POLITECNICO DI MILANO
Scuola di Ingegneria dell'Informazione

POLO TERRITORIALE DI COMO

**Master of Science in
Computer Engineering**

Classification of Microphones of Mobile Devices via Blind Channel Estimation

**Supervisor:
Prof. Marco Tagliasacchi**

**Assistant Supervisor:
Patrick Aichroth**

**Master Graduation Thesis by:
Luca Cuccovillo
Student Id. Number 764873**

Academic Year 2011-2012

to my family, for their endless love, support and trust,
to my friends in Italy, that despite the distance were always close to me

to the efforts spent so far and to the challenges that make the life so intriguing

Abstract

The present work concerns microphone classification, a research topic still in its early stages strictly related with audio authenticity and forgery detection. In this paper we propose a new method of microphone classification based on a recent single-microphone blind channel estimation algorithm, focusing on audio recordings from mobile devices: this method aims to correctly identify the source device with high accuracy, even if the sample audio file was compressed at extremely low bit-rates. A Support Vector Machine (SVM) was used in order to perform closed-set identification experiments, and to assess the performance of the algorithm.

Acknowledgments

I would like to acknowledge and express a special thank to my thesis advisor and professor, Marco Tagliasacchi, for his assistance throughout the work, and for believing in me from the start. I would also like to acknowledge Patrick Aichroth and Christian Dittmar for inviting me in Germany, for their advises, help and constant support. Being in Germany has been both a motivation and an encouragement: somewhere else, and without the collaboration with the Fraunhofer Institute for Digital Media Technology all of this would not have been possible.

It is my very pleasure to extend a very special thank to the entire team of the Media Management and Delivery group of the Fraunhofer IDMT, that supplied me with every resource that I needed in order to conduct my research. I spent a very pleasant time with them: their kindness, their patience and their friendliness are priceless. In particular a sincere thank goes to Peggy Walther and Patrick Rönsch, for their help during the recording sessions.

Last, but certainly not least, I am pleased to extend an heartfelt thank to Sebastian Mann, for his valuable and continuous presence. Sebastian helped me constantly, provided me with comments and suggestions, and thanks to him I was able to enjoy even more the time spent at the Fraunhofer.

I could not have completed this thesis without each and every one of the peoples mentioned here, and for that I'm grateful beyond words.

Contents

List of Figures

Chapter 1

Introduction

During the last years multimedia contents have spread all over the world: the duplication of a digital object has become quite a straightforward procedure, as well as its coding and decoding, transmission, editing and storage. Although some sources provide us with authentic information, many others contain forged content: concerns regarding how to validate multimedia data constantly arise, highlighting the need of methods and tools to assess their authenticity and quality.

A research topic still in its early stages strictly related with audio authenticity and forgery detection is the microphone classification: the goal of microphone classification is to blindly identify the microphone device involved during the recording. This is possible thanks to the presence of some traces embedded in the audio files, created by the unique physical characteristics of the recording device involved.

The first attempt in this direction dates back to 2007 thanks to the efforts of Kraetzer, Oermann, Dittmann and Lang [1]: this research can be considered the source of all the following proposals, since it demonstrated at once the feasibility of microphone classification. In the following years, thanks to the efforts of several groups the accuracy of microphone classification improved a lot from that first attempt, but the research in this field is far from being finished. Another consequence of the first attempt by Kraetzer's group was also the environment classifications - whose goal is to blindly identify the place where a recording has been made - which we will not discuss in this context.

Our work is meant to be a proposal for a completely new classification algorithm, based on a recent work by Gupta et al. [13]. We chose not to use a purely statistical approach or a model based directly on spectral or perceptual audio features, in favor of an higher-level representation. This high-level representation is built on top of Gupta's proposal, i.e. an algorithm for the blind estimation of the magnitude response of a channel, using the observations from a single microphone.

The frequency response of the recording device can be considered as a channel on its own: starting from its estimate we searched for a feature vector that could lead us to a correct classification. Moreover, from the start we chose to focus our attention not only on PCM encoded audio files, but also on sources that underwent compression, i.e. AAC, MP3 or AMR encoding.

This happened because of the possible application of a highly reliable microphone classification framework: authenticity issues often concern recordings from cellphones and smartphones, as well as conversations that took place on Skype. On each and every one of this use cases the hypothesis of handling a PCM encoded audio source is unrealistic, and thus requires a possibly high robustness toward compression.

Moreover, this robustness must hold also when extremely-low target bitrates are chosen for the encoding, as happening with the Adaptive Multi-Rate (AMR) compression scheme: the bitrates offered by the AMR range from a maximum of 12.2 kbps to a minimum of a 4.75 kbps, and this encoder is present as the default encoding options on most of the cellphones available on the market. On the other hand, both the MP3 and the AAC encoding are well-known and extremely diffused, to the point that is reasonable to expect them to be involved in a recording.

Our first opinion when considering the impact of the encoding on the classification was that it would have caused a great loss in accuracy, by crushing any useful information about the channel of the recording device. However, as we will extensively show throughout the entire work, this is not true: our algorithm is able to handle MP3 and AAC compressed audio file, up to the point that is possible to correctly identify the single device with only an extremely small influence from the target bitrate. With AMR compressed audio file, instead, the identification of the single device is still possible, but not reliable enough; however we are already able to classify the general model of the device involved, despite the low bitrate and the clearly perceivable great lost in terms of quality of the recording.

For the sake of clarity, we chose to devote each of the following chapters to a different aspect of the algorithm that we are going to expose. The reader will notice several references between different chapters: this has been done in order not to consider each of them a monolithic block and to underline the relationships between each different level of the framework, that was built by a continuous process of constant refinements, driven both by theoretical considerations and by practical evaluations of the results.

In Chapter 2 we will state the problem by the means of the use cases considered in our work. We will also underline how these use cases affect our microphone classification framework, and how they have been addressed. Moreover, we will provide the process flow of each use case, with a description of each stage involved, and with practical considerations about the processing steps directly influencing or framework.

In Chapter 3 we will expose the state of the art on microphone classification. Due to microphone classification still being in its early stages, we provided a critic overview of each algorithm proposed in the past: this was done both for the sake of completeness and in order to allow the reader to critically compare our algorithm with the previous one by exposing both pros and cons of each proposal. In the final section we tried to show the connection between microphone classification and other audio authenticity related fields, as well as works that we can reasonably suppose to be emerging in the near future.

In Chapter 4 our proposal is described thoroughly, starting from the blind channel identification on which we rely on, and ending with a complete description of the final classification algorithm. For the sake of clarity, in this chapter we chose to expose in two different sections the logical design and the implementation choices that are not due to the algorithm itself: the logical design is meant to be a reference for the implementation details, and we believe that this choice reflects the continuous process of practical evaluation of theoretical considerations that has been the real constant during the research work

In Chapter 5 we will expose the results achieved. In the first section of the chapter the test content is defined. Each following section of the chapter is organized with the same scheme: first the different test sets are differentiated; afterward for each test set the result are exposed; finally some considerations about the results are made, with complete references and critic comparisons with the best results from the state of the art.

In Chapter 6 we will make a brief summary of the results achieved, we will consider possible improvements of the framework, and we will discuss a possible applications of this work in the field of tampering detection.

Interested readers in the appendices will find details about the implementation framework with references to the software involved, the complete numerical results, and some of the listings of the octave implementation. The appendix B.2, in particular, reports the early research stages of the feature vector, and clearly shows the existing dualism between theory and evaluations.

Chapter 2

Problem Statement

In this chapter we will focus on the use cases involved in our work. We will also underline how these use cases affect our microphone classification framework, and how they have been addressed.

2.1 Use Cases

Several use cases could be related to the recordings from mobile devices, but we chose to focus on two of them, i.e. mobile recording with smart-phones and Skype interview: these two use cases are the ones where recently the authenticity has been an issue, and for which the possibility to perform a correct microphone classification could be a great help in the near future.

2.1.1 Mobile Recording with Smart-Phones

This use case concerns recordings made by a single mobile device, with one audio source as the input. On smart-phones usually several recording applications are available: depending on the application, there's the possibility to control the sampling rate, the audio encoder, or both.

"Audio source" in this context is intentionally an ambiguous term: in principle, due to the algorithm underlying our work, the "audio source" we refer to is one speaker, or more than one, whose voice is been recorded. However, we cannot ensure that this assumption holds: the "audio source" could be the playback of the loudspeaker, as well as prolonged silence or noise

2.1.2 Skype Interview

This use case concerns recordings made by at least two mobile devices, with two different audio sources as the input, e.g. a conversation between a journalist and a politician. The recording is made on one of the two sides and contains both the content by the "primary" audio source and the content by the "secondary" audio source. With the term "primary" audio source we mean the audio source which is interacting with the device that is recording the conversation, and that has not undergone any kind of transmission; on the other hand, the "secondary" audio source present in the recording has been sent on the network from another location and - as we will in the process flow description - has undergone a much longer processing chain than the other one.

The final audio file is a mixture of the primary and the secondary audio source which has been encoded and stored. Again, the term “audio source” must retain his ambiguity in order to accommodate any possible sound signal acquired by the microphone.

2.1.3 Built-in Microphones and Headset Microphones

Nowadays most of the mobile devices can be equipped with headset microphones of several different types. Regardless of the use case, we need to consider both the possibility of a recording from the built-in microphone of a mobile device, and from an headset microphone plugged in.

Since the headsets are often provided directly by the same manufacturer of the mobile device, not only we should expect similarities between different instances of the same model, but also similarities between headsets microphones and built-in microphones which are akin[1]: a situation that makes our task more difficult, as well as more interesting.

2.2 Process Flow

We will now provide the process flow of the use case, with a description of each stage involved. Our process flows will follow an outline somehow similar to the one previously stated by Kreatzer et al. [4], but that enriches their description and properly address our two use cases. Each stage of each flow will be described thoroughly.

2.2.1 Process Flow of the Mobile Recording

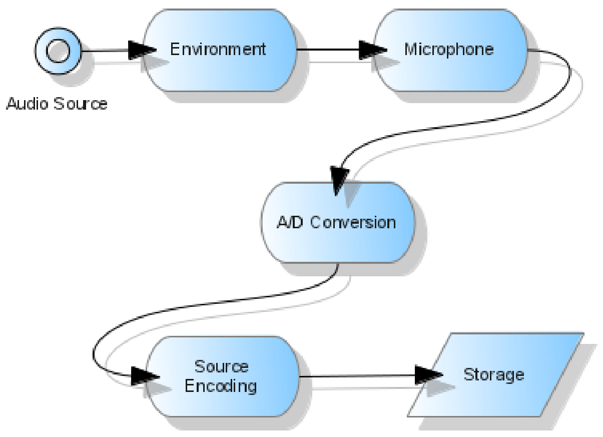


Figure 2.1: Mobile Recording - Process Flow

[1]e.g. between the original headset microphone of an iPhone 3gs and the built-in microphone of an iPhone 3gs.

As shown in Figure 2.1 the mobile recording use case can be represented by a five-stage pipeline:

2.2.1.1 Audio Source

From now on, we will denote with $s_i(t)$ an audio signal in time-domain; thus $S_i(f)$, its representation in frequency-domain, can be easily achieved by a Fourier transformation. The following holds:

$$\begin{cases} S_i\left(f\right) = \mathcal{F}\left(s_i\left(t\right)\right) \\ s_i\left(t\right) = \mathcal{F}^{-1}\left(S_i\left(f\right)\right) \end{cases} ;\ \forall i$$

where $\mathcal{F}(\cdot)$ denotes the Fourier Transform, and $\mathcal{F}^{-1}(\cdot)$ the inverse Fourier Transform.

Let $s_1(t)$ be the audio source in Figure 2.1; then, if the audio source is created by a single loudspeaker with multiple drivers[2] which is playing an audio signal $s_0(t)$ we can model the process with:

$$S_1(f) = \sum_{N_{driver}} \int_l^u F_{driver}(f) \cdot S_0(f)\, df + N_{ls}(f) \tag{2.1a}$$

In ideal circumstances the upper and lower frequency values u and l, and the amplifying function $F_{driver}(f)$ could be simplified into a constant amplifying factor, and the thermal noise $N_{ls}(f)$ generated in the playback by the loudspeaker ls is constant and negligible. Otherwise, If there's no loudspeaker present, we can simply assume that

$$S_1(f) = S_0(f), \tag{2.1b}$$

where $s_0(t)$ denotes the audio signal produced by the source.

2.2.1.2 Environment

Before the audio signal is collected by the microphone there are mainly two possible distortions occurring, as shown in equation 2.2:

$$S_2(f) = S_1(f) \cdot F_{echo}(f) + N_{envi}(f), \tag{2.2}$$

Where, of course, either equation 2.1a or equation 2.1b holds.

The product between $S_1(f)$ and $F_{echo}(f)$ describes the object and the environment that reflect the signal, and is used to simulate the possible distortion caused by echoes and reverberations[3]. The possible distortions caused by the environmental noise are denoted by $N_{envi}(f)$.

2.2.1.3 A/D Conversion

The A/D conversion involves both the sampling process and the quantization process. Let us denote with $s_i[n]$ the digital counterpart of a signal $s_i(t)$: $s_i[n]$ is the result of a quantization process with $nBits$ bits per sample applied on the values assumed by $s_i(t)$ at discrete moments $t = n \cdot T$; $\forall n \in \mathbb{Z}$, where $f_s = \frac{1}{T}$ denotes the sampling frequency in Hz.

[2]e.g. full-range, sub-woofer, woofer, mid-range or tweeter

[3]when the recording process is accomplished in an anechoic environment, then $F_{echo}(f)$ can be considered as a constant value of 1

The A/D conversion stage can be modeled as follows:

$$s_2[n] = \text{quantization}\left(nBits, \text{sampling}\left(f_s, s_2\left(t\right)\right)\right),$$

where $s_2(t)$ is the inverse Fourier Transform of $S_2\left(f\right)$ stated in 2.2. In this stage, usually the user is able to control at least the sampling frequency f_s.

2.2.1.4 Source Encoding

The source encoding in this context will be defined as follows:

$$s_3\left[n\right] = \text{source-encoding}\left(algorithm, bitrate, s_2\left[n\right]\right).$$

The input *algorithm* refers to one of the source encoding algorithms suitable for audio data, and together with the input *bitrate* is user-defined, with some limitations due mostly to the recording application involved. More details will be provided shortly afterward, in Section 2.2.3.

2.2.1.5 Storage

Finally, whatever the chosen encoding , the audio file is stored on the recording device. We will assume that any further transmission of $s_3\left[n\right]$ will be error-free, since there are no time-constraint on it.

2.2.2 Process Flow of the Skype Interview

In Figure 2.2 we can see the process flow of the Skype interview. Is straightforward to notice how this scenario involves much more steps than the previous one. Moreover, there are a lot of operations that the users cannot control, because of Skype influence or due to the transmission of the network of the signal.

In the process flow figure we have both the source encoding and the channel encoding: we can differentiate the two of them in the following way:

- The source encoding attempts to compress the data in order to transport it more efficiently, in a lossless or in a lossy fashion. We can consider the PCM encoding as the less efficient compression - i.e. there's no compression at all. The source encoding mainly addresses the size of the data.

- The channel encoding aims to find a code that allows us to transmit the data efficiently and safely over a network, where safe means "with as less errors and corruptions as possible". Usually, this is done by using Forward Error Correction codes and Error Detection Codes, with more than one layer involved. The channel encoding mainly addresses the transmission of the data.

As we will see in details, both of them are involved in the secondary audio source processing chain, and their influence cannot be considered negligible if we consider the final recording, as we will demonstrate in Chapter 5.

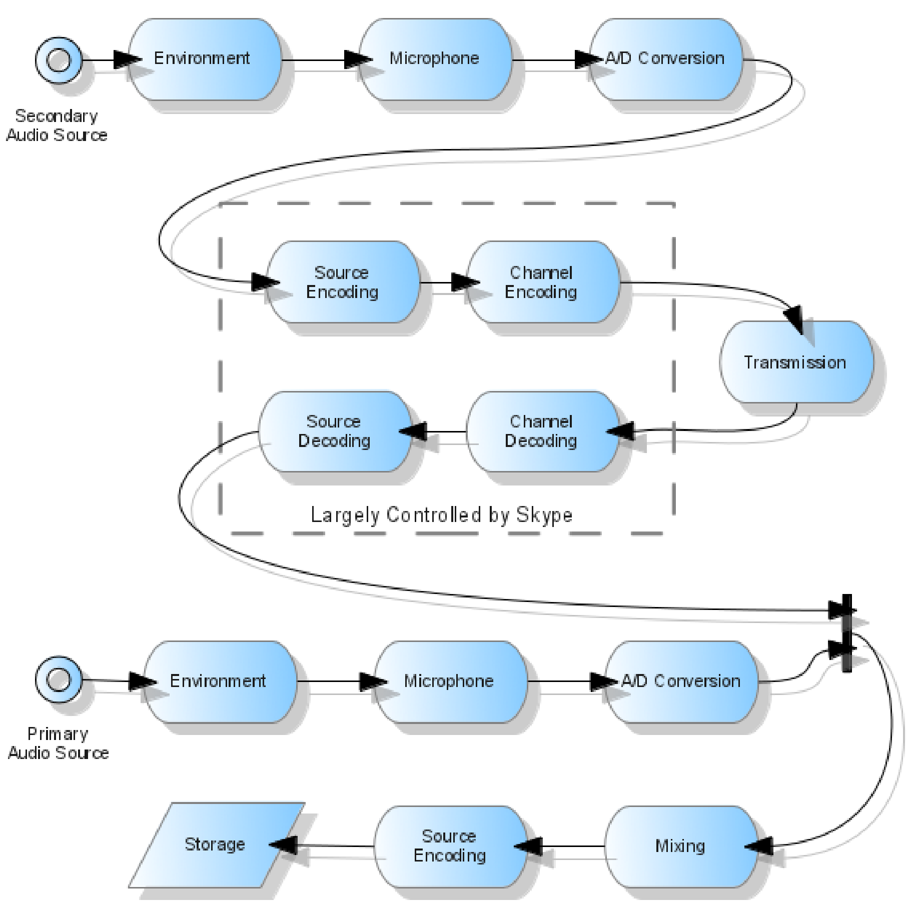


Figure 2.2: Skype Interview - Process Flow

2.2.2.1 Primary Audio Source: Definition

From now on, we will denote with $s_{j,i}(t)$ an audio signal in time-domain; thus $S_{j,i}(f)$, its representation in frequency-domain, can be easily achieved by a Fourier transformation. The following holds:

$$\begin{cases} S_{j,i}\left(f\right) = \mathcal{F}\left(s_{j,i}\left(t\right)\right) \\ s_{j,i}\left(t\right) = \mathcal{F}^{-1}\left(S_{j,i}\left(f\right)\right) \end{cases} ;\ \forall i,j$$

where $\mathcal{F}(\cdot)$ denotes the Fourier Transform, and $\mathcal{F}^{-1}(\cdot)$ the inverse Fourier Transform. The index j will be used in order to differentiate between the primary - i.e. $j=1$ - and secondary - i.e. $j=2$ - audio source.

Let $s_{1,1}(t)$ be the primary audio source in Figure 2.2; then, if the audio source is created by a single loudspeaker with multiple drivers[4] which is playing an audio signal $s_{1,0}(t)$ we can model the process with:

$$S_{1,1}(f) = \sum_{N_{driver}} \int_{l}^{u} F_{1,driver}(f) \cdot S_{1,0}(f)\, df + N_{1,ls}(f) \qquad (2.3a)$$

[4]e.g. full-range, sub-woofer, woofer, mid-range or tweeter

In ideal circumstances the upper and lower frequency values u and l, and the amplifying function $F_{1,driver}(f)$ could be simplified into a constant amplifying factor, and the thermal noise $N_{1,ls}(f)$ generated in the playback by the loudspeaker $1, ls$ is constant and negligible. Otherwise, If there's no loudspeaker present, we can simply assume that

$$S_{1,1}(f) = S_{1,0}(f), \tag{2.3b}$$

where $s_{1,0}(t)$ denotes the audio signal produced by the source.

2.2.2.2 Primary Audio Source: Environment

Before the primary audio signal is collected by the microphone there are mainly two possible distortions occurring, as shown in equation 2.4:

$$S_{1,2}(f) = S_{1,1}(f) \cdot F_{1,echo}(f) + N_{1,envi}(f), \tag{2.4}$$

Where, once again, either equation 2.3a or equation 2.3b holds.

The product between $S_{1,1}(f)$ and $F_{1,echo}(f)$ describes the object and the environment that reflect the signal, and is used to simulate the possible distortion caused by echoes and reverberations[5]. The possible distortions caused by the environmental noise are denoted by $N_{1,envi}(f)$.

2.2.2.3 Primary Audio Source: A/D Conversion

Let us denote with $s_{j,i}[n]$ the digital counterpart of a signal $s_{j,i}(t)$: $s_{j,i}[n]$ is the result of a quantization process with $nBits_j$ bits per sample applied on the values assumed by $s_{j,i}(t)$ at discrete moments $t = n \cdot T_j$; $\forall n \in \mathbb{Z}$, where $f_{s,j} = \frac{1}{T_j}$ denotes the sampling frequency in Hz. As before, the index j will be used in order to differentiate between the primary - i.e. $j = 1$ - and secondary - i.e. $j = 2$ - audio source.

The A/D conversion stage can be modeled as follows:

$$s_{1,2}[n] = \text{quantization}\left(nBits_1, \text{sampling}\left(f_{s,1}, s_{1,2}(t)\right)\right),$$

where $s_{1,2}(t)$ is the inverse Fourier Transform of $S_{1,2}(f)$ stated in 2.4.

2.2.2.4 Secondary Audio Source: Definition

Let $s_{2,1}(t)$ be the secondary audio source in Figure 2.2; then, if the audio source is created by a single loudspeaker with multiple drivers[6] which is playing an audio signal $s_{1,0}(t)$ we can model the process with:

$$S_{2,1}(f) = \sum_{N_{driver}} \int_l^u F_{2,driver}(f) \cdot S_{2,0}(f)\, df + N_{2,ls}(f) \tag{2.5a}$$

In ideal circumstances the upper and lower frequency values u and l, and the amplifying function $F_{2,driver}(f)$ could be simplified into a constant amplifying

[5] when the recording process is accomplished in an anechoic environment, then $F_{1,echo}(f)$ can be considered as a constant value of 1

[6] e.g. full-range, sub-woofer, woofer, mid-range or tweeter

factor, and the thermal noise $N_{2,ls}(f)$ generated in the playback by the loudspeaker $2, ls$ is constant and negligible. Otherwise, If there's no loudspeaker present, we can simply assume that

$$S_{2,1}(f) = S_{2,0}(f), \tag{2.5b}$$

where $s_{2,0}(t)$ denotes the audio signal produced by the source.

2.2.2.5 Secondary Audio Source: Environment

Before the secondary audio signal is collected by the microphone there are mainly two possible distortions occurring, as shown in equation 2.6:

$$S_{2,2}(f) = S_{2,1}(f) \cdot F_{2,echo}(f) + N_{2,envi}(f), \tag{2.6}$$

Where, as before , either equation 2.5a or equation 2.5b holds.

The product between $S_{2,1}(f)$ and $F_{2,echo}(f)$ describes the object and the environment that reflect the signal, and is used to simulate the possible distortion caused by echoes and reverberations[7]. The possible distortions caused by the environmental noise are denoted by $N_{2,envi}(f)$.

