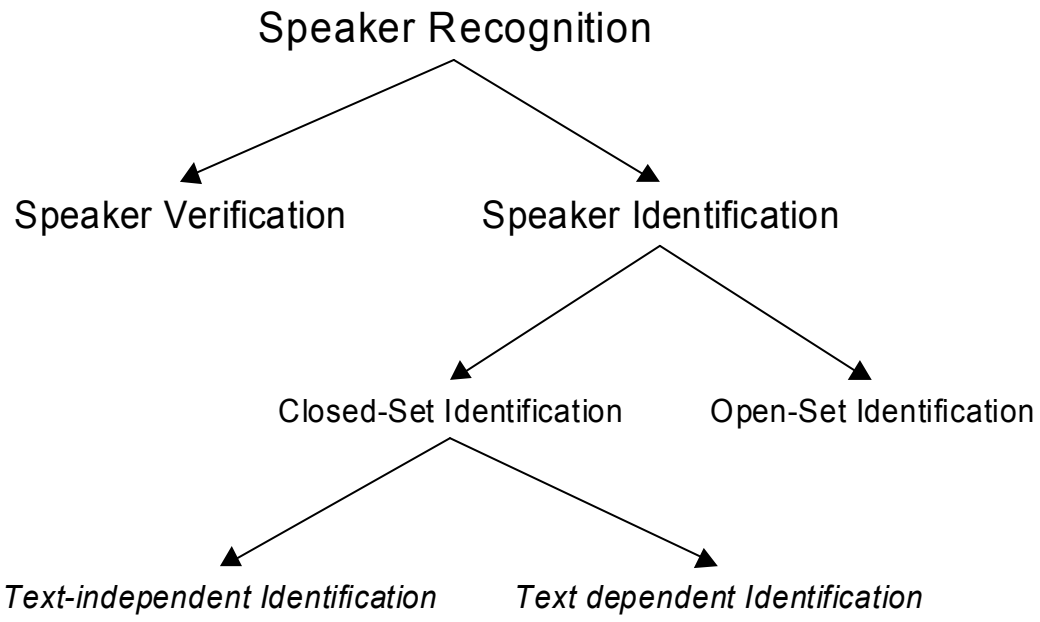

Figure 1.1 Identification Taxonomy

The process of speaker identification is divided into two main phases. During the first phase, *speaker enrollment*, speech samples are collected from the speakers, and they are used to train their models. The collection of enrolled models is also called a *speaker database*. In the second phase, *identification* phase, a test sample from an unknown speaker is compared against the speaker database. Both phases include the same first step, *feature extraction*, which is used to extract speaker dependent characteristics from speech. The main purpose of this step is to reduce the amount of test data while retaining speaker discriminative information. Then in the enrollment phase, these features are modeled and stored in the speaker database. This process is represented in Figure 1.2.

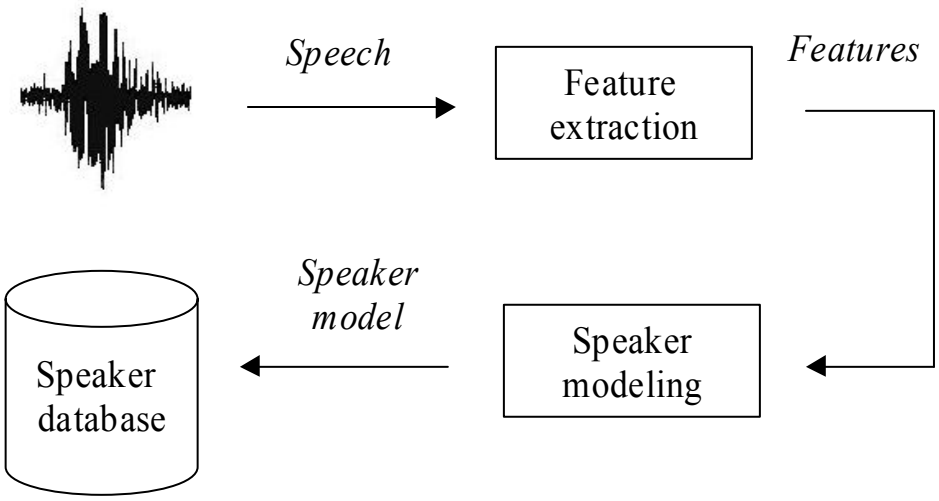

Figure 1.2 Enrollment Phase

In the identification step, the extracted features are compared against the models stored in the speaker database. Based on these comparisons the final decision about speaker identity is made. This process is represented in Figure 1.3.

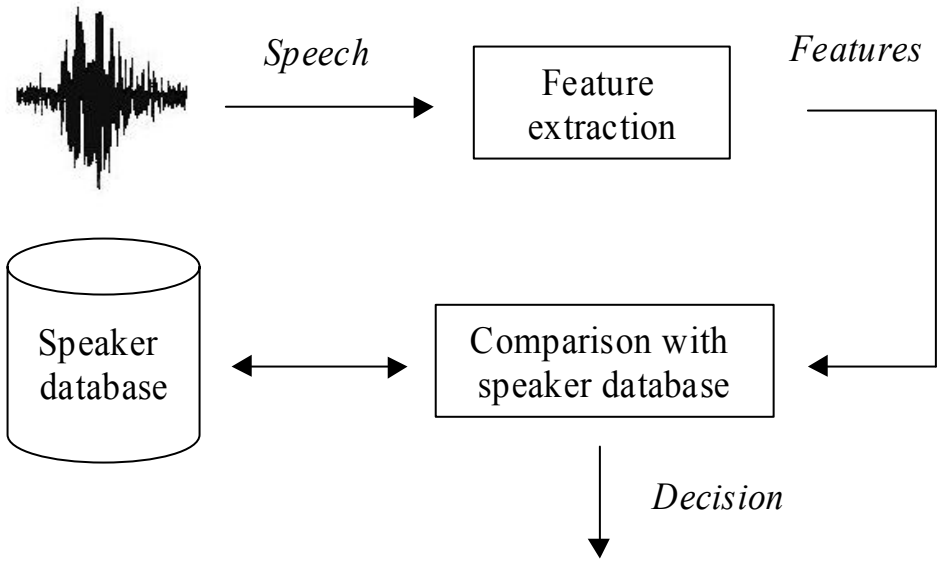

Figure 1.3 Identification Phase

However, these two phases are closely related. For instance, identification algorithm usually depends on the modeling algorithm used in the enrollment phase. This thesis mostly concentrates on the algorithms in the identification phase and their optimization.

1.2 Applications

Practical applications for automatic speaker identification are obviously various kinds of security systems. Human voice can serve as a key for any security objects, and it is not so easy in general to lose or forget it. Another important property of speech is that it can be transmitted by telephone channel, for example. This provides an ability to automatically identify speakers and provide access to security objects by telephone. Nowadays, this approach begins to be used for telephone credit card purchases and bank transactions. Human voice can also be used to prove identity during access to any physical facilities by storing speaker model in a small chip, which can be used as an access tag, and used instead of a pin code. Another important application for speaker identification is to monitor people by their voices. For instance, it is useful in information retrieval by speaker indexing of some recorded debates or news, and then retrieving speech only for interesting speakers. It can also be used to monitor criminals in common places by identifying them by voices.

In fact, all these examples are actually examples of real time systems. For any identification system to be useful in practice, the time response, or time spent on the identification should be minimized. Growing size of speaker database is also common fact for practical systems and can also lead to system optimization.

1.3 Thesis Description

Nowadays, speaker identification is not anymore just a theory. Applications based on it are widely used around the word and found their

appropriate places in the industry. But even though a lot of work has already done in this field [6,8,20], it is still not a solved problem. The research in the area of speaker identification still continues and at present there are a few basic techniques that have shown their effectiveness in practice and called “classical” by scientists. The goal of this work is to make general overview of these techniques and then analyze them from the real time system point of view. The main requirement, which is set by real-time system, is a fast identification time. Therefore, the main emphasize in this work is set on the optimization approaches for identification algorithms.

To give a better understanding, we start from the very beginning. In Chapter 2, we study the fundamentals of digital signal processing theory used in speaker identification, and model of biometric characteristics of human speech production organs. This model will serve us as a basis for techniques described in the next chapters. In Chapter 3, we study different methods for the extraction of the speaker characteristics from speech signal. In Chapter 4, we discuss possible ways for modeling of extracted characteristics and methods, used to calculate the dissimilarity value between unknown speech sample and the stored speaker models. In Chapter 5, we analyze the methods described in the two previous chapters and discuss some possible optimization approaches. In Chapter 6, we provide a novel approach for the optimization problem, which is evaluated in practice in Chapter 7. Finally, we finish this work by giving short discussion and conclusions in Chapter 8.

Chapter 2

Identification Background

In this chapter we discuss theoretical background for speaker identification. We start from the digital signal processing theory. Then we move to the anatomy of human voice production organs and discuss the basic properties of the human speech production mechanism and techniques for its modeling. This model will be used in the next chapter when we will discuss techniques for the extraction of the speaker characteristics from the speech signal.

2.1 DSP Fundamentals

According to its abbreviation, *Digital Signal Processing (DSP)* is a part of computer science, which operates with special kind of data – *signals*. In most cases, these signals are obtained from various sensors, such as microphone or camera. DSP is the mathematics, mixed with the algorithms and special techniques used to manipulate with these signals, converted to the digital form [45].

2.1.1 Basic Definitions

By *signal* we mean here a relation of how one parameter is related to another parameter. One of these parameters is called *independent parameter* (usually it is time), and the other one is called *dependent*, and represents what we are measuring. Since both of these parameters belong to the continuous range of values, we call such signal *continuous signal*. When continuous signal is passed through an *Analog-To-Digital converter (ADC)* it is said to be *discrete* or *digitized* signal. Conversion works in the following way: every time period, which occurs with frequency called *sampling frequency*, signal value is taken and *quantized*, by selecting an appropriate value from the range of

possible values. This range is called *quantization precision*, and usually represented as an amount of bits available to store signal value. Based on the *sampling theorem,* proved by Nyquist in 1940 [45], digital signal can contain frequency components only up to one half of the sampling rate. Generally, continuous signals are what we have in nature while discrete signals exist mostly inside computers. Signals that use time as the independent parameter are said to be in the *time domain*, while signals that use frequency as the independent parameter are said to be in the *frequency domain*.

One of the important definitions used in DSP is the definition of *linear system.* By *system* we mean here any process that produces *output* signal in a response on a given *input* signal. A system is called linear if it satisfies the following three properties: *homogeneity, additivity* and *shift invariance* [45]. Homogeneity of a system means that change in the input signal amplitude corresponds to the change in the output signal. Additivity means that the output of the sum of two signals results in the sum of the two corresponding outputs. And finally, shift invariance means that any shift in the input signal will result in the same shift in the output signal [8,38,45].

2.1.2 Convolution

An *impulse* is a signal composed of all zeros except one non-zero point. Every signal can be decomposed into a group of impulses, each of them then passed through a linear system and the resulting output components are synthesized or added together [45]. The resulting signal is exactly the same as obtained by passing the original signal through the system.

Every impulse can be represented as a shifted and scaled *delta function*, which is a *normalized* impulse, that is, sample number zero has a value of one and all other samples have a value of zero. When the delta function is passed through a linear system, its output is called *impulse response*. If two systems are different they will have different impulse responses. According to the properties of linear systems every impulse passed through it will result in the

scaled and shifted impulse response and scaling and shifting of the input are identical to the scaling and shifting of the output [38,45]. It means that knowing systems impulse response we know everything about the system [8,38,45].

Convolution is a formal mathematical operation, which is used to describe relationship between three signals of interest: input and output signals, and the impulse response of the system. It is usually said that the output signal is the input signal convolved with the system's impulse response. Mathematical equation of convolution for discrete signals is represented in the following (convolution is denoted as a star):

$$y[i] = x[i] * h[i] = \sum_{j=0}^{M-1} h[j]x[i-j] \qquad \textbf{(2.1)}$$

where *y[i]* is the output discrete signal, *x[i]* is the input discrete signal and *h[i]* is *M* samples long system's impulse response *flipped left-for-right*. Index *i* goes through the size of the output signal. Mathematics behind the convolution does not restrict how long the impulse response is. It only says that the size of the output signal is the size of the input signal plus the size of the impulse response minus one.

Convolution is very important concept in DSP. Based on the properties of linear systems it provides the way of combining two signals to form a third signal. A lot of mathematics behind the DSP is based on the convolution. In detail it is described in [8,38,45].

2.1.3 Discrete Fourier Transform

Fourier transform belongs to the family of linear transforms widely used in DSP based on decomposing signal into sinusoids (sine and cosine waves). Usually in DSP we use the *Discrete Fourier Transform* (DFT), a special kind of Fourier transform used to deal with aperiodic discrete signals [45]. Actually there are an infinite number of ways how signal can be decomposed but

sinusoids are selected because of their *sinusoidal fidelity* that means that sinusoidal input to the linear system will produce sinusoidal output, only the amplitude and phase may change, frequency and shape remain the same [45].

Discrete Fourier Transform changes an *N* point input signal into two *N/2+1* point output signals. The output signals represent the amplitudes of the sine and cosine components scaled in a special way that is represented by the equations:

$$C_k[i] = \cos(2 \cdot k \cdot i \cdot \pi / N)$$
$$S_k[i] = \sin(2 \cdot k \cdot i \cdot \pi / N) \quad \textbf{(2.2)}$$

where C_k are *N/2+1* cosine functions and S_k are *N/2+1* sine functions, index *k* runs from zero to *N/2*. These functions are called *basis functions*. Actually zero samples in resulting signals are amplitudes for zero frequency waves, first samples for waves which make one complete cycle in *N* points, second for waves which make two cycles and so on. Signal represented in such a way is called to be in *frequency domain* and obtained coefficients are called *spectral coefficients* or *spectrum*. Frequency domain contains exactly the same information as the time domain and every discrete signal can be moved back to the time domain, using operation called *Inverse Discrete Fourier Transform (IDFT)*. Because of this fact, the DFT is also called *Forward DFT* [45]. Schematically DFT is represented in Figure 2.1.

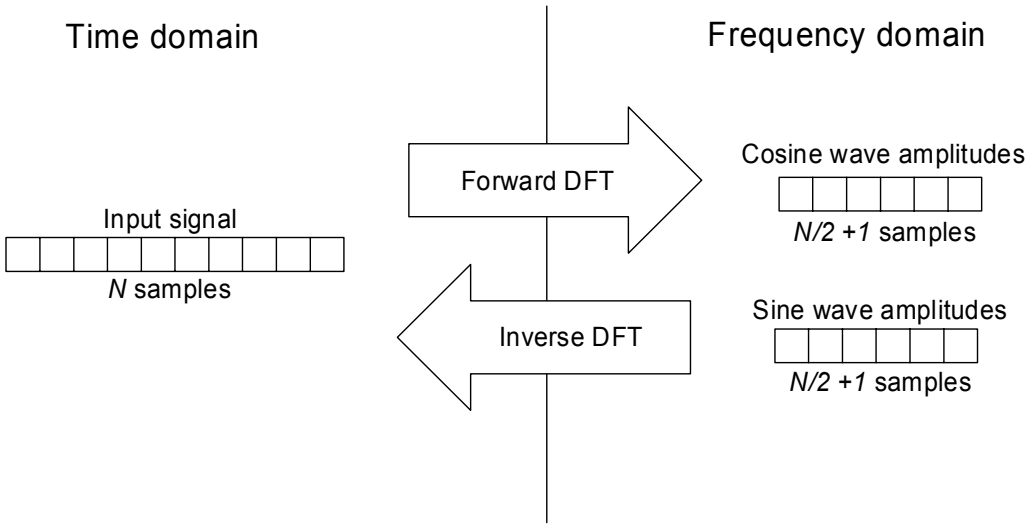

Figure 2.1 Discrete Fourier Transform

The amplitudes for cosine waves are also called *real part* (denoted as *Re[k]*) and for sine waves are called *imaginary part* (denoted as *Im[k]*). This representation of frequency domain is called *rectangular* notation. Alternatively, the frequency domain can be expressed in the *polar* notation. In this form, real and imaginary parts are replaced by magnitudes (denoted as *Mag[k]*) and phases (denoted as *Phase[k]*) respectively [45]. The equations for conversion from rectangular notation to the polar notation are as follows:

$$Mag[k] = \sqrt{(\mathrm{Re}[k]^2 + \mathrm{Im}[x]^2)}$$
$$Phase[k] = \arctan\left(\frac{\mathrm{Im}[k]}{\mathrm{Re}[k]}\right) \quad \textbf{(2.3)}$$

There are two main reasons why DFT became so popular in DSP. First is *Fast Fourier Transform (FFT)* algorithm [45], developed by Cooley and Tukey in 1965, which opened a new era in DSP because of the efficiency of the FFT algorithm. The second reason is the *convolution theorem* [45], which states that convolution in time domain is a multiplication in frequency domain and

vice versa. This makes possible to use high-speed convolution algorithm, which convolves two signals by passing them through the Fast Fourier Transform, multiplying and using Inverse Fourier Transform computing convolved signal. More details about Fourier Transform can be found in [8,38,45].

2.1.4 Filters

By *filter* we mean here a method to manipulate with signals defined as a linear system. There are two main uses for filters: signal *separation* and signal *restoration*. Signal separation is needed when the signal was interfered with the other not useful signals or noise. Signal restoration is needed when the signal was distorted for example due to the transform through a long wire or bad quality recording. There are two main types of filters: *analog* and *digital*. Analog filters are cheap and have a large dynamic range in frequency and amplitude. However, digital filters can achieve thousands better performance [45].

Easiest way to implement a digital filter is to convolve the input signal with the filters impulse response. Based on the length of its impulse responses, filters are usually divided into *Infinite Impulse Response (IIR)* filters and *Finite Impulse Response (FIR)* filters. There are also few types of responses: *step response* and *frequency response*. Each of these responses can be used to completely define filter. Step response is the output signal of the filter when input is a *step function*, which is defined as a transition from one level of signal to another. This type of responses can be used to define filters, which are able to divide signal into regions with similar characteristics. The frequency response can be found by taking discrete Fourier transform of the impulse response. It can be useful to define filters, which are able to block undesirable frequencies in input signals or separate one band of frequencies from another, such as high-pass, band-pass and band-reject filters.

Digital filter theory is important in speaker identification, since it allows by a given signal to analyze origin of it or in this case the unknown speaker. There are also few minor uses for filters like a noise removal or other types of filtering to achieve better results in signal analyzing. More details about filter design and implementation can be found in [8,38,45].

2.2 Human Speech Production Model

Undoubtedly, ability to speak is the most important way for humans to communicate between each other. Speech conveys various kind of information, which are essentially the meaning of information speaking person wants to impart, individual information representing speaker and also some emotional filling. Speech production begins with the initial formalization of the idea which speaker wants to impart to the listener. Then speaker converts this idea into the appropriate order of words and phrases according to the language. Finally, his brain produces motor nerve commands, which move the vocal organs in an appropriate way [14]. Understanding of how human produce sounds forms the basis of speaker identification.

2.2.1 Anatomy

The sound is an acoustic pressure formed of compressions and rarefactions of air molecules that originate from movements of human anatomical structures [20]. Most important components of the human speech production system are the *lungs* (source of air during speech), *trachea* (windpipe), *larynx* or its most important part *vocal cords* (organ of voice production), *nasal cavity* (nose), *soft palate* or *velum* (allows passage of air through the nasal cavity), *hard palate* (enables consonant articulation), *tongue*, *teeth* and *lips*. All these components, called *articulators* by speech scientists, move to different positions to produce various sounds. Based on their production, speech sounds can also be divided into consonants and voiced and unvoiced vowels [8,20].

From the technical point of view, it is more useful to think about speech production system in terms of an acoustic filtering operations that affect the air going from the lungs. There are three main cavities that comprise the main acoustic filter. According to [8] they are *nasal*, *oral* and *pharyngeal* cavities. The articulators are responsible for changing the properties of the system and form its output. Combination of these cavities and articulators is called *vocal tract*. Its simplified acoustic model is represented in Figure 2.2.