2.2.2.6 Secondary Audio Source: A/D Conversion

The A/D conversion stage can be modeled as follows:

$$s_{2,2}[n] = \text{quantization}\left(nBits_2, \text{sampling}\left(f_{s,2}, s_{2,2}\left(t\right)\right)\right),$$

where $s_{2,2}(t)$ is the inverse Fourier Transform of $S_{2,2}\left(f\right)$ stated in 2.6.

2.2.2.7 Secondary Audio Source: Source Encoding

The source encoding of the secondary audio source is the first stage where the influence of Skype is really high. According to Skype, the default audio codec for all Skype-to-Skype calls is SILK [29], which was integrated for the first time in version 4.0 beta 3 from January 7, 2009. Previous releases are supposed to use a proprietary wide-band audio codec, i.e. the internet Speech Audio Codec (iSAC) [30]. Of course, is possible that other codecs are involved.

What we now for sure, is that Skype seems to adapt its encoding options depending both on the content of the audio input and on the network conditions, e.g. the bandwidth and the number of transmission errors. Other than switching between different bitrates, SILK is also supposed to change the sampling frequency acquired by the sound card and to match the encoding complexity to the CPU resources available [29]; similar features can be found in the iSAC codec [30].

We can try to summarize this step with

$$s_{2,3}\left[n\right] = \text{source-encoding}_{Skype}\left(algorithm\left(C, N, R\right), bitrate\left(C, N, R\right), s_{2,2}\left[n\right]\right),$$

where $algorithm(\cdot)$ denotes the codec chosen by Skype and $bitrate(\cdot)$ the target bitrate. Both variables are supposed to depend on the detected content C, on the network conditions N and on the CPU resources available R.

[7]when the recording process is accomplished in an anechoic environment, then $F_{2,echo}(f)$ can be considered as a constant value of 1

2.2.2.8 Secondary Audio Source: Channel Encoding

We can suppose that, as for the source encoding, the channel encoding involved is Skype is not negligible: it has to integrate the time-constraints related to the real-time processing needed by the conversation while relying on the internet network and its limitations. Unfortunately we don't know the specification of this algorithm, but we can try to express it on the following way:

$$s_{2,4}[n] = \text{channel-encoding}_{IP}\left(\text{channel-encoding}_{Skype}\left(s_{2,3}[n]\right)\right).$$

This formulation, of course, is meant to underline the interaction between the non real-time Internet Protocol algorithms and the Skype real-time constraints and algorithms.

2.2.2.9 Secondary Audio Source: Transmission

It would be meaningless to formalize all the transmission procedure herein: we will just let $s_{j,i}^{trans}$ be the signal $s_{j,i}$ after the transmission, i.e.

$$\begin{cases} s_{j,i}^{trans}(t) = s_{j,i}(t - \Delta t)\,;\ \forall t \\ s_{j,i}^{trans}[n] = s_{j,i}[n]\,;\ \forall n \end{cases}, \text{ in ideal conditions.}$$

2.2.2.10 Secondary Audio Source: Channel Decoding

The channel decoding addresses the transmission error overs the network, and can be formalized as follows:

$$s_{2,5}[n] = \text{channel-decoding}_{Skype}\left(\text{channel-decoding}_{IP}\left(s_{2,3}^{trans}[n]\right)\right).$$

We can suppose that the Skype channel-encoding/decoding schema is meant to correct as much missing or corrupted samples as possible, relying on information about the data structure - which is unknown to the Internet Protocol. It's also legitimate to suppose that, when an error correction is not feasible, the bits of the missing or corrupted samples are marked as incorrect data: the source-decoder of the SILK speech-codec, whose specifications are available online, is errors-aware.

2.2.2.11 Secondary Audio Source: Source Decoding

The source decoding can be formalized simply as

$$s_{2,6}[n] = \text{source-decoding}_{Skype}\left(s_{2,5}[n]\right).$$

$s_{2,6}[n]$ is a digital audio file of sampling frequency $f_{s,Skype}$: a priori we cannot assume that $f_{s,2} = f_{s,Skype}$, especially when the network conditions are bad. Moreover, we know that the perceived quality of $s_{2,6}[n]$ is time-variant, and that when to many errors occur some samples of $s_{2,6}[n]$ are missing. In other terms, we can assume that

$$s_{2,6}[n] \neq \text{source-decoding}_{Skype}\left(s_{2,3}[n]\right), \text{ in general.}$$

2.2.2.12 Mixed Recordings: Definition

After the decoding of the secondary audio source, we are finally able to mix together the two signals in a single audio file. This cannot be done directly, since most likely $f_{s,Skype} \neq f_{s,1}$: the program used in order to record the conversation first needs to resample both the audio streams to a common sampling frequency $f_{s,1+2}$, i.e.

$$\begin{cases} s_{1,3} = resampling(s_{1,2}, f_{s,1+2}) \\ s_{2,7} = resampling(s_{2,6}, f_{s,1+2}) \\ s_{1+2,0}[n] = s_{1,3}[n] + s_{2,7}[n] \end{cases} .$$

Usually $f_{s,1+2}$ is defined by the user of the recording application.

2.2.2.13 Mixed Recordings: Source Encoding

The source encoding will be defined again as follows:

$$s_{1+2,1}[n] = \text{source-encoding}\left(algorithm, bitrate, s_{1+2,0}[n]\right).$$

The input *algorithm* refers to one of the source encoding algorithms suitable for audio data, and together with the input *bitrate* is user-defined, with some limitations due to the recording application involved. Important details about this source encoding will be provided shortly afterward, in Section 2.2.3.

2.2.2.14 Mixed Recordings: Storage

As we did for the process flow of the mobile recording, also for the Skype interview use case we can assume that at the end, whatever the encoding specified, the audio file is stored on the recording device. We will also assume again that any further transmission of $s_{1+2,1}[n]$ will be error-free, since there are no time-constraint on it.

2.2.3 Source Encoding Options

As we know several audio codecs exists, and of course it's possible to apply a further source encoding after the storage step. We will now consider the specific case addressed in our process flows, i.e. the encoding options available when the source encoding is performed during the recording itself.

2.2.3.1 Mobile Recording

Depending on the recording application, there are several options available: usually it's possible to select between PCM encoding, MP3 encoding and AAC encoding, with different bitrates and sampling frequency. However, this is not always the default, even for smart-phones: the default source-encoding algorithm - for most of the simple mobile phones but also for some of the smart-phones default recording applications - is the Adaptive Multi-Rate coding (AMR), a compression scheme proposed by 3GPP [31, 32].

The AMR codec consists of eight source codecs with bit-rates of 12.2, 10.2, 7.95, 7.40, 6.70, 5.90, 5.15 and 4.75 kbit/s and is based based on the Code-Excited Linear Predictive (CELP) coding model: in this CELP model, for each 20 ms frame 10 Lineal Predictive (LP) filter coefficients, together with the gains and the indexes of two excitation vectors from an adaptive and fixed codebooks are computed; after their transmission, at the decoder the speech is synthesized by filtering the reconstructed excitation signal - i.e. the sum of the two excitation vectors - through the LP synthesis filter.

2.2.3.2 Skype Interview

As for the mobile recording, also in the Skype interview use case it's possible to select between PCM encoding, MP3 encoding and AAC encoding, with different bitrates and sampling frequency, depending on the specific application involved during the process. Whatever the encoding, it's important to remember that the audio data coming from the secondary audio source has been already encoded, transmitted and decoded with the processing steps stated in Section 2.2.2.

For this use case, on the other end, we believe that the presence of an AMR source-encoding is really unlikely.

Chapter 3

State of the Art on Microphone and Environment Classification

In the last years multimedia contents have spread all over the world: the duplication of a digital object has become quite a straightforward procedure, as well as its coding and decoding, transmission, editing and storage. Although some sources provide us with authentic information, many others contain forged content: concerns regarding how to validate multimedia data constantly arise, highlighting the need of methods and tools to assess their authenticity and quality.

Recently Gupta et al. gave a general survey on current developments and future trends in detecting the forgery of digital audio files [2], while the REWIND consortium provided a comprehensive overview of multimedia footprint detection and footprint parameter estimation, regarding both image, video and audio data [3].

The aim of this chapter is to point out the state-of-the art on microphone/device and environment classification: for the sake of clarity, in Section 3.0 we provide a context model for microphone forensics proposed by Kreatzer et al. [4], in order to highlight the different stages of the recording process pipeline, and how each of them alters the original source.

In Section 3.1 and Section 3.2 we discuss both microphone/device classification and environment classification techniques, each of them relying on a specific stage of the recording pipeline previously defined.

Lastly, in Section 3.3 we discuss possible approaches to editing/tampering detection, that exploit microphone/device and environment classification, together with different forgery detection techniques.

3.0 A Context Model for Microphone Forensics

Recording Process Pipeline

In the following sections of this chapter we will refer to the context model introduced by Kreatzer et al. [4], which describes all the recording process by using the five-stage pipeline shown in Fig. 3.1.

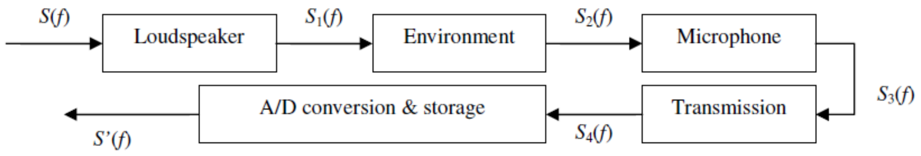


Figure 3.1: Recording Process Pipeline

Let a function $S(t)$ denote the original audio signal in time-domain; thus $S(f)$, its representation in frequency-domain, can be easily achieved by a Fourier transformation:

$$S(f) = \mathcal{F}(S(t))$$

In Figure 3.1 $S_1(f)$, $S_2(f)$, $S_3(f)$ and $S_4(f)$ denote the analogue audio signals after each processing segment, while $S^{'}(f)$ denotes the final digital audio signal achieved; obviously the time domain counterpart of $S^{'}(f)$ is computed via inverse Fourier transform as $S^{'}(t) = \mathcal{F}^{-1}(S^{'}(f))$.

This process pipeline clearly is the one that we extended in the previous chapter: it presents both some similarities and some important differences that we will mention in the following sections.

3.0.1 Loudspeaker Stage

The loudspeaker influence is always present, and modeled explicitly by the means of Equation 3.1

$$S_1(f) = \sum_{N_{driver}} \int_l^u F_{driver}(f) \cdot S(f)\, df + N_{ls}(f) \tag{3.1}$$

that is equivalent to Equation 2.1a. The main difference with our model where the loudspeaker is modeled as a special case - is that in this context model the loudspeaker is considered to be always present inside the pipeline.

3.0.2 Environment Stage

The environment stage is shown in equation 3.2:

$$S_2(f) = S_1(f) * F_{echo}(f) + N_{envi}(f) \tag{3.2}$$

The main difference with the nearly-equivalent Equation 2.4 is the convolution of $S_1(f)$ and $F_{echo}(f)$ instead of their product. The consistency of this stage is the characteristic exploited by environment-based classification techniques proposed by Malik et al. [5, 6], that we are going to discuss in depth in Section 3.2.

3.0.3 Microphone Stage

Equation 3.3 simulates the process of a microphone collecting the signal:

$$S_3(f) = \int_{spectrum} F_{mic}(f) \cdot S_2(f)\, df + N_{mic}(f) + N_{ENF}(f) \tag{3.3}$$

$F_{mic}(f)$ denotes the frequency response function of the microphone, $N_{mic}(f)$ denotes the thermal noise that the microphone generates, and $N_{ENF}(f)$ denotes the electric network frequency (ENF) influence[1].

Microphone/device classification techniques based on statistical pattern recognition [4, 1], that we are going to discuss thoroughly in Section 3.1, attempt to detect intrinsic fingerprint traces left by $F_{mic}(f)$, assuming that the specificity of a microphone comes from the unique vibration behavior of its diaphragm, as well as the diaphragm's interaction with the other parts. Other influences to be considered are the orientation of the microphone to sound sources, the microphone mounting and the aging phenomena of the microphone, modeled as multiplicative influences O (orientation), M (mounting) and A (aging):

$$F_{mic}(f) = F_{inf}(O, M, A) \cdot F_{diaphragm}(DiaphCharacteristics)$$

Usually $N_{mic}(f)$ can be considered as constant and negligible during the classification, since its influence is rather minor compared to the on coming from $F_{mic}(f)$. On the other hand, while it seems that orientation and aging phenomena have no impact at all on microphone classification accuracy, the mounting can have a strong influence if it affects the reverberation behavior[2].

Our own classification method, as we will see, is based on an hi-level model of the transfer function $F_{mic}(f)$.

3.0.4 Transmission and A/D Conversion Stages

Equation 3.4 and 3.5 model the transmission of the signal from the microphone to the A/D conversion device, and the process of storing the audio as an audio file:

$$S_4(f) = \int_{spectrum} F_{tran}(f) \cdot S_3(f)\, df + N_{tran}(f) \tag{3.4}$$

$$S'(f) = \int_0^{f_N} F_{samp}(f) \cdot S_4(f)\, df + N_{quant}(f) + N_{thermal}(f) \tag{3.5}$$

In equation 3.4 $F_{tran}(f)$ denotes the distortion coming from the transmission of the signal, while $N_{tran}(f)$ denotes the thermal noise coming from the transmission environment. In equation 3.5 f_N denotes the Nyquist frequency, $N_{quant}(f)$ denotes the quantization noise, and $N_{thermal}(f)$ the thermal noise of the A/D device. None of the microphone/device and environment classification techniques proposed so far exploited these stages.

We can notice that both these two stages are modeled in a completely different way than in our own pipeline, and that the encoding is not addressed at all.

[1]Further details in Section 3.3

[2]e.g. if the microphone is lying on a table

3.1 Microphone Classification Techniques

In this section we will discuss different microphone classification techniques: in order to assess advantages and disadvantages of each of them the general outline is provided, as well as test setup and experimental results. Before starting our discussion, we will also provide some information about general adoptable strategies.

All the classification techniques that we are going to introduce in this section rely only on the microphone stage of the context model previously proposed[3].

3.1.1 General Strategies

We are able to identify two main aspects concerning microphone classification strategies: the classification technique is supervised or unsupervised, and either an inter-device analysis or an intra-device is performed.

Supervised Classification In supervised classification different models[4] can be applied. Each of them passes a so-called supervised learning algorithm, that is a procedure to infer a function starting from *supervised* (i.e. labeled) data. The inferred function is called *classifier*, and should be able to identify the actual class of any input between those involved during the training.

Unsupervised Classification The goal of unsupervised classification techniques[5] is the partitioning of data into groups while the group affiliation of the data is not known in advance (i.e. *clustering*). Since - hopefully - data belonging to a group share common or similar traits, usually the partitioning is defined by a distance measure of some sort.

Inter-Device Analysis Inter-device analysis refers to a classification performed between *different models* of microphones, usually from different manufacturers. Although this case seems simpler, it's possible that microphones share the same transducer technology, hence leading to misclassification.

Intra-Device Analysis Intra-device analysis refers to a classification performed between microphones of the *same model and manufacturer*. It's mainly possible thanks to the unique vibration behavior of each microphone's diaphragm.

Within microphone classification supervised classifiers outperform clustering algorithms, hence most of the methods relies to a supervised classifier. Conversely, both inter-device and intra-device classification were investigated and enhanced.

3.1.2 Statistical Pattern Recognition Based Approach

In 2005 Oermann et al. presented a new concept for audio classification and analysis [7], based on an introduced Verifier-Tuple which enables a detailed analysis of every kind of media. Their proposal led Kraetzer et al. to a statistical

[3]see Subsection 3.0.3 of Section 3.0

[4]e.g. decision trees, regression analysis, support vector machines, Naive Bayes

[5]e.g. k-means, mixture models, hierarchical clustering

pattern recognition based approach [4, 1], in order to determine whether inter-device and intra-device classification are possible, both with unsupervised and supervised techniques.

3.1.2.1 Verifier-Tuple for Audio Forensic

"A speaker environment is determined through its characteristic background sounds and the used microphone". Starting from this statement Oerman et al. [7], in order to specify the actual speaker's environment, developed a concept for a detailed analysis of room and device characteristics. The fundamental assumption was that the extraction of background features of an audio stream can provide an informative basis for determining its origin location[6] and the used microphone.

The Verifier-Tuple consists of four parts: the syntax, the executive semantic, the functional semantic and the interpretative semantic. Since each of them refers to a different level this multilayer structure enables a more detailed analysis and classification of information, thus providing a new powerful approach for audio forensic.

Even if this work was only theoretic, in the following years it led many other researches to evaluate the discriminative power for microphone and environment classification of different known statistical features: it provided the basis for the concept of microphone/device and environment classification, as well as a concept scalable also for other media such as text, image, video or 3d-complexes.

3.1.2.2 Inter-Device Statistical Approach

In 2007 Kraetzer et al. proposed a first approach for digital media forensic to determine both the used microphones and the environments of recorded digital audio samples [1]. Starting from the concept of the Verifier-Tuple, the main idea of this work is that from syntactical audio features additional higher level semantic features can be derived up, hence allowing a microphone/device and environment classification that was never done before.

Based from this idea, the work proposes three hypotheses and evaluates them with many tests:

1. Is it possible to correctly classify the microphone used for the generation of a recording?
2. Is it possible to correctly classify the location where the recording was made?
3. Does feature selection (feature reduction) improve the classification accuracy?

We will focus for now on hypotheses 1 and 3: further discussion about hypothesis 2 will be held in Subsection 3.2.1 of Section 3.2.

The test set is composed of 10 different audio files, recorded for the inter-device classification as mono audio data by 4 microphones from different manufacturers in 10 different rooms[7].

[6]e.g. a quiet room, a noisy street, an airport or a train station

[7]i.e. 400 audio files of 18.5 sec with 44.1 kHz sampling rate and 16 bit quantization

The genre of the recorded audio files is variable: there are music files, speech data, noise, artificial sounds, silence and instrumental records. Both Naive Bayes classifiers and K-means clustering algorithm are tested, in a feature space composed by the complete feature set[8] of AMSL Audio Steganalysis Toolset.

Average results for the Naive Bayes classifier are in the range [61.37%,75.99%] depending on the room, while those obtained by K-means clustering are in the range [30.13%, 43.57%]. In order to correctly evaluate these results it's important to notice that, since it's a 5-class classification problem (recordings from the four microphones and the original data), just "guessing" the result would be equal to a 20% performance. Moreover, the choice to record files of variable genres makes the classification much harder, since complex signals mask the peculiar coloring effect of each microphone.

In the case of Naive Bayes classifier the increasing of the number of feature vectors per file results in an increasing classification accuracy on microphones, while the results of the K-means clustering seem to be independent of the number of feature vector supplied for the tests. Unfortunately the classification accuracy is reduced by feature selection: every reduction of the feature space adversely affects performances, both for supervised and unsupervised classification.

The small size of the test set implies that a generalization based on these results is not possible. Nevertheless, this first investigation demonstrated at once that microphone classification starting from multiple low-level features is possible, and it's been a reference for many following researches. Despite the decrease in accuracy due to feature reduction it's interesting to notice the importance of Mel-cepstrum domain based features even in this first and somehow "rough" attempt: more recent works have made an extensive use of these features, as we will see.

3.1.2.3 Intra-Device Statistical Approach

In 2011 Kraetzer et al. proposed a second work, this time concerning intra-device classification [4]. After the formalization of the context model reported in Section 3.0, they made an attempt to correctly classify two well distinct set of microphones[9]; the recorded material used for the practical investigations has been generated by the same reference files proposed in [1]. A total of 590 features per frame were computed by AMSL Audio Feature Extractor, including both time domain based features, frequency domain based features and Mel-cepstrum domain based features[10].

This work focused on two empirical investigation, about

1. The identification of suitable classification algorithms for statistical pattern recognition based classification, evaluating 74 supervised classifiers and 8 clustering algorithms

2. The determination of suitable features for the pattern recognition, and the impact of feature selection to the classification accuracy

[8]i.e. 7 time domain based features and 56 Mel-cepstrum domain based features

[9]i.e. an homogeneous set of four Røde NT6 condenser microphones and another homogeneous set of four Beyerdynamic Opus 69 dynamic microphones

[10]Including 2nd order derivative MFCCs and FMFCCs, that were not present in the previous work

3. The determination of the influence of changes in the microphone orientation and mounting on the classification performance

4. The performance achieved in using the statistical pattern recognition based microphone classification for the detection of audio signal composition

We will now discuss only points 1, 2 and 4, since results for point 3 have been reported previously in Subsection 3.0.3 of Section 3.0.

As well as in [1] supervised classifiers outperformed clustering algorithms: in a 5-class problem none of the clustering algorithms was able to show an accuracy better than 32.675%, with a further decrease of the maximum classification accuracy to 27.6% after a reduction to the 20 most significant features. On the other hand supervised classifiers achieved a maximum accuracy of 75.88% for Røde microphones, and of 82.51% for Beyer microphones[11]; if only the 20 best features are used the accuracy drops in average for about 7.11%, but the average computation time is reduced by factor 32.7.

The best features were uncovered by a principle component analysis, that highlighted 187 components out of 590 being responsible for 95% of the sample variance. Global features showed no significance in intra-device microphone classification, while the second order derivative FMFCCs clearly outperformed every other class of features: within the top 30 features, Mel-cepstrum domain based ones occupy 23 ranks.

In order to correctly evaluate the performance of the statistical pattern recognition based microphone classification for the detection of audio signal composition, four different cases were inspected:

1. Microphone recordings of one known[12] microphone made in different locations composed into one stream

2. One known microphone pasted into a stream of a completely different known microphone

3. One unknown microphone pasted into a stream of a completely different known microphone

4. One unknown microphone pasted into a stream of a completely different unknown microphone

Classifiers involved in these tests were Naive Bayes, SMO, RandomCommittee and RandomForest. In cases 1,2 and 3 RandomCommittee and RandomForest achieved outstanding results, while Naive Bayes classifier's performances were really poor, and SMO's ones were significant, but surely less than optimal: in case 3, that is the most likely in recording authentication, accuracy reported was 51% for Naive Bayes, 84% for SMO, 96% for RandomForest and 97% for RandomCommittee. Results for test 4 showed, as expected, that supervised classifiers are unsuited for unknown sources.

Unfortunately also in this work the small size of the test set implies that a generalization based on this results is not possible. However, this paper highlights that intra-device classification is possible, and both the great relevance of

[11] a detailed listing of the classifier ranking is present on http://omen.cs.uni-magdeburg.de/itiamsl/mitarbeiter/christiankraetzer/ publications.html

[12] i.e. used during the training of the classifier

microphone classification in audio forgery detection and the great need of unsupervised microphone classification techniques, since supervised ones are poor in audio files containing only unknown sources. To have put so much emphasis on the importance of Mel-cepstrum domain based features will surely be useful in the near future, since they could be a starting point for further researches.

3.1.3 Fourier Coefficients Based Approach

In 2009 Buchholz et al. investigated whether is possible to identify the microphone model used in making a certain audio recording by using only Fourier coefficients [8]. Since this work was focused on microphone model classification as opposed to microphone identification, only inter-device analysis was performed, with several supervised classifiers.

3.1.3.1 Concept

The purpose of this paper was to investigate about the following questions:

1. Is it possible to determine a microphone model relying only on Fourier coefficient characteristics of a recording made using that microphone?
2. Which classifier is the most accurate one for this peculiar classification setup?

Fourier coefficients are usually characteristic for the sounds recorded, and not for the device recording it: in order to achieve a suitable description of the microphone involved Fourier coefficients have to be computed only in near-silence segments of the audio files, since it's likely for them to contain mostly noise.