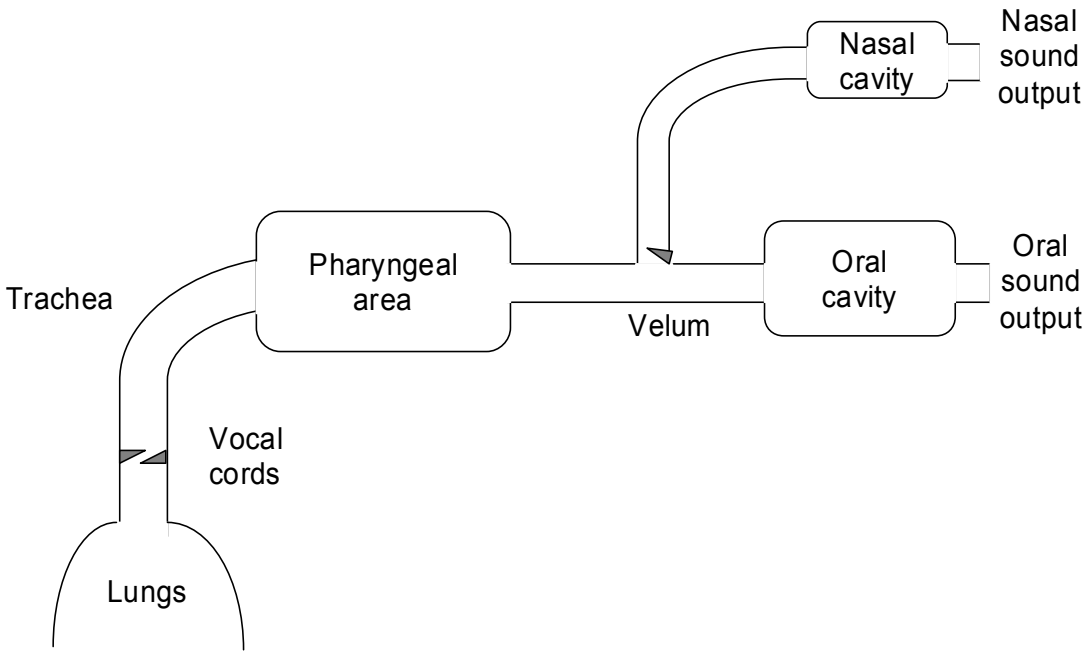

Figure 2.2 Vocal tract model

Speech production can be divided into three stages: first stage is the sound source production, second stage is the articulation by vocal tract, and the third stage is sound radiation or propagation from the lips and/or nostrils [14]. A v*oiced sound* is generated by vibratory motion of the vocal cords powered by the airflow generated by expiration. The frequency of oscillation of vocal cords is called the *fundamental frequency*. Another type of sounds - *unvoiced sound* is produced by turbulent airflow passing through a narrow constriction in the vocal tract [6,8].

In a speaker recognition task, we are interested in the physical properties of human vocal tract. In general it is assumed that vocal tract carries most of the speaker related information [6,8,20,39]. However, all parts of human vocal tract described above can serve as speaker dependent characteristics [6,8,39]. Starting from the size and power of lungs, length and flexibility of trachea and ending by the size, shape and other physical characteristics of tongue, teeth and lips. Such characteristics are called *physical distinguishing factors*. *A*nother aspects of speech production that could be useful in discriminating between speakers are called *learned factors*, which include speaking rate, dialect, and *prosodic* effects [6].

2.2.2 Vocal Model

In order to develop an automatic speaker identification system, we should construct reasonable model of human speech production system. Having such a model, we can extract its properties from the signal and, using them, we can decide whether or not two signals belong to the same model and as a result to the same speaker.

Modeling process is usually divided into two parts: the excitation (or source) modeling and the vocal tract modeling [8]. This approach is based on the assumption of independence of the source and the vocal tract models [6,8]. Let us look first at the *continuous-time* vocal tract model called *multitube lossless model* [8], which is based on the fact that production of speech is characterized by changing the vocal tract shape. Because the formalization of such a time-varying vocal-tract shape model is quite complex, in practice it is simplified to the series of concatenated lossless acoustic tubes with varying cross-sectional areas [8], as shown in Figure 2.3.

This model consists of a sequence of tubes with cross-sectional areas A_k and lengths L_k. In practice the lengths of tubes assumed to be equal [8]. If a large amount of short tubes is used, then we can approach to the continuously varying cross-sectional area, but at the cost of more complex model. Tract

model serves as a transition to the more general *discrete-time* model, also known as *source-filter model*, which is shown in Figure 2.4 [8].

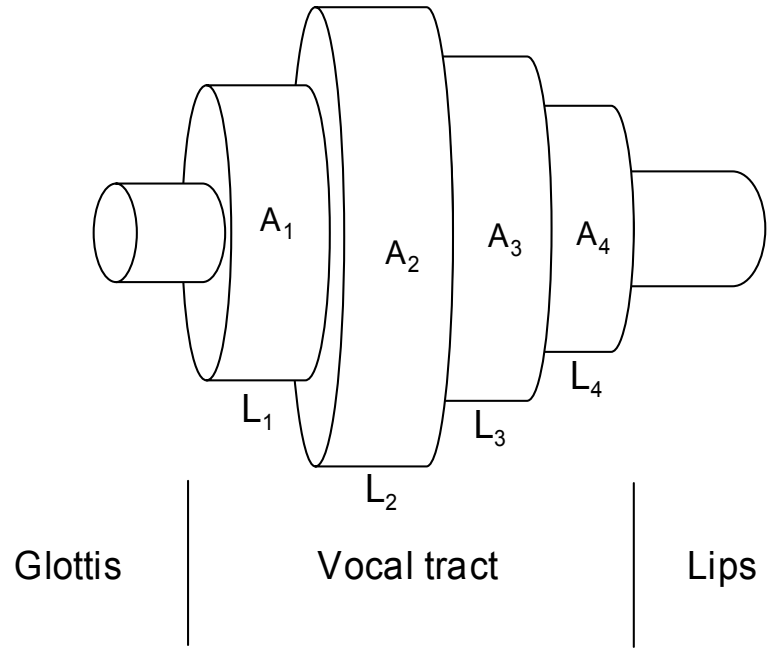

Figure 2. 3 Multitube lossless model

In this model, the voice source is either a periodic pulse stream or uncorrelated white noise, or a combination of these. This assumption is based on the evidence from human anatomy that all types of sounds, which can be produced by humans, are divided into three general categories: voiced, unvoiced and combination of these two (2.2.1). Voiced signals can be modeled as a basic or fundamental frequency signal filtered by the vocal tract and unvoiced as a white noise also filtered by the vocal tract. Here *E(z)* represents the *excitation function*, *H(z)* represents the *transfer function*, and *s(n)* is the output of the whole speech production system [8]. Finally, we can think about vocal tract as a digital filter, which affects source signal and about produced sound output as a filter output. Then based on the digital filter theory we can extract the parameters of the system from its output.

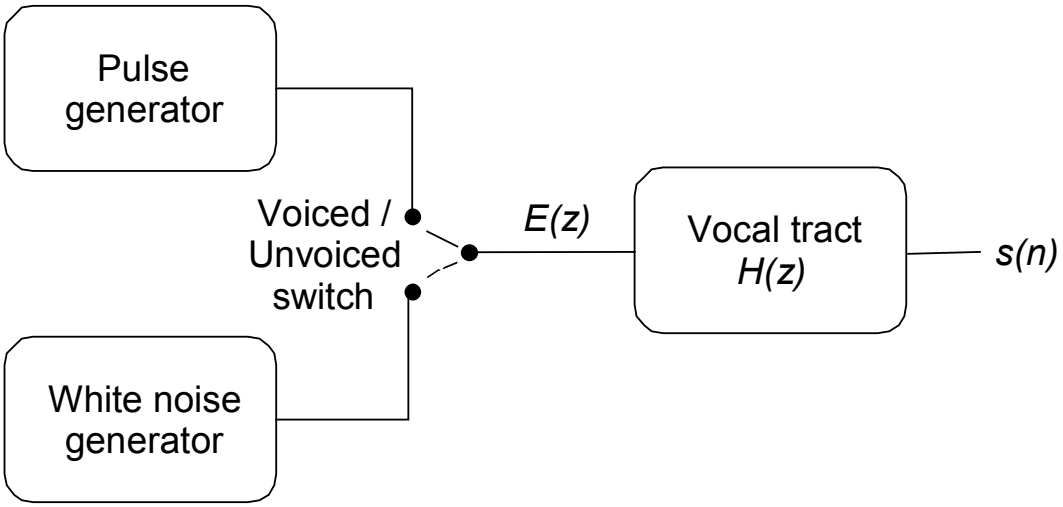

Figure 2.4 Source-filter model

The issues described in this chapter serve as a basis for developing speaker identification techniques described in the next chapter. More details about speech production system modeling can be found in [6,8,20,39].

Chapter 3

Feature Extraction

In this chapter we discuss the possible ways of extracting speaker discriminative characteristics from speech signal.

3.1 Introduction

The acoustic speech signal contains different kind of information about speaker. This includes “high-level” properties such as dialect, context, speaking style, emotional state of speaker and many others [35]. A great amount of work has been already done in trying to develop identification algorithms based on the methods used by humans to identify speaker. But these efforts are mostly impractical because of their complexity and difficulty in measuring the speaker discriminative properties used by humans [35]. More useful approach is based on the “low-level” properties of the speech signal such as *pitch* (fundamental frequency of the vocal cord vibrations), *intensity*, *formant frequencies* and their *bandwidths*, *spectral correlations*, *short-time spectrum* and others [1].

From the automatic speaker recognition task point of view, it is useful to think about speech signal as a sequence of *features* that characterize both the speaker as well as the speech. It is an important step in recognition process to extract sufficient information for good discrimination in a form and size which is amenable for effective modeling [17]. The amount of data, generated during the speech production, is quite large while the essential characteristics of the speech process change relatively slowly and therefore, they require less data. According to these matters *feature extraction* is a process of reducing data while retaining speaker discriminative information [8,17].

Based on the issues described above, we can define requirements that should be taken into account during selection of the appropriate speech signal characteristics or features [1,35]:

- discriminate between speakers while being tolerant of intra-speaker variabilities,
- easy to measure,
- stable over time,
- occur naturally and frequently in speech,
- change little from one speaking environment to another,
- not be susceptible to mimicry.

Of course, practically, it is not possible to meet all of these criteria and there will be always a trade-off between them, based on what is more important in the particular case.

The speech wave is usually analyzed based on spectral features. There are two reasons for it. First is that the speech wave is reproducible by summing the sinusoidal waves with slowly changing amplitudes and phases. Second is that the critical features for perceiving speech by humans ear are mainly included in the magnitude information and the phase information is not usually playing a key role [14].

3.2 Short-Term Analysis

Because of its nature, the speech signal is a slowly varying signal or *quasi-stationary*. It means that when speech is examined over a sufficiently short period of time (20-30 milliseconds) it has quite stable acoustic characteristics [8]. It leads to the useful concept of describing human speech signal, called "*short-term analysis*", where only a portion of the signal is used to extract signal features at one time. It works in the following way: predefined length window (usually 20-30 milliseconds) is moved along the signal with an

overlapping (usually 30-50% of the window length) between the adjacent frames. Overlapping is needed to avoid losing of information. Parts of the signal formed in such a way are called *frames.* In order to prevent an abrupt change at the end points of the frame, it is usually multiplied by a *window function*. The operation of dividing signal into short intervals is called *windowing* and such segments are called *windowed frames* (or sometime just *frames*). There are several window functions used in speaker recognition area [14], but the most popular is *Hamming window function*, which is described by the following equation:

$$w(n) = 0.54 - 0.46\cos\left(\frac{2n\pi}{N-1}\right) \qquad \textbf{(3.1)}$$

where *N* is the size of the window or frame. A set of features extracted from one frame is called *feature vector*. Overall overview of the short-term analysis approach is represented in Figure 3.1.

In the next subchapters we describe a few features, commonly used in speaker recognition. More details about feature selection and extraction can be found in [1,6,8,14,17,39,35].

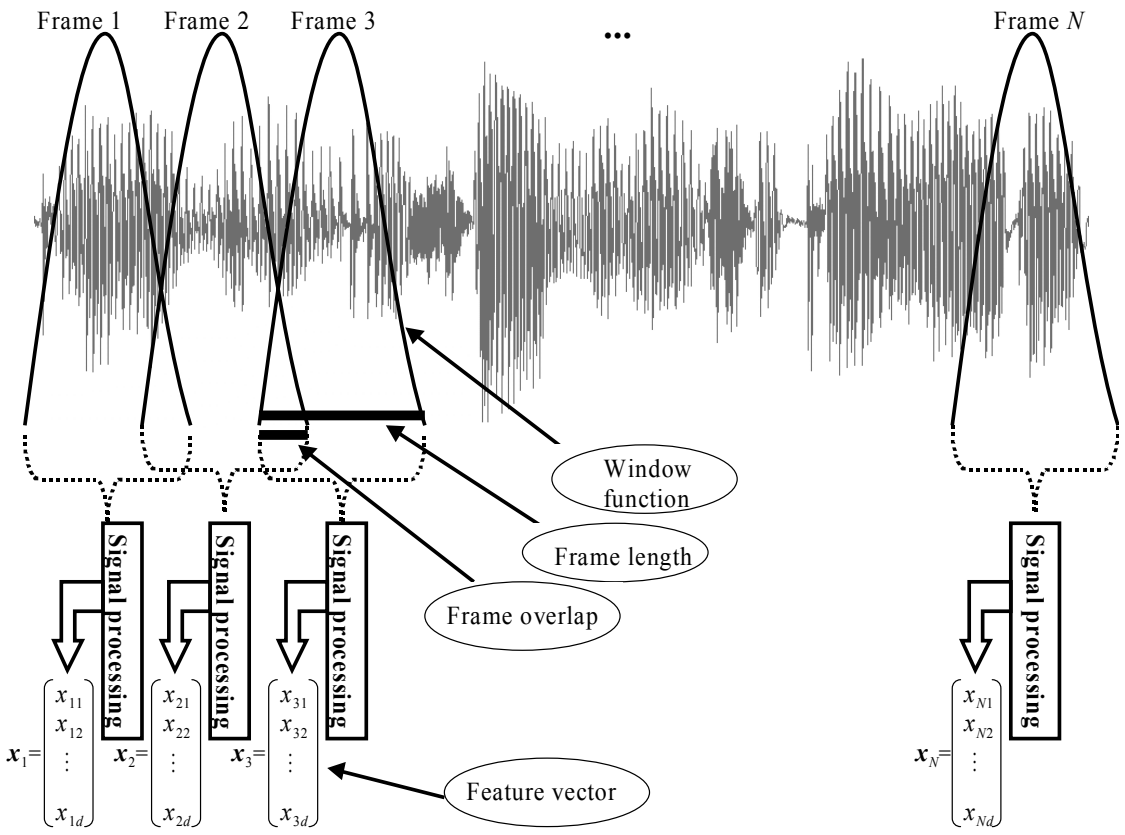

Figure 3.1 Short-Term Analysis

3.3 Cepstrum

According to the issues described in the subchapter (2.2.2), the speech signal *s(n)* can be represented as a "quickly varying" source signal *e(n)* convolved with the "slowly varying" impulse response *h(n)* of the vocal tract represented as a linear filter [8]. We have access only to the output (speech signal) and it is often desirable to eliminate one of the components. Separation of the source and the filter parameters from the mixed output is in general difficult problem when these components are combined using not linear operation, but there are various techniques appropriate for components combined linearly. The *cepstrum* is representation of the signal where these two components are resolved into two additive parts [8]. It is computed by taking the inverse DFT of the logarithm of the magnitude spectrum of the frame. This is represented in the following equation:

$$cepstrum(frame) = IDFT\,(\,\log(|DFT(frame)|)) \qquad \textbf{(3.2)}$$

Some explanation of the algorithm is therefore needed. By moving to the frequency domain we are changing from the convolution to the multiplication. Then by taking logarithm we are moving from the multiplication to the addition. That is desired division into additive components. Then we can apply linear operator inverse DFT, knowing that the transform will operate individually on these two parts and knowing what Fourier transform will do with quickly varying and slowly varying parts. Namely it will put them into different, hopefully separate parts in new, also called *quefrency* axis [8]. Let us look at the speech magnitude spectrum in Figure 3.2 [8].

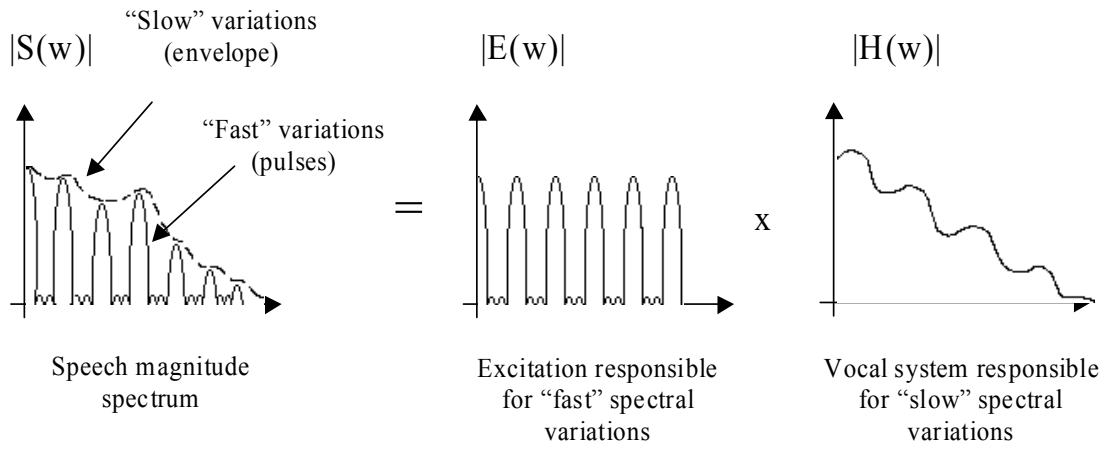

Figure 3.2 Speech magnitude spectrum

From the Figure 3.2 we can see that the speech magnitude spectrum is combined from slow and quickly varying parts. But there is still one problem: multiplication is not a linear operation. We can solve it by taking logarithm from the multiplication as described earlier. Finally, let us look at the result of the inverse DFT in Figure 3.3 [8].

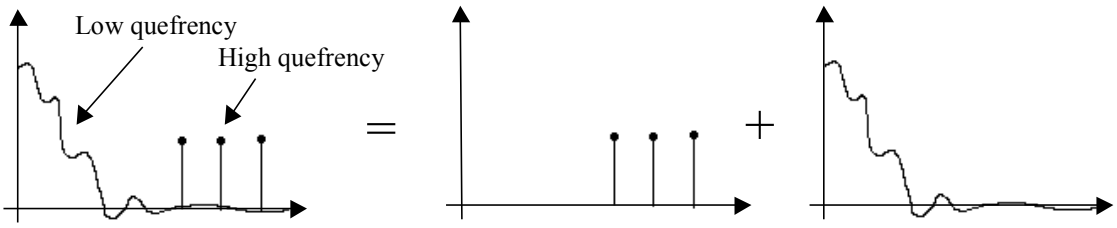

Figure 3.3 Cepstrum

From this figure we can see that two components are clearly distinctive now. Cepstrum is explained in more details in [8,17,39].

3.4 Mel-Frequency Cepstrum Coefficients

Mel-frequency cepstrum coefficients (MFCC) are well known features used to describe speech signal. They are based on the known evidence that the information carried by low-frequency components of the speech signal is phonetically more important for humans than carried by high-frequency components [8]. Technique of computing MFCC is based on the short-term analysis, and thus from each frame a MFCC vector is computed.

MFCC extraction is similar to the cepstrum calculation except that one special step is inserted, namely the frequency axis is warped according to the mel-scale. Summing up, the process of extracting MFCC from continuous speech is illustrated in Figure 3.4.

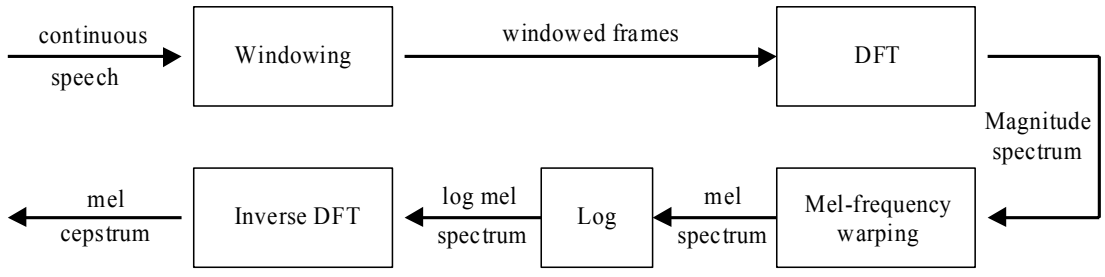

Figure 3.4 Computing of mel-cepstrum

As described above, to place more emphasize on the low frequencies one special step before inverse DFT in calculation of cepstrum is inserted, namely mel-scaling. A "*mel*" is a unit of special measure or scale of *perceived pitch* of a tone [8]. It does not correspond linearly to the normal frequency, indeed it is approximately linear below 1 kHz and logarithmic above [8]. This approach is based on the psychophysical studies of human perception of the frequency content of sounds [8,39]. One useful way to create mel-spectrum is to use a filter bank, one filter for each desired mel-frequency component. Every filter in this bank has triangular bandpass frequency response. Such filters compute the average spectrum around each center frequency with increasing bandwidths, as displayed in Figure 3.5.

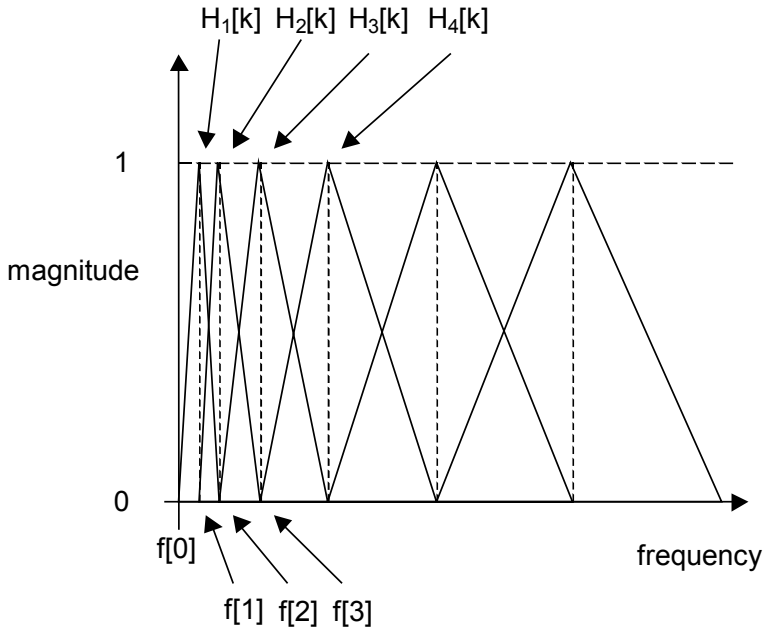

Figure 3.5 Triangular filters used to compute mel-cepstrum

This filter bank is applied in frequency domain and therefore, it simply amounts to taking these triangular filters on the spectrum. In practice the last step of taking inverse DFT is replaced by taking *discrete cosine transform (DCT)* for computational efficiency.

The number of resulting mel-frequency cepstrum coefficients is practically chosen relatively low, in the order of 12 to 20 coefficients. The zeroth coefficient is usually dropped out because it represents the average log-energy of the frame and carries only a little speaker specific information. However, MFCC are not equally important in speaker identification [3] and thus some coefficients weighting might by applied to acquire more precise result. Different approach to the computation of MFCC than described in this work is represented in [34] that is simplified by omitting filterbank analysis. More details about MFCC can be found in [6,8,17,20,39].

3.5 Linear Predictive Coding

Another widely used in speaker recognition area method for speech signal analysis is based on *Linear predictive coding (LPC)* (also know as *auto-regressive modeling* or *AR-modeling*). LPC is based on the speech production source-filter model described in subchapter (2.2.2) and it assumes that this model is an *all-pole model*. As described in [8] the speech production system can be ideally characterized by the pole-zero system function and such assumption to use only poles has two main reasons. First reason is the simplicity, and as we will see LPC will result in simple *linear* equations. Second reason is that based on human perception mechanism, human ear is fundamentally “phase deaf” and phase information is less important. All-pole model can exactly preserve magnitude spectral dynamics (the “information”) in the speech but may not retain the phase characteristics [8].

The main idea behind LPC is that given speech sample can be approximated as a linear combination of the past speech samples [39]. LPC models signal *s(n)* as a linear combination of its past values and present input (vocal cords excitation). Because in speaker recognition task the present input is generally unknown it is simply ignored [6]. Therefore, the LPC

approximation depends only on the past values, which is represented by the equation:

$$\hat{s}(n) = -\sum_{k=1}^{p} a_k \cdot s(n-k) \qquad \textbf{(3.3)}$$

where *ŝ(n)* is an approximation of the present output, *s(n-k)* are past outputs, *p* is the prediction order, and a_k are the model parameters called the *predictor coefficients*. Prediction error is defined as the difference between real and predicted output, also called as *prediction residual*.

In speaker recognition task, we can use LPC based on the short-term analysis approach. Because of the quasi-stationary nature of speech, we can compute a set of prediction coefficients from every frame. Then we can use these coefficients as features to describe the signal and therefore, the speaker. In practice, prediction order is set to 12-20 coefficients, depending on the sampling rate and the number of poles in the model. More details about selection of prediction order can be found in [20].

Thus, the basic problem in LPC analysis is to determine prediction coefficients from the speech frame. There are two main approaches how to derive them. The classical *least-square* method selects prediction coefficients to minimize the mean energy in prediction error over a frame of speech [44]. Examples of this method are *autocorrelation* and *covariance* methods. More details about these two common methods can be found in [6,8,20,39,44]. Another approach is called *lattice*, which permits instantaneous updating of the coefficients [44]. In other words, LPC parameters are determined sample by sample. This method is more useful for real-time application.

In speaker recognition area the set of prediction coefficients is usually converted to the so-called *linear predictive cepstral coefficients (LPCC)*, because cepstrum is proved to be the most effective representation of speech signal for speaker recognition [1]. An important fact is that it can be done

directly from the LPC parameter set. The relationship between cepstrum coefficients c_n and prediction coefficients a_k is represented in the following equations [1]:

$$
\begin{aligned}
c_1 &= a_1 \\
c_n &= \sum_{k=1}^{n-1} (1 - k/n) \cdot a_k \cdot c_{n-k} + a_n, \qquad 1 < n \le p \\
c_n &= \sum_{k=1}^{n-1} (1 - k/n) \cdot a_k \cdot c_{n-k}, \qquad n > p
\end{aligned}
\tag{3.4}
$$

where p is a prediction order. It is usually said that the cepstrum, derived in such a way represents the "smoothed" version of the spectrum [8]. More details about LPC and LPCC can be found in [1,6,8,20,39,44].

3.6 Alternatives and Conclusions

MFCC and LPCC described above are well known techniques used in speaker identification to describe signal characteristics, relative to the speaker discriminative vocal tract properties. They are quite similar as well as different. Both MFCC and LPCC result in the cepstrum coefficients, but the method of computation differs. MFCC are based on the filtering of spectrum using properties of human speech perception mechanism. On the other hand, LPCC are based on the autocorrelation of the speech frame. There is no general agreement in the literature about what method is better. However, it is generally considered that LPCC are computationally less expensive while MFCC provide more precise result [17]. The reason of such opinion is based on that all-pole model used in the LPC provides a good model for the voiced regions of speech and quite bad for unvoiced and transient regions [39]. The main drawback of LPCC is that it does not resolve the vocal tract characteristics from the glottal dynamics [8], which vary from person to person

and might be useful in speaker identification, whereas MFCC just pay less attention to them. However, some authors do not agree with the psychoacoustic analysis on which MFCC are based [49]. More broad discussion about the advantages and disadvantages of MFCC and LPCC can be found in [8].

As alternatives for the methods described in this work, a few different approaches can be suggested. First approach is to improve either MFCC or LPCC. For example, well-known technique to improve recognition is to add the first-order derivatives of cepstrum coefficients called *delta features* to every feature vector [8]. Such features capture the time dynamics of cepstrum coefficients from frame to frame. Another technique to improve recognition accuracy of systems based on MFCC is proposed in [10]. This method is based on the adding of information about the pitch into the feature vectors. Yet another approach is to combine MFCC and LPCC. This method can be found in [8].

Finally, other types of features can be used in speaker identification, such as *perceptual linear prediction cepstrum coefficients (PLPCC)* [20,30] or *eigen-MLLR coefficients* [52]. Experimental evaluation of recognition accuracy of the MFCC, LPCC and PLPCC was made in [42] and result of this report is that all features perform poorly without some form of channel compensation, however, with channel compensation MFCC slightly outperform other types [42].

Cepstrum representation of the speech signal has shown to be useful in practice. However, it is not without drawbacks. The main disadvantage of the cepstrum is that it is quite sensitive to the environment and noise [8]. Therefore, in practice speech signal is usually preprocessed to achieve more precise representation. This process usually includes noise removal [8,43] and pre-emphasis [8,50]. One approach for separating speaker information and environment can be found in [43]. More details about cepstrum and other feature extraction methods can be found in [1,6,8,14,17,20,30,39,40,42].

Chapter 4

Feature Matching and Speaker Modeling

In this chapter we discuss techniques for modeling of features extracted from the speech signal, and methods, which are allowing to compute dissimilarity between unknown speech sample and stored speaker models.

4.1 Introduction

In the previous chapter we were discussing so called *measurement* step in the speaker identification where a set of speaker discriminative characteristics is extracted from the speech signal. In this chapter, we go through the next step called *classification*, which is a decision making process of determining the author of a given speech signal based on the previously stored or learned information [1]. This step is usually divided into two parts, namely *matching* and *modeling*. The modeling is a process of enrolling speaker to the identification system by constructing a model of his/her voice, based on the features extracted from his/her speech sample. The matching is a process of computing a *matching score*, which is a measure of the similarity of the features extracted from the unknown speech sample and speaker model [6].

There are two main approaches for solving the classification problem in the speaker identification, namely *template matching* and *stochastic matching* [6]. The template method can be dependent or independent of time. In the time-dependent template approach the model consists of a sequence of feature vectors extracted from a fixed phrase. During identification a matching score is produced using *dynamic time warping (DTW)* algorithm to align and measure the similarity between the template and test phrase [6,40]. This method can be used for text-dependent identification systems. For text-independent systems there is a variation of template matching called *feature averaging* [17], which uses the mean of some feature over a relatively long

period of time to distinguish among speakers, based on the distance to the average feature. An alternative stochastic approach is to build probabilistic model of the speech signal that describes its time-varying characteristics [35]. This method refers to the modeling of speakers by probability distributions of feature vectors and its classification decision is based on the probabilities or likelihoods [17]. In the following text we go shortly trough the most popular and well-known techniques used in modeling and matching. More details about classification step in speaker identification can be found in [1,6,17,35,40].

4.2 Vector Quantization

Vector quantization (VQ) is a process of mapping vectors from a vector space to a finite number of regions in that space. These regions are called *clusters* and represented by their central vectors or *centroids*. A set of centroids, which represents the whole vector space, is called a *codebook*. In speaker identification, VQ is applied on the set of feature vectors extracted from the speech sample and as a result, the speaker codebook is generated. Such codebook has a significantly smaller size than extracted vector set and referred as a speaker model. Actually, there is some disagreement in the literature about approach used in VQ. Some authors [6] consider it as a template matching approach because VQ ignores all temporal variations and simply uses global averages (centroids). Other authors [17,35] consider it as a stochastic or probabilistic method, because VQ uses centroids to estimate the modes of a probability distribution [17]. Theoretically it is possible that every cluster, defined by its centroid, models particular component of the speech. But practically, however, VQ creates unrealistically clusters with rigid boundaries in a sense that every vector belongs to one and only one cluster [17].

Mathematically a VQ task is defined as follows: given a set of feature vectors, find a partitioning of the feature vector space into the predefined

number of regions, which do not overlap with each other and added together form the whole feature vector space. Every vector inside such region is represented by the corresponding centroid [46]. The process of VQ for two speakers is represented in Figure 4.1.

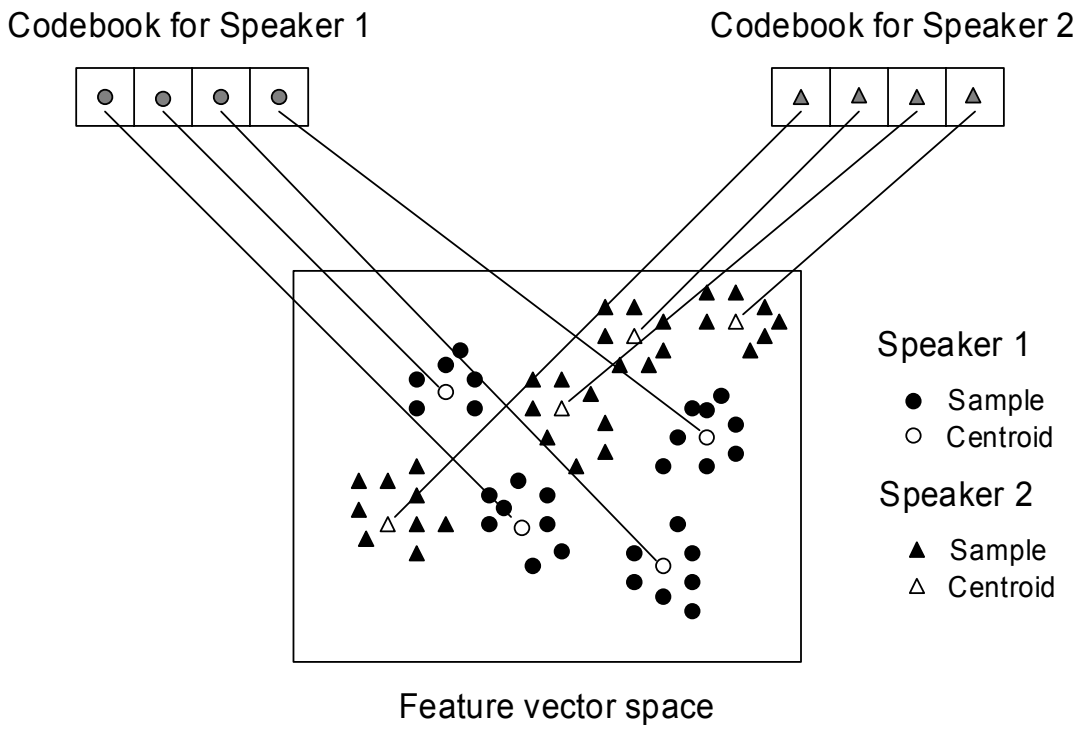

Figure 4.1 Vector quantization of two speakers

The are two important design issues in VQ: the method for generating the codebook and codebook size [24]. Known clustering algorithms for codebook generation are [24]:

- *Generalized Lloyd algorithm (GLA)*,
- *Self-organizing maps (SOM)*,
- *Pairwise nearest neighbor (PNN)*,
- *Iterative splitting technique (SPLIT)*,
- *Randomized local search (RLS)*.

According to [24], iterative splitting technique [12] should be used when the running time is important but RLS [13] is simpler to implement and generates better codebooks in the case of speaker identification task. Codebook size is a trade-off between running time and identification accuracy. With large size, identification accuracy is high but at the cost of running time and vice versa [24]. Experimental result obtained in [24] is that saturation point choice is 64 vectors in codebook. The *quantization distortion* (quality of quantization) is usually computed as the sum of squared distances between vector and its representative (centroid) [13]. The well-known distance measures are *Euclidean*, *city block distance*, *weighted Euclidean* and *Mahalanobis* [6,36]. They are represented in the following equations:

$$d_C(x,y) = \sum_{i=1}^{N} |x_i - y_i| \qquad \text{City block distance}$$

$$d_E(x,y) = (x-y)^T \cdot (x-y) = \sum_{i=1}^{N} (x_i - y_i)^2 \qquad \text{Euclidean distance} \qquad \textbf{(4.1)}$$

$$d_M(x,y) = (x-y)^T \cdot D^{-1} \cdot (x-y) \qquad \text{Weighted Euclidean distance}$$

where *x* and *y* are multi-dimensional feature vectors and *D* is a weighting matrix [6,36]. When *D* is a covariance matrix weighted Euclidean distance also called *Mahalanobis distance* [6,36]. A set of observation was made in [36] concerning the choice of distance for speaker identification task. Their conclusion is that weighted Euclidean distance where *D* is a diagonal matrix and consists of diagonal elements of covariance matrix is more appropriate, in a sense that it provides more accurate identification result. The reason for such result is that because of their nature not all components in feature vectors are equally important [3] and weighted distance might give more precise result.

During the matching a matching score is computed between extracted feature vectors and every speaker codebook enrolled in the system. Commonly it is done as a partitioning extracted feature vectors, using centroids from speaker codebook, and calculating matching score as a quantization distortion. Another choice for matching score is *mean squared error (MSE)*, which is computed as the sum of the squared distances between the vector and nearest centroid divided by number of vectors extracted from the speech sample. MSE formula is represented in the following:

$$MSE(X,C) = \frac{1}{N}\sum_{i=1}^{N}\min_{j}(d(x_i, c_j))^2 \qquad \textbf{(4.2)}$$

where X is a set of N extracted feature vectors, C is a speaker codebook, x_i are feature vectors, c_i are codebook centroids and d is any of distance functions. However, these methods are not adapted to the speaker identification. More realistic approaches are proposed in [22,25], which are based on the assigning of weights to the code vectors according to their discrimination power or the correlations between speaker models in the database.

The final identification decision is made based on the matching score: speaker who has a model with the smallest matching score is selected as an author of the test speech sample. More details about vector quantization can be found in [13,15,46].

4.3 Gaussian Mixture Modeling

Another type of speaker modeling techniques is *Gaussian mixture modeling (GMM)*. This method belongs to the stochastic modeling and based on the modeling of statistical variations of the features. Therefore, it provides a statistical representation of how speaker produces sounds [40].

A *Gaussian mixture density* is a weighted sum of component densities, as represented in the following equation [6,17,41]:

$$p(x) = \sum_{i=1}^{M} p_i \cdot b_i(x) \qquad \textbf{(4.3)}$$

where M is a number of components, x is a multi-dimensional feature vector, $b_i(x)$ are the components densities and p_i are the mixture weights or prior probabilities. To ensure that the mixture is a proper density, the prior probabilities should be chosen to sum to unity [17]. Each component density is given by equation:

$$b_i(x) = \frac{1}{(2 \cdot \pi)^{\frac{N}{2}} \cdot \left|\Sigma_i\right|^{\frac{1}{2}}} \cdot \exp\left\{-\frac{1}{2} \cdot (x - \mu_i)' \cdot \Sigma_i^{-1} \cdot (x - \mu_i)\right\} \qquad \textbf{(4.4)}$$

where N is a dimensionality of feature vector x, μ_i is a mean vector and Σ_i is a covariance matrix for *i-th* component [17,41]. For the identification each speaker is represented by his/her GMM, which is parameterized by the mean vectors, covariance matrices and mixture weights from all component densities. The number of components must be determined, either by some clustering algorithm or by automatic speech segmenter [17,41]. An initial model can be obtained by the estimating of parameters from the clustered feature vectors whereas proportions of vectors in each cluster can serve as a mixture weights. Means and covariances are estimated from the vectors in each cluster. After the estimation, the feature vectors can be reclustered using component densities (likelihoods) from the estimated mixture model and then model parameters are recalculated. This process is iterated until model parameters converge [17,41]. This algorithm is called *Expectation Maximization (EM)* and explained in detail in [41]. In identification phase, mixture densities are calculated for every feature vector for all speakers and

speaker with maximum likelihood is selected as the author of a speech sample.

The GMM has several forms depending on the choice of covariance matrix. The model can have covariance matrix per one component density, per one speaker or shared for all speakers [41]. The underlying reasons behind the using of GMM in speaker identification are well explained in [41]. More information about GMM can be found in [1,6,17,41].

4.4 Decision

The next step after computing of matching scores for every speaker model enrolled in the system is the process of assigning the exact classification mark for the input speech. This process depends on the selected matching and modeling algorithms. In template matching, decision is based on the computed distances, whereas in stochastic matching it is based on the computed probabilities. This process is represented in Figure 4.2.