Other parameters taken in account were both the size of FFT windows and the amplitude value for near-silence threshold: by increasing the size of FFT the frequency resolution is higher and the model is more detailed, but the number of selected windows is lower; by increasing the amplitude value for the threshold the number of selected windows is higher, but portions of audible audio signal are selected together with noise.

3.1.3.2 Feature Extraction and Experimental Results

Since the basic idea was to classify the microphone for each recorded file by relying only on the FFT coefficients of the noise portion of the audio recordings, the feature extraction is made by several steps:

1. Each audio file f is divided into equally-spaced non-overlapping windows W_f with size $2n$ samples[13]
2. Each windows is selected only if the maximum amplitude does not exceed a variable[14] near-silence threshold t and thus is assumed to contain no content but background noise

[13] $n \in \{256, 2048\}$

[14] for $n = 256$ $t \in \{0.01, 0.025, 0.05, 0.1, 0.2, 0.225, 0.25, 0.5, 0.1\}$
for $n = 2048$ $t \in \{0.01, 0.025, 0.05, 0.1, 0.25, 0.35, 0.4, 0.5, 0.1\}$

3. Selected windows are transformed in the frequency domain using the FFT, and the amplitude portion of the complex valued Fourier coefficients is computed

4. For each file the amplitudes of the Fourier coefficients are aggregated summing up the amplitudes representing the same harmonic in different windows, and the resulting feature vector is normalized

After the feature extraction each file is characterized by a unique fixed-length feature vector, regardless of its length.

For the actual classification task the WEKA machine learning tool was used: from it's broad range of classification algorithms Naive Bayes, SMO, Simple Logistic, J48 decision tree, IB1 and IBk[15] were selected. The parameter k for IBk was set equal to 2, and all classification tests were performed with a 10-fold stratified splitting strategy: the sample set was divided into ten subsets of equal size, all containing about the same number of samples from each microphone class, and each subset was used as the test set in turn, while the remaining nine subsets were combined and used as the training set.

As well as in works by Kreatzer et al. [4] the genre of the recorded audio files is variable: there are music files, speech data, noise, artificial sounds, silence and instrumental records.

Seven different microphones were classified, and the classification accuracy changed depending both on thresholds and FFT lengths, as expected: for $n = 2048$ accuracy is higher, but less windows are selected, while for $n = 256$ overall accuracy decreases, but more windows were selected. Best classification results are obtained with the Simple Logistic classifier, with about 93.5% ($n = 2048$) and 90.6% ($n = 256$).

Unfortunately these results are not totally reliable: the initial assumption was to capture the intrinsic characteristics of each microphone using only the Fourier coefficients computed from the noise present in near-silent segments. Despite this, in order not to reject too many windows, the threshold has to be set not lower than 0.1 of the maximum amplitude: a value at which portions of audible audio signal are definitely selected together with noise. Furthermore, these results also contains windows for which a feature vector was not present, that were classified by guessing[16].

There is clearly room for improvement: for instance, overlapping-windows would increase the number of FFT windows selected, thus reducing the effect of randomness on the frequency histogram and decreasing the number of samples that are not within a selected window; if an original version of the signal could be obtained, it should be feasible to subtract the original signal from the recording one, thus achieving only the distortions and noise introduced by the microphone; furthermore, other features could be used in order to better characterize near-silence segments.

However, this work showed that it is indeed feasible to determine the microphone model based on audio recording, also only by using a simple but really fast feature extraction method such as the one proposed, even it the actual implementation of the concept model appears unsuited.

[15]i.e. 1-nearest neighbor and k-nearest neighbor

[16]to the point that for very small thresholds the percentage of correctly classified samples actually exceeds the percentage of samples that can be classified

3.1.4 Mel-Frequency Cepstrum Based Approach

In 2010 Garcia-Romero and Epsy-Wilson presented a study on the automatic identification of the acquisition device when only access to the output speech recordings is possible [9]. They introduced a supervised inter-device microphone classification algorithm, relying on an only-means adapted Universal Background Model - Gaussian Mixture Model (UBM-GMM) trained on an Mel-Frequency Cepstral Coefficients (MFCCs) characterization of the speech recordings, and on a Support Vector Machine (SVM) classifier. Also Linear-Frequency Cepstral Coefficients (LFCCs) were tested, but since their average identification rate despite the higher dimensionality was identical to that of MFCCs, we will focus on the second ones.

3.1.4.1 Device Characterization

In order to achieve a blind-passive mechanism for device characterization, since the signals available not only contain information about the device but also about the speech content variability, a probabilistic characterization that could overcome speakers' characteristics was required.

The solution proposed was the adoption of an only-means adapted UBM-GMM architecture with 2048 mixtures and diagonal covariance matrices: for each recording a GMM was built, where the frequency content information was represented by either 23 MFCCs or 38 LFCCs, and all the discriminant information of the models was captured by the means of the GMM thanks to a MAP adaptation process that only updates the means of the speaker's GMM with respect to the UBM. After this process a Gaussian supervector (GSV) was built by stacking the means of the mixture components, thus representing each speech recording with a point in a high-dimensional vector space.

This procedure resulted in a fixed-length [17] template to represent variable-length speech recordings: a desirable property, since in principle the intrinsic fingerprint of a device should be independent of the amount of data acquired.

3.1.4.2 Classifier Training and Experimental Results

GSVs computed from each file were used in order to train a SVM classifier: a two-fold cross-validation setup resulted in an average of 280 positive GSV exemplars and 7 times as much negative GSV exemplars to train the SVM models for each partition, starting from the ICSI subset of the NIST 2006 Speaker Recognition Evaluation Database.

Classifier performance in microphone classification were outstanding: the average accuracy across microphones is 99.0%, with no apparent confusion patterns observable. A similar test was performed also for landlines telephone handset, obtaining an average accuracy of 93.2%, and with most of the errors remaining within the same transducer class.

Results achieved in this work show a great improvement compared to those reported by Kraetzer et al., and are more general since the number of recordings and microphone involved is much bigger. However, this GSV-SVM system should be tested also for intra-device microphone classification, in order to eval-

[17]GSV dimension is equal to the number of mixtures times the number of MFCCs

uate possible degradations of the performances due to the strict correlation between microphones of the same model and manufacturer.

3.2 Environment Classification Techniques

In this section we will discuss different environment classification techniques: as in Section 3.1, in order to assess advantages and disadvantages of each of them the general outline is provided, as well as test setup and experimental results.

The classification techniques that we are going to introduce in this section rely mainly on the environment stage of the context model previously proposed[18].

3.2.1 Statistical Pattern Recognition Based Approach

We will now report the actual results concerning environment identification of the approach proposed by Kraetzer et al. [1] in 2007, whose general outline and test setup has been already provided in Subsection 3.1.2.2 of Section 3.1.

In particular, we are going to discuss their answer to the following:

> Is it possible to correctly classify the location where the recording was made?

Since a set of 10 rooms was considered, a random correct classification of the rooms would occur with a likelihood of 10%. Average results for the Naive Bayes classifier are in the range of [23.97%,41.54%] depending on the microphone, while those obtained by K-means clustering are in the range of [10.99%, 26.49%], when 100 feature vectors per file were used; the accuracy dropped when increasing the number of feature vectors per file.

The results for the clustering algorithm are lower than the results for the Naive Bayes classifier: while the best classification accuracy of 26.49% could still be considered significant, the lowest result of 10.99% is close to just “guessing” the room. More precise information is need about results of Naive Bayes classifier: the confusion matrix reported by the authors highlights that in seven cases out of ten the highest classification result is achieved for the room the recording was made in; only three rooms out of ten were “totally” misclassified.

It’s not surprising that within this paper the environment identification was less accurate than the microphone identification, since the filtering effect of a microphone is probably much stronger and more unique than the filtering effect of a room environment. However, these results are positive, since the feature vectors were the same both in microphone classification and in environment classification despite the different stages in the recording pipeline.

3.2.2 Audio Reverberation Time Based Approach

In 2010 Malik and Farid proposed a technique to model and estimate the amount of reverberation in an audio recording [5]: since reverberation depends on the shape and composition of a room, differences in the estimated reverberation time can be used in an environment classification setting.

[18]see Subsection 3.0.2 of Section 3.0

3.2.2.1 Mathematical Model

Within this work, the decay of an audio signal $x(t)$ is modeled with a multiplicative decay and additive noise:

$$y(t) = d(t) \cdot x(t) + n(t),$$

where

$$d(t) = \exp(-t/\tau).$$

The decay parameter τ embodies the extent of the reverberation, and can be estimated using a maximum likelihood estimator. This can be done assuming that the signal $x(t)$ is a sequence of N independently and identically-distributed (*iid*) zero mean and normally distributed random variables, and that this signal is uncorrelated to the noise $n(t)$, which is also a sequence of N *iid* zero mean and normally distributed random variables with variance σ_n. with this assumption $y(t)$ - the observed signal[19] - is a random variable with its probability density function $P_{y(t)}(\cdot)$ and likelihood function $L(\cdot)$.

The decay parameter τ is estimated by maximizing the log-likelihood function $\mathcal{L}(\cdot)$, by setting its partial derivatives equal to zero and solving for the desired τ:

$$\frac{\partial \mathcal{L}}{\partial \sigma} = -\frac{N}{\sigma} + \frac{1}{\sigma^3} \sum_{k=0}^{N-1} \frac{y^2(k)}{\gamma^2(k)} \tag{3.6}$$

$$\frac{\partial \mathcal{L}}{\partial \tilde{\tau}} = -\sum_{k=0}^{N-1} \frac{k\tilde{\tau}^{2k-1}}{\gamma^2(k)} \left(\frac{y^2(k)}{\gamma^2(k)} - 1 \right) \tag{3.7}$$

where

$$\gamma(t) = \sqrt{\exp(2t/\tau) + \sigma_n^2)}$$

$$\tilde{\tau} = \exp(-1/\tau)$$

Although equation 3.6 can be solved analytically, $\tilde{\tau}$ in equation 3.7 cannot: an iterative non-linear minimization is required, consisting in two primary steps. In the first step σ is estimated by setting equation 3.6 equal to zero and solving for σ, and in the second one $\tilde{\tau}$ is estimated by maximizing the log-likelihood function $\mathcal{L}(\cdot)$ using a standard gradient descent. A previous estimate of σ_n is required, and it's achieved from the noise floor following the signal.

3.2.2.2 Experimental Results

The mathematical method was tested in three different experiments:

1. Digital generated audio data, built according to the *iid* zero mean and normally distributed assumption
2. Audio recordings with digital reverberation simulated with an image-source model
3. Human speech recorded in four different environments[20]

[19] $y(t) = \mathcal{F}^{-1}\left(S_1(f) * F_{echo}(f)\right)$, see equation 3.2

[20] i.e. outdoors, in a small office, in a large office, and in a stairwell

Since we are interested both in environment classification and in audio forensic let's focus on the third experiment, in which the same speaker read the same text in all the recordings: the audio was recorded using a commercial-grade microphone, and each recording was previously pre-processed with a speech enhancement filter, because of the considerable background noise. Then, the reverberation time was estimated from fourteen positions in each of the recorded audio segments, manually selecting them considering that the speech in those positions decayed to the noise floor. The mean and standard deviation estimates for the reverberation time τ were meaningful, and were confirmed by a one-way analysis of variance.

Despite the individual estimates of the reverberation decay time can't fully characterize a speaker's environment, an abrupt variation between them could be a strong clue while trying to detect a forgery: the contribution of this research has been relevant, and provided a simple - yet powerful - model to detect a feature produced by the environment stage of the recording pipeline, that until now was not included explicitly in the statistical pattern recognition based approach previously proposed. However, this techniques has some disadvantages: for instance, the manual selection of the positions suitable for the reverberation time estimation, the negative influence of a noisy environment, and the number of frames suitable for the estimation, that could be too low.

3.2.3 Blind De-Reverberation Based Approach

In 2012 Malik and Zhao proposed a supervised environment classification algorithm relying on blind de-reverberation [6]: the feature extraction is made not on the microphone signal, but on the estimate of the reverberant signal, that is fully characterized from the geometric properties of the environment.

3.2.3.1 General Outline

As we saw in the context model previously proposed, during the environment stage the acoustic environment leaves its fingerprint in the final recording, due to the shape and the composition of the room. A common way to express the combined effect of direct signal, reflected signals and background noise is the following[21]:

$$\tilde{x}(t) = s(t) + h_{RIR}(t) * s(t) + \eta(t)$$

where $s(t)$ denotes the direct signal, $h_{RIR}(t)$ denotes the room impulse response, and $\eta(t)$ denotes the background noise.

The proposed system estimates the reverberation signal

$$r(t) = h_{RIR}(t) * h_{Mic}(t) * s(t)$$

from the final audio recording[22] $y(t)$, assuming flat microphone response and negligible distortions in order to get rid or microphone impulse response, microphone distortion and transcoding distortion:

$$y(t) = s(t) + r(t) = s(t) + h_{RIR}(t) * s(t)$$

[21] $\tilde{x}(t) = \mathcal{F}^{-1}(S_2(f))$, see equation 3.2

[22] hence the presence of the microphone impulse response $h_{Mic}(t)$

The process of separating $r(t)$ from $y(t)$ is called de-reverberation: usually $h_{RIR}(t)$ can be modeled by a finite impulse response (FIR) filter, provided that the filter is of sufficient length. However, under blind de-reverberation it is not possible to measure or derive FIR filter response directly: to overcome this limitation a perceptually relevant estimate of the FIR filter clock is used, in order to estimate both the dry signal $\tilde{s}(t)$ and the reverberant signal $\tilde{r}(t)$.

The estimated reverberant signal is then used for feature extraction: the proposed scheme used Mel-Frequency Cepstral Coefficients (MFCCs) and Logarithmic Mel-Spectral Coefficients (LMSCs) in order to build a 48-dimensional feature space able to characterize the acoustic reverberation.

3.2.3.2 Classifier Training and Experimental Results

Performance of the proposed scheme has been tested using a data set containing 284 speech recording in nine different environments, made with four different microphones[23]: the input signal has been pre-emphasized before the de-reverberation. Then for each audio-frame of the estimated reverberant signal a 48-dimension feature vector is extracted and used for training and testing of a SVM classifier: 50% of recordings from each category were randomly selected for the training phase, and the rest 50% were used to verify the performance of the proposed scheme.

In the first experiment microphone dependent performance was tested: the SVM classifier was trained and tested using feature vectors extracted from the recording captured using the same microphone type. The average classification accuracy were between 92% and 94%, so the accuracy is actually independent from the microphone type.

In the second experiment microphone independent blind performance was tested: the SVM classifier was trained using feature vectors coming with a microphone, then tested with recordings coming from a completely different microphone, obtaining only a negligible ($<1\%$) performance loss.

In the third experiment a blind environment classification was tested: the SVM classifier was trained on a data set containing registration made in a small office, in a restroom, in an hallway and outdoors, and then tested on a data set containing the same acoustic environments, but not recorded in the same places of the training data set. Apart form large misclassification of the hallway to the restroom, both complex and concrete structures, all the other environments were correctly labeled.

In the fourth and last experiment the performance gain due to the de-reverberation were investigated: the SVM was trained and tested using feature vectors extracted from the unprocessed speech recordings: the average classification accuracies for with (and without) de-reverberation based identification systems were 94% (84%), 92% (86%), 93% (86%), 92% (86%).

This approach highlights the importance of the environment stage in the recording process, and the need to exploit it in actual environment classification. The system has yet to be tested with non-speech signal and with more microphone models in order to generalize these results; an unsupervised classifier should be tested in order to assess if this algorithm works with unknown acoustic environments.

[23]i.e. $M1$ & $M2$ were Behringer ECM8000, while $M3$ & $M4$ were left and right built-in Mic of ZOOM R16 Recorder

3.3 Editing/Tampering Detection

In this section first we will provide a brief overview of some audio forgery detection techniques, then we are going to introduce possible hybrid approaches concerning audio forensic that exploit both microphone and environment classification for editing/tampering detection.

3.3.1 Electrical Network Frequency

When an electronic equipment is used to record audio data, usually it captures not only the sound, but also the implicit electric characteristics of the device; if the recording device is AC-powered, it's not uncommon for it to record also the Electrical Network Frequency (ENF) due to the coupling of the powered circuit and the capturing circuit.

ENF is the distributed electricity generated by the power plants: power plants generates electricity, and the transmission of electricity from these power plants to the end-users is typically organized as a number of grids, in which power cables are directly connected. Since electricity is generated by mechanical devices the variations in source, pressure and the rotation speed of the coils affect the generated frequency and current: each power plant generates electricity with a designated utility frequency, and in order to provide almost-constant electric frequency to the users each power plant has to monitor and regulate the generated frequency. The inevitable adaptation delay induces a time-varying, slowly and slightly deviation around the specified utility frequency, that is distributed to the electrical grid. This variation makes the ENF suitable for audio forensics:

- The ENF is distributed only on the connected grids: this property can be used to localize the location of a recording device
- The ENF is time-varying. This property can be used to localize the time duration of the recorded audio clip
- The ENF is generated by the power plants: they can be trusted sources to provide ENF histories, and this makes matching the ENF in a given audio clip meaningful
- Even without the trusted ENF history for matching, the phase continuity of the ENF in the given audio clip can be used to detect whether the clip has been manipulated

According to the above characteristics, the typical usage is to match the extracted ENF components with the trusted ENF components: it they are not matched, either the time/location is wrong or the audio clip is tampered. Sometimes ENF cannot be matched because the ENF history for the specific power grid is missing; when this happens tampering can be detected with a certain probability with further analysis.

As shown by Nicolalde and Apolinario [10], a possible forgery detection approach could be to exploit the phase properties of the ENF in order to detect tampering or editing. When a tampering is made it is very unlikely that a person in the urge of forging an evidence - even with technical background - would be able not to introduce a slight discontinuity in the ENF.

This is especially true because even a well-trained ear would fail to recognize these slight variation, so without prior expertise in this subject, it's likely to ignore this very specific detail. The weakness of ENF based approaches is that ENF is not present in every recording, for instance in ones made by portable devices or electret microphones: in other cases where is not possible to extract the ENF even if it's present in the recording, because it's masked by other components of the audio stream .

3.3.2 Local Noise Level Estimation

In 2012 Pan et al. proposed an effective forgery detection method that detects splicing thanks to the estimation of local noise level [11].

This method relies on the audio kurtosis, which represents the peakedness of the distribution of the signal sampling values. Except for a few outliers, most of the kurtosis values in authentic audio files fall into a narrow range around the mean value: thanks to this property, a better estimate of signal noise level can be achieved.

In order to detect splicing, both a global noise level estimation and a local noise level estimation are needed: in splicing regions the local noise level gets higher, hence revealing a possible tampering.

This method has been tested both in a synthetic audio splicing forgery and in a realistic one. In the first test the signal spliced into the original file was an Average withe Gaussian Noise (AWGN): the true SNR values were 10 dB, 15 dB, 20 dB and 30 dB; the estimated means (and standard deviations) were 10.04 (0.13) dB, 15.01 (0.24) dB, 20.00 (0.41) dB and 30.73 (3.53) dB.

In the second test some words were substituted from an audio file into another one, and during the manipulation process the tampering section was carefully chosen so that the resulting sentence were still meaningful. Furthermore both volume and speed of the splicing audio components were tuned, in order to make the forged audio signal realistic: the detection results demonstrate that the individual splicing segments in the forged audio signal exhibit significant noise level differences, thus providing strong evidence of tampering.

Even if this method has to be further deployed in order to be totally reliable, this fast and blind local noise level estimation algorithm can be really useful in forgery detection, and has the advantage of not to require specific knowledge of the recording device or the file format.

3.3.3 Blind Channel Identification

In 2009 Gaubitch et al. proposed an algorithm that can identify accurately the magnitude spectrum of an unknown channel in noise-free conditions [12]. Even if this work was not devoted to forgery detection or audio authentication, it still provide a method that could be tested both in microphone and environment classification.

The objective of blind channel identification is to estimate the magnitude response of an unknown channel starting only from the recorded file. The method proposed relies on the Long Term Average Speech Spectrum (LTASS), so in a forensic contest it could be useful to validate court evidence. Within this work LTASS has been found by measurement from the complete training set of the

TIMIT database, and the average between male and female LTASS has been used, instead of the approximate formula defined in the ITU-T recommendation.

First experiments has been made on digitally generated channels, both in noise-free condition and with different kinds of noise[24]. Unfortunately, the estimation accuracy was reduced, and the degradation varied largely depending on the long term spectral characteristics of the noise.

Last experiments, instead, were made with real measured channels: the first one was a microphone in noise-free conditions, whose frequency response was estimated obtaining a 3.87 dB spectral distance; the second one was a noisy telephone network from NTIMIT, and the spectral distance between the estimated frequency response and the real one was 36.13 dB. In the second test it's interesting to see how the large estimation error can be attributed to the high frequency portion ($\geq$ 4 kHz), that usually is buried in noise on telephone networks.

In 2011, after two years, Gaubitch et al. proposed a second algorithm devoted to blind channel identification [13], somehow similar to the previous one, but based on a Gaussian Mixture Model (GMM) instead that on the LTASS. This approach requires less data in order to work, and at the same time provides a better estimate of the channel: the variation in estimation accuracy for different utterances is greatly reduced, and also the overall accuracy is improved by about 2 dB. Moreover, the GMM-based algorithm is more robust to noise compared to the LTASS-based one, especially for what concerns withe Gaussian Noise (WGN).

Blind channel identification algorithms such as the ones proposed by Gaubitch et al. [12, 13] could be effectively applied both on microphone classification and environment classification: the channel involved could be either the microphone coloring effect or the environment reverberant property, and the actual estimate could be adapted to be a feature vector.

3.3.4 Future Works

Future works in audio forgery detection should try to use different layers in order to perform a reliable tampering/editing detection: right now ENF based techniques and local noise level estimation based technique are able to perform splicing detection, but can only highlight possible suspect regions by the means of their borders.

A strong contribution could come from unsupervised microphone/device and environment classification systems: they could be used to validate a local splicing detection by checking the neighbor of the detected boundary for inconsistencies about the microphone or the environment. Equally, if microphone/device and environment classification is used as the main core of tampering detection, ENF based techniques and local noise level estimation based techniques could be used locally in order to refine the boundary within the region belonging to a source and the other one.

As we saw, apart from the first attempt by Kreatzer et al. [1], there's never been a study focusing on both microphone/device classification and environment classification, despite their correlation. A possible first step in this direction could be to evaluate how de-reverberation affects microphone classifica-

[24]i.e. babble noise, car noise and withe Gaussian Noise (WGN)

tion: it's true that the coloring effect of the microphone is usually stronger than the environment contribution, but it's possible that a previous de-reverberation could enhance the microphone classification accuracy, as well as provide useful information about the environment.

Chapter 4

Proposed Method

In the following chapter we will provide a complete description of our proposal for a new microphone classification algorithm.

For the sake of clarity, we will keep the logical design as much unrelated as possible from the implementation choices, which are not due to the algorithm itself: Section 4.1 is about the logical design, and is meant to be a reference for Section 4.2, which concerns our Octave implementation.

4.1 Logical Design

The classification framework is composed of several phases, each one related to the other one: in order to fully understand the relationship between them a general outline is provided, followed by in-depth details about each single phase previously stated.

4.1.0 Outline

Our strategy strongly relies on the last, recent work by Gupta et al. [13]: their research provides an algorithm for the blind estimation of the magnitude response of a channel, using the observations from a single microphone. Their proposal is meant to give a meaningful estimate of the channel present during the recording: in our work the channel is the microphone frequency response itself.

Our efforts were devoted to effectively exploit this estimates in order to achieve the desired microphone classification: we had to find a robust way to successfully classify different recording devices, relying only on the estimates provided by the channel estimation algorithm.

Even if until now clustering algorithms showed almost no effectiveness in the field of microphone classification, at first we focused on the k-means algorithm: this choice led us to create a feature vector which could successfully discriminate between pairs of devices with an high accuracy; a feature vector which was the starting point for the one used in the final classification algorithm.

The whole process can be split into several phases, that we will now describe briefly:

1. Training of a Gaussian Mixture Model of the Clean Speech:
   in order to successfully obtain a meaningful channel estimate, we need to train a GMM. This GMM provides a general model for the clean speech signal[1], starting from noiseless recordings of sentences produced by female and male speakers.

2. Blind Estimation of the Magnitude Response of the Acquisition Device:
   once the GMM has been obtained, we exploits its parameters in order to re-build the possible clean speech which originated the recording under examination; then, we achieve an estimate of the magnitude response by subtracting the power spectrum of this possible clean speech from the power spectrum of the actual recording.

3. Feature Vector Computation:
   once obtained all the channel estimates of the recordings in the testing set we have to compute a meaningful feature vector for each recording. The feature vector must be able to deal both with the noise from the environment and with the noise due to the content variability affecting the channel estimates; moreover, it must provide valuable information, even for compressed audio files.

4. Clustering or Classification:

   (a) Clustering: once the feature vectors are built, we are able to partition the test set into clusters: we associate each cluster to a single recording device, and it's finally possible to discriminate between them. In this context the range of each feature could be highly related to the final result, e.g when the mean squared distance is used with no normalization of each feature component: the larger the range of a single feature, the bigger its influence on the final distance and on the outcome of the clustering procedure.

   (b) Classification: as for the clustering, it's strictly related to the feature vector computation phase. In our framework the classification is also based on a closed-set assumption.

Our method proved to be effective both for the clustering and for the classification, and works on real recordings where also environmental noise and music are sometimes present, despite the assumption from [13] of a noiseless speech-only content. Moreover, it's been tested also with encoded[2] audio files, and proved to be sufficiently robust even against lossy processing.

Further in-depth details about each single phase of the outline are provided in the following sections.

[1]i.e. an ideal speech signal which has not undergone any spectral modification
[2]i.e. AMR, MP3, AAC

4.1.1 Training of GMMs for Clean Speech

Our training set is made of several sentences produced by two male and two female speakers. The sentences were acquired by an hi-quality microphone in a noiseless environment, and were originally sampled at 48 kHz.

Afterward, the recordings were down-sampled to 8 kHz; as we will see in Section 4.1.2, this sampling frequency is really important for us, since all the test recordings will be also down-sampled to 8 kHz.

In order to train the GMM, we tried to follow as much as possible the algorithm proposed by Gupta et al. [13]. The modified algorithm is the following:

1. HTK-MFCCs Computing:
   the first step is the computation of the Hidden Markov Model Toolkit (HTK) Mel-Frequency Cepstral Coefficients (MFCCs) [15, 16] instead of the classic MFCCs[3]. Let $s(n)$ denote one our training recording:

   (a) Pre-emphasis filtering of the speech content: $\tilde{s}(n) = s(n) - \alpha \cdot s(n-1)$, where α denotes the pre-emphasis coefficients. We put $\alpha = 0$, since in [13] no mention were made about pre-emphasis[4].

   (b) Windowing: $s_l(n) = windowing\,(\tilde{s}(n), 32ms, 50\%, hanning)$. The speech content is split in frames $s_l(n)$ of length 32ms, with an overlap of 50%, using an hanning window[5].

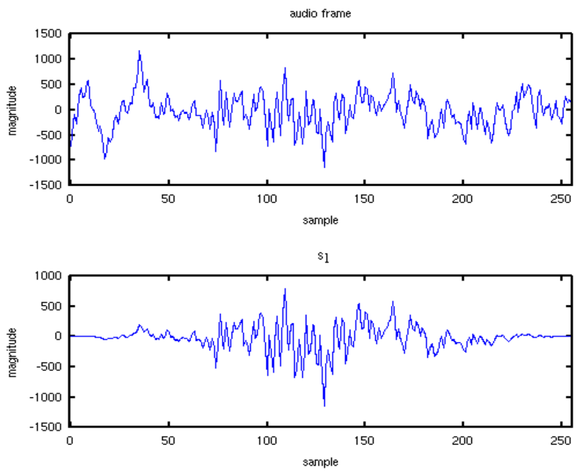


Figure 4.1: HTK-MFCCs: Windowing

[3]this operation allows the computation of smoother channel estimates, as noticed during the development and reported in appendix B.2.

[4]moreover, RASTA filtering integrates the speech pre-emphasis by itself [17]

[5]$hanning(n) = 0.5 \cdot \left(1 - cos\left(\frac{2\pi n}{N-1}\right)\right)$, where N denotes the window length

(c) Magnitude spectrum computation: $MAG_l(k) = |S_l(k)| = |\mathcal{F}(s_l(n))|$, where $\mathcal{F}(\cdot)$ denotes the Fourier transform.

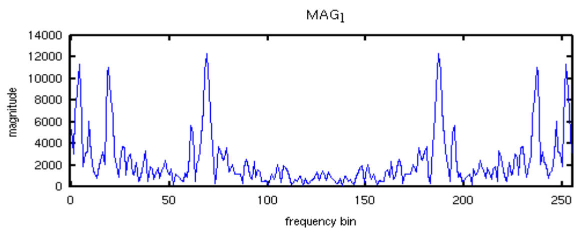


Figure 4.2: HTK-MFCCs: Magnitude Spectrum Computation

(d) Triangular filter bank application to the first half of the magnitude spectrum: $FBE_l = h_{triang} \cdot MAG_l\left(k \in \left[1\,; \frac{N_{fft}}{2}+1\right]\right)$, where h_{triang} denotes a triangular filter bank with uniformly spaced filters on Mel scale, and N_{fft} the number of points in the Fourier transform.

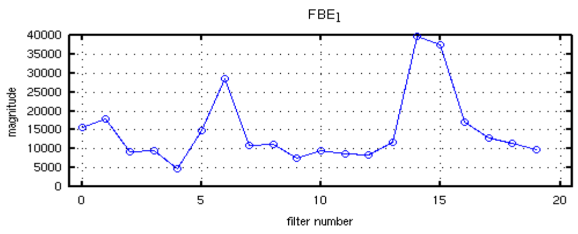


Figure 4.3: HTK-MFCCs: Filter Bank

(e) Conversion of the logarithm of the filter bank output into 13 MFCCs: $MFCC_l = dct^{-1}(log(FBE_l))$.

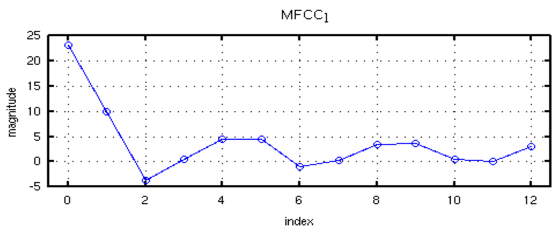


Figure 4.4: HTK-MFCCs: MFCCs computation

(f) HTK cepstral liftering of MFCCs with cepstral liftering parameter $L = 22$: $HTK\text{-}MFCC_l(k) = MFCC_l(k) \cdot \left(1 + \frac{1}{2}L \cdot sin\left((k-1)\frac{\pi}{L}\right)\right)$

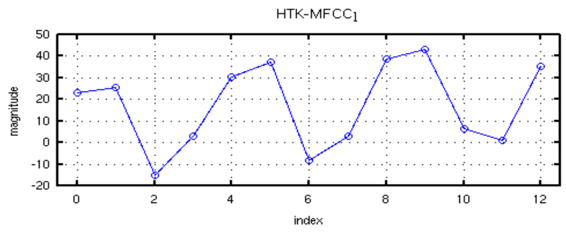


Figure 4.5: HTK-MFCCs: HTK cepstral liftering

2. RASTA filtering of the HTK-MFCCs:
the time series of the k-th HTK-MFCCs is RASTA filtered in the frequency domain itself, i.e.

$$RASTA\text{-}HTK\text{-}MFCC(k) = HTK\text{-}MFCC(k) * RASTAfilter$$

where $*$ denotes the convolution operation and

$$HTK\text{-}MFCC(k) = \left[HTK\text{-}MFCC_1(k), \ldots, HTK\text{-}MFCC_{N_{frames}}(k)\right].$$

$RASTAfilter$ is the filter defined in the z-domain by the following transfer function:

$$H_{RASTAfilter}(z) = 0.1 \cdot \frac{2 + z^{-1} - z^{-3} - 2z^{-4}}{1 - 0.94z^{-1}}.$$

3. Training of a 1024-mixture Gaussian Mixture Model:
the GMM is trained using as feature vectors all the RASTA-HTK-MFCCs computed excluding the coefficients corresponding to $k = 0$.
Let us denote with $rasta_{S,l}$ the RASTA-HTK-MFFCs of the l-th frame of the training file S; with the training we obtain

(a) the GMM parameters, i.e. the priors π_i, the means μ_i and the diagonal covariances Σ_i of each mixture.

(b) the relative mixture probability[6] of all the frames of each training file, defined as

$$p\left(z_i = 1 | rasta_{S,l}\right) = \frac{\pi_i \cdot \mathcal{N}\left(rasta_{S,l} | \mu_i, \Sigma_i\right)}{\sum_{m=1}^{M} \pi_m \cdot \mathcal{N}\left(rasta_{S,l} | \mu_m, \Sigma_m\right)}$$

where $z_i = 1$ denotes that the i-th mixture is generating the current frame.
The relative probabilities of the L_S frames available from the whole training set[7] are used to build the matrix $P_S \in \mathbb{R}^{M \times L_S}$, where $M = 1024$ denotes the number of mixtures.

[6]i.e.the probability that the feature vector $rasta_{S,l}$ belongs to the i-th mixture

[7]$L_S \neq N_{frames}$, since N_{frames} refers to a single audio file.

4. Computation of a normalized log power spectrum of each frame:
we normalize the log power spectrum of each training recording by subtracting its mean:

$$Z_{S,l} = log\left(|S_l|\right) - \frac{1}{N_{fft}} \sum_{k=1}^{Nfft} log\left(|S_l(k)|\right), \forall l$$

and compress all the normalized log power spectra in the matrix Z_S, where $Z_S \in \mathbb{R}^{L_S \times N_{fft}}$.

5. Computation of the average log spectrum of the speech:
finally, we combine both the relative mixture probabilities and the normalized log power spectra into a weighted average:

$$\hat{Z}_S = P_S \cdot Z_S.$$

In the average log spectrum of the speech matrix $\hat{Z}_S \in \mathbb{R}^{M \times N_{fft}}$ the i-th row represents the average log-spectrum corresponding to the i-th mixture of the GMM.

The average log spectrum of the speech $\hat{Z}_S$ that we obtained starting from the speech corpus previously described is shown in Figure 4.6.

The choice of the speech corpus used in order to train the GMM is highly relevant: e.g., in Figure 4.7 is shown the average log spectrum of the speech that we can obtain starting from the clean subset of the NOIZEUS speech corpus [14], a speech corpus that contains 30 IEEE sentences produced by three male and three female speakers. The thirty sentences are selected from the IEEE database so as to include all phonemes in the American English language; moreover, the sentences were originally sampled at 25 kHz and then down-sampled to 8 kHz. Finally, a pass-band filtering was performed, in order to simulate the common channel response present in everyday telephone lines.

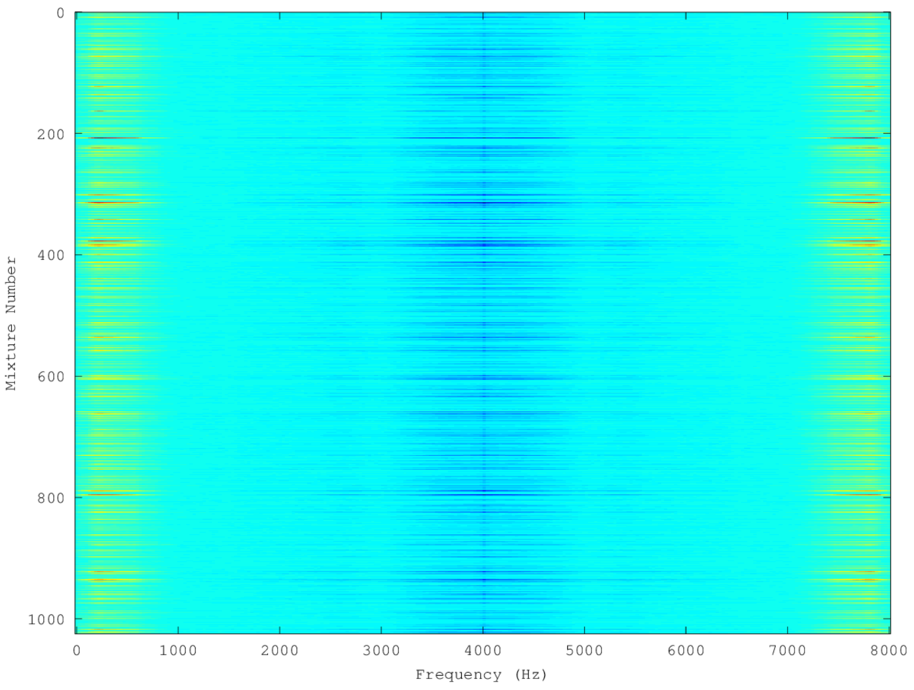


Figure 4.6: $\hat{Z}_S$ computed from our Self-Produced Speech Corpus

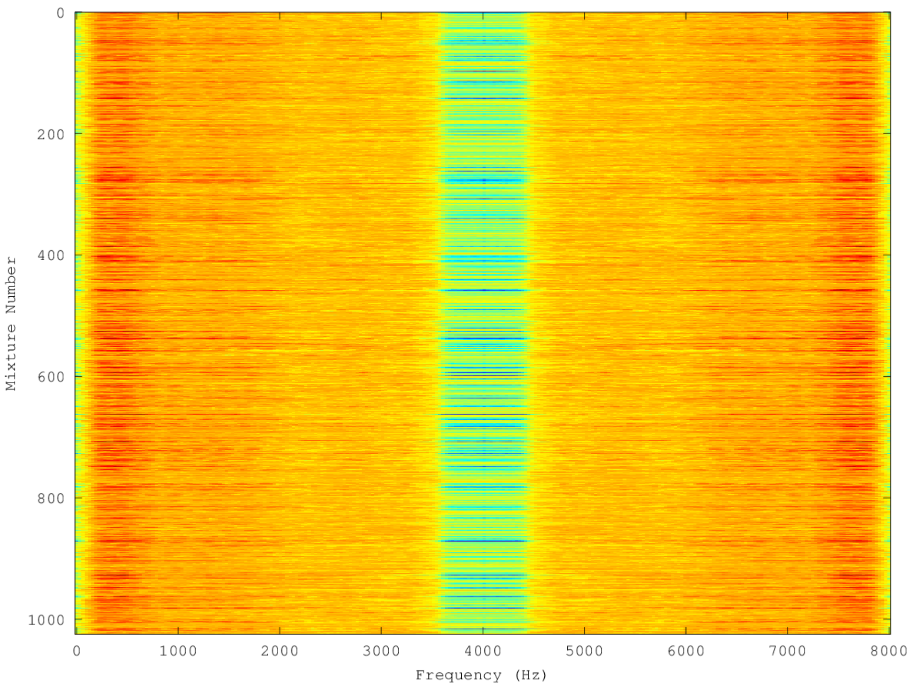


Figure 4.7: $\hat{Z}_S$ computed from the NOIZEUS Speech Corpus

The average log spectrum of the speech $\hat{Z}_S$ is a reference of an ideal speech signal which has not undergone any spectral modification: the RASTA filtering makes the HTK-MFCC independent from the channel[8], and allows us to build a GMM that represents every possible phoneme that could be present in a speech recording. In Figure 4.6 and 4.7 each row[9] represents the spectrum of a specific phoneme, while each column represents a specific frequency bin.

4.1.2 Blind Estimation of the Channel Response

The channel estimation of the magnitude of the channel response relies both on the parameters (π_i, μ_i, Σ_i) of the Gaussian Mixture Model previously trained, and on the average log spectrum of the speech $\hat{Z}_S$. In order to estimate the magnitude response of the channel present, we have to process the testing recording $x(n)$ as follows:

1. (optional) Re-sampling of the test recording down to Fs=8000: in order to be able to compare our model of the ideal speech with a recording, first we have to down-sample the recording - if needed, of course - to the same sampling frequency of the audio recordings present in the training set, i.e 8000 Hz. The resampled version of $x(n)$ is $\tilde{x}(n)$

2. Windowing: $x_l(n) = windowing\,(\tilde{x}(n),\, 32ms,\, 50\%,\, hanning)$. The recording is split in frames $s_l(n)$ of length 32ms, with an overlap of 50%, using an hanning window, as in the training phase.

3. RASTA-HTK-MFCCs Computing: following the same steps shown in Section 4.1.1, in order to obtain the correct feature vector $rasta_{X,l}$ of each frame we have to

   (a) Compute the 13 HTK-MFCCs.

   (b) RASTA filter the time series of each HTK-MFCC.

   (c) Drop the 0-th RASTA-HTK-MFCCs

4. Computation of a normalized log power spectrum of each frame: as before, we normalize the log power spectrum of the recording by subtracting its mean

$$Z_{X,l} = log\,(|X_l|) - \frac{1}{N_{fft}} \sum_{k=1}^{Nfft} log\,(|X_l(k)|)\,,\, \forall l$$

   and compress all the normalized log power spectra in the matrix $Z_X \in \mathbb{R}^{L_x \times N_{fft}}$. $X_l = \mathcal{F}\,(x_l)$, where $\mathcal{F}\,(\cdot)$ denotes the Fourier transform of length N_{fft}.

5. Computation of the relative mixture probability of each frame: using the parameters (π_i, μ_i, Σ_i) of the Gaussian Mixture Model previously trained, we compute the probability that the feature vector $rasta_{X,l}$ belongs to the i-th mixture with

$$p\,(z_i = 1|rasta_{X,l}) = \frac{\pi_i{\cdot}\mathcal{N}\,(rasta_{X,l}|\mu_i, \Sigma_i)}{\sum_{m=1}^{M} \pi_m{\cdot}\mathcal{N}\,(rasta_{X,l}|\mu_m, \Sigma_m)}$$

[8]e.g. the speaker's gender and peculiar timbre, the microphone involved, the environment reverberation

[9]i.e. each GMM mixture

and compress this information into the matrix $P_X \in \mathbb{R}^{M \times L_X}$, where L_X denotes the number of frames of the initial recording $x(n)$.

6. Estimate of the real power spectrum of the recording: using the relative mixture probability matrix P_X as a selection matrix on the average log spectrum of the speech $\hat{Z}_S$, we are able to estimate the non-filtered ideal speech source of the test recordings

$$\tilde{Z}_X = P_X^t \cdot \hat{Z}_S$$

where the superscript t denotes the matrix transpose.

7. Computation of the Channel Estimate: we can retrieve our desired estimate by subtracting the real normalized log power spectrum from the estimate of its real source. In order to to that, we have to

   (a) Normalize between 0 and 1 both $\tilde{Z}_X$ and Z_X, applying

$$\begin{cases} \tilde{Z}_{X,norm} = \underset{[0,1]}{norm}(\tilde{Z}_X) \\ Z_{X,norm} = \underset{[0,1]}{norm}(Z_X) \end{cases}, \underset{[0,1]}{norm}(\cdot) = \frac{(\cdot) - min(\cdot)}{max(\cdot) - min(\cdot)} \tag{4.1}$$

   (b) Compute the estimate $\hat{h}$ of the magnitude response of the original channel h, by applying

$$\hat{h} = \frac{\left(Z_{X,norm} - \tilde{Z}_{X,norm}\right)^t \cdot \mathbf{1}}{L_X} \tag{4.2}$$

   where $\mathbf{1}$ is a $L_X \times 1$ vector with all elements equal to one.

A possible estimate of the magnitude response of the channel is shown in Figure 4.8:

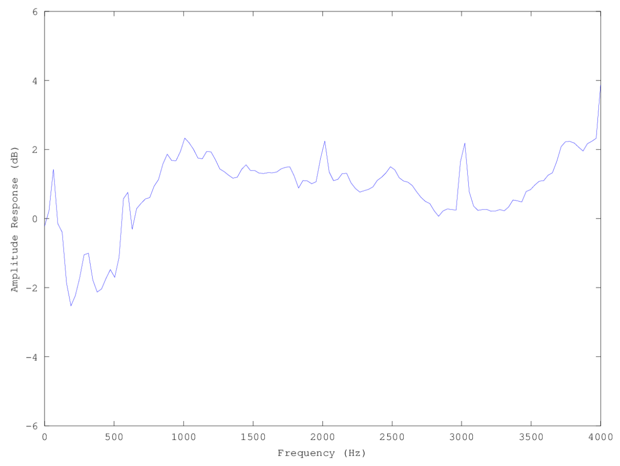


Figure 4.8: Single Channel Estimate from a Dell Latitude D630

Of course, the algorithm for the blind channel estimation is still affected by the content: as reported in [12, 13] this happens because the speakers involved in the recordings of the GMM are different from those involved in the test recording, and when the recordings are affected by some noise; moreover, as we will see in Chapter 5, the content of our test recordings is intentionally far from being the ideal one, thus enhancing the differences between channel estimates from different recordings of the same device.

In Figure 4.9 and 4.10 we can see how, despite such a noise in the channel estimates, several amplitude responses from a single device present a unique pattern regardless of the content, and how the general shape - i.e. the peculiar coloring effect of a single device - changes for different device:

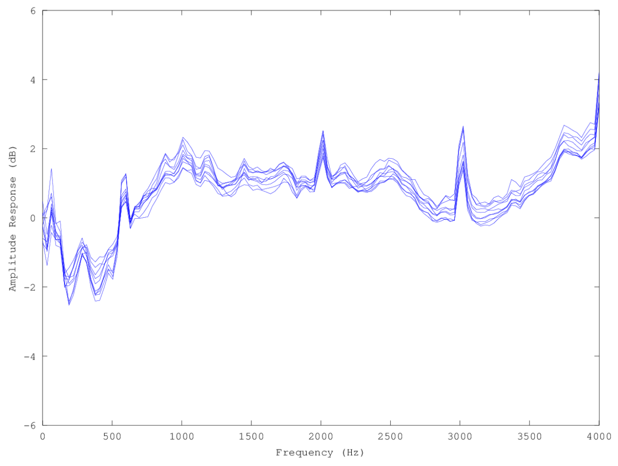


Figure 4.9: Multiple Channel Estimates from a Dell Latitude D630

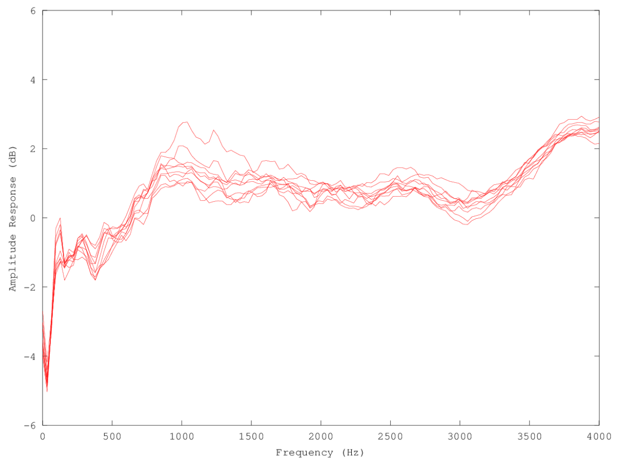


Figure 4.10: Multiple Channel Estimates from an iPhone 3gs

4.1.3 Feature Vector Computation

Since we want to be able to differentiate between different devices, we have to compute a meaningful feature vector for each recording, starting from the information available. The description below concerns the feature vector involved during the following SVM classification: for a similar description concerning the feature vector developed through the clustering attempts please refer to the appendix B.