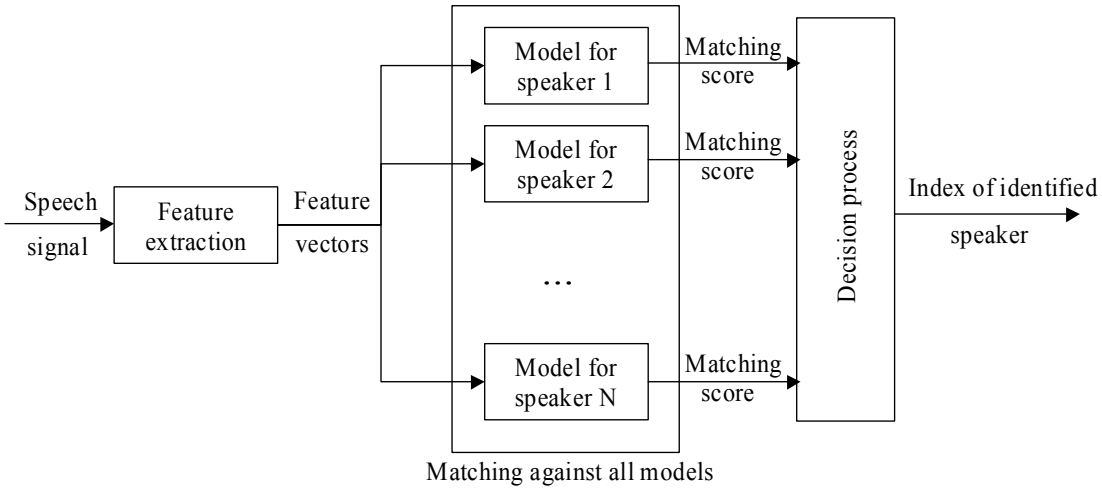

Figure 4.2 Decision process

In template matching, the speaker model with smallest matching score is selected, whereas in stochastic matching, the model with highest probability is selected. Practically, decision process is not so simple and for example for so called open-set identification problem the answer might be that input speech signal does not belong to any of the enrolled speaker models. More details about decision process can be found in [6,17].

4.5 Confidence of Decision

After performing identification it might be useful to measure the *confidence* of the decision. It might be needed in the open-set task when the speaker model may not exist in the speaker database or, based on confidence threshold, identification result might be classified as reliable or not. Unreliable tests can be for example further processed by human [17]. The underlying assumption in confidence measurements is that maximum score for correct identification is in general higher than scores for incorrect identifications and therefore, a *confidence measure* is a quantification of this assumption [17]. According to [17], the confidence measure is a number from 0 to 1, where 0 corresponds to the no confidence at all and 1 to the certainty.

In stochastic models, identification process results in a measure of likelihood or conditional probability [6]. There are several methods of confidence measure based on likelihoods. For speaker identification two different methods are proposed in [17]. The first method is based on the *significance testing*. In order to estimate the confidence, a two-term mixture model of obtained score is constructed:

$$p(x) = P(C_F) \cdot f_F(x) + P(C_T) \cdot f_T(x) \qquad \textbf{(4.5)}$$

where x denotes the score of identified speaker, C_F and C_T denote the classes of incorrect and correct identifications respectively, $f_F(x)$ and $f_T(x)$ denote the distributions of incorrectly and correctly identified speakers, $P(C_T)$ is the probability of correct identification and $P(C_F)=1-P(C_T)$ is the probability of incorrect identification. Both $f_F(x)$ and $f_T(x)$ are assumed to be normal distributions, and four parameters associated with them as well as $P(C_T)$ can be estimated, for instance, by using cross-validation [17]. The *significance confidence measure* is a measure of how far on the tail of the distribution $f_F(x)$ the identification result occurred. Such confidence measure (CM) is defined as follows:

$$CM(x) = 1 - \int_{x}^{\infty} f_F(x)dx \qquad \textbf{(4.6)}$$

The higher the confidence, the more we trust that matching score is too high to be incorrect. The problem with this approach is that it does not use the probability of incorrect classification [17]. Another approach to attack this problem is based on Bayes rule [17]. Bayes confidence measure is defined as follows:

$$P(C_T|x) = \frac{P(C_T)\cdot f_T(x)}{P(C_F)\cdot f_F(x) + P(C_T)\cdot f_T(x)} \qquad \textbf{(4.7)}$$

which is a probability that matching score x is a correct identification. More details about these two approaches can be found in [17].

However, in template matching models the result is deterministic and based on the distance calculation between model and input feature vectors and therefore, we can not use the probability theory apparatus. The likelihood in such models can be approximated by exponentiating the matching score [6]:

$$L = \exp(-a \cdot d) \tag{4.8}$$

where *d* is a distance value and *a* is a positive constant, which is set empirically [6]. In this way having the matching scores as the likelihoods, we can use the probabilistic methods described above to calculate confidence measures. In this work, we propose another approach for measuring confidence in template matching models. It is based on the assumption that distribution of matching scores follows a Gaussian shape. Proposed confidence measure is represented as follows:

$$C(d) = \exp\left(-\frac{1}{2} \cdot \left(\frac{d}{\sigma} \right)^2 \right) \tag{4.9}$$

where *d* is a distance or matching score returned by matching function and *σ* is a parameter selected based on how strong we are measuring confidence, or in other words, based on what is more important: either do not accept incorrect identification or prevent incorrect rejection. The intuitive idea behind this approach is that we are quite confident in the matching score if it is clearly different from other distances. The parameter *σ* is selected in the following way: first compute the mean of all distances, then select *σ* from the interval from zero to mean. More close to the mean we select *σ* more higher confidence will be assigned for the matching score and vice versa.

Another novel approach was recently proposed in [21]. Confidence in this work is measured based on the duration of speech samples used for modeling and identification, level of noise in speech signal and overlapping of speaker models. This fusion technique is shown to have high accuracy for both stochastic and template matching models. Also the influence of the amount of models enrolled in the system (population size) on the confidence measure is studied in [37]. In real-time systems, the confidence measure might be used

as a stopping criterion, e.g. when it reaches some predefined threshold, there are no reasons anymore to continue identification. More details about confidence measurement can be found in [6,17,21,37].

4.6 Alternatives and Conclusions

The issues described in this chapter actually fall into the more general topic, namely *pattern recognition*, which aims to classify object of interest into one of a number of classes [48]. Therefore, the methods applicable for pattern recognition are applicable for speaker identification as well. VQ and GMM are the most well studied techniques for speaker identification. Both of these methods aim to produce reasonable model for high accuracy identification. However, VQ works mostly as a quantifier rather than modeler and therefore, in practice it produces reduced number of feature vectors rather than speaker model. Whereas GMM models stochastic processes, which underlie speech signal, and therefore, it produces more accurate speaker model for robust identification [41]. GMM is based on the broader theory, *Hidden Markov Models*, which got its name “hidden” because it models hidden or not observable stochastic process (speech production) that can be observed through another stochastic process (speech signal) [8,20,35]. However, VQ has its own place in the speaker identification and has shown good results in practice [15,22,24,46]. It outperforms GMM in the tasks where the small amount of training data is available and sufficiently fast modeling (training) time is necessary [9,11]. VQ approach dominated early work in speaker identification whereas stochastic modeling has been developed recently and offers more flexible and theoretically meaningful probabilistic score [6]. Comparison of GMM with other techniques can be found in [11,41,42,51].

As an alternatives to the two techniques described above few methods can be suggested. First of all, these are modified GMM’s techniques [17,47], modified VQ [11] and combination of VQ and GMM [9,19]. A novel and fast

developing nowadays approach to speaker identification problem is *neural network (NN)* based methods [53]. Instead of training of an individual model for each speaker neural networks are trained to model differences among known speakers and therefore, requires less amount of parameters and more efficiently performs in training and identification phases [53]. Some comparison of NN approach and GMM can be found in [51]. More details about feature matching and speaker modeling can be found in [6,8,14,17,20,35,39,41].

4.7 Remarks

In chapters 2,3,4 we were discussing about general techniques used in speaker identification area. These methods serve as a basis for future investigations and ideas behind them still lead researchers to the new discoveries. Nowadays it is obvious that it is possible to recognize speakers from their voices using computers, at least under laboratory environments and within small speaker populations. Nowadays research in speaker identification area is mostly concentrated on the developing fast and robust algorithms, which can work in difficult, from the identification task point of view, conditions, such as in noise or using poor environments. The motivation for future work is driven by practical and economical applications of automatic speaker recognition. In the next chapters we judge these basic techniques from the real-time speaker identification task point of view and also propose few solutions for this kind of identification problems.

Chapter 5

Real-Time Speaker Identification

Speaker identification is a computationally expensive task and requires a large amount of computations to identify the unknown speaker. In this chapter, we analyze the speaker identification methods from the running time point of view. We do not discuss here classical optimization problems but concentrate only on the specific for speaker identification area approaches to optimization. We start from the analysis of basic techniques, described in the previous chapters. Then we discuss possibilities of their optimization.

5.1 Introduction

In this context, by *real-time system* we refer to a system, which works under some time constraints. These constraints are defined using so called *response time*, which is a length of time from the moment when the task for the system was set and the moment when the system replied with the answer [29]. Usually, time constraints are divided into two types: *hard* and *soft*. Under hard constraints, when the system can not accomplish its task in proper time it should stop executing of the task and reply with failure, whereas under soft constraints system can continue executing its task. More details about real-time systems can be found in [29].

By *real-time speaker identification (RTSI)* we mean here the process of identification, which works at the same time when the unknown person is speaking. More precise, RTSI system is a soft real-time system with response time is set to the length of the input speech sample. However, speaker identification is the time-consuming process and a growing population size dramatically decreases identification time, because matching score should be computed for every speaker enrolled in the system. Therefore, some optimization is required. As a motivation for necessity of optimization, a typical

example of the growing identification time as a function of population size is represented in Figure 5.1.

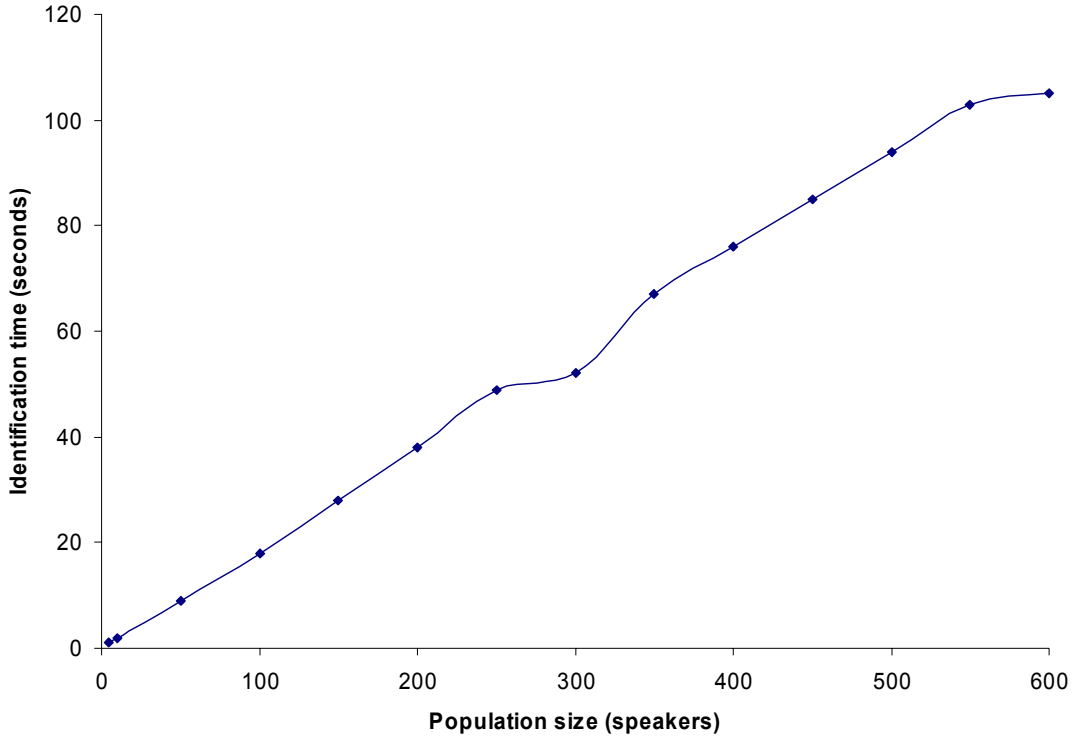

Figure 5.1 Dependency of identification time on population size

However, we do not consider here optimization of speaker modeling, because it can be done once off-line and used during the many identifications. At the speaker modeling step, accuracy of modeling is more important for speaker identification rather than computation speed.

5.2 Front-End Analysis and Optimization

In chapter 3, we discussed two popular types of features, the mel-frequency cepstral coefficients (MFCC) and linear predictive cepstral coefficients (LPCC). In this subchapter, we analyze the time complexities of these algorithms and also discuss some optimization issues.

5.2.1 MFCC and LPCC Analysis

As it was described in chapter 3, MFCC and LPCC are computed based on the short-term analysis or, in other words, a vector of MFCC or LPCC is computed for every speech frame. Knowing the time needed to extract one MFCC or LPCC vector we can easily compute the time needed to extract vectors from the whole speech sample. We also compute approximate amount of operations instead of *order* of algorithm or classical asymptotic time complexity, because for real-time case problem size (frame size) is relatively small and order analysis does not make sufficient sense.

Let us assume that the analysis frame has N samples. At first, frame is multiplied by a window function. It takes N operations. Then the FFT is taken from the speech sample. Time complexity of the FFT is $N \cdot logN$ [45]. Next step is to take the magnitude of the complex frequency spectrum. The time complexity for this is also N operations. Next, the frequencies are warped according to the mel scale. This step depends on the amount of mel-filters M and their bandwidths. However, the bandwidths of filters vary for different filters, depending on the order in the filter bank, as a function of the filter center frequency. Let L denote the sum of all filter bandwidths $L_1,\ldots,L_M$. The time complexity for mel filtering is approximately L operations, because one mel-frequency coefficient is computed as a sum of multiplication of all frequencies in one interval on filter coefficients [8,20]. In other words, computing of the *i-th* mel-frequency coefficient takes L_I operations and thus, computing of all MFCC takes approximately $L_1+\ldots+L_M=L$. Therefore, we need to resolve L as a function of N. If we look carefully at Figure 2.9, we can see that every filter is exactly covered by its two adjacent filters, or, in other words, the sum of filter bandwidths can be approximated as a two times N. Thus, the time complexity for mel warping is approximately $2 \cdot N$ operations. Finally, discrete cosine transform is taken, for which the time complexity is $M \cdot K$, where K is a number of desired MFCC.

Summing up, the time complexity of computing of the MFCC is approximately $N+N\cdot logN+N+2\cdot N+M\cdot K= N\cdot logN+4\cdot N+M\cdot K$ operations. Dominating parameter in this equation is the frame size N, because, for example, for 8kHz sampling frequency and 20 milliseconds window size N equals to 160, while usually M is set to three times natural logarithm of sampling frequency (usually 29) and K is set to 15.

As discussed in chapter 3, there are two main methods used in computing of linear predictive coefficients, which are then transformed to the cepstrum. Namely, they are autocorrelation and covariance methods. In both methods one matrix equation is solved to find predictive coefficients [20]. The rank of these matrix equals to the number of predictive coefficients and therefore, can be computed in constant time p^2, where p is a prediction order. In both cases matrixes are symmetric with respect to its main diagonal and have only p different elements located in appropriate places [20]. According to the algorithm in [20] first element is computed by N operations, second by $N-1$, and so on. Finally, p-*th* element is computed by $N-p$ operations. Summing up, computing of matrix coefficients takes approximately $N\cdot p+(p+1)/2$ operations. The final step is to compute cepstrum from these coefficients, which also depends only on the number of needed cepstrum coefficients and has approximately $K\cdot(K+1)/2$ operations, where K is a desired amount of cepstrum coefficients. Summing up, the time complexity for computing LPCC is $p^2+N\cdot p+(p+1)/2+ K\cdot(K+1)/2$. More dominating factors here are the frame size N and prediction order. Much expensive part in this computation is computing of autocorrelation coefficients. Prediction order is selected to minimize prediction error and practically it is set to 10-20. Numerical examples for these algorithms are presented in Table 1.

From this table we can see that LPCC can be computed approximately 1.2 times faster than MFCC. Note also, that the mel-scaling greatly increase the speed of computing of the cepstrum coefficients, because after it, significantly lower input size is provided to the final inverse Fourier transform.

Table 5.1 Numerical examples for time complexities of feature extraction algorithms

Sampling frequency	Window size	Vector size	Number of computations (MFCC)	Number of computations (LPCC, prediction order 10)
8kHz	20ms	24	2460	2311
8kHz	20ms	12	2136	1867
16kHz	30ms	24	6891	5511
16kHz	30ms	12	6543	5067

5.2.2 Front-End Optimization

Nevertheless the time complexity analysis made in previous subchapter is quite rough, it shows advantage of LPCC over MFCC. The main reason of this result is that in MFCC computing computationally expensive Fourier transform is used, even though it is computed using the fast algorithm (FFT). However, as discussed in chapter 3, the advantage of MFCC is their more precise characterizing, comparing with the LPCC, of speech signal. On the other hand, these methods are widely used in the speaker identification and good results are reported for both of them [6,44].

In our case of real-time system, this time complexity is not so important, because the problem size (frame size) is relatively small and both of these algorithms are well studied and, in general, no essential improvements can be done for them. We concentrate mostly on the abilities to improve computation speed based on the problem nature. As we know, human's speech does not consist only from connected speech sounds, but there are always some silent regions between them. By removing these parts of speech, we can greatly improve identification speed, because the amount of frames will be reduced. Another approach might be to classify speech frames as a speaker discriminative or not, and use only discriminative.

By silence we mean here the region of speech signal, which does not contain speech information. For instance, it can be pauses between words or sentences, filled by background noise. Examples of silent regions are represented in Figure 5.2.

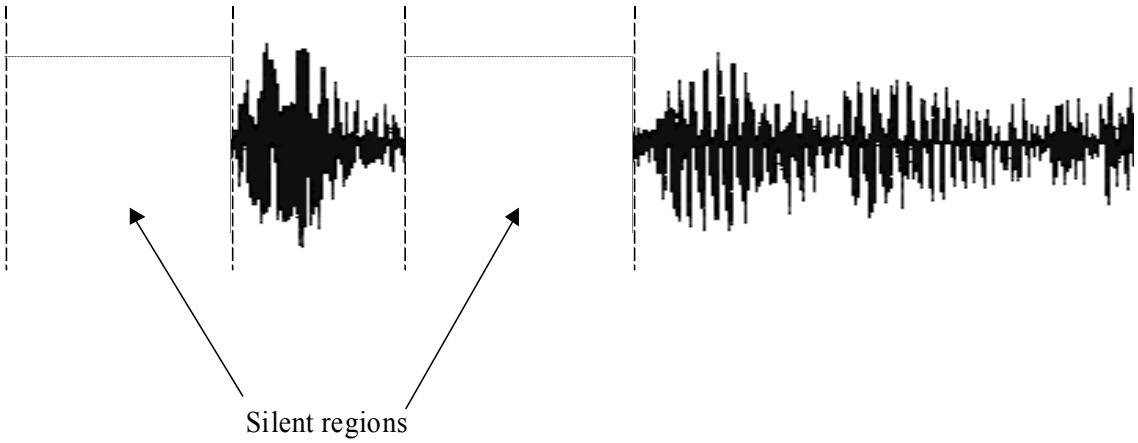

Figure 5.2 Silence detection

Silence detection is usually based on the measuring some signal characteristics, for instance, the following [31]:

- Relative energy level,
- Zero crossing rate,
- First autocorrelation coefficient,
- First LPC linear predictor coefficient,
- First mel-frequency cepstrum coefficient,
- Normalized prediction error.

The easiest method proposed in this work to detect silent regions in speech is based on the computing of variations of the signal samples in speech frame, against the frame mean. If variations are big enough, the frame is considered as a speech frame, otherwise as a silence. Silent region is detected in the following way. First, the mean of the frame samples is computed, then cumulative sum of absolute magnitude of differences between

samples and mean is collected. Then if this sum exceeds predefined threshold the frame is considered as a speech frame, otherwise as a silent frame. This process is represented in the following equation:

$$\mu = \frac{1}{N} \cdot \sum_{n=1}^{N} s_k(n)$$
$$\Omega = \sum_{n=1}^{N} \left| s_k(n) - \mu \right| \qquad \textbf{(5.1)}$$

where $s_k(n)$ are the signal samples for frame k, μ is a mean and Ω is a cumulative sum, compared with the threshold. Although this method is quite simple, our experiments show that it performs good for clean speech and poor for noisy speech. However, the threshold should be set different for different sound recording hardware.

Another algorithm for silence detection is proposed in [4]. In this work, the authors designed a novel *explicit energy-based speech detection algorithm*. It measures the energy of the speech frame and makes decision based on this energy level. Proposed algorithm works in four steps. First, speech is preprocessed by high-pass filtering. Then two energy thresholds are calculated using the following equations:

$$T_1 = E_{\min} \cdot \left(1 + 2 \cdot \log_{10} \frac{E_{\max}}{E_{\min}} \right)$$
$$T_2 = T_1 + 0.25 \cdot \left(SL - T_1 \right) \qquad \textbf{(5.2)}$$

where E_{min} is the minimum energy value, computed as a minimum speech sample value for whole signal, E_{max} is the peak energy value for the speech signal, and SL is the average level of the signal above T_1, computed using the following equation:

$$SL = \frac{\sum_{i} s_k(i)}{\sum_{i} i} \qquad \textbf{(5.3)}$$

where i is the index for all frames having $s_k > T_1$. Using these threshold rough boundaries of speech are estimated. At the third step, these boundaries are refined using *zero-crossing rate*. Finally, silent frames are eliminated. This algorithm yielded good results both in accuracy and identification speed, and therefore, it might be used for real-time speaker identification [4].

In the methods described above, frames are rejected before the feature extraction, or, in other words, independently on the feature extraction algorithm. Another approach is to use extracted features to classify frames as a silent or not. However, since feature extraction is required, they provide less input for computationally expensive matching step, and thus improve identification speed. Such methods are proposed in [18,32]. In [32] two classes, speech and silence are discriminated based on finding of the linear function, which maximizes between classes and minimizes within classes and operates on the MFCC. In [18] cepstral features are used to compute energy of the frame and based on this energy we can decide does this frame contain speech or not.

An approach based on the selection of input frames, which at best contribute to the speaker identification, is proposed in [2]. This method is based on calculating of the likelihoods for the speech frames using Gaussian modeling. Noise removal or other types of speech enhancements can also be used as an optimization technique because analysis of enhanced speech will produce more precise results and therefore, less data will be needed for identification. More details about silence detection and noise removal can be found in [4,16,20,26,31].

5.2.3 Remarks

In this subchapter, we analyzed the two basic methods used in feature extraction from the time complexity point of view. Linear predictive cepstrum showed its advantage over mel-frequency cepstrum in this sense. However, there is a trade-off between the computation speed and the identification accuracy, and thus, the algorithm for feature extraction should be selected based on what is more important in a particular case. We also proposed optimization method, which is based on the classification of the speech as a silent or not. However, there is a trade-off between time, spent on the silence detection and the identification speed up and accuracy. Exact amount of frames, dropped as a silent frames depends on many factors, such as speaking style and speed. Practically, silence removal reduces identification time by 5-10 percents.

5.3 Feature Matching Analysis and Optimization

In this subchapter, we discuss matching step analysis and optimization. Modeling step is left out from discussion, since modeling can be done offline, and it does not affect the real-time identification. Also updating of speaker models after identification can be done independently of speaker, using already recorded data. Also, heuristic optimization for matching, proposed in this work, namely speaker pruning is left in separate chapter.

5.3.1 Analysis of Matching Step in VQ and GMM

Matching step in VQ consists in the computing of quantization distortion between feature vectors and speaker model. In the previous chapter, we described several approaches to its computation. Most of them are based on the finding of the nearest centroid for every feature vector and computing of distortion as a sum of distances between vector and its centroid. Let us assume that after feature extraction step, we have N feature vectors, and M speaker models are enrolled in the system and each speaker model has K

centroids. The number of operations needed for matching step is equal to $N \cdot M \cdot K$, because in order to find nearest centroid for every vector we have to calculate distance to every centroid in the model and select centroid with the smallest distance. Then we should repeat this process for every speaker model. This process is represented in Figure 5.3.

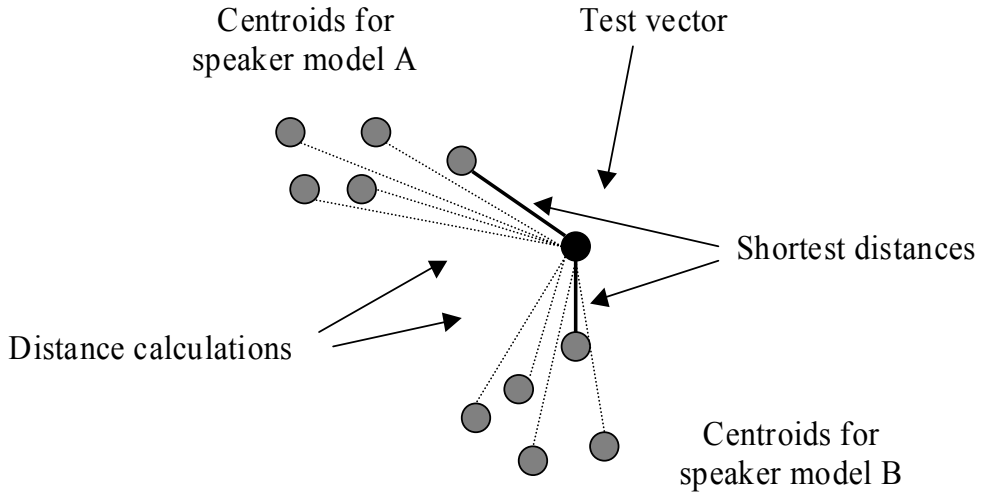

Figure 5.3 Matching of one vector in VQ

However, in real time applications amount of feature vectors is relatively small and can be dropped. The size of feature vectors also affects computation time. The time complexity for distance calculation can be approximated as an $O(p)$, where p is a vector size. Therefore, the result time complexity is $O(M \cdot K \cdot p)$.

In the GMM approach, matching step consists in the computing of probability densities for every feature vectors. Let N denote amount of feature vectors, K amount of densities in the model and M amount of speaker models, enrolled in the system. Again as it was in the VQ case, the time needed for computation of matching score takes $O(N \cdot M \cdot K)$. Because we compute all densities in the model for every feature vector and this process is repeated for every model, enrolled in the system. Computation of model densities depends on the chosen type of covariance matrix. If this matrix is diagonal, then computation of model densities can be approximated as $2 \cdot p$ operations, but in the case of full matrix it takes approximately p^2+p operations. However,

practically only diagonal matrix is used because full matrix requires dramatically greater amount of computations. Summing up, time complexity for GMM matching is $O(M \cdot K \cdot p)$. Again we dropped amount of feature vectors because in real-time case it is relatively small.

The preceding discussion showed that in both cases time complexity can be approximated as $O(M \cdot K \cdot p)$, where M is an amount of models in the system, K is a model size and p is a feature vector dimensionality. In general, according to our analysis, if diagonal matrix is used in GMM for both approaches computation time is almost the same. The main factors that affect computation time are model size and number of enrolled speakers, because computation should be done between every feature vector and every model. However, these approaches to computing matching scores are straightforward and in the rest of this chapter we discuss the possibilities to improve them.

5.3.2 Matching Optimization

Computing of matching score for every enrolled model is the most computationally expensive step in speaker identification. Such approach, when we straightforward compute score for every model and find the best model by searching the minimum matching score, will lead to the identification system with high computational requirements. For instance, as we can see from Figure 5.1 growing population size dramatically decreases identification speed.

For VQ, this problem can be leaded to the more general problem, namely *searching in metric spaces* [7], because it uses some metric functions to compute distances. Using algorithms from this theory, nearest centroid in a codebook can be found more quickly and therefore, overall identification time can be reduced. One of the solutions for fast searching in metric space was presented in [5], which is suitable for discrete-valued distance functions or for functions, which have finite amount of values. They propose a tree, called *Burkhard-Keller tree (BKT)*, which is constructed in the following way. First,

one vector *q* is arbitrary selected from the set of vectors *U* as a root of tree. Then for every distance value *i>0,* a set of vectors $U_i=\{u \in U, d(u,q)=i\}$ which have such distance to the root, is selected and for every non-empty set U_i one child is created. Then this algorithm is repeated for every child until there are more then one vector in any of the children. First step of this algorithm is represented in Figure 5.4.

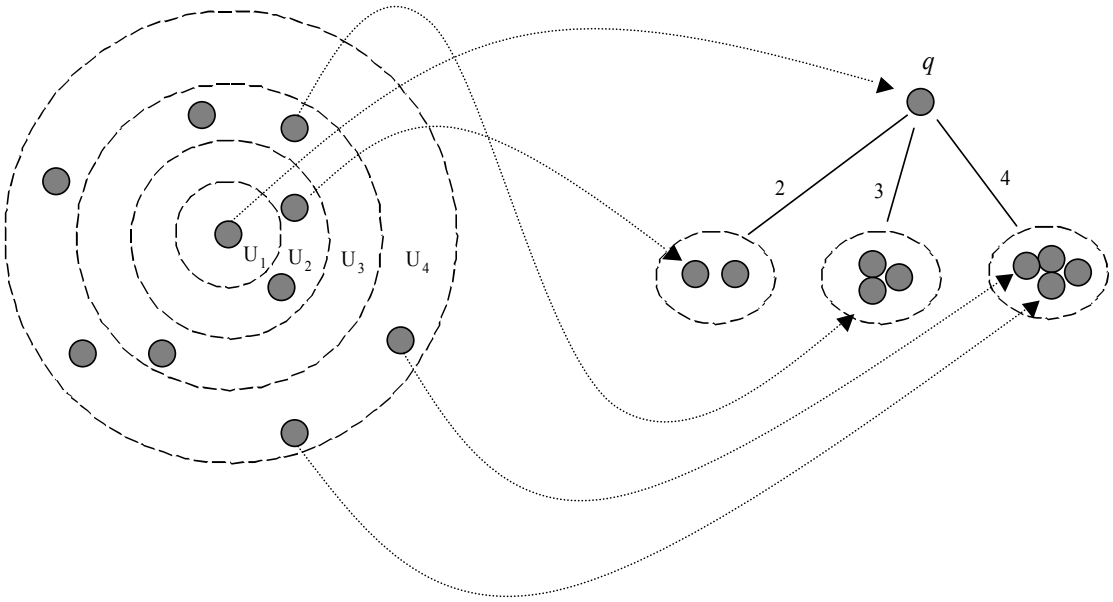

Figure 5.4 Building of Burkhard-Keller tree

In the searching stage with a given vector *t* and a searching radius *r* we calculate the distance to the root *d(q,t)* and enter into the all children *i* such that $d(q,t)-r \le i \le d(q,t)+r$. Each time we arrive to a leaf *u* we calculate *d(u,t)* and if it less then *r* we report the element *u*. The triangular inequality of distance functions ensures that we cannot miss the answer [7]. However, this method is suitable only for discrete-valued distance functions and report vectors that have distance less or equal to *r* from given vector *t*, and therefore, some modification for VQ is required. One approach to such modification proposed in [7] is to start search with zero radius *r* and increase it until at least one vector is found. To move from the continuous distance function to the

discrete-valued it should be quantized using suitably small step. Approach to the building of a tree for searching with the continuous distance function also presented in [7]. They propose binary tree where children are created based on the mean of the distances from the root to all children. Those, which have distance less than mean, are moved to the left child, others are moved to the right child. This process continues recursively for all children, which contain more than one vector. At the searching stage it works as described for Burkhard-Keller tree. More precise overview of techniques used to search in metric spaces can be found in [7].

Another approach to improve identification speed, proposed in this work, is to reduce amount of test vectors by forming a codebook of them using VQ. This method produces centroids for test data, which are used in comparisons with speaker models instead of the whole test set. Such approach is useful for different modeling techniques because it does not depend on the modeling algorithm. Schematically this process is represented in Figure 5.5.

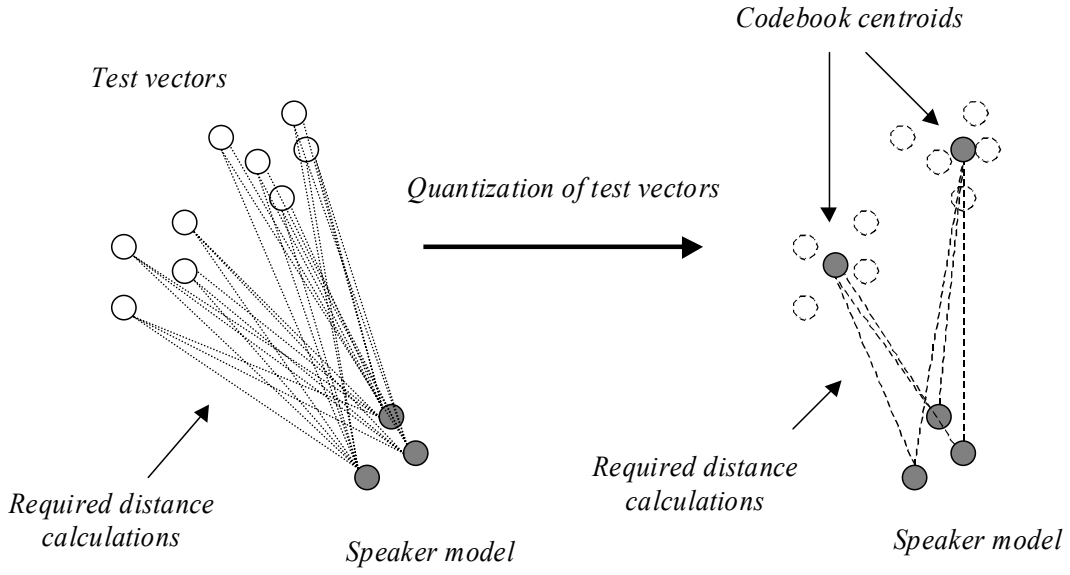

Figure 5.5 Quantization of test data

Based on our experiments this approach greatly reduces computation load for identification system without degrading significantly identification accuracy. However, there is always a trade-off between time, spent on the codebook creation, and time, gained by reducing the amount of vectors. Because codebook generation also takes time, there is some maximum amount of centroids, which can replace the test data without decreasing of identification speed, comparing with direct computation.

For example, well-known clustering algorithm GLA [27] takes approximately $M \cdot N$ distance calculations for one step, where M is a number of clusters, used for quantization of test data, and N is amount of test vectors. Let G be the number of GLA steps and K is the size of speaker model. Summing up, the number of distance calculations, required with quantization of test data, equals to the number of distance calculations needed for codebook generation $M \cdot N \cdot G$ plus $M \cdot K$ calculations for matching quantized data with speaker model. Therefore, it is $M \cdot N \cdot G + M \cdot K$ distance calculations. On the other hand, matching without quantization requires $N \cdot K$ distance calculations. Numerical examples for the number of distance calculations with GLA steps fixed to 3 and model size fixed to 64 are presented in Table 5.2.

From this table we can see that quantization of test data is useful with different amount of test vectors. However, there is always a trade-off between matching speed-up and identification accuracy. For the real time systems this approach can also be useful. Even though the amount of test vectors is relatively small, we can replace few input vectors by one code vector and greatly improve identification speed.

Table 5.2 Comparison of matching with and without quantization

Number of test vectors (N)	Number of clusters (M)	Matching with quantization	Matching without quantization	Speed-up ratio
5	1	79	320	4,05
10	5	470	640	1,36
50	10	2140	3200	1,49
100	20	7280	6400	0,87
200	20	13280	12800	0,96
500	20	21280	32000	1,02
1000	20	61280	64000	1,04
5000	20	301280	320000	1,06

For GMM modeling there are no well-known techniques which can essentially improve identification speed. Because for every model a set of densities should be computed and, for instance, this process does not fit to the mathematical requirements for distance functions and therefore, mathematical apparatus for metric spaces can not be used here.

5.3.3 Remarks

In this subchapter we analyzed two basic techniques for speaker modeling. From the first time analysis, under certain conditions VQ and GMM do not show any great difference in identification speed. However, VQ deals with a quite old and therefore, well studied area in mathematics, namely searching in metric space. And there is a great amount of different techniques, which can reduce number of computation by pre-computing information about speaker models [7]. On the other hand, GMM approach is quite new for speaker identification [41] and different techniques for its speed improvements

are nowadays under investigation. We also proposed our approach, which can reduce amount of comparisons by reducing the number of test vectors.

5.4 Conclusions

In this chapter we discussed known speaker identification techniques from the real-time systems point of view. The main requirement, which is set by the meaning of real-time system, is fast identification time. However, there is always a trade-off between identification speed and accuracy. Based on the analysis in this chapter we can conclude that matching phase in the identification is the most time consuming part, because it requires large amount of comparisons between high-dimensional vectors. For example, from table 5.1 we can see that computation of one 12 dimensional feature vector, even using slowest method and high quality speech, requires about 6543 operations. On the other hand, based on Table 5.2 the matching of this vector requires distance computations between this vector and all speaker models. If we assume for example, that there are only 20 speakers in the database, modeled by 64 codebook centroids each, the amount of computation equals to $20 \cdot 64 \cdot 12 = 15306$ operations. The amount of computation grows rapidly if the size of speaker database increases. Practically, feature extraction takes less than 5-10 percent of time, spent on the identification. Therefore, the main efforts should be leaded on the matching optimization.

In this chapter we also discussed few optimization methods. However, we did not consider a general approaches to speed improvements. Indeed, we discussed methods related only to the area of speaker identification. The main approaches were silence removal and fast searching algorithms in the high-dimensional vector spaces.

Chapter 6

Speaker Pruning

As we discussed in previous chapter, speaker identification requires a great amount of computations, which are mostly distance calculations between test vectors and speaker models. We also discussed some possible ways to improve identification speed, such as silence removal and algorithms for fast searching in metric spaces. In this chapter, we propose our own approach to this problem [23], which can be combined with these two methods. Our method is heuristic, in a sense that it improves identification speed at the cost of a little growing of the probability of incorrect identification. Therefore, there is a trade-off between the running time and identification accuracy.

6.1 Principle of Speaker Pruning

In this work, by *speaker pruning* we mean the continuous process of reducing amount of speaker models involved into the matching step by pruning from it models, to which much probable unknown speaker voice does not belong. The main idea behind this technique is that at the beginning we do not know anything about unknown speaker voice. But when more data comes into the system we can realize what models are close to the unknown speech sample and what models are far away from it and can be ignored in the next computations. This process is illustrated in Figure 6.1. The ellipses represent the speaker models and the "**x**" dots are the feature vectors of unknown speaker. Speaker identification using speaker pruning works in the following way. At the beginning matching function is computed between test vectors and all speaker models. Then, depending on the pruning algorithm, when there is enough data some speaker models are dropped and not anymore used in matching.

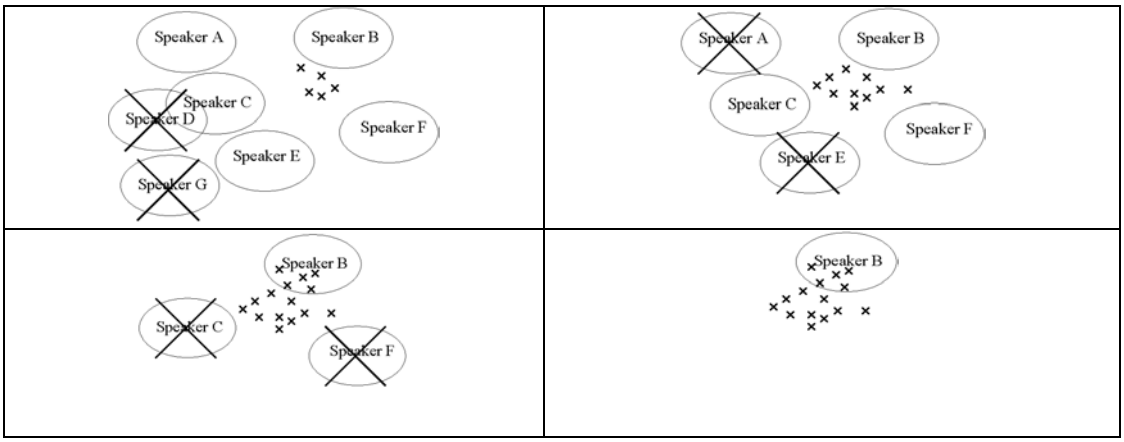

Figure 6.1 Principle of speaker pruning

This process continues until finally only one speaker model is left. If there is not enough test data to finish pruning process, final decision is made between remained speaker models based on the value of the matching function. Following issues should be taken into account during development of speaker pruning algorithm [23]:

- what are the features,
- how speaker models are represented and what is a matching function,
- what is the pruning criterion (when the speaker model should be pruned),
- how many vectors should be extracted prior to next pruning,
- how many speakers are pruned at each iteration.

Let us suppose that feature vectors are extracted and inserted into so called *input buffer* independently of speaker pruning process and identification only deals with input buffer of feature vectors. Speaker pruning is iterative process and depends on two main factors:

1. *pruning interval*,
2. *pruning criterion*.