In order to achieve a robust classification algorithm, for each audio file x we compute three multidimensional features, denoted in the following as f_1, f_2 and f_3.

These three features, despite being similar to each other, have different meanings: in f_1 we keep all the information obtained by the channel estimation algorithm; f_2 is meant to be a descriptor of the correlation between the channel estimate and the original spectrum of the audio file; finally, f_3 defines the properties of the approximated spectrum of the audio file x.

Let $Z_{X,norm}$ be the normalized power of x defined in equation 4.1, $\hat{h}$ the channel estimate of x defined in 4.2, K the number of frequency bins on the positive axis and

$$\hat{p} = \frac{(Z_{X,norm})^t \cdot \mathbf{1}}{L_X},$$

where $\mathbf{1}$ is a $L_X \times 1$ vector with all elements equal to one. f_1 is processed in order to fully enhance the information obtained by the channel estimation algorithm, as follows:

1. Apply a variable gain on the sum between the channel estimate of x and the average value of the approximated normalized power of x, dependent on the power of $\hat{h}$:

$$h_1 = \frac{\hat{h} + \overline{(\hat{p})}}{\left\|\hat{h}\right\|^2},$$

where $\overline{(\hat{p})}$ denotes the average value of $\hat{p}$, and $\|\cdot\|$denotes the l^2-norm.

2. Compute the first derivative $h_1^{'}$ of h_1, as the inter-sample difference of h_1:

$$h_1^{'}(k) = h_1(k) - h_1(k-1)\,, \forall k \in [2, K]\,.$$

3. Compute the second derivative $h_1^{''}$ of h_1, as the inter-sample difference of $h_1^{'}$:

$$h_1^{''}(k) = h_1^{'}(k) - h_1^{'}(k-1)\,, \forall k \in [3, K]\,.$$

4. Build the feature f_1 by collecting the processed data:

$$f_1 = \left[h_1\,, h_1^{'}\,, h_1^{''}\right].$$

In order to compute f_2 we follow a slightly different procedure, that is meant to fully enhance the description of the correlation between the channel estimate and the original spectrum of the audio file:

1. Compute h_0, as the entry-wise division[10] between the channel estimate of x and the approximated normalized power of x:
$$h_0 = \hat{h}\,./\hat{p}. \tag{4.3}$$
2. Apply a variable gain on h_0, dependent on its power:
$$h_2 = \frac{h_0}{\|h_0\|^2}.$$
3. Compute the first derivative $h_2^{'}$ of h_2, as the inter-sample difference of h_2:
$$h_2^{'}(k) = h_2(k) - h_2(k-1)\,,\,\forall k \in [2, K]\,.$$
4. Compute the second derivative $h_2^{''}$ of h_2, as the inter-sample difference of $h_2^{'}$:
$$h_2^{''}(k) = h_2^{'}(k) - h_2^{'}(k-1)\,,\,\forall k \in [3, K]\,.$$
5. Apply a variable gain on $h_2^{'}$ and $h_2^{''}$, dependent on the power of each vector:
$$\begin{cases} \widetilde{h_2^{'}} = \frac{h_2^{'}}{\|h_2^{'}\|^2} \\ \widetilde{h_2^{''}} = \frac{h_2^{''}}{\|h_2^{''}\|^2} \end{cases}.$$
6. Build the feature f_2 just by collecting the processed data:
$$f_2 = \left[h_2\,,\,\widetilde{h_2^{'}}\,,\,\widetilde{h_2^{''}}\right].$$

f_3 defines the properties of the approximated spectrum of the audio file x. Other than to ehnhace its descriptive power, the processing of $f_3$is also meant to eliminate as much as possible redundant information between f_1 and f_3, and to reduce the noise due to the content of the recording: $\hat{p}$, that is an approximation of the spectrum, is generated both by the microphone frequency response and by the content, and we must ensure that this influence is not strong enough to affect by any means the outcome of the classification. The procedure can be described as follows:

1. Apply a variable gain on the sum between the approximated normalized power of x and the average value of the channel estimate of x, dependent on the power of $\hat{P}$:
$$h_3 = \frac{\hat{p} + \overline{\left(\hat{h}\right)}}{\|\hat{p}\|^2}.$$
where $\overline{\left(\hat{h}\right)}$ denotes the average value of $\hat{h}$, and $\|\cdot\|$denotes the l^2-norm.

[10] $C = A\,./B \Longrightarrow C(i,j) = A(i,j)/B(i,j)\,\forall(i,j),\ A, B, C \in \mathbb{R}^{I \times J}$.

2. Normalize h_3 between 0 and 1:

$$h_{3,norm} = \underset{[0,1]}{norm}\left(h_3\right).$$

where $\underset{[0,1]}{norm}\left(\cdot\right)$ is the same function defined in equation 4.1.

3. Compute the first derivative $h_3^{'}$ of h_3, as the inter-sample difference of $h_{3,norm}$:

$$h_3^{'}(k) = h_{3,norm}(k) - h_{3,norm}(k-1)\,,\ \forall k \in [2, K]\,.$$

4. Compute the second derivative $h_3^{''}$ of h_3, as the inter-sample difference of $h_3^{'}$:

$$h_3^{''}(k) = h_3^{'}(k) - h_3^{'}(k-1)\,,\ \forall k \in [3, K]\,.$$

5. Compute h_4, which contains the absolute value of each component of $h_3^{''}$:

$$h_4(k) = \left|h_3(k)\right|.$$

6. Apply a variable gain on $h_3^{'}$ and h_4, dependent on the power of each vector:

$$\begin{cases} \widetilde{h_3^{'}} = \frac{h_3^{'}}{\|h_3^{'}\|^2} \\ \\ \widetilde{h_4} = \frac{h_4}{\|h_4\|^2} \end{cases}$$

7. Normalize between 0 and 1 both $h_3^{'}$ and h_4:

$$\begin{cases} \widetilde{h_{3,norm}^{'}} = \underset{[0,1]}{norm}\left(\widetilde{h_3^{'}}\right). \\ \\ \widetilde{h_{4,norm}} = \underset{[0,1]}{norm}\left(h_4\right). \end{cases}$$

8. Build the feature f_2 just by collecting the processed data:

$$f_3 = \left[h_{3,norm}\,,\ \widetilde{h_{3,norm}^{'}}\,,\ \widetilde{h_{4,norm}}\right].$$

The final feature vector f, of course, is computed by collecting together the three features:

$$f = [f_1\,,\ f_2\,,\ f_3]\,. \tag{4.4}$$

A visual comparison of the three features is present in Figure 4.11, where the x-axis represents the dimension index; the features reported refer to the audio file whose channel estimate is showed in Figure 4.8.

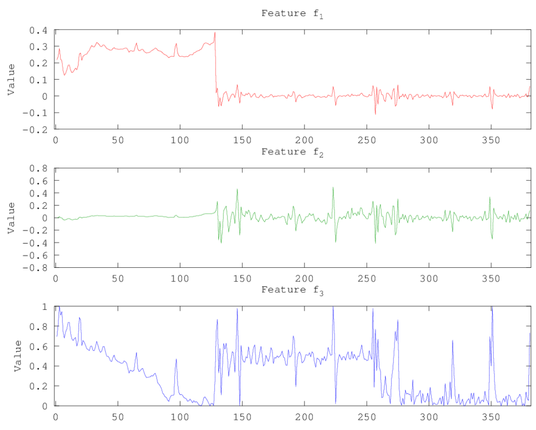


Figure 4.11: Comparison of Feature f_1, f_2, and f_3

4.1.4 Clustering

As told in Section 4.1.0, before focusing on the classification we tried to partition the feature vectors from each test recording into clusters. This has been both a key point for the development of the final feature vector and a very hard task, and required also a strong assumption, i.e. the previous knowledge of the number of recordings from each device.

Our clustering is accomplished by a variation of the well-known k-means clustering algorithm, and uses two alternative feature vectors $f_3^{'}$ and $f_4^{'}$, whose evolution is reported in appendix B. Let us denote with $f_{3,i}^{'}$ and $f_{4,i}^{'}$ the feature vectors $f_1^{'}$ and $f_2^{'}$ of the i-th recording, with N_k the number of recordings from the k-th device, with N_{test} the total number of test recordings, with N_{kmeans} the total number of repetition of the k-means, and with $IDX \in \mathbb{R}^{1\times N_{test}}$ the vector containig the clustering labels returned by the kmeans. The clustering algorithm is the following:

1. Build the data-set $F_1 = \left\{ f_{1,1}^{'}\,,\, f_{1,2}^{'}\,,\, \ldots\,,\, f_{1,N_{test}}^{'} \right\}$.

2. Let j represent the number of repetitions past the current one; ue to the kmeans being sensitive to the initial positions of the centroids we chose to alternate between complete random initializations and initializations from sampled starting points:

$$[IDX\,,\, kmeans\, score] = \begin{cases} kmeans(F_1,\;\, random\, starting\, point) & j\, is\, even \\ kmeans(F_1,\;\, sampled\, starting\, point) & else \end{cases}$$

3. If IDX is a possible configuration, i.e. each cluster k has the correct number of elements N_K - a condition that doesn't imply the correctness of the classification - then

   (a) if IDX is a new[11] configuration, set $score(IDX) = kmeans\, score$ and $votes(IDX) = 1$.

[11]e,g $IDX = [1, 2, 2, 1]$ and $IDX = [2, 1, 1, 2]$ are the same, like $IDX = [1, 3, 2]$ and $IDX = [3, 1, 2]$, but $IDX = [1, 1, 2, 2]$ and $IDX = [1, 2, 2, 1]$ are different

   (b) else, set $votes(IDX) = votes(IDX) + 1$.

4. If $j < N_{kmeans} \wedge \sum_i votes(IDX_i) < \frac{N_{kmeans}}{10}$ repeat from point 2.

5. If $\sum_i votes(IDX_i) \neq 0$, i.e. the feature vector $f_1^{'}$ was able to find a valid data partitioning, skip forward to point 10.

6. Build the data-set $F_2 = \left\{ f_{2,1}^{'} \,,\, f_{2,2}^{'}, \, \dots \,,\, f_{2,N_{test}}^{'} \right\}$.

7. Let j represent the number of repetitions past the current one:

$$[IDX\,,\, kmeans\,score] = \begin{cases} kmeans(F_2,\ random\,starting\,point) & j\,is\,even \\ kmeans(F_2,\ sampled\,starting\,point) & else \end{cases}$$

8. If IDX is a possible configuration, then

   (a) if IDX is a new configuration, set $score(IDX) = kmeans\,score$ and $votes(IDX) = 1$.

   (b) else, put $votes(IDX) = votes(IDX) + 1$.

9. If $j < N_{kmeans} \wedge \sum_i votes(IDX_i) < \frac{N_{kmeans}}{10}$ repeat from point 7.

10. Compute the final score:

$$score_{final}(IDX_i) = \alpha \cdot \frac{votes(IDX_i)}{\sum_k votes(IDX_k)} + (1-\alpha) \cdot \frac{min(score(IDX_i))}{score(IDX_i)}; \ \forall i \tag{4.5}$$

   where $\alpha \in [0;1]$ selects the trade-off between relying on the number of classifications that chose the i-th possible partitioning, or on its k-means score.

11. Select the final partitioning:

$$IDX_{best} = IDX_i \,|\, score(IDX_i) = \max_k \{score(IDX_k)\}\,.$$

As we saw, our classification algorithm at first tries to work only with the feature vectors $f_1^{'}$: if it's not possible for the feature vector $f_1^{'}$to find a valid data partitioning due to the presence of local minima, the same procedure is held working only with a different feature vector $f_2^{'}$.

This is done because, despite the different initialization policies used in order to address the problem of the dependence of the k-means result from the starting point, sometimes the noise on the high frequency component is simply too strong, and doesn't allow us to find a possible partitioning. If both feature vectors $f_1^{'}$ and $f_2^{'}$are not sufficient to achieve a possible classification, we mark the test set as unlabeled.

Finally, in order to achieve the desired microphone classification, we associate each cluster to a single recording device, following the content of IDX_{best}.

4.1.5 Classification

The classification is performed with a Support Vector Machine (SVM) in a closed-set fashion. Let us denote with $f_{i_k k} \in \mathbb{R}^{N_{features}}$ the feature vector f of the k-th device computed from its i-th recording as defined in equation 4.4, with N_k the number of recordings from the k-th device, with N_{device} the total number of devices, with l_k the class-label of the k-th device and with N_{class} the number of classes that we desire to classify.

We will now split the training and the testing phase in two distinct parts. The training phase procedure is the following:

1. Build the data-set and assign the proper labels:

$$\left\{ \begin{array}{l} F_{training} = \{f_{i_k k}\}\,; \forall i_k \in \{1, N_k\} \\ \\ L_{training} = \{l_k \cdot \mathbf{1}_{N_k}\} \end{array} \right. \quad \forall k \in \{1, N_{device}\},$$

   where $\mathbf{1}_{N_k}$ is a $N_k \times 1$ vector with all elements equal to one.

2. Feature Selection:
   let us denote with $f_{ji_k k}$ the j-th component of the feature vector $f_{i_k k} \in \mathbb{R}^{N_{features}}$; for each $f_{i_k k}$ the feature selection[12] returns

$$f^{select}_{i_k k} = \{f_{ji_k k}\}\,;\, \forall j \in J^{select} \subseteq \{1, N_{features}\}\,.$$

3. Dimension Normalization:
   after the feature selection, we need to perform a normalization of each dimension of the feature vector, i.e.

$$f^{norm}_{ji_k k} = \frac{f^{select}_{ji_k k} - \underset{i_k k}{min}\left(f^{select}_{ji_k k}\right)}{\underset{i_k k}{max}\left(f^{select}_{ji_k k}\right) - \underset{i_k k}{min}\left(f^{select}_{ji_k k}\right)}(b-a)+a;\, \forall j \in J^{select},$$

   where $a < b$. The most common values of a and b are $a = 0$, $b = 1$ and $a = -1$, $b = 1$. The dimension normalization returns the following values:

$$\left\{ \begin{array}{l} F^{norm}_{training} = \left\{f^{norm}_{ji_k k}\right\} \\ \\ min^j = \underset{i_k k}{min}\left(f^{select}_{ji_k k}\right) \\ \\ max^j = \underset{i_k k}{max}\left(f^{select}_{ji_k k}\right) \\ \\ a_{training} = a \\ \\ b_{training} = b \end{array} \right. ;\, \forall j \in J^{select}.$$

4. SVM training:
   the SVM training requires to select a proper kernel function and - since

[12]details about the feature selection specific for the final implementation can be found in Section 4.2.5.3.

we will use a multi-class SVM - to choose between different training algorithms, such as one-versus-rest or one-versus-one. After this we can perform the training of the SVM, which returns

$$Model_{SVM} = training\left(F^{norm}_{training}\,,\, L_{training}\,,\, kernel\,,\, algorithm\right).$$

For details about the specific settings involved during this last step please refer to Section 4.2.

The classification phase proceeds in a similar fashion as the previous one:

1. Build the data-set:

$$F_{testing} = \{f_{i_k k}\}\,; \forall i_k \in \{1, N_k\}\, \forall k \in \{1, N_{device}\}.$$

2. Feature Selection:
let us denote with $f_{ji_k k}$ the j-th component of the feature vector $f_{i_k k} \in \mathbb{R}^{N_{features}}$; for each $f_{i_k k}$ the feature selection returns

$$f^{select}_{i_k k} = \{f_{ji_k k}\}\,;\, \forall j \in J^{select} \subseteq \{1, N_{features}\}\,.$$

3. Dimension Normalization:
after the feature selection, we need to perform a normalization of each dimension of the feature vector, by using the parameters computed during the training phase

$$f^{norm}_{ji_k k} = \frac{f^{select}_{ji_k k} - min^j}{max^j - min^j}\left(b_{training} - a_{training}\right) + a;\ \forall j \in J^{select}.$$

This time the normalization phase returns only

$$F^{norm}_{testing} = \left\{f^{norm}_{ji_k k}\right\},$$

i.e. a set of $N_{testing}$ vectors of dimension $1 \times J^{select}$.

4. SVM classification:
finally, the SVM can perform the classification:

$$L_{testing} = training\left(F^{norm}_{testing}\,,\, Model_{SVM}\right).$$

Where $L_{testing}$ is a vector containing the desired label.

4.2 Implementation Details

For each step of the algorithm stated in the previous section we will now focus on specific implementation choices, following the same outline. This implementation choices, even if somehow related to the logical design, are mainly due to our general framework, which has been developed in GNU Octave [21], as reported in appendixA.

4.2.1 Training of the GMM of the Clean Speech

During the training of the GMM of the clean speech, the most important issue concerned the RASTA filtering [17]. The RASTA filter is an IIR filter with the transfer function

$$H_{RASTA}(z) = 0.1 \cdot z^4 \cdot \frac{2 + z^{-1} - z^{-3} - 2 \cdot z^4}{1 - 0.94 \cdot z^{-1}}.$$

The issue arose due to the length of the filter impulse response: this problem is known, and is usually overcome by setting the initial state IS of the filter using

$$IS = x\,(1) \cdot (-1\,,\, -1\,,\, 1\,,\, 1)^t\,,$$

where $x(1)$ is the first sample of the sequence that we are going to filter, i.e. for each i-th HTK-MFCC of the j-th audio file we have:

$$\begin{cases} IS_{ij} = HTK\,MFCC_{ij}(1) \cdot (-1\,,\, -1\,,\, 1\,,\, 1)^t \\ \\ RASTA - HTK\,MFCC_{ij} = filter\,(H_{RASTA}(z)\,,\, HTK\,MFCC_{ij}\,,\, IS_{ij}) \end{cases}$$

where the function $filter\,(H(z)\,,\, data\,,\, IS)$ filters the signal $data$ with the transfer function $H(z)$, and with the additional constraint of the initial state IS.

Unfortunately, Octave doesn't allow us to impose such a condition: in Figure 4.12 we can see the RASTA-HTF MFCCs obtained directly applying $H_{RASTA}(z)$ with no constraints.

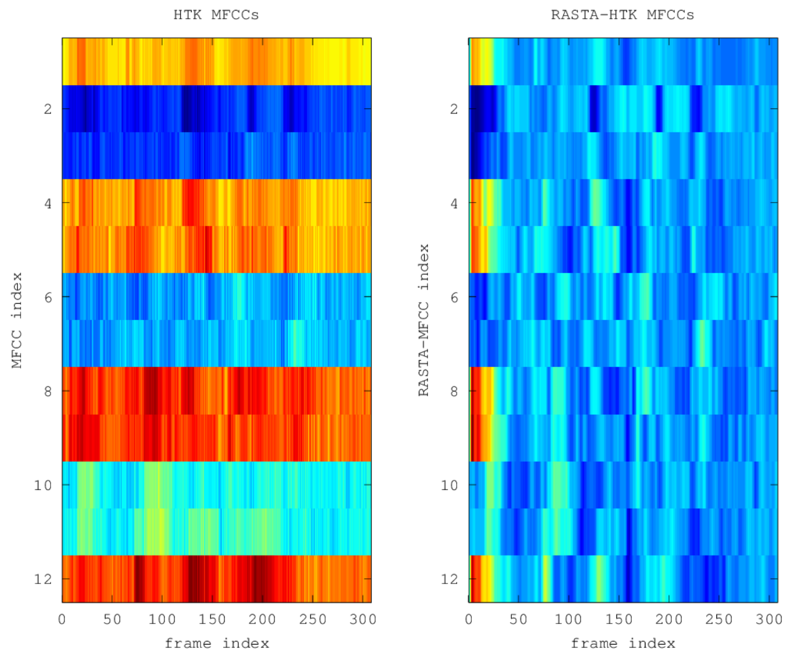


Figure 4.12: RASTA filtering without the initial condition IS

Of course such a behavior is undesirable, since there's a huge difference between the first frames and the following ones.

In order to overcome this fault without altering neither the content nor the Octave built-in function - for the sake of portability - we used the following procedure for the RASTA filtering:

1. Replicate the i-th HTK-MFCC time series $HTK\,MFCC_{ij}$ of the j-th audio file 10 times and obtain $HTK\,MFCC\_CAT_{ij}$.

2. Apply the RASTA filter without any constraint on the initial state directly on the new time series:

   $$R-HTK\,MFCC\_CAT_{ij} = filter\left(H_{RASTA}(z)\,,\,HTK\,MFC\_CAT_{ij}\right).$$

3. Select the last $L_{HTK\,MFCC_{ij}}$ samples of $R-HTK\,MFCC\_CAT_{ij}$, where $L_{MFCC_{ij}}$ is the length of $HTK\,MFCC_{ij}$, in order to obtain the final coefficients:

   $$RASTA\text{–}HTK\,MFCC_{ij}\,(k) = R\text{–}HTK\,MFCC\_CAT_{ij}\left(k + 9 \cdot L_{MFCC_{ij}}\right),$$

   where $k \in \left[0\,,L_{MFCC_{ij}} - 1\right]$.