Pruning interval defines when to check speaker model against pruning criterion, because it is computationally not efficient to check models every feature vector. This factor specifies the amount of vectors taken from the input

buffer before the next pruning iteration. This process is represented in Figure 6.2.

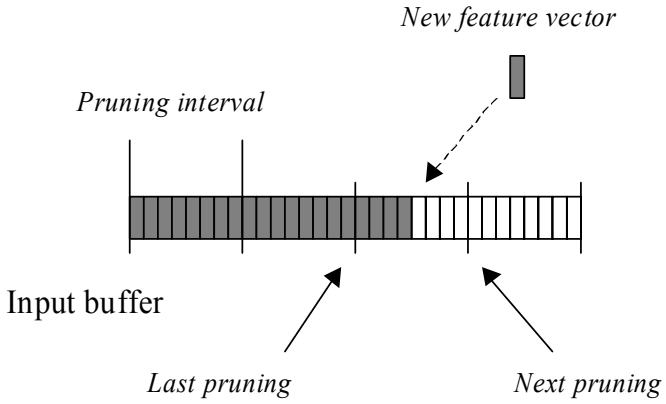

Figure 6.2 Pruning interval

Pruning criterion describes the way in which speaker models are pruned. All speaker models are checked against this criterion and those who meet it are pruned from identification process. Two proposed variants of pruning criterion are discussed in detail in the following subchapters. The following notations are used:

X	Feature vectors of the unknown speaker
C_i	The model of i-th speaker
$D(X,C_i)$	Matching function between vector sequence X and speaker model C_i
M	The number of new vectors read at each iteration
K	The number of pruned speakers at each iteration

It is also supposed that matching function D computes dissimilarity between vector sequence and speaker model. Stochastic matching functions can be in general transformed to this form.

6.2 Static Pruning

In this subchapter we discuss the first proposed variant for speaker pruning.

6.2.1 Principle

The basic idea of *static pruning* is to prune *K* worst speaker models at every pruning interval. To do that, an ordered list of speaker models is maintained. At each iteration, *K* models, which have the smallest matching score, are pruned out from the list. As the new vectors appear in the input buffer, the matching scores are updated and list is sorted. This process continues until only one model is left in the list. Note that, in practice, updating of matching scores can be done fast by using cumulative values of matching function. The pseudocode for static pruning is given in Figure 6.3.

STATIC PRUNING (*M,K*) **RETURNS** speaker model *C*

Let $C=\{C_1,\ldots,C_N\}$ be the set of all speaker models

Let $X=\emptyset$

WHILE (*C* contains more than one model **AND** vectors left in the buffer) **DO**

    Insert *M* new vectors from the input buffer to the set *X*

    Update matching scores for all C_i in *C*

    Remove *K* models with the lowest matching score from *C*

END

RETURN *C*

Figure 6.3 Static pruning

This process is controlled by two parameters. They are pruning interval *M* and number of speaker models pruned at each iteration *K*. One iteration of static pruning is illustrated in Figure 6.4.

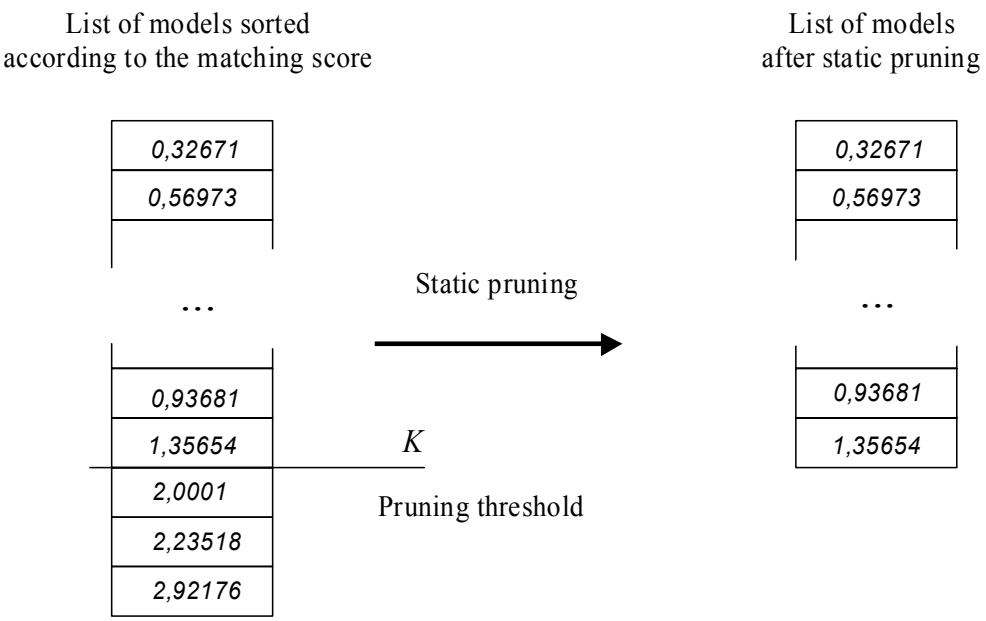

Figure 6.4 One iteration of static pruning

Also the static pruning is a simple algorithm, it shows good results in practice and its implementation is very easy. Its implementation consists only in addition of a special counter for the new feature vectors. When this counter reaches a predefined value *M*, the models are sorted and *K* worst models are pruned.

6.2.2 Complexity analysis

Let us assume that feature vectors have *L* elements and every model has the same size *S*. Then, computing of matching score between one vector and speaker database will take approximately $N \cdot S \cdot L$ operations, where *N* is amount of models in the database. Further, one iteration of static pruning takes $M \cdot S \cdot L \cdot T$, where *T* is amount of models, which are not pruned yet, plus number of operations, needed for model sorting. Simple sorting algorithm, like bubble sort, takes T^2 operations, where *T* is a number of models to sort. More powerful sorting algorithms work faster and take $T \cdot log(T)$ operations, but for simplicity of analysis we use simple algorithm. Let *n* be the current iteration number, then *T* equals to $N-(n-1) \cdot K$, because *K* models are pruned at each

iteration. Therefore, the final equation for the number of operations, needed for one iteration of static pruning, is given by equation 6.1.

$$M \cdot S \cdot L \cdot (N-(n-1) \cdot K)+(N-(n-2) \cdot K)^2 \quad \textbf{(6.1)}$$

The first term in this equation corresponds to the matching of the remained models and the second term corresponds to the sorting of the remained models, based on the computed matching scores. Note that we start sorting only on the second iteration, after we have computed matching scores. Based on this equation, we can compute the cumulative number of operation needed for *n* iterations:

$$\begin{aligned} &\textit{Iteration 1:} && M \cdot S \cdot L \cdot N \\ &\textit{Iteration 2:} && M \cdot S \cdot L \cdot (N-K)+N^2 \\ &\textit{Iteration 3:} && M \cdot S \cdot L \cdot (N-2 \cdot K)+(N-K)^2 \\ &\dots \\ &\textit{Iteration n:} && M \cdot S \cdot L \cdot (N-(n-1) \cdot K)+(N-(n-2) \cdot K)^2 \end{aligned}$$

Summing these numbers of operations, we get the number of operations, made after *n* iterations:

$$M \cdot S \cdot L \cdot N+ M \cdot S \cdot L \cdot N \cdot (N-K)+N^2+\dots+ M \cdot S \cdot L \cdot N \cdot (N-(n-1) \cdot K)+(N-(n-2) \cdot K)^2=$$

$$= M \cdot S \cdot L \cdot (N+N-K+N-2 \cdot K+\dots+(n-1) \cdot K)+N^2+(N-K)^2+\dots+(N-(n-2) \cdot K)=$$

$$=M \cdot S \cdot L \cdot (n \cdot N-n \cdot (n-1) \cdot K/2)+(n-1) \cdot N^2-2 \cdot N \cdot K-4 \cdot N \cdot K-\dots-2 \cdot (n-2) \cdot N \cdot K+$$

$$+K^2+4 \cdot K^2+\dots+(n-2)^2 \cdot K^2=$$

$$= M \cdot S \cdot L \cdot (n \cdot N-n \cdot (n-1) \cdot K/2)+(n-1) \cdot N^2-(n-1) \cdot (n-2) \cdot N \cdot K+(n-1) \cdot (n-2) \cdot (2 \cdot n-3)K^2/6$$

On the other hand, matching without pruning requires $P \cdot S \cdot L \cdot N$ operations, where *P* is a number of test vectors. To compare it with equation (6.1) we fixed

some of the parameters to see how these two methods behave during the time. We compare static pruning and full search by varying the number of feature vectors, available in the input buffer. Iteration number for static pruning can be resolved from the amount of feature vectors in the input buffer *P* as *n* = *P*/*M*. The fixed parameters are *M*=10, *S*=64, *L*=12, *N*=500. The results for two different *K*=10 and *K*=20 are represented in Figure 6.5.

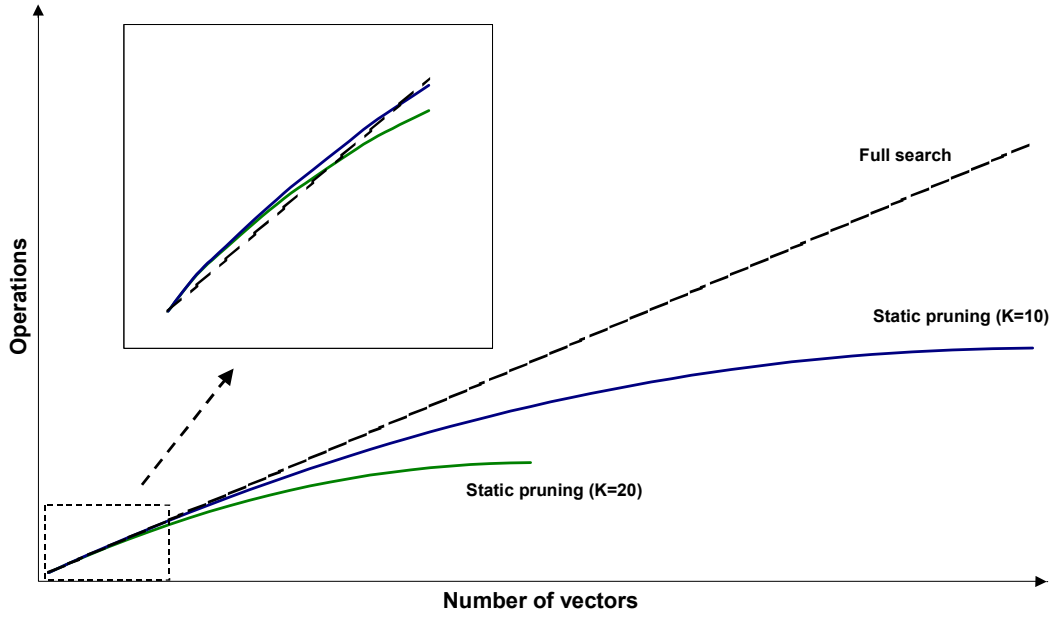

Figure 6.5 Static pruning complexity

From this figure we can see, that with the small amount of test vectors static pruning requires more computations, but when more vectors are available it starts work faster, because some of the models are pruned and are not used in computations. We can also see that pruning parameters also affect computation load. For example, growing number of pruned speakers increase identification speed. Note also, that after *N*/*K* iterations static pruning will stop, because all models except one will be already pruned.

6.3 Adaptive Pruning

In this subchapter we discuss the second proposed pruning variant.

6.3.1 Principle

In the adaptive pruning, the pruning criterion is *data-driven*. This means that the number of speakers pruned at each iteration depends on the current distribution of the matching scores of remained models. Based on the mean value μ and standard deviation σ of the matching score distribution, a pruning threshold θ is set and all models, which have matching score above this threshold are pruned. After pruning, the distribution of matching scores changes, and therefore, the mean and standard deviation must be recomputed. The pseudocode for adaptive pruning is given in Figure 6.6.

ADAPTIVE PRUNING (*M, η*) **RETURNS** speaker model *C*

Let $C=\{C_1,\ldots,C_N\}$ be the set of all speaker models

Let $X= \emptyset$

WHILE (*C* contains more than one model **AND** vectors left in the buffer) **DO**

    Insert *M* new vectors from the input buffer to the set *X*

    Update matching score for all C_i in *C*

    Compute μ and σ of the distribution $\{ D(X,C_i) \mid C_i \in C\}$

    Let $\theta = \mu + \eta \cdot \sigma$ be the pruning threshold

    Remove all models *i* from set C satisfying $D(X,C_i) > \theta$

END

RETURN *C*

Figure 6.6 Adaptive pruning

This adaptive pruning is controlled by two parameters. First is the pruning interval *M,* and the second parameter η determines the “degree” of the thresholding. The larger it is, the less speakers are pruned and vice versa. One iteration of adaptive pruning is represented in Figure 6.7.

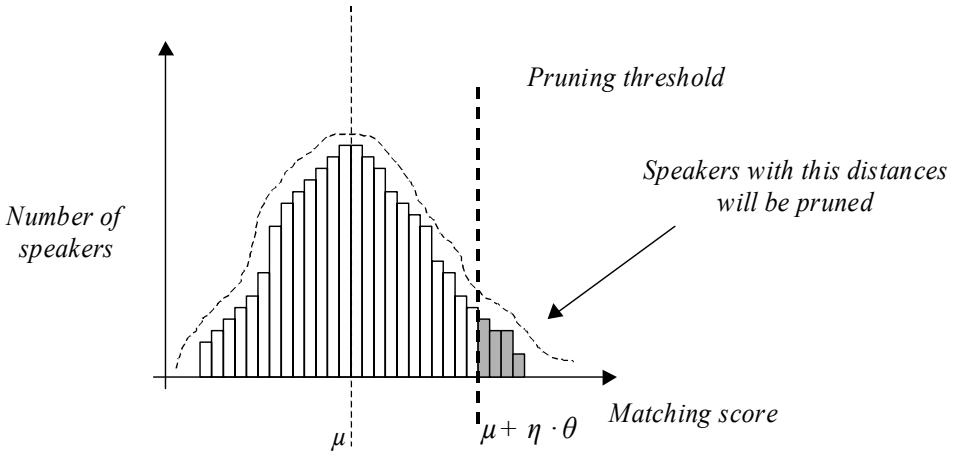

Figure 6.7 One iteration of adaptive pruning

This method has the following mathematical interpretation. In our experiments we found out that distribution of matching scores follows more or less a Gaussian curve. Because of this, the pruning threshold corresponds to the certain *confidence interval* of the normal distribution, and η specifies its width [23]. According to the probability theory, for Gaussian distribution interval $[\mu - \theta, \mu + \theta]$ contains 68 percent of speaker, and interval $[\mu - 2\cdot\theta, \mu + 2\cdot\theta]$ contains 95 percent [33]. For example, if η equals one we prune 16 percent of speakers, or if it equals two we prune 2,5 percent [33]. In the first case the probability that the correct speaker was not pruned is at least 84 percent, and in the second case at least 97,5 percent.

6.3.2 Complexity analysis

Let us assume, that distribution of matching scores follows ideally the Gaussian shape at every iterations, and thus, at every iteration the same percentage of speaker models is pruned. Therefore, at every iteration q percents of remained models are pruned. Let us also assume that feature vectors have L elements and every model has size S. Then calculating of matching score between one vector and speaker database will take approximately $N\cdot S\cdot L$ operations, where N is an amount of models in database.

Further, one iteration of adaptive pruning takes $M \cdot S \cdot L \cdot T$, where T is an amount of models, which are not pruned yet, plus number of operations, needed for threshold calculation. Threshold calculation is simple and takes T operations to find mean and T operations to find standard deviation. However, we do not calculate threshold at first iteration, and therefore, one iteration takes $M \cdot S \cdot L \cdot T+2 \cdot Q$ operations, where Q is a number of not pruned models at previous iteration. To calculate T and Q, let us look at the number of speakers, remained at each iteration:

Iteration 1: $N-q \cdot N$

Iteration 2: $N-q \cdot N-q \cdot (N-q \cdot N)=N- 2 \cdot q \cdot N+q^2 \cdot N$

Iteration 3: $N- 2 \cdot q \cdot N+q^2 \cdot N-q \cdot (N- 2 \cdot q \cdot N+q^2 \cdot N) = N- 3 \cdot q \cdot N+3 \cdot q^2 \cdot N-3 \cdot q^3 \cdot N$

…

Let n be the current iteration number, then based on the Binomial theorem we can calculate the number of remained speakers as a function of n:

$$N \cdot (1-q)^n \qquad \textbf{(6.2)}$$

Based on this equation we can compute the final number of operations for iteration n:

$$M \cdot S \cdot L \cdot N \cdot (1-q)^{n-1}+2 \cdot N \cdot (1-q)^{n-2} \qquad \textbf{(6.3)}$$

Summing these numbers of operations for every iteration, we get the following:

$$M \cdot S \cdot L \cdot N+ M \cdot S \cdot L \cdot N \cdot (1-q)+2 \cdot N+ M \cdot S \cdot L \cdot N \cdot (1-q)^2+2 \cdot N \cdot (1-q)+\ldots+$$

$$+M \cdot S \cdot L \cdot N \cdot (1-q)^{n-1}+2 \cdot N \cdot (1-q)^{n-2}=$$

$$= \sum_{i=1}^{n} (1-q)^{i-1} \cdot \left(M \cdot S \cdot L \cdot N + \frac{2 \cdot N}{1-q} \right) =$$

$$= \frac{1-(1-q)^n}{q} \cdot \left(M \cdot S \cdot L \cdot N + \frac{2 \cdot N}{1-q} \right)$$

Matching without pruning requires $P \cdot S \cdot L \cdot N$ operations, where P is a number of test vectors. To compare it with adaptive pruning, we fixed some of the parameters to see how these two methods behave during the time. We compare adaptive pruning and full search by varying the number of feature vectors, available in the input buffer. Iteration number for adaptive pruning can be resolved from the amount of feature vectors in the input buffer P as $n = P/M$. The fixed parameters are M=10, S=64, L=12, N=500. The results for two different η=1 and η=2 are represented in Figure 6.8. The percentage of speaker models pruned at every iteration q can be resolved from η based on the assumption, that distribution follows Gaussian shape. For η=2 we have q=0.025 and for η=1 we have q=0.16.

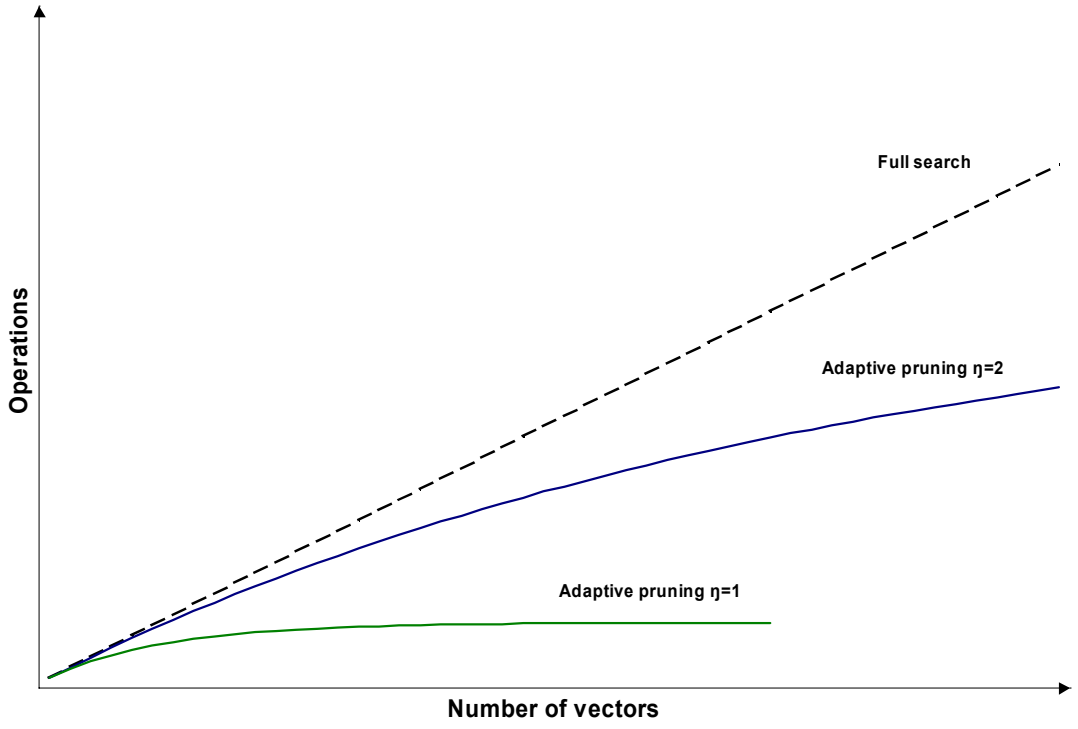

Figure 6.8 Adaptive pruning complexity

From this figure we can see, that even at the beginning adaptive pruning requires less or the same amount of computations. We also can see, that pruning threshold can significantly change computation load. However, it should be chosen accurately to prevent high error rate.

6.4 Discussion

In this chapter, we proposed a speaker pruning algorithm, which is a novel approach to improving identification speed during the matching step. We proposed two pruning variants, static and adaptive. Also we compared these two methods with the full search, and the main conclusion from it is that both algorithms work well and outperform full search, except static pruning is useful only for sufficiently large amount of test vectors. Combining Figure 6.5 and Figure 6.8 we can compare two variants between each other. This comparison is represented in Figure 6.9. From this figure we can see that adaptive pruning is more useful for cases with small amount of test vectors. However, for large number of vectors static pruning outperforms adaptive, because it prunes every iteration the same number of speakers, whereas adaptive variant prunes less and less speakers. It is also the reason why static pruning stops earlier than adaptive. However, the analysis presented in this chapter is only theoretical and it does not tell us anything about identification error rate. It only shows us the computational load for different parameters and algorithms dynamics over time. Note also, that analysis for static pruning is exact, whereas analysis for adaptive variant is made under certain assumptions and therefore, practical justification is needed. In the next chapter we present experiments on the real data for both variants.

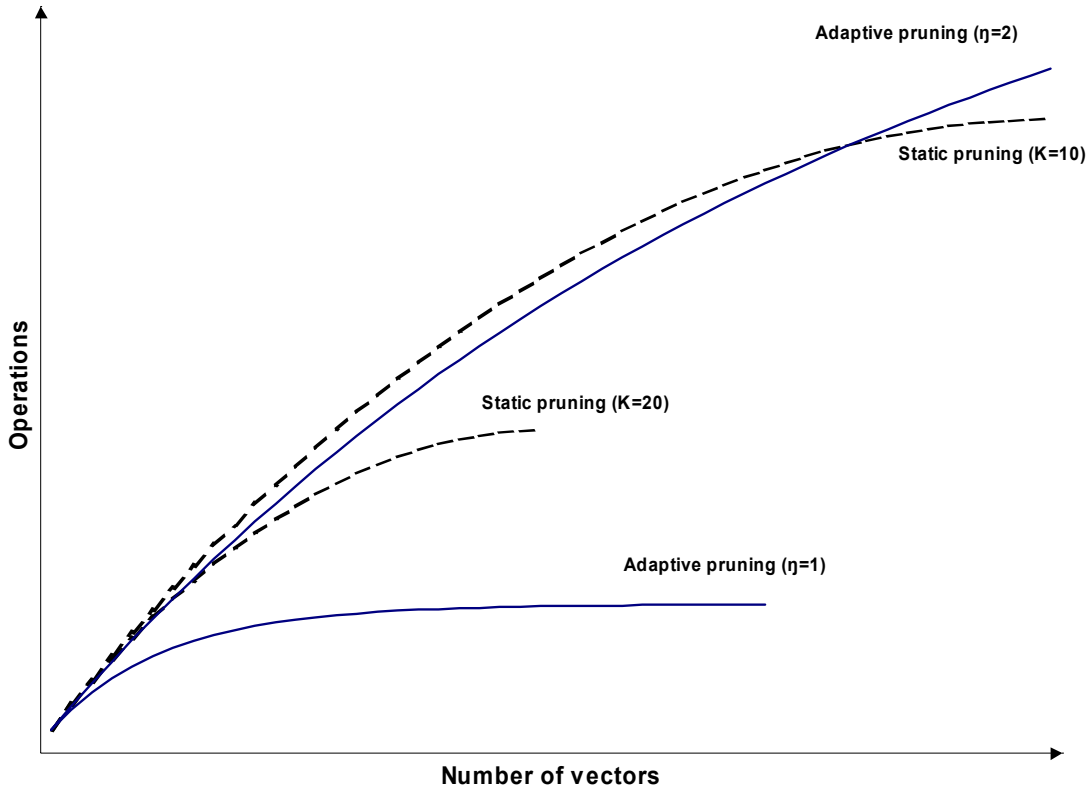

Figure 6.9 Complexity of static and adaptive pruning

For future work, we plan to extend the algorithm to use time-depended values for parameters, which are controlling the pruning. For example, pruning interval should be initially large to give ability to stabilize for matching scores, and then it should be gradually minimized to make the identification faster. Pruning threshold can also be extended. It can be based on probability that correct speaker already has the minimum matching score. When it is high enough the pruning threshold can be increased to prune more speakers and vice versa.

Chapter 7

Experiments

In this chapter, we report the results from our experiments. In order to evaluate empirically speaker pruning algorithms proposed in the previous chapter, we created appropriate speaker identification software. First, we present experiments, which show the underlying basis for pruning. Then we show how the proposed algorithms improve identification speed. Finally, we show the comparisons result between two proposed variants of speaker pruning and also discuss the possible reasons for the results.

7.1 Experiments conditions

In our experiments, we used feature vectors composed from 12 lowest mel-frequency cepstral coefficients (MFCC) computed using 27 mel-spaced filters. The 0-th coefficient was excluded, because it carries a little of speaker specific information. Analysis frame was windowed by 30 milliseconds Hamming window with 10 milliseconds overlapping. The signal was pre-emphasized by the filter $H(z)=1-0.97 \cdot z^{-1}$ and silence frame were removed before the feature extraction. All sample durations in these experiments refer to the silence-removed speech. For speaker modeling we used vector quantization (VQ) as described in subchapter 4.1. All speakers were modeled by a codebook of 64 vectors using GLA [27] as the clustering algorithm. The matching function was the mean squared error between the test vector sequence and speaker codebook. The decision was simply the speaker with minimum error.

As a test materials for our experiments we used the American English TIMIT corpus [28]. The length of training data (silence removed speech) was on average 8.8 seconds. Speech data was sampled using 8 kHz sampling frequency and 16-bit resolution. All speech samples are recorded under noisy-free laboratory conditions. For tests we used different workstations, equipped

with Pentium 4 processor, 256 megabytes of memory and Windows XP operating system.

7.2 Results

In this subchapter, we show the results of our experiments. Every chart is preceded by the short explanation what we measured and why and followed be the short discussion about results.

7.2.1 Pruning basis

First, we start from the basis for speaker pruning. We ran a few tests to see how the matching function behaves during the identification and when it stabilizes. Chart in Figure 7.1 represents the variation of matching score depending on the available test vectors for 20 different speakers. One vector refers to the one analysis frame.

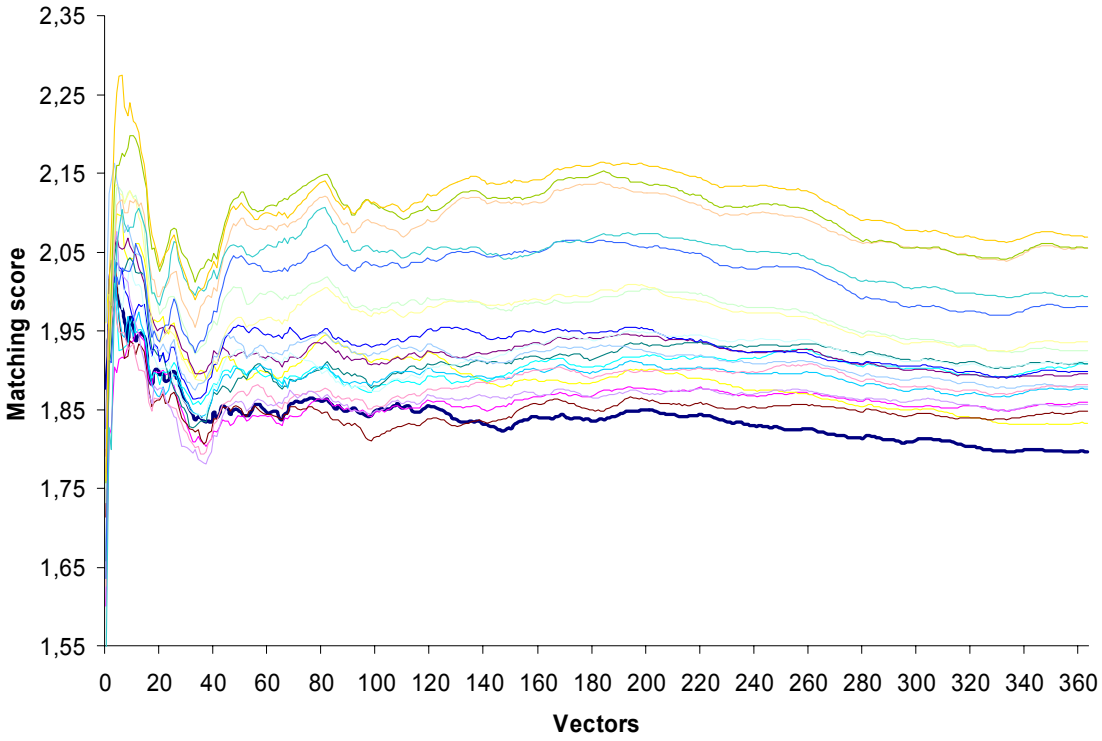

Figure 7.1 Distance stabilization

In this figure, the bold line represents the owner of the test sample. It can be seen that at the beginning the matching score for correct speaker is somewhere among the other scores. But after large enough amount of new vectors extracted from the test speech, it becomes close with only few scores and at the end it becomes the smallest score. Actually, this is the underlying reason for speaker pruning: when we have more data we can drop some of the models from identification.

In the next test, we analyze how much vectors we need to identify speaker correctly. We ran identification tests for 100 and for all 630 speakers from TIMIT corpus and marked the amount of vectors where the system started to identify speaker correctly. Then we transformed these values to the error rate in the following way. For every amount of feature vectors we calculated what is the identification error rate if we use only this amount of vectors. The result of these tests is represented in Figure 7.2.

From this figure we can see that for TIMIT corpus, about 90 percent of speakers are already identified after approximately 120 feature vectors for 630 speakers and 50 vectors for 100 speakers. This corresponds to the test sample of lengths about 2,5 and 1 second of non-silent speech respectively. This is another reason of pruning, because the curves decrease very fast and it means that correct speaker is usually found after surprisingly small amount of test data. Another observation is that for small speaker database less number of test vectors is required to identify speakers.

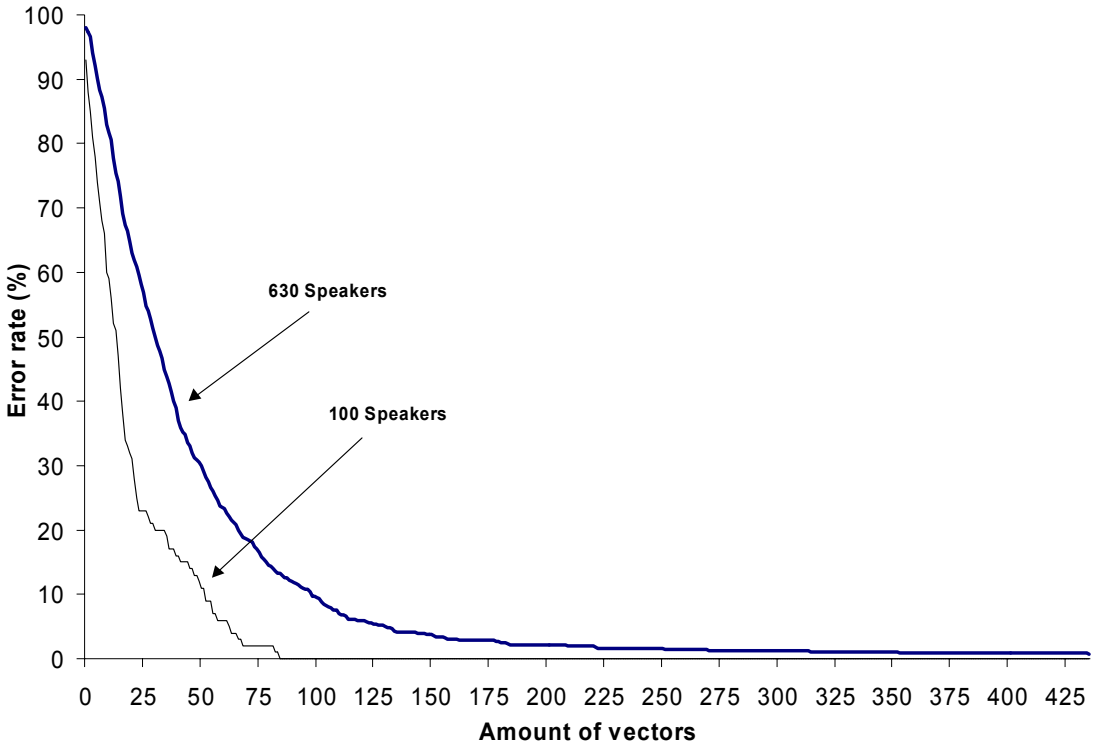

Figure 7.2 Identification error rate depending on amount of test data

7.2.2 Static pruning

In the next experiment, we consider the trade-off between identification error rate and average time spent on the identification for static pruning. By varying the pruning interval or number of pruned speakers we expect different error rates and different identification times. From several runs with different parameter combination we can plot the error rate as a function of average identification time. To obtain such dependency, we fixed three values for pruning interval and were varying the number of pruned speakers. The results are shown in Figure 7.3.

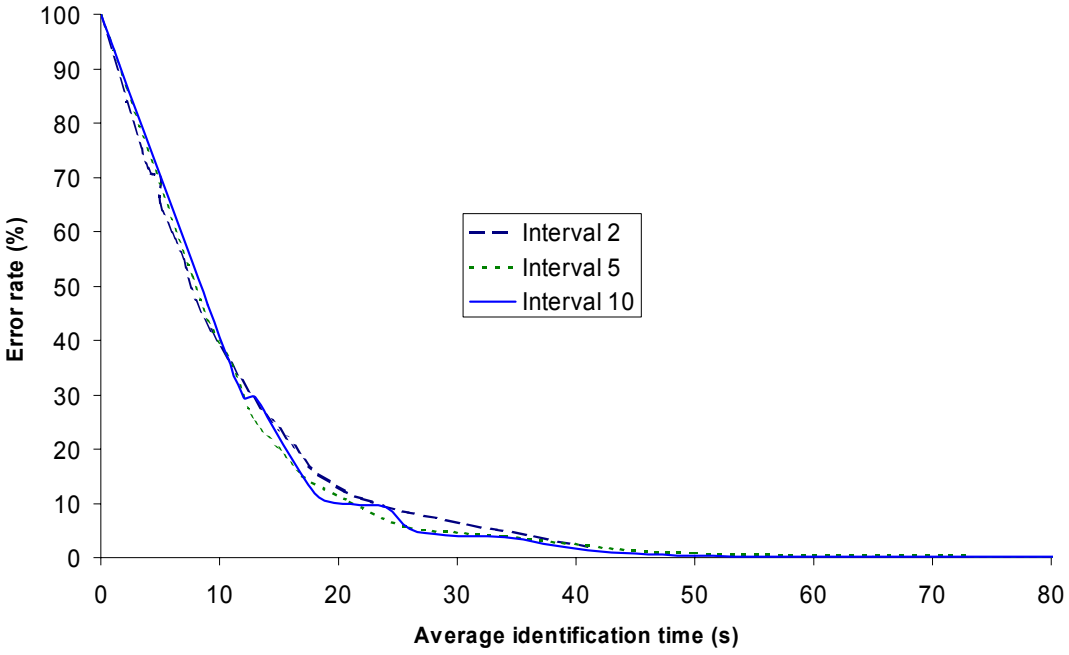

Figure 7.3 Evaluation of static variant using different pruning intervals

From this figure we can see that all curves follow almost the same shape. It is because in order to have fast identification we have to choose either small pruning interval or large number of pruned speakers. On the other hand, in order to have low error rate we have to choose large interval or small number of pruned speakers. The main conclusion from this figure is that these two parameters compensate each other.

7.2.3 Adaptive pruning

The idea of adaptive pruning is based on the assumption that distribution of matching score follows more or less the Gaussian curve. In Figure 7.4 we can see the distributions of matching scores for two typical identifications.

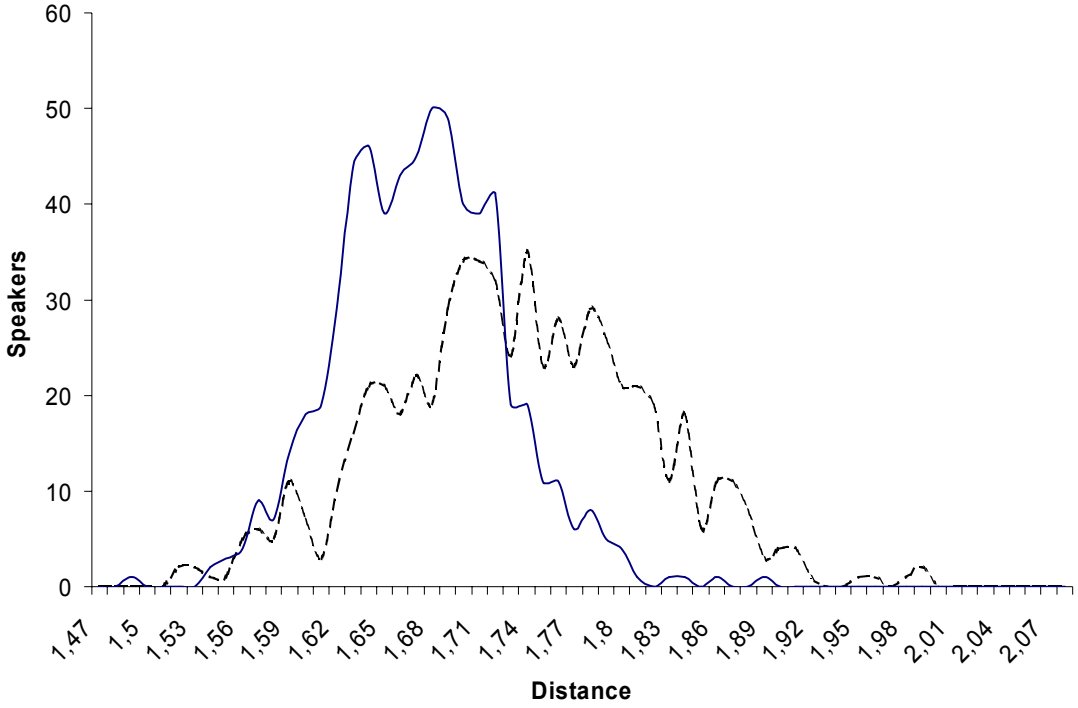

Figure 7.4 Examples of the matching score distribution

From this figure we can see that distribution is not exactly follows the Gaussian curve but its shape is almost the same. In the next experiment, we consider the trade-off between identification error rate and average time spent on the identification for adaptive pruning. By fixing the parameter η and varying the pruning interval we obtained desired dependency. The results are shown in Figure 7.5.

From this figure we can see that parameter η has some minor effect only for small identification times. For high identification times, curves do not show significant differences. Again, as in the case of static pruning, algorithm parameters compensate each other. This means, that for fast identification, we have high error rate, whereas for low error rate we have relatively slow identification speed. Algorithm parameters have only a minor effect.