In Figure 4.13 we can the RASTA-HTK-MFCCs obtained with this different algorithm, which allows us to avoid the faulty behavior which was previously occurring on the first frames, and to correctly apply the filter regardless of the frame index:

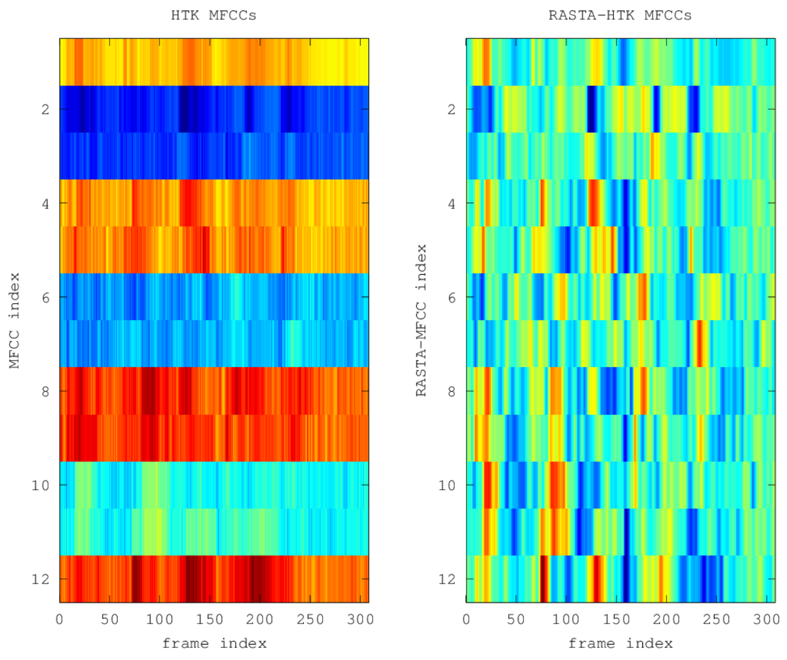


Figure 4.13: RASTA filtering with the initial condition IS

4.2.2 Blind Estimation of the Channel Response

As we saw in equation 4.2, in order to compute our channel estimate we require to compute the difference between $Z_{X,norm}$ and $\tilde{Z}_{X,norm}$. A visual representation of the two log power spectrum can be found in Figure 4.14:

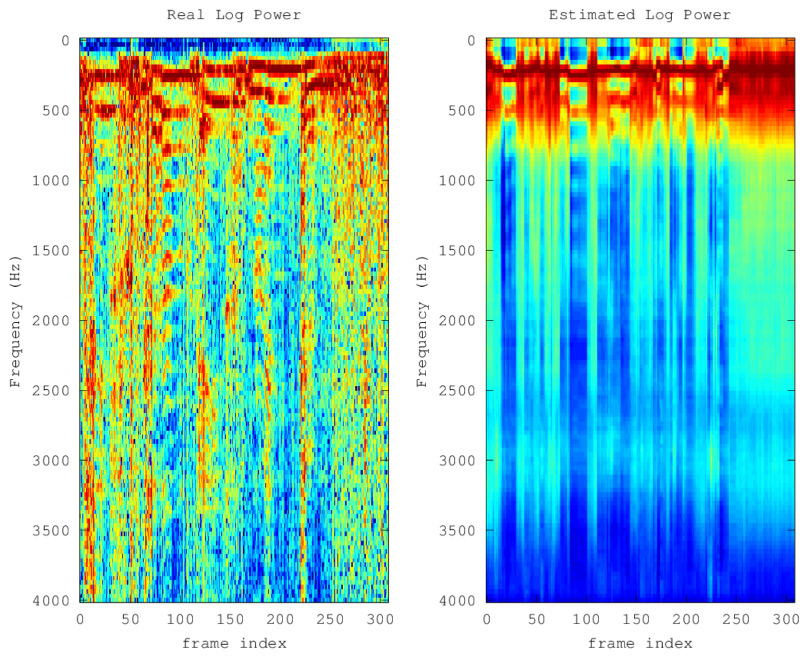


Figure 4.14: $Z_{X,norm}$ and $\tilde{Z}_{X,norm}$ in Gupta et al. [13]

We found that is possible to achieve a less noisy estimation[13] by selecting only a subset of the original frames: let $n \in [1\,,\, N_{frames}] = N$ be the frame index and $k \in [1\,,\, N_{fft}] = K$ be the frequency bin index, where N_{frames} and N_{fft} are the total numbers of the frames and of the frequency bins. The selection procedure is the following:

1. Select a subset of frame indexes:

$$\bar{n} \in Selection \iff \underset{k \in K}{max}\left(|X\left(\bar{n}, k\right)|\right) \leq \tau \cdot \underset{n \in N}{max}\,\underset{k \in K}{max}\left(|X\left(n, k\right)|\right),$$

   where X is the STFT of the audio file x, and $\tau = 0.1$ is a user-defined threshold.

2. Build the matrices $Z_{X,norm}^{Selection}$ and $\tilde{Z}_{X,norm}^{Selection}$ by using only the frames in the selection, i.e

$$\begin{cases} z \in Z_{X,norm}^{Selection} \iff z(n,k) \in Z_{X,norm} \wedge n \in Selection \\ \tilde{z} \in \tilde{Z}_{X,norm}^{Selection} \iff \tilde{z}(n,k) \in \tilde{Z}_{X,norm} \wedge n \in Selection \end{cases} ;\ \forall n \in N\,,\ \forall k \in K.$$

[13] i.e. an estimation with a lower distance between channel estimates from different recordings acquired by the same device.

$Z_{X,norm}^{Selection}$ and $\tilde{Z}_{X,norm}^{Selection}$ are both matrices of size $N_{Selection} \times N_{fft}$, where $N_{Selection}$ is the total number of selected frames. The algorithm stated in Section 4.1.2 is left unchanged except for the equation 4.2, that is replaced by

$$\hat{h} = \frac{\left(Z_{X,norm}^{Selection} - \tilde{Z}_{X,norm}^{Selection}\right)^t \cdot \mathbf{1}}{N_{Selection}} \tag{4.6}$$

where $\mathbf{1}$ is a $N_{Selection} \times 1$ vector with all elements equal to one. A visual representation of the two log power spectrum can be found in Figure 4.15:

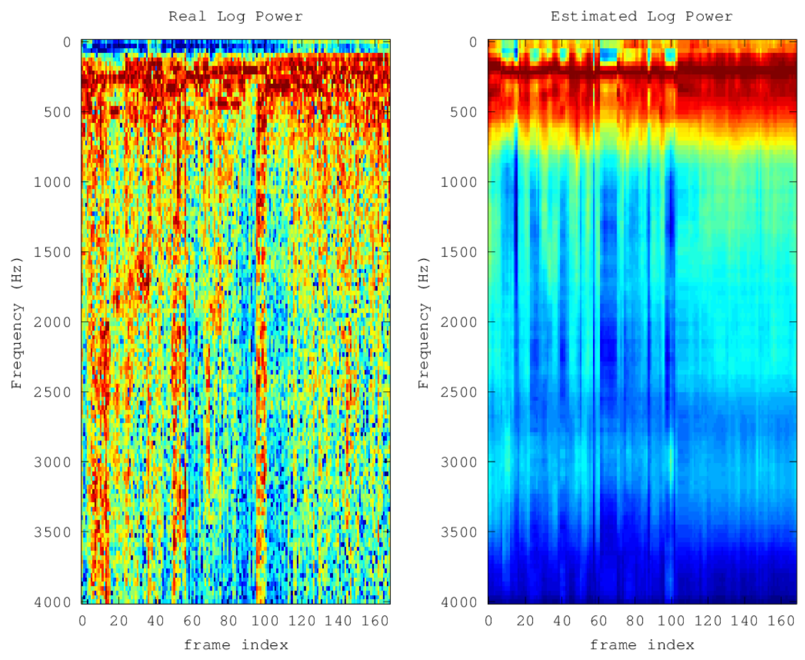


Figure 4.15: $Z_{X,norm}^{Selection}$ and $\tilde{Z}_{X,norm}^{Selection}$ in our proposal

As we can see, our selection algorithm reduces the number of frames where the speech content is present, so that the channel estimate is computed relying mostly on the silent parts of the audio files, where the influence of the coloring effect of the microphone is higher[14].

We chose not to include this selection in the algorithm discussed in 4.1.2 because we found that the effectiveness of this selection is somehow related to the content: if the content includes not only direct speech but also some playback from a loudspeaker, the loudspeaker channel influence is so strong that the final classification by the means of the SVM has a noticeable drop in accuracy. However, the presence of the loudspeaker violates our original assumption about the microphone frequency response being the only channel present, and for sources not involving a loudspeaker our proposed selection procedure is extremely useful, so that we chose to include it in the final version whose results

[14] i.e. the assumption exploited by [8]

will be provided in Chapter 5. For information about the performance of the SVM classifier without this selection refer to appendix C.

4.2.3 Feature Vector Computation

The feature vector computation proceeds as detailed in 4.1.3. The only slight difference is that, for the sake of numerical stability, 4.3 has been replaced with

$$h_0 = \hat{h} \,./ \left(\hat{P} + \epsilon \cdot \mathbf{1}^t\right),$$

where $\mathbf{1}$ is a row vector of the same length of $\hat{P}$ and ϵ denotes the machine epsilon of the system.

4.2.4 Clustering

The most important issue left about the clustering phase is the value of α in 4.5: if $\alpha = 0$ we rely only on the best k-means score; if $\alpha = 1$ we select the solution with most votes.

During the development we noticed that, for our encoded testing audio files, the compression increases the noise on the channel estimates and gives birth to global minima that are the best in terms of kmeans-score, but that are really hard to reach[15]: when this happens, if we rely only on the kmeans-score we are led to select the best result despite a strong hint on a different partitioning, which is usually given by a massive amount of clustering attempts that select the sub-optimal - and often correct - result. In order to avoid such an undesirable behavior in our implementation we set $\alpha = \frac{1}{2}$.

This value of α was not defined in Section 4.1.4 because it could be related to our specific test content: hopefully, future researches will be able to ensure that our assumption on the relationship between the number of votes and the correctness of the solution holds always.

4.2.5 Classification

The SVM classification involves plenty of settings in order to be fully effective. The algorithm presented in 4.1.5 is totally unrelated to the test content, but we can't claim this to be true also for the settings that we are now going to introduce.

4.2.5.1 Dimension Normalization

The dimension normalization influences both the SVM parameter selection and the results of the Feature Selection. Moreover, is known that it can the accuracy of an SVM can severely degrade if the data is not normalized [23].

In our implementation is possible to choose the normalization between 0 and 1, i.e.

$$f_{ji_kk}^{norm} = \frac{f_{ji_kk}^{select} - \underset{i_kk}{min}\left(f_{ji_kk}^{select}\right)}{\underset{i_kk}{max}\left(f_{ji_kk}^{select}\right) - \underset{i_kk}{min}\left(f_{ji_kk}^{select}\right)}; \forall j \in J^{select}$$

[15]i.e. they have only a few votes.

or the normalization between -1 and 1, i.e.

$$f_{ji_kk}^{norm} = \frac{f_{ji_kk}^{select} - \underset{i_kk}{min}\left(f_{ji_kk}^{select}\right)}{\underset{i_kk}{max}\left(f_{ji_kk}^{select}\right) - \underset{i_kk}{min}\left(f_{ji_kk}^{select}\right)} \cdot 2 - 1; \forall j \in J^{select}.$$

The normalization is carried out in a different way depending on the input dataset: if both the training set and the testing set are provided, the procedure is exactly the one stated in Section 4.1.5. Otherwise, if only the training set is provided, all the data are normalized at once: we can assume that the influence from a single audio file is negligible, considering that the whole set is tested in a leave-one-out fashion[16].

4.2.5.2 SVM Parameter Selection

The SVM parameter selection is the most important setting affecting the SVM classification [23]. In our implementation is possible to choose between a linear kernel function and Radial Basis Function (RBF): for each of them, we found the best performing parameter selection, computed by an exhaustive coarse-grid search on the parameter space:

Linear Kernel Function: the parameter selection is performed on the complete PCM encoded training-set, and the best parameter selection is the one returning the best cross-validation result. For linear kernel function only the parameter c can be selected, so that the procedure can be formalized as follows:

1. $\forall c \in C = \left\{2^{-5}, 2^{-4}, 2^{-3}, \ldots, 2^{15}\right\}$ compute the cross-validation value cv_c.
2. select $\bar{c} : cv_{\bar{c}} = \underset{c}{max}\,(cv_c)$.

Radial Basic Function: the parameter selection is performed on the complete PCM encoded training-set, and the best parameter selection is the one returning the best cross-validation result.

1. $\forall\,(c,\gamma) \in C \times \Gamma = \left\{2^{-5}, 2^{-4}, 2^{-3}, \ldots, 2^{15}\right\} \times \left\{2^{-15}, 2^{-14}, \ldots, 2^{3}\right\}$ compute the cross-validation value $cv_{(c,\gamma)}$.
2. select $(\bar{c},\bar{\gamma}) : cv_{(\bar{c},\bar{\gamma})} = \underset{(c,\gamma)}{max}\left(cv_{(c,\gamma)}\right)$.

The procedure above, using our test content, returned the following values:

Linear Kernel Function: the best cross-validation is achieved by

$$c_{LKF} = 2^5$$

.

Radial Basis Function: the best cross-validation is achieved by the pair

$$(c_{RBF}, \gamma_{RBF}) = \left(2^8, 2^{-9}\right)$$

.

[16]see Section 4.2.5.4 for further details

Once selected, these parameters are applied regardless of the test file encoding, since in principle we are not aware about the previous encoding: this is the reason why, even in the final implementation, we kept both of them as mandatory - hence editable - inputs.

A second parameter related to the SVM defines the selection algorithm for multi-class problems, i.e. one-vs-one or one-vs-all [18]. Its choice is also left to the user, and the default value both for the LIBSVM library and for our implementation is the one-vs-one.

4.2.5.3 Feature Selection

The feature selection (as known as feature reduction) is a procedure meant to reduce the number of features involved during the classification, possibly increasing the final accuracy, and definitely reducing the time complexity of the classification. It's influenced both by the dimension normalization and by the SVM parameter selection.

As stated in 4.1.5, the feature selection acts as following: let us denote with f_{ji_kk} the j-th component of the feature vector $f_{i_kk} \in \mathbb{R}^{N_{features}}$; for each f_{i_kk} the feature selection returns

$$f_{i_kk}^{select} = \{f_{ji_kk}\}\,;\ \forall j \in J^{select} \subseteq \{1, N_{features}\}\,.$$

How can we choose the best subset J^{select} of features? In order to do so, we relied on a tool included in the LIBSVM library[17], that acts in the following way:

1. Normalize each dimension of the feature vectors.
2. Compute the F-score of the j-th feature F_j defined in [28], and sort all the features according to this score: The $1 \times N_{features}$ vector *sorting* holds the sorted indexes of the features, i.e.

$$F_{sorting(i)} \geq F_{sorting(i+1)};\ \forall i \in [1\,,\, N_{features}-1]\,.$$

3. Start with $select = N_{features}$, $best_{score} = 0$, $best_{index} = 0$.
4. Apply the current feature selection:

$$\begin{cases} f_{i_kk}^{select} = \{f_{ji_kk}\}\,;\ \forall j \in J^{select} \\ J^{select} = \{sorting\,(i)\}\,;\ \forall i \in [1\,,\, select] \end{cases}.$$

5. Compute the cross-validation value $cv_{current}$of the SVM trained with the current feature selection. then
   (a) If $cv_{current} > best_{score}$ update the best:

$$\begin{cases} best_{score} = cv_{current} \\ best_{index} = select \end{cases}.$$

   (b) Else, do nothing.

[17]see appendix A for further details.

6. Update the selection by halving the number of elements included:
$$select = \left\lfloor \frac{select}{2} \right\rfloor ,$$
where $\lfloor \cdot \rfloor$ denotes the largest previous integer of $\cdot$.
7. If $select > 0$ repeat from point 4.
8. Return the selection with the best score, i.e.
$$J^{select} = \{sorting\,(i)\}\,;\,\forall i \in [1\,,\,best_{index}]\,.$$

Since our goal is to be able to deal also with compressed audio files, we applied the tool on different training sets, obtained with different encoding and rates. After this preliminary operation, according to our possible previous knowledge on the test content, we are able to choose between

1. J^{select}_{PCM} is obtained using as training set the whole test content encoded as raw PCM.
2. J^{select}_{MP3} is obtained using as training set the whole test content encoded with MP3 at 32 kbps.
3. J^{select}_{AAC} is obtained using as training set the whole test content encoded with AAC at 16 kbps.
4. J^{select}_{AMR} is obtained using as training set the whole test content encoded with AMR at 4.75 kbps.
5. J^{select}_{MERGE} is the union between the previous selections, i.e.
$$J^{select}_{MERGE} = J^{select}_{PCM} \cup J^{select}_{MP3} \cup J^{select}_{AAC} \cup J^{select}_{AMR}\,.$$
6. $J^{select}_{DEFAULT}$ is the selection including all the features, i.e.
$$J^{select}_{DEFAULT} = \{1\,,\,2\,,\,3\,,\,\dots\,,\,N_{features}\}\,.$$

4.2.5.4 Cross-Validation

When we don't have a testing set a leave-one-out cross-validation is performed on the training set. This procedure has been used both for the parameter selection and for the evaluations reported in Chapter 5, and follows an algorithm derived from the one stated in Section 4.1.5.

Let us denote with $f_{i_k k} \in \mathbb{R}^{N_{features}}$ the feature vector f of the k-th device computed from its i-th recording as defined in Equation 4.4, with N_k the number of recordings from the k-th device, with N_{device} the total number of devices, with l_k the class-label of the k-th device and with N_{class} the number of classes that we desire to classify.

Starting from the whole training set, the procedure is the following:

1. Build the data-set and assign the proper labels:
$$\begin{cases} F = \{f_{i_k k}\}\,;\forall i_k \in \{1, N_k\} \\ \\ L = \{l_k \cdot \mathbf{1}_{N_k}\} \end{cases} \quad \forall k \in \{1, N_{device}\},$$
where $\mathbf{1}_{N_k}$ is a $1 \times N_k$ vector with all elements equal to one.

2. Feature Selection:
   let us denote with f_{ji_kk} the j-th component of the feature vector $f_{i_kk} \in \mathbb{R}^{N_{features}}$; for each f_{i_kk} the feature selection returns

$$f^{select}_{i_kk} = \{f_{ji_kk}\}\,;\ \forall j \in J^{select} \subseteq \{1, N_{features}\}\,.$$

3. Dimension Normalization:
   after the feature selection, we need to perform a normalization of each dimension of the feature vector, i.e.

$$f^{norm}_{ji_kk} = \frac{f^{select}_{ji_kk} - \underset{i_kk}{min}\left(f^{select}_{ji_kk}\right)}{\underset{i_kk}{max}\left(f^{select}_{ji_kk}\right) - \underset{i_kk}{min}\left(f^{select}_{ji_kk}\right)}(b-a)+a;\ \forall j \in J^{select},$$

   where $a < b$. The most common values of a and b are $a = 0$, $b = 1$ and $a = -1$, $b = 1$. As we can see, this time the normalization involves at once both the training and the testing set, since we can assume that the influence of the single testing audio file is negligible.

4. Set the starting values of the evaluation variables:

$$test\_file = 1,\ N_{correct} = 0,\ M_{conf} = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix} \in \mathbb{R}^{N_{class} \times N_{class}},$$

   where $N_{correct}$ denotes the total number of correct classifications, and M_{conf} denotes the resulting confusion matrix.

5. Perform all the evaluations:
   both $F^{norm} = \{f^{norm}_i\}$ and $L = \{l_i\}$ are sets of $N_{tot} = \sum_{k=1}^{N_{device}} N_k$ elements, so we can proceed as follows:

   (a) Split the training set and the testing set:

$$\begin{cases} F^{norm}_{testing} = \left\{f^{norm}_{test\_file}\right\} \\ F^{norm}_{training} = F^{norm} \backslash F^{norm}_{testing} \\ L_{testing} = \left\{l_{test\_file}\right\} \\ L_{training} = L \backslash L_{testing} \end{cases}.$$

   (b) Train the SVM:

$$Model_{SVM} = training\left(F^{norm}_{training}\,,\ L_{training}\,,\ kernel\,,\ algorithm\right).$$

   (c) Classify the t-th audio file with the computed model:

$$l_{SVM} = training\left(F^{norm}_{testing}\,,\ Model_{SVM}\right).$$

   (d) Update the result:
   if $l_{SVM} \neq l_{testing}$

$$M_{conf}\left(l_{SVM}, l_{test\_file}\right) = M_{conf}\left(l_{SVM}, l_{test\_file}\right) + 1$$

else

$$\begin{cases} M_{conf}\left(l_{test\_file}, l_{test\_file}\right) = M_{conf}\left(l_{test\_file}, l_{test\_file}\right) + 1 \\ N_{correct} = N_{correct} + 1 \end{cases}.$$

(e) Update the variable selector:

$$test\_file = test\_file + 1.$$

(f) If $test\_file \leq N_{tot}$ repeat from point (a).

6. Compute the final accuracy:

$$accuracy = \frac{N_{correct}}{N_{tot}}.$$

The confusion matrix M_{conf} is left unprocessed, since it contains information both about the total number N_{tot} of files involved and about the number of recordings per class. If the user prefers to transform it in the confusion matrix with percentages inside the computation is straightforward: let $M_{conf,sum}$ be the $1 \times N_{class}$vector built by computing the sum of each column of M_{conf}, i.e.

$$M_{conf,sum} = \left(\sum_{i=1}^{N_{class}} M\left(i,1\right), \sum_{i=1}^{N_{class}} M\left(i,2\right), \ldots, \sum_{i=1}^{N_{class}} M\left(i,N_{class}\right) \right).$$

Then

$$M_{conf,\%} = 100\% \cdot M_{conf} \,./\, \left(\mathbf{1}^{t}_{N_{class}} \cdot M_{conf,sum}\right)$$

where ./ denotes the entry-wise division[18], and $\mathbf{1}_{N_{class}}$ is a $1 \times N_{class}$ vector with all elements equal to one.

[18] $C = A\,./B \Longrightarrow C(i,j) = A(i,j)/B(i,j)\,\forall(i,j),\ A, B, C \in \mathbb{R}^{I\times J}$.

Chapter 5

System Evaluation

In order to provide a valuable evaluation of our framework we will now report the final results obtained with the algorithm defined in Chapter 4. Readers interested in the evolution of the feature vector or in the proceeding of the development part will find further information in appendix B.

5.1 Test Specifications

The test specifications have different purposes: we chose to apply the algorithm to realistic recordings, leaving aside unrealistic situations such as anechoic rooms with no environmental noise, and to address also non-speech-only recordings from the playback of a loudspeaker, where more than one audio channel is involved.

All of this, in order to inquire both the applicability degree of this method and possible flaws due to the restrictions of the blind channel estimation algorithm we rely on.

5.1.1 Training Content for the GMM of the Clean Speech

The training content of the GMM is composed by high quality speech recording, acquired at 48 kHz. The utterances, both in English and in German, are spoken by two male and two female speakers and recorded in a low-noise environment, for a total time of about 30 minutes.

The training content was then subsampled down to 8 kHz, i.e. the sampling frequency of possible AMR encoded test content, as well as of recordings from telephone transmissions.

5.1.2 Test Content for the Framework Evaluation

The test content for the framework evaluation is composed by 26 audio files per device, and is composed both of sentences spoken by three male and two female speakers, different from the speakers involved in the recording included in the training set, and of recordings containing both speech and music content. For the latter, three different loudspeaker were involved, in order to highlight possible degradation due to the low quality of the playback device.

The test recordings are acquired in a lossless fashion - i.e. PCM encoded - and the lossy compression is performed only afterwards[1]. The recording sessions took place in a quiet office, and therefore present a low but clearly perceivable environmental noise.

For each compression we chose different bit-rates:

AAC: 128 kbps, 96kbps, 64 kbps, 48 kbps, 32 kbps, 16 kbps.

AMR: 12.20 kbps, 10.20 kbps, 7.95 kbps, 7.4 kbps, 6.7 kbps, 5.9 kbps, 5.15 kbps, 4.75 kbps.

Mp3: 192 kbps, 128 kbps, 96kbps, 64 kbps, 32 kbps

As we saw in Chapter 4, in order to be able to compute the channel estimates from an audio file, we have to ensure that the sampling frequency of the training set of the GMM and of the test file are the same; thus, we had to downsample the original recordings down to 8 kHz.

For the Mp3 and the AAC encoded audio files the resampling occurred after the encoding. On the other hand, the AMR encoding requires from the input audio files both a sampling frequency of exactly 8 kHz, and a bit depth of 16 bit: the resampling was performed before the encoding - unlike what stated in our original process flow.

5.1.3 Devices Involved in the Testing Phase

The testing phase involved 8 model of devices, each one present twice[2], for a total of 16 different devices. Between all of them, we can identify two subsets[3]:

Built-in Microphone: Dell Latitude D630, Google Phone, iPhone 3GS, iPhone 4S and Samsung Galaxy S II.

Headset Microphone: Logitec QuickCam Pro 9000, Logitec Headset, Samsung Galaxy S II headset, iPhone 3GS headset, iPhone 4S headset.

All the recordings from the same subset are acquired at once, in order to minimize possible interferences during the classification due to the environmental conditions.

5.1.4 Sampling Frequency

As stated in Chapter 4 and in the previous sections of this chapter, ideally we need each audio file to be resampled exactly down to 8 kHz. Unfortunately, due to some limitations of the Octave framework, this was not always possible: files whose original sampling frequency was 44100 Hz were resampled down to 8018 Hz.

By doing this we implicitly considered the difference of 18 Hz as negligible, also because the number of STFT frequency bins is the same. Apart from the AMR encoded audio files, whose processing involved the Shibatch Sampling Rate Converter[4], rather than relying on an external resampler we chose to use

[1]See appendix A for details about each encoder involved.
[2]Or two slightly different models from the same manufacturer
[3]see Section 2.1.3.
[4]See appendix A for details and motivations

the one provided by Octave - that we can fully control - and to tolerate the approximation $8018 \approxeq 8000$.

5.2 Clustering Result: Summary and Evaluation

The second section of this chapter is focused on the clustering results: the state of the art on microphone classification still lacks clustering algorithms able to discriminate between different devices with a reasonable degree of confidence[5].

Despite this, during the development of our framework, we chose to focus at first on the k-means clustering algorithm. We did this both in order to try reduce the gap between the clustering and the classification world and in order to achieve a robust feature vector: in principle, a feature vector strong enough to blindly differentiate between different devices should be able to perform even better if applied to a supervised classification method.

5.2.1 Clustering with Two Devices

As we stated in Section 4.1, our clustering procedure relies on two assumption: we know both the number of devices present in the clustering set and the number of recording per device.

Let $A = \left\{r_a^A\right\}$ and $B = \left\{r_b^B\right\}$ denote the initial recording sets, where $a \in [1,\, N_A]$ and $b \in [1,\, N_B]$. N_A denotes the number of recordings of the device A and N_B denotes the number of recordings of the device B. We define the testing set $R = \{r_k\}$ as follows:

$$R = \left\{r_1^A, \ldots, r_{N_A}^A, r_1^B, \ldots, r_{N_B}^B\right\}.$$

Let $l_k \in \{L_1, L_2\}$ be the clustering label of the test recording r_k, where $L_1 \neq L_2$: we mark a clustering attempt as *correct* if and only if

$$\begin{cases} l_{k_1} = L_1 & \forall k_1 \in \{1, \ldots, N_A\} \\ l_{k_2} = L_2 & \forall k_2 \in \{N_A + 1, \ldots, N_A + N_B\} \end{cases}.$$

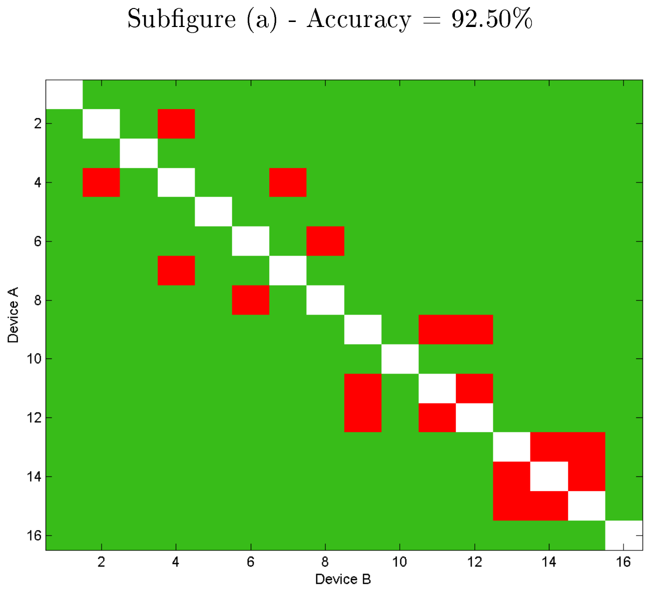
We perform this procedure for all the possible couples of devices: if we have a total of N devices, then we have $\sum_{n=1}^{N-1} n = \frac{1}{2}(N-1) \cdot N$ couples in total.

In the next sections we will provide a graphical representation of the results obtained by this procedure while varying the test set: each axis of the figures will represent the chosen device and the color found on the corresponding square will represent the classification outcome - *correct* (green), or *wrong* (red).

[5]see Chapter 3.

5.2.1.1 Full Test Set

The full test set include all the the test content from all the 16 devices, for a total of 120 possible couples. A graphical representation of the clustering outcome is provided in Figure 5.1:



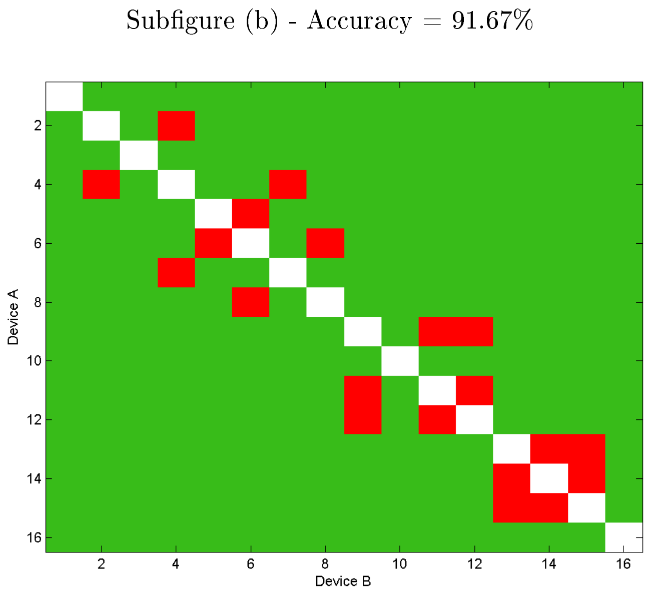


Figure 5.1: Clustering with Two Devices - Full Test Set

Figure 5.1 (a) refers to the results obtained starting with audio files encoded with PCM or with MP3 at 96 kbps, while Figure 5.1 (b) refers only to MP3 encoded audio files, with bitrates 192 kbps, 128 kbps, 64 kbps and 32 kbps.

The devices involved are labeled numerically in both subfigures as follows:

Label	Device
1	$Dell_1$, with the *built-in* microphone
2	$Dell_1$, with the *headset* microphone
3	$Dell_2$, with the *built-in* microphone
4	$Dell_2$, with the *headset* microphone
5	$GooglePhone_1$, with the *built-in* microphone
6	$GooglePhone_1$,with the *headset* microphone
7	$GooglePhone_2$,with the *built-in* microphone
8	$GooglePhone_2$,with the *headset* microphone
9	$iPhone_1$, with the *built-in* microphone
10	$iPhone_1$,with the *headset* microphone
11	$iPhone_2$,with the *built-in* microphone
12	$iPhone_2$,with the *headset* microphone
13	$GalaxyS2_1$, with the *built-in* microphone
14	$GalaxyS2_1$,with the *headset* microphone
15	$GalaxyS2_2$,with the *built-in* microphone
16	$GalaxyS2_2$,with the *headset* microphone

As stated in Section 4.1.4, we avoided the case where recordings from the same device are present: as a consequence, in both figures the first diagonal is empty, since $Device\,A \neq Device\,B$ is always true.

5.2.1.2 Built-in Test Set

For the built-in test set the 8 devices involved are labeled as follows:

Label	Device
1	$Dell_1$, with the *built-in* microphone
2	$Dell_2$, with the *built-in* microphone
3	$GooglePhone_1$, with the *built-in* microphone
4	$GooglePhone_2$,with the *built-in* microphone
5	$iPhone_1$, with the *built-in* microphone
6	$iPhone_2$,with the *built-in* microphone
7	$GalaxyS2_1$, with the *built-in* microphone
8	$GalaxyS2_2$,with the *built-in* microphone

For a resulting total of 28 possible couples. With this test set the outcome of the clustering algorithm is the same both for PCM encoded and for MP3 compressed audio files, and shown in Figure 5.2:

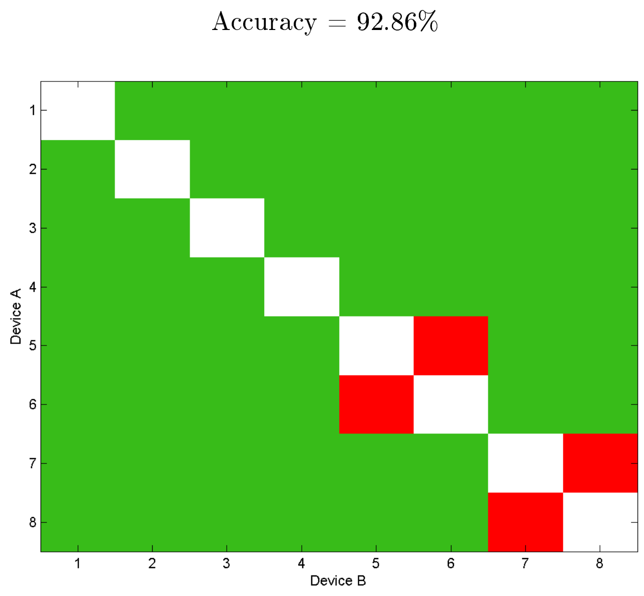


Figure 5.2: Clustering with Two Devices - Built-in Test Set

From this figure is clearly visible that the clustering between devices of the same model and manufacturer is, as we expected, more difficult than the clustering performed on devices from different models. More specifically, our modified version of the k-means can't differentiate between the two different $iPhone_i$ or between the two different $GalaxyS2_i$.

5.2.1.3 Headset Test Set

In a similar way, for the headset test set the 8 devices involved are labeled as follows:

Label	Device
1	$Dell_1$, with the *headset* microphone
2	$Dell_2$, with the *headset* microphone
3	$GooglePhone_1$,with the *headset* microphone
4	$GooglePhone_2$,with the *headset* microphone
5	$iPhone_1$,with the *headset* microphone
6	$iPhone_2$,with the *headset* microphone
7	$GalaxyS2_1$,with the *headset* microphone
8	$GalaxyS2_2$,with the *headset* microphone

For a resulting total of 28 possible couples. The outcome of the clustering algorithm is the same both for PCM encoded and for MP3 compressed audio files, as happened using the previous test set, and is shown in Figure 5.3:

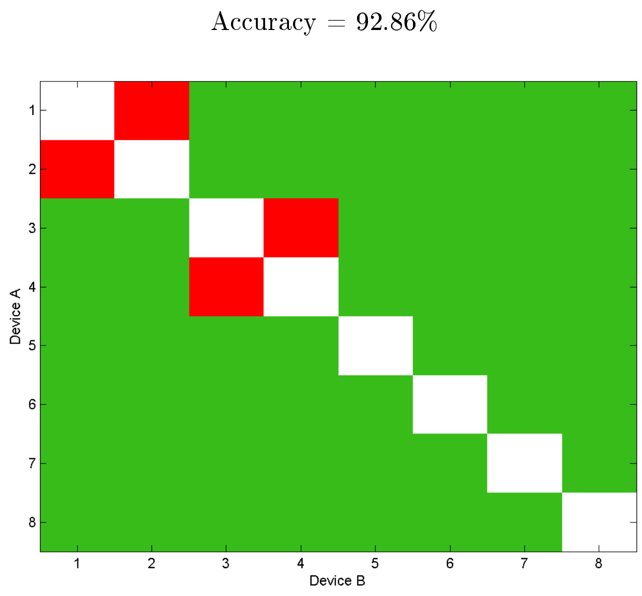


Figure 5.3: Clustering with Two Devices - Headset Test Set

Once again the misclassifications occur between couples of namely identical devices, i.e. the two different $Dell_i$ and the two different $GooglePhone_i$.

5.2.2 Clustering with Three Devices

Let $A = \{r_a^A\}$, $B = \{r_b^B\}$ and $C = \{r_c^C\}$ denote the initial recording sets, where $a \in [1, N_A]$,$b \in [1, N_B]$ and $c \in [1, N_C]$. N_A denotes the number of recordings of the device A, N_B denotes the number of recordings of the device B and N_C denotes the number of recordings of the device C. We define the testing set $R = \{r_k\}$ as follows:

$$R = \{r_1^A, \dots, r_{N_A}^A, r_1^B, \dots, r_{N_B}^B, r_1^C, \dots, r_{N_C}^C\}.$$

Let $l_k \in \{L_1, L_2, L_3\}$ be the clustering label of the test recording r_k, where $L_1 \neq L_2$, $L_1 \neq L_3$ and $L_2 \neq L_3$: we mark a clustering attempt as *correct* if and only if

$$\begin{cases} l_{k_1} = L_1 & \forall k_1 \in \{1, \dots, N_A\} \\ l_{k_2} = L_2 & \forall k_2 \in \{N_A + 1, \dots, N_A + N_B\} \\ l_{k_3} = L_3 & \forall k_3 \in \{N_A + N_B + 1, \dots, N_A + N_B + N_C\} \end{cases}.$$

If we have a total of N devices, then we have $\binom{N}{3} = \frac{N!}{3! \cdot (N-3)!}$ triples available to choose from in order to create the test set.

5.2.2.1 Full Test Set

We didn't perform the clustering with three devices on the full test set, because the results obtained by using only two devices already show that misclassifications occur both between namely identical devices and between microphones and headsets provided directly by the same manufacturer. Since we had no other reasonable criteria suitable on order to check the whole test set without, we chose to investigate on the built-in subset and on the headset subset separately, and to avoid triples involving devices between which we can expect a strong relation beforehand.

5.2.2.2 Built-in Test Set

The built-in test set corresponds to a total number of $\binom{8}{3} = 56$ possible triples. We didn't use the whole testing set but only 32 triples, i.e. all those triples where namely identical devices are *not* present. With this selection we are sure that the model of the devices involved in each test set is different and we expected some misclassifications occurring with triplets both one Google Phone and one Galaxy S II, since both are produced by Samsung.

The final results for the built-in test set, however, show a 100% accuracy across all the triplets: this means that our final feature vector with our modified version of the k-means algorithm is able to correctly separate sets including recording from three built-in microphones from mobile phones, as long as these devices are *not* namely identical.

5.2.2.3 Headset Test Set

The headset test set, as the built-in test set, corresponds to a total number of 56 devices. For the same reasons expressed before we reduced this number down to 32 triples, where namely identical devices are *not* present.

Again, we achieved a 100% accuracy across all the triplets, showing that, also for headset microphones for mobile phones, as long as three devices are not namely identical we are able to correctly partition them.

5.2.3 Clustering with Four Devices

Let $A = \left\{r_a^A\right\}$, $B = \left\{r_b^B\right\}$, $C = \left\{r_c^C\right\}$ and $D = \left\{r_d^D\right\}$ denote the initial recording sets, where $a \in [1, N_A]$,$b \in [1, N_B]$, $c \in [1, N_C]$ and $d \in [1, N_D]$. N_A denotes the number of recordings of the device A, N_B denotes the number of recordings of the device B, N_C denotes the number of recordings of the device C and N_D denotes the number of recordings of the device D. We define the testing set $R = \{r_k\}$ as follows:

$$R = \left\{r_1^A, \dots, r_{N_A}^A, r_1^B, \dots, r_{N_B}^B, r_1^C, \dots, r_{N_C}^C, r_1^D, \dots, r_{N_D}^D\right\}.$$

Let $l_k \in \{L_1, L_2, L_3, L_4\}$ be the clustering label of the test recording r_k, where $L_1 \neq L_2$, $L_1 \neq L_3$, $L_1 \neq L_4$, $L_2 \neq L_3$, $L_2 \neq L_4$and $L_3 \neq L_4$: we mark a clustering attempt as *correct* if and only if

$$\begin{cases} l_{k_1} = L_1 & \forall k_1 \in \{1, \dots, N_A\} \\ l_{k_2} = L_2 & \forall k_2 \in \{N_A + 1, \dots, N_A + N_B\} \\ l_{k_3} = L_3 & \forall k_3 \in \{N_A + N_B + 1, \dots, N_A + N_B + N_C\} \\ l_{k_4} = L_4 & \forall k_3 \in \{N_A + N_B + N_C + 1, \dots, N_A + N_B + N_C + N_D\} \end{cases}.$$

If we have a total of N devices, then we have $\binom{N}{4} = \frac{N!}{4! \cdot (N-4)!}$ quadruples available to choose from in order to create the test set.

5.2.3.1 Full Test Set

As we did for the evaluation of our clustering algorithm with three devices, we chose not to perform any evaluation on the full test set, since we had not a proper selection criteria between the available quadruples.

5.2.3.2 Built-in Test Set

Once again, we selected only the quadruples where namely identical devices are *not* present, i.e. 16 quadruples out of the $\binom{8}{4} = 70$ possible ones.

Since we have 4 couples of microphones from namely identical devices, this is also the biggest meaningful test case of our clustering algorithm available for the built-in test set: we achieved a 100% accuracy, demonstrating that our clustering algorithm and our feature vector, as long as there's not a strong correlation between the devices included in the test set, is able to create a correct partitioning of recordings from built-in microphones of mobile devices.

5.2.3.3 Headset Test Set

As for the built in test set, we selected only the quadruples where namely identical devices are *not* present, i.e. 16 quadruples out of the $\binom{8}{4} = 70$ possible ones and the biggest meaningful test case of our clustering algorithm available for the headset test set.

Also on the headset test set we achieved a 100% accuracy, demonstrating that our clustering algorithm and our feature vector, as long as there's not a strong correlation between the devices included in the test set, is able to create a correct partitioning of recordings from headset microphones for mobile devices.

5.2.4 Final Considerations

The clustering algorithm proposed seems to be extremely powerful, if compared with the previous attempt from Kraetzer et al. [1]. In their work the accuracy is less than 35% for a 5-device problem: our algorithm would probably outperform this accuracy if applied to a 5-device problem, since we can not expect such a huge drop in accuracy.

Regardless of this comparison, these results demonstrate that our algorithm, thanks to our feature vector, can differentiate between recordings from unrelated devices, when both the number of devices and the number of recordings per device are known: our initial purpose was met, and the algorithm works both with PCM encoded files and with MP3 encoded files.

For extensive numerical results please refer to appendix B.

5.3 Classification Result: Summary and Evaluation

The evaluation of the classification algorithm proposed was performed by the leave-one-out cross-validation of the data-set defined in Section 4.2.5.4.

We have a total of 20 full data-set for this evaluation, each one corresponding to a different encoding: like this, we are able to evaluate possible degradation of the algorithm due to the compression, and we are able to see whether our initial goal of being able to perform a classification robust to low-bitrate compression - which is likely to be present on mobile devices - was met or not.

In addition to the encoding algorithm and its bitrate, during the evaluations we also changed the definition of class: in the device classification each class represent the single device as a single instance, i.e.

$$\begin{cases} class\,(Dell_i) \neq class\,(GooglePhone_j) \neq class\,(iPhone_k) \neq class\,(GalaxyS2_h) \\ class\,(Dell_1) \neq class\,(Dell_2) \\ class\,(GooglePhone_1) \neq class\,(GooglePhone_2) \\ class\,(iPhone_1) \neq class\,(iPhone_2) \\ class\,(GalaxyS2_1) \neq class\,(GalaxyS2_2) \end{cases} ;$$

in the model classification different instances of a single model belongs to a same class, i.e.

$$\begin{cases} class\,(Dell_i) \neq class\,(GooglePhone_j) \neq class\,(iPhone_k) \neq class\,(GalaxyS2_h) \\ class\,(Dell_1) = class\,(Dell_2) \\ class\,(GooglePhone_1) = class\,(GooglePhone_2) \\ class\,(iPhone_1) = class\,(iPhone_2) \\ class\,(GalaxyS2_1) = class\,(GalaxyS2_2) \end{cases} .$$

Our feature vector, from the start, was meant to be used on the device classification. We chose to investigate the model classification in order to understand how much our feature vector is suitable to such a scenario, where the high-detailed representation of the channel that we build in order to be able to distinguish between different instances of the same model could actually become a burden for the classifier, due to some overfitting problems.

The results will be presented in a graphical way, by the means of confusion matrices and of accuracy graphs. Readers interested in the extended text results will find them in appendix C.

5.3.1 Device Classification

During the device classification evaluation for each test set we performed two kind of classification: the first one acts directly on the complete test set, and we will call it *intra-device* classification, since different instances of the same model are present during the classification; the second one, that we will call *inter-device* classification, split the data set in several subsets, so that for each subset one instance of each model is present.

As we saw when considering the clustering problem, in our test we have two sources of misclassifications: the first one is the presence of two devices per model, while the second one is the presence of headsets built from the same manufacturer of the mobile device they are meant to be used with.

We will address the first problem by using the *inter-device* classification, and the second one by performing the same tests on the built-in test set and on the headset test set, in a disjoint fashion.

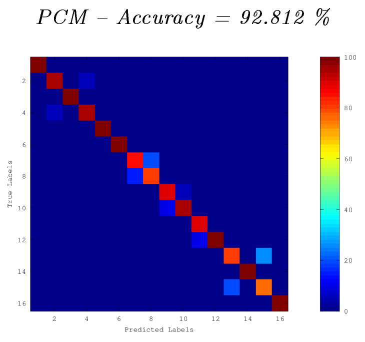
All the following results are achieved with RBF kernel functions with parameters $(c_{RBF}, \gamma_{RBF}) = \left(2^8, 2^{-9}\right)$, with the feature selection $J^{select} = J^{select}_{PCM}$, and with a dimension normalization between -1 and 1.

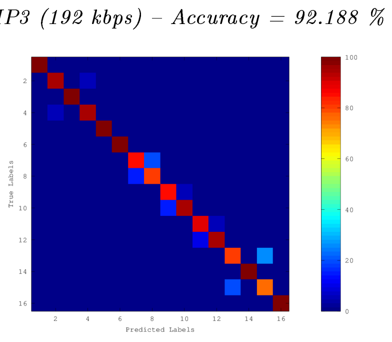
5.3.1.1 Full Test Set

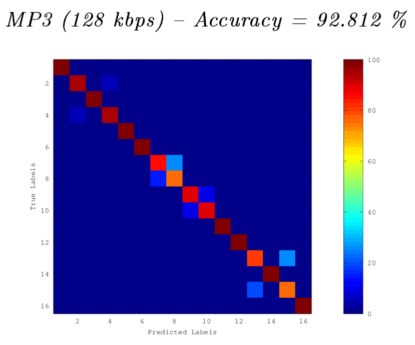
The full test set is composed of 16 different devices, each one representing a single class, that will be labeled numerically as follows:

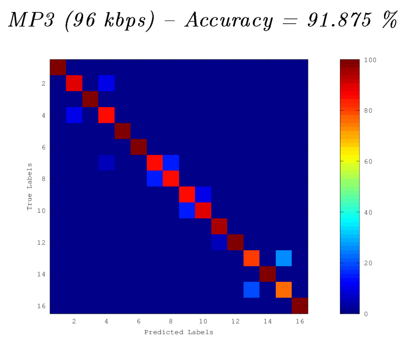
Label	Device	Class
1	$Dell_1$, with the *built-in* microphone	$Dell_1^{built-in}$
2	$Dell_1$, with the *headset* microphone	$Dell_1^{headset}$
3	$Dell_2$, with the *built-in* microphone	$Dell_2^{built-in}$
4	$Dell_2$, with the *headset* microphone	$Dell_2^{headset}$
5	$GooglePhone_1$, with the *built-in* microphone	$GooglePhone_1^{built-in}$
6	$GooglePhone_1$,with the *headset* microphone	$GooglePhone_1^{headset}$
7	$GooglePhone_2$,with the *built-in* microphone	$GooglePhone_2^{built-in}$
8	$GooglePhone_2$,with the *headset* microphone	$GooglePhone_2^{headset}$
9	$iPhone_1$, with the *built-in* microphone	$iPhone_1^{built-in}$
10	$iPhone_1$,with the *headset* microphone	$iPhone_1^{headset}$
11	$iPhone_2$,with the *built-in* microphone	$iPhone_2^{built-in}$
12	$iPhone_2$,with the *headset* microphone	$iPhone_2^{headset}$
13	$GalaxyS2_1$, with the *built-in* microphone	$GalaxyS2_1^{built-in}$
14	$GalaxyS2_1$,with the *headset* microphone	$GalaxyS2_1^{headset}$
15	$GalaxyS2_2$,with the *built-in* microphone	$GalaxyS2_2^{built-in}$
16	$GalaxyS2_2$,with the *headset* microphone	$GalaxyS2_2^{headset}$

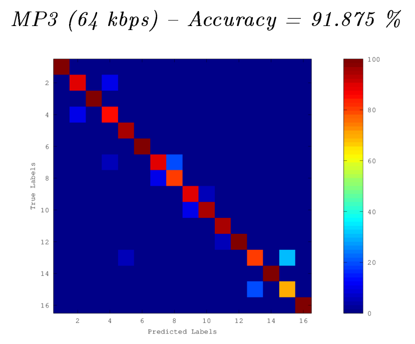
The intra-device classification involves all 16 classes in a single classification task. For each compression algorithm we had several confusion matrices, one per bitrate: a graphical representation of the confusion matrices of the intra-device classification is shown in Figure 5.4.