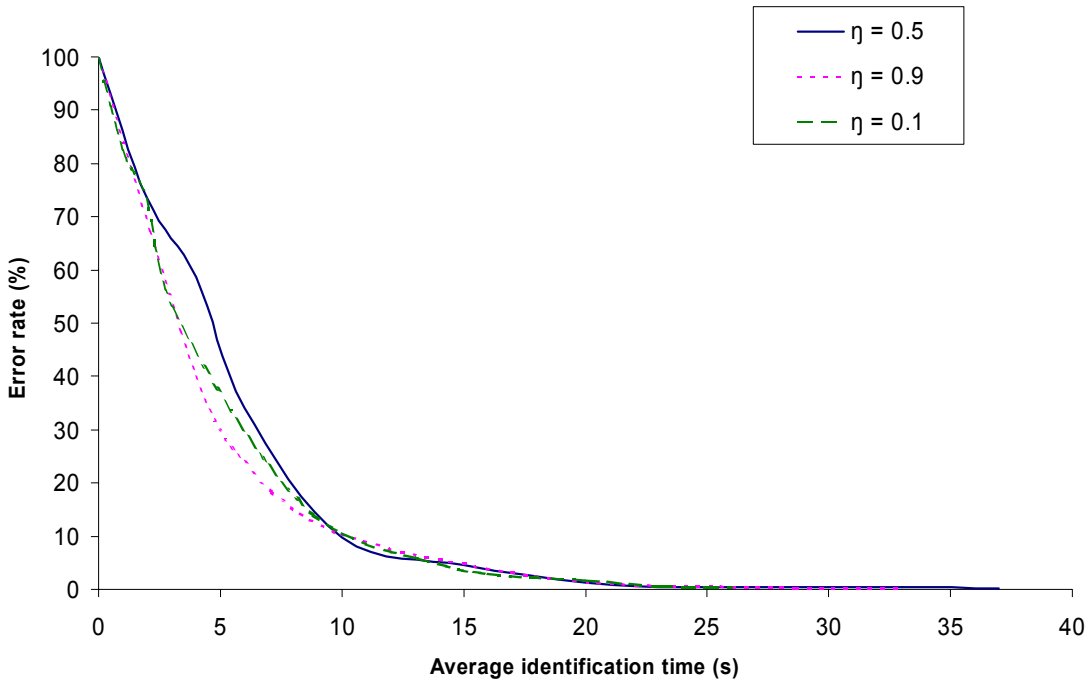

Figure 7.5 Evaluation of adaptive variant using different values for *η*-parameter

We also were interested how the distribution and pruning threshold in the adaptive pruning variant change over the time after few iterations of pruning. This observation is represented in Figure 7.6.

From this figure we can see an example of how the distributions and thresholds change over time. Our general observation is that the variance of the matching scores decreases with time, because after pruning, models with large distances are pruned and only models, which are close to the unknown speaker, are remained.

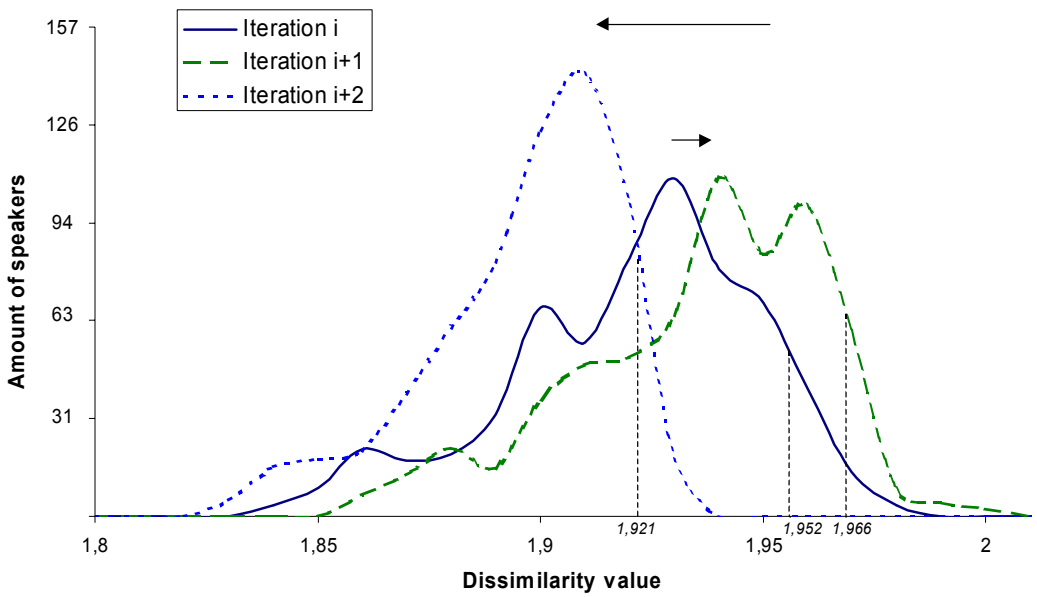

Figure 7.6 Moving of the matching score distribution and the pruning threshold over time

7.2.4 Comparisons

The results for best parameter combinations for the static and adaptive variants are shown in Figure 7.7.

We can easily see from this figure that adaptive variant works better in general. It reaches lower error rate with the same identification time. For example, with error rate of 0.46 percent the adaptive variant works in 24 seconds of speech, whereas the static method spends over 60 seconds to reach the same error rate.

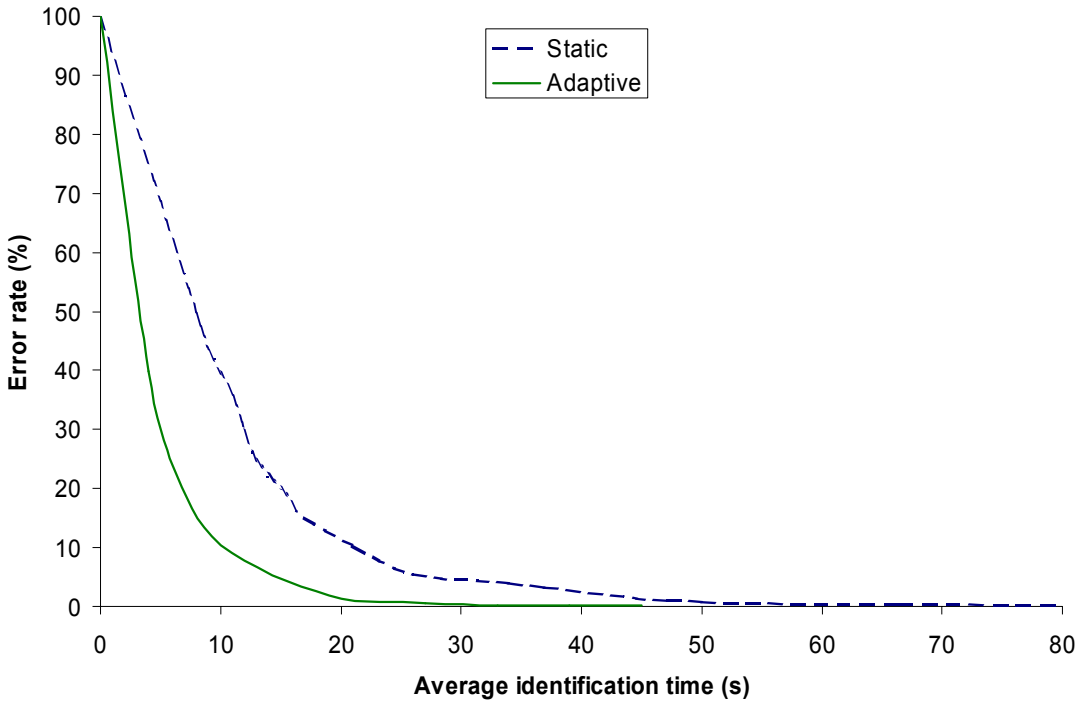

Figure 7.7 Comparison of the static and adaptive pruning

We also ran few tests with larger pruning intervals and η parameter (up to M = 30 and η = 2.0). However, these results were poor and these tests are not included in this work. We also ran the full test for all 630 speakers without any pruning. In this case, we reach identification time on average 230 seconds with 0.15 percent error rate. Therefore, the speed-up is significant in both static and adaptive cases. The high identification rate is due to the fact that TIMIT was recorded in noise-free laboratory conditions.

7.3 Discussion

In this chapter, we presented our experiments on proposed pruning algorithms. First, we presented observations, which gives us the motivation for pruning algorithms. Then we tested both variants on the TIMIT speech corpus. The main result is that adaptive pruning outperforms the static variant on every

case. We also studied the influence of the parameters on the algorithm performance and concluded that in both cases there is no significant difference what values to choose for parameters in a sense of fixed error rate or identification speed. In other words, if we need certain error rate we can choose different sets of parameters to reach it but the difference between achieved identification speeds will be not significant, and vice versa. Note also, that in previous chapter we made theoretical analysis for pruning variants, which shows us the computational load for different parameters. Whereas in this chapter we made practical analysis, which shows us the relationship between the identification speed and error rate, and therefore, is more reliable.

Chapter 8

Conclusions

In this work, we studied and analyzed different techniques for speaker identification. In the first part, we started from the identification background, which is based on the digital signal theory and modeling of the speaker vocal tract. Then we discussed various techniques for reducing amount of test data or feature extraction. Further, we studied most popular speaker modeling methods, which are commonly used in the speaker identification. In the second part, we studied techniques, discussed in the previous part, from the real-time systems point of view. We proposed different optimization approaches to the speaker identification. However, we discussed only methods related to the speaker identification area, and left out from discussion general optimization methods.

We proposed a speaker pruning as a novel approach to reducing amount of distance calculations in the matching step. This method is heuristic, and therefore, improves identification speed at the cost of increasing of the probability of incorrect identification. We proposed two variations of the pruning algorithm and made approximate time complexity analysis for this methods and concluded that it significantly improves matching step. Finally, we studied speaker pruning empirically and found out that theory analysis was correct and it really improves identification speed. We also compared different parameter combinations for both variants of speaker pruning.

From this work we can conclude that in speaker identification process matching between test vectors and speaker models is the most time consuming part. It takes about 90 percent of all time spent on the identification. Therefore, optimization efforts should be concentrated on the matching optimization. Based on our experiments and theoretical analysis, we can also conclude that proposed speaker pruning is useful in practice. For instance, the error rate of 0.46 percent can be reached using adaptive pruning

in 24 seconds, whereas for full search we reached error rate of 0.15 percent in 230 seconds. These two results in general can not be compared because using pruning we reach result faster but the full search is more accurate. Therefore, pruning should be used in applications where the identification time is more important. We also proposed future directions for improvements of speaker pruning algorithm. We plan to extend the algorithm to use time-depended values for parameters, which are controlling the pruning. This is the topic for future research.

List of References

[1] B. S. Atal, “Automatic Recognition of Speakers from their Voices”, *Proceedings of the IEEE*, vol 64, 1976, pp 460 – 475.

[2] L. Besacier, J.F. Bonastre, “Frame Pruning for Speaker Recognition”, *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference*, Vol. 2, pp. 765-768.

[3] Z. Bin, W. Xihong, C. Huisheng, “On the Importance of Components of the MFCC in Speech and Speaker Recognition”, *Center for Information Science, Peking University*, China, 2001.

[4] D. Burileanu, L. Pascalin, C. Burileanu, M. Puchiu, “An Adaptive and Fast Speech Detection Algorithm”, *Proc. TSD 2000 - Third International Workshop on Text, Speech and Dialogue*, Brno, Czech Republic, September 13-16, 2000.

[5] W. Burkhard and R. Keller, “Some approaches to best-match file searching”, *Comm. Of the ACM*, 16(4):230-236, 1973.

[6] J.P. Campbell, “Speaker Recognition: A Tutorial”, *Proc. of the IEEE*, vol. 85, no. 9, Sept 1997, pp. 1437-1462

[7] E. Chavez, G. Nevarro, R. Bayeza-Yates, J. Marroquin, “Searching in Metric Spaces”, *ACM Computing Surveys (CSUR)* September 2001 Volume 33, pp. 273-321.

[8] J. R. Deller, J. H. L. Hansen, J. G. Proakis, *Discrete-Time Processing of Speech Signals*, Piscataway (N.J.), IEEE Press, 2000.

[9] M. Do, M. Wagner, “Speaker Recognition with Small Training Requirements Using a Combination of VQ and DHMM”, *Proc. of Speaker Recognition and Its Commercial and Forensic Applications*, pp. 169-172, Avignon, France, April 1998.

[10] H. Ezzaidi, J. Rouat, D. O’Shaughnessy, “Towards Combining Pitch and MFCC for Speaker Identification Systems”, Aalborg, *Eurospeech 2001* – Scandinavia.

[11] T. Filho, R. Messina, E. Cabral, “Learning Vector Quantization in Text-Independent Automatic Speaker Identification”, *5-th Brazilian Symposium on Neural Networks* December 09 - 11, 1998 Belo Horizonte, MG, Brazil, pp. 135-139.

[12] P. Fränti, T. Kaukoranta, O. Nevalainen, “On the Splitting Method for Vector Quantization Codebook Generation”, *Optical Engineering*, 36 (11), pp. 3043-3051, November 1997.

[13] P. Fränti, J. Kivijärvi, “*Randomized Local Search Algorithm for the Clustering Problem*”, *Pattern Analysis and Applications*, 3 (4), 358-369, 2000.

[14] S. Furui, *Digital Speech Processing, Synthesis and Recognition,* New York, Marcel Dekker, 2001.

[15] S. Furui, “Vector-Quantization-Based Speech Recognition and Speaker Recognition Techniques”, *IEEE Signals, Systems and Computers*, 1991, Volume 2, pp. 954-958.

[16] N.R. Garner, P.A. Barrett, D.M. Howard, A.M. Tyrrell, “Robust Noise Detection for Speech Detection and Enhancement”, *IEEE Electronic Letters* 13-th February 1997, Vol. 33, No 4, pp. 270-271.

[17] H. Gish and M. Schmidt, “Text Independent Speaker Identification”, *IEEE Signal Processing Magazine*, Vol. 11, No. 4, 1994, pp. 18-32.

[18] J.A. Haigh, J.S. Mason, “Robust Voice Activity Detection using Cepstral Features”, *Computer, Communication, Control and Power Engineering*. Proceedings. TENCON '93, 1993 IEEE Region 10 Conference, Part: 30000 , 1993, Vol. 3, pp. 321-324

[19] P. Hedelin and J. Skoglund, “Vector quantization based on Gaussian mixture models”, *IEEE Transactions on Speech and Audio Processing*, Vol. 8, No 4, July 2000, pp. 385-401.

[20] X. Huang, A. Acero and H.-W. Hon, *Spoken language processing*, Upper Saddle River, New Jersey, Prentice Hall PTR, 2001.

[21] M. C. Huggins, J. J. Grieco, “Confidence Metrics for Speaker Identification”, *ICSLIP 2002*, Denver, pp. 1381-1384

[22] T. Kinnunen and P. Fränti, “Speaker Discriminative Weighting Method for VQ-Based Speaker Identification”, *Proc. 3rd International Conference on audio- and video-based biometric person authentication (AVBPA))*, pp. 150-156, Halmstad, Sweden, 2001.

[23] T. Kinnunen, E. Karpov, P. Fränti, “A speaker pruning algorithm for real-time speaker identification”, *submitted to ICASSP 2003*.

[24] T. Kinnunen, T. Kilpeläinen, P. Fränti, “Comparison of Clustering Algorithms in Speaker Identification”, *Proc. IASTED Int. Conf. Signal Processing and Communications (SPC 2000)*, pp. 222-227, Marbella, Spain, 2000.

[25] T. Kinnunen, I. Kärkkäinen, “Class-Discriminative Weighted Distortion Measure for VQ-based Speaker Identification”, *Springer-Verlag Berlin Heidelberg 2002*, *Volume 2396, pp 681*-688.

[26] L. Liao, M. Gregory, “Algorithms for Speech Classification”, *ISSPA* 1999, Brisbane, Australia.

[27] Y. Linde, A. Buzo, R. Gray, “An algorithm for Vector Quantizer Design”, *IEEE transactions on Communications*, Vol. 28 (1), 84-95, January 1980.

[28] *Linguistic Data Consortium*, http://www.ldc.upenn.edu/

[29] J. W. S. Liu, *Real-time systems*, Upper Saddle River, (N.J.), Prentice Hall, 2000.

[30] V. Mantha, R. Duncan, Y. Wu, J. Zhao, A. Ganapathiraju, J. Picone, “Implementation and Analysis of Speech Recognition Front-Ends”, Southeastcon '99. *Proceedings. IEEE*, 1999, pp. 32-35.

[31] J. Marks, “Real Time Speech Classification and Pitch Detection”, *IEEE Communications and Signal Processing*, 1988. Proceedings., COMSIG 88. Southern African Conference, pp. 1-6.

[32] A. Martin, D. Charlet, L. Mauuary, “Robust Speech/Non-Speech Detection using LDA applied to MFCC”, *IEEE Acoustics, Speech, and Signal Processing*, 2001 IEEE International Conference, Vol. 1, pp. 237-240.

[33] J. S. Milton, and J. C. Arnold , *Introduction to Probability and Statistics*, Singapore, McGraw-Hill International Edition, 1990

[34] S. Molau, M. Pitz, R. Schluter, H. Ney, “Computing Mel-Frequency Cepstral Coefficients on the Power Spectrum”, *Acoustics, Speech, and Signal Processing*, 2001 IEEE International Conference, Volume: 1, 2001, pp. 73-76

[35] J. M.Naik, “Speaker Verification: A Tutorial”, *IEEE Communications Magazine*, January 1990, pp.42-48.

[36] S. Ong, S. Sridharan, Cheng-Hong Yang, Miles Moody, “Comparison of Four Distance Measures for Long Time Text-Independent Speaker Identification”, *ISSPA*, 1996, pp. 369-372

[37] S. Ong, M. Moody, S. Sridharan, “Confidence Analysis for Text-Independent Speaker Identification: Inspecting the Effect of Population Size”, *IEEE International Symposium on Speech, Image Processing and Neural Networks*, April 1994, Hong Kong, pp. 611-613.

[38] J. G. Proakis and D. G. Manolakis, *Digital Signal Processing, Principles, Algorithms, and Applications*, New York, Macmillan Publishing Company, 1992.

[39] L. Rabiner and B.-H. Juang, *Fundamentals of Speech Recognition*, Englewood Cliffs (N.J.), Prentice Hall Signal Processing Series, 1993.

[40] D. A. Reynolds, “An Overview of Automatic Speaker Recognition Technology”, *ICASSP* 2002, pp 4072-4075.

[41] D. Reynolds, R. Rose, “Robust Text-Independent Speaker Identification Using Gaussian Mixture Speaker Models”, *IEEE transactions on speech and audio processing*, Vol. 3, No1, 1995, pp. 72-83

[42] D. A. Reynolds, “Experimental Evaluation of Features for Robust Speaker Identification”, *IEEE Transactions on Speech and Audio Processing*, Vol. 2, No 4, October 1994, pp. 639-643.

[43] L. Rigazio, P. Nguyen, D. Kryze, J.-C. Junqua, “Separating Speaker and Environment Variabilities for Improved Recognition in Non-Stationary Conditions”, *Eurospeech 2001 – Scandinavia*.

[44] D. O’Shaughnessy, ” Linear Predictive Coding”, *IEEE Potentials* -- Vol. 7, 1988, no. 1, p. 29-32.

[45] S. W. Smith, *The scientist and Engineer’s Guide to Digital Signal Processing*, California Technical Publishing, 1999, http://www.dspguide.com (was valid at 20.12.2002)

[46] F. K. Soong, A. E. Rosenberg, L. R. Rabiner and B. H. Juang, “A Vector Quantization Approach to the Speaker Recognition”, *AT&T Technical Journal*, Vol. 66, pp. 14-26, Mar/Apr 1987.

[47] R.Stapert, J. Mason, “A Segmental Mixture Model for Speaker Recognition”, *Eurospeech 2001 – Scandinavia*.

[48] S. Theodoridis, K. Koutroumbas, *Pattern recognition*, San Diego, Academic Press, 1999

[49] S. Umesh, L. Cohen, D. Nelson, “Fitting the Mel Scale”, *Acoustics, Speech, and Signal Processing*, 1999 IEEE International Conference, Volume: 1, 1999, pp. 217 –220.

[50] R. Vergin, D. O'Shaughnessy, "Pre-Emphasis and Speech Recognition", *Electrical and Computer Engineering*, 1995. Canadian Conference, Volume: 2, pp. 1062-1065.

[51] C. Vivaracho, J. Ortega-Garcia, L. Alonso, Q. Moro, "A Comparative Study of MLP-Based Artificial Neural Networks in Text-Independent Speaker Verification against GMM-Based Systems", *Eurospeech 2001 – Scandinavia*.

[52] N. J.-C. Wang, W.-H. Tsai, L.-S. Lee, "Eigen-MLLR Coefficients as New Feature Parameters for Speaker Identification", *Eurospeech 2001 – Scandinavia*.

[53] X. Yue, D. Ye, C. Zheng, X. Wu, "Neural Networks for Improved Text-Independent Speaker Identification", *IEEE Engineering in Medicine and Biology Magazine*, Vol. 22, 2002, pp. 53-58.