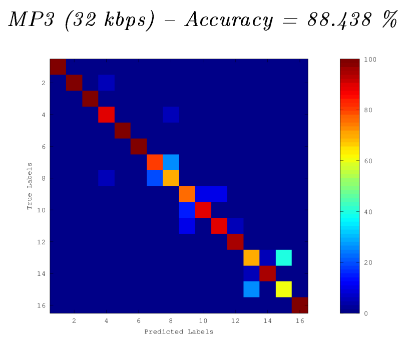


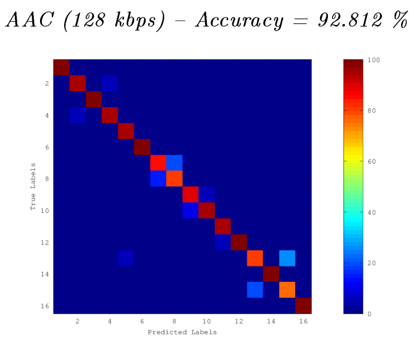


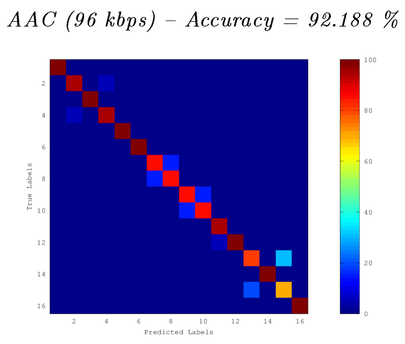

MP3 (128 kbps) – Accuracy = 92.812 %
True Labels
Predicted Labels

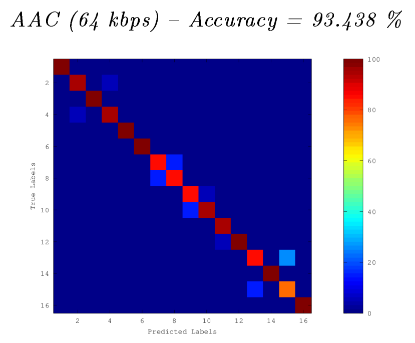

MP3 (96 kbps) – Accuracy = 91.875 %
True Labels
Predicted Labels

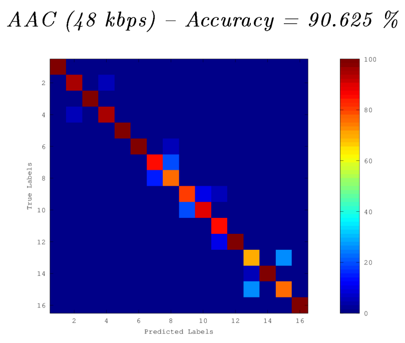

MP3 (64 kbps) – Accuracy = 91.875 %
True Labels
Predicted Labels


MP3 (32 kbps) – Accuracy = 88.438 %
True Labels
Predicted Labels

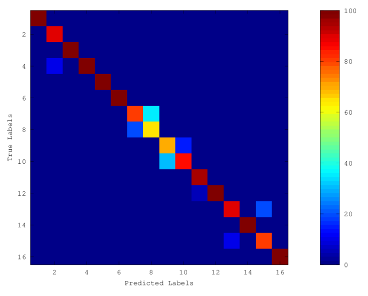

AAC (128 kbps) – Accuracy = 92.812 %
True Labels
Predicted Labels


AAC (96 kbps) – Accuracy = 92.188 %
True Labels
Predicted Labels

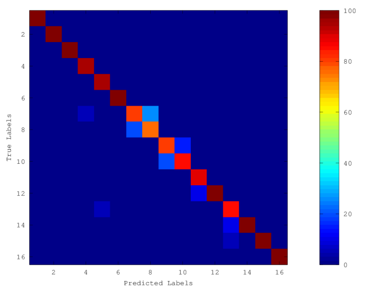

AAC (64 kbps) – Accuracy = 93.438 %
True Labels
Predicted Labels


AAC (48 kbps) – Accuracy = 90.625 %
True Labels
Predicted Labels

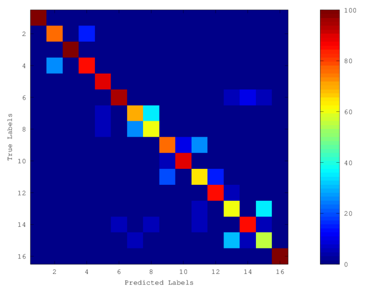
AAC (32 kbps) – Accuracy = 90.938 %



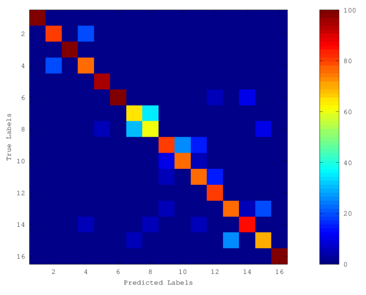
AAC (16 kbps) – Accuracy = 92.812 %



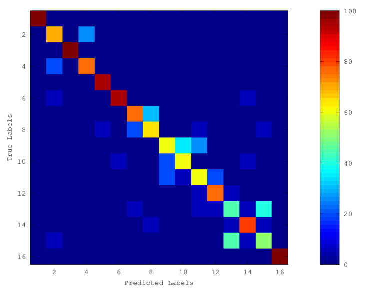
AMR (12.2 kbps) – Accuracy = 80.625 %



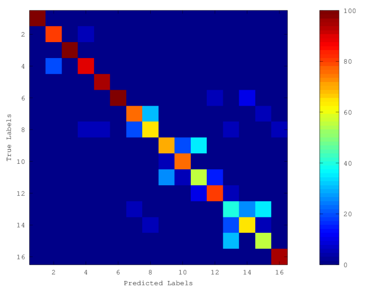
AMR (10.2 kbps) – Accuracy = 82.188 %



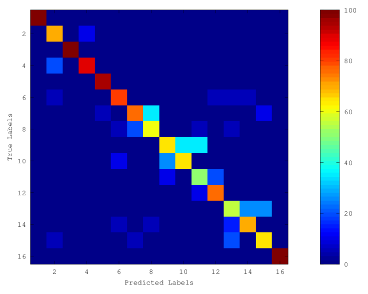
AMR (7.95 kbps) – Accuracy = 75.312 %



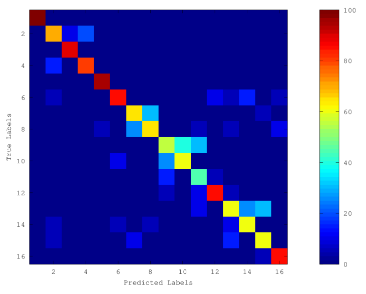
AMR (7.4 kbps) – Accuracy = 77.5 %



AMR (6.7 kbps) – Accuracy = 75.938 %



AMR (5.9 kbps) – Accuracy = 72.5 %



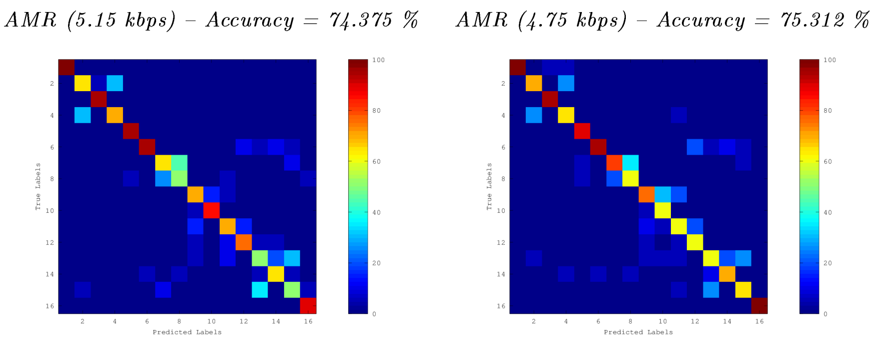


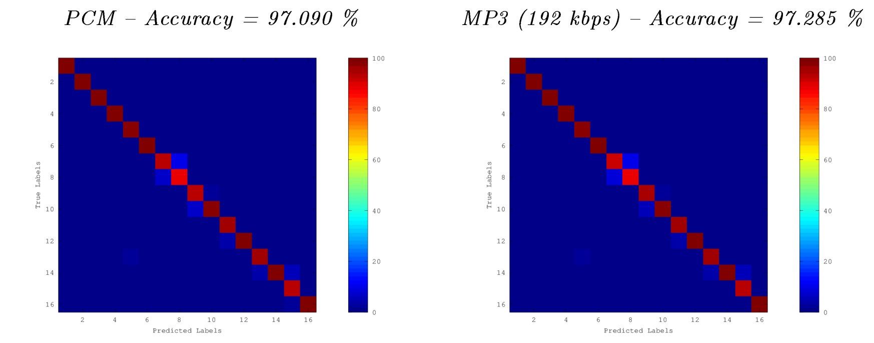
Figure 5.4: Intra-Device Classification - Test Set

The confusion matrices show the presence of a tight correlation between namely identical devices, especially between the two iPhones, the two Samsung Galaxy, and the dell headsets.

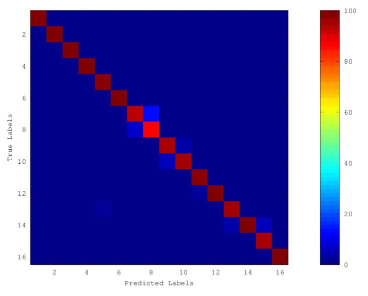
In order to remove this dependency and to assess the quality of our classification method against supposedly unrelated devices the inter-device classification is performed: during the inter-device classification 8 classes per test subset are present, and the cross-validation is performed on all the different test configurations. The test configurations are selected in order to avoid the presence in the test set of different instances of the same model, i.e. the main source of misclassification: $\forall\,(a,b,c,d,e,f,g,h,i) \in \{1,2\}^8$

$$
\begin{aligned}
L = \{l \;:\; & \left(l = Dell_a^{built-in}\right) \vee \left(l = Dell_b^{headset}\right) \vee \dots \\
& \left(l = GooglePhone_d^{built-in}\right) \vee \left(l = GooglePhone_e^{headset}\right) \vee \dots \\
& \left(l = iPhone_f^{built-in}\right) \vee \left(l = iPhone_g^{headset}\right) \vee \dots \\
& \left(l = GalaxyS2_h^{built-in}\right) \vee \left(l = GalaxyS2_i^{headset}\right) \qquad \}
\end{aligned}
$$

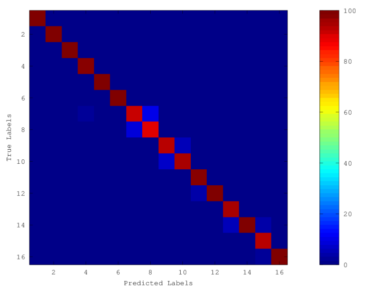
The cross-validation results from all the possible 256 configurations are aggregated together in a single confusion matrix, and the whole procedure is repeated for all the available compression algorithms. A graphical representation of the confusion matrices resulting from the inter-device classification is shown in Figure 5.5.



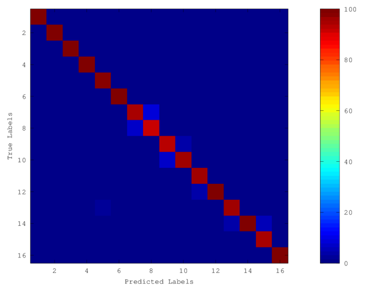
MP3 (128 kbps) – Accuracy = 97.197 %



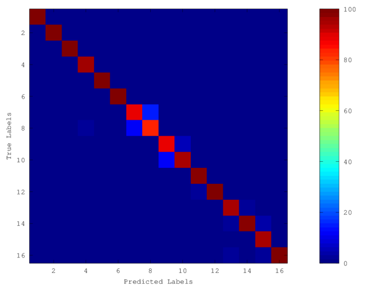
MP3 (96 kbps) – Accuracy = 96.796 %



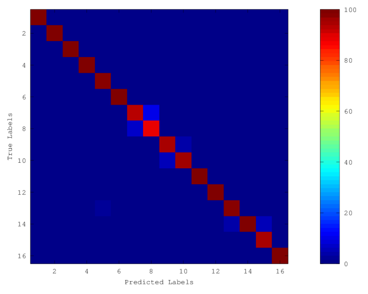
MP3 (64 kbps) – Accuracy = 97.375 %



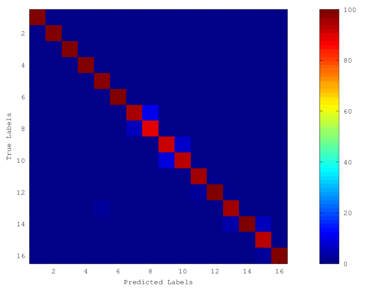
MP3 (32 kbps) – Accuracy = 95.964 %



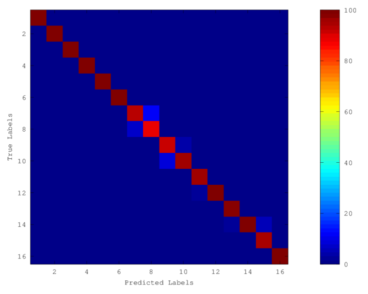
AAC (128 kbps) – Accuracy = 97.495 %



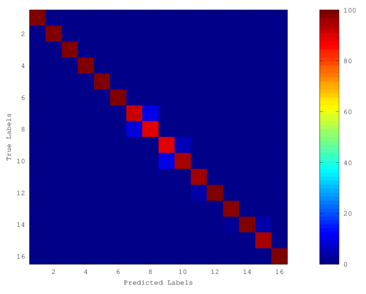
AAC (96 kbps) – Accuracy = 97.041 %

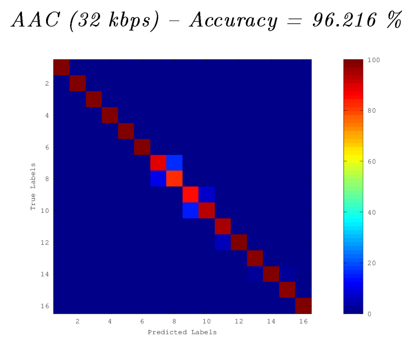


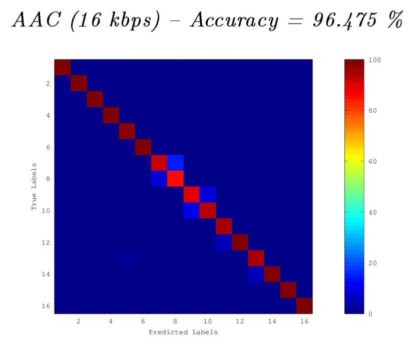
AAC (64 kbps) – Accuracy = 97.139 %

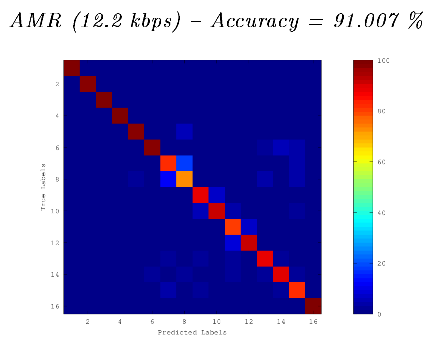


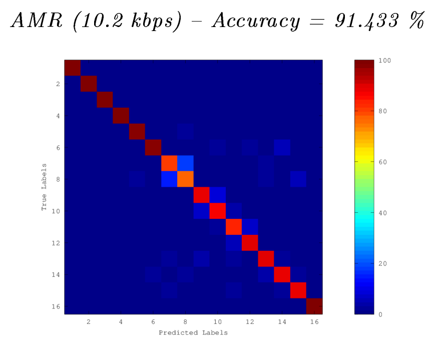
AAC (48 kbps) – Accuracy = 97.124 %

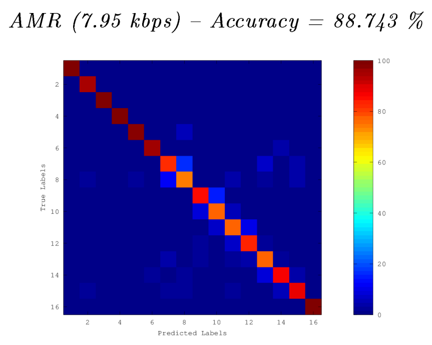


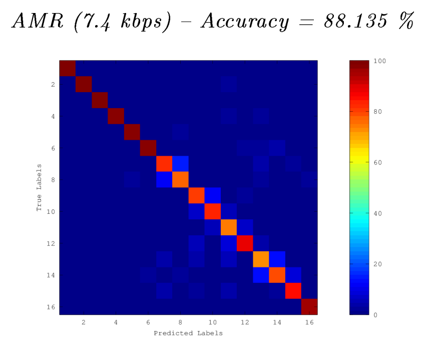

AAC (32 kbps) – Accuracy = 96.216 %
True Labels
Predicted Labels

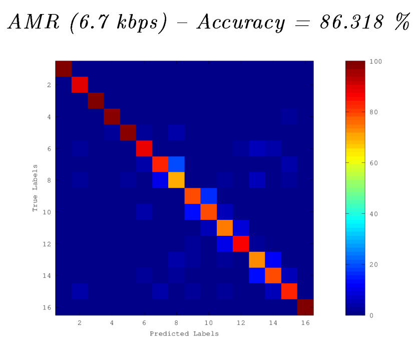

AAC (16 kbps) – Accuracy = 96.475 %
True Labels
Predicted Labels

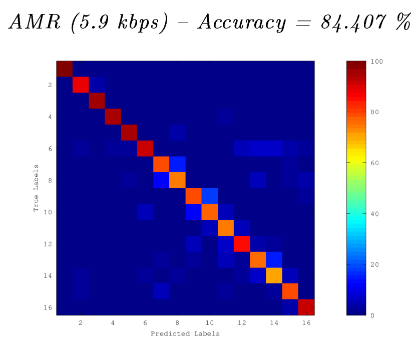

AMR (12.2 kbps) – Accuracy = 91.007 %
True Labels
Predicted Labels


AMR (10.2 kbps) – Accuracy = 91.433 %
True Labels
Predicted Labels


AMR (7.95 kbps) – Accuracy = 88.743 %
True Labels
Predicted Labels


AMR (7.4 kbps) – Accuracy = 88.135 %
True Labels
Predicted Labels


AMR (6.7 kbps) – Accuracy = 86.318 %
True Labels
Predicted Labels


AMR (5.9 kbps) – Accuracy = 84.407 %
True Labels
Predicted Labels

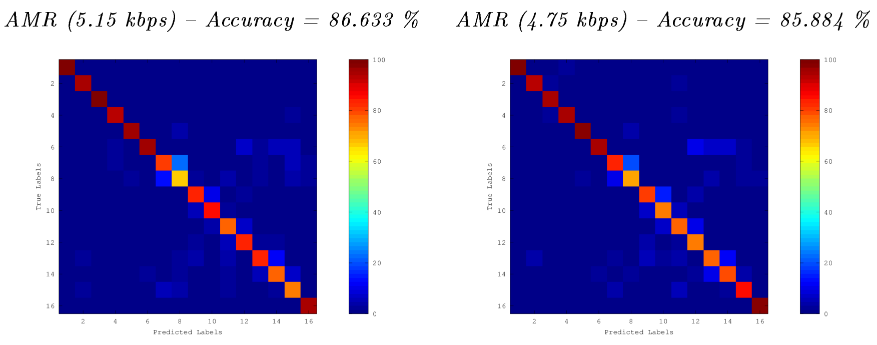


Figure 5.5: Intra-Device Classification - Full Test Set

The results show a clear improvement, as well as the presence of a secondary source of misclassification: as mentioned when considering the clustering problem, headsets built from the same manufacturer of the mobile device they are meant to be used with creates a strong source of misclassifications. We can also notice a dependency between the accuracy and the coding algorithm, which is summarized in Figure 5.6 and 5.7

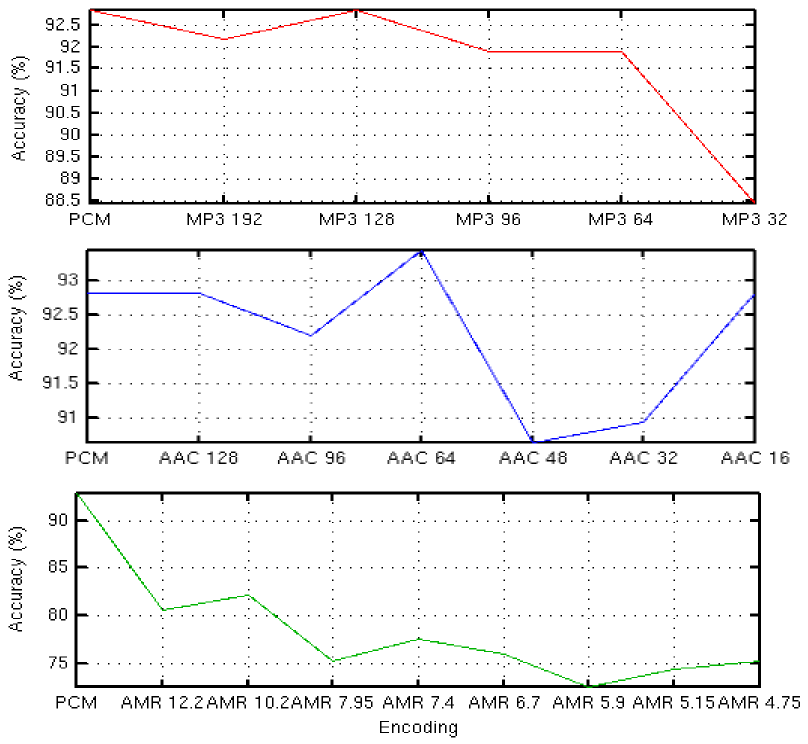


Figure 5.6: Intra-Device Classification - Full Test Set: Accuracy Vs Encoding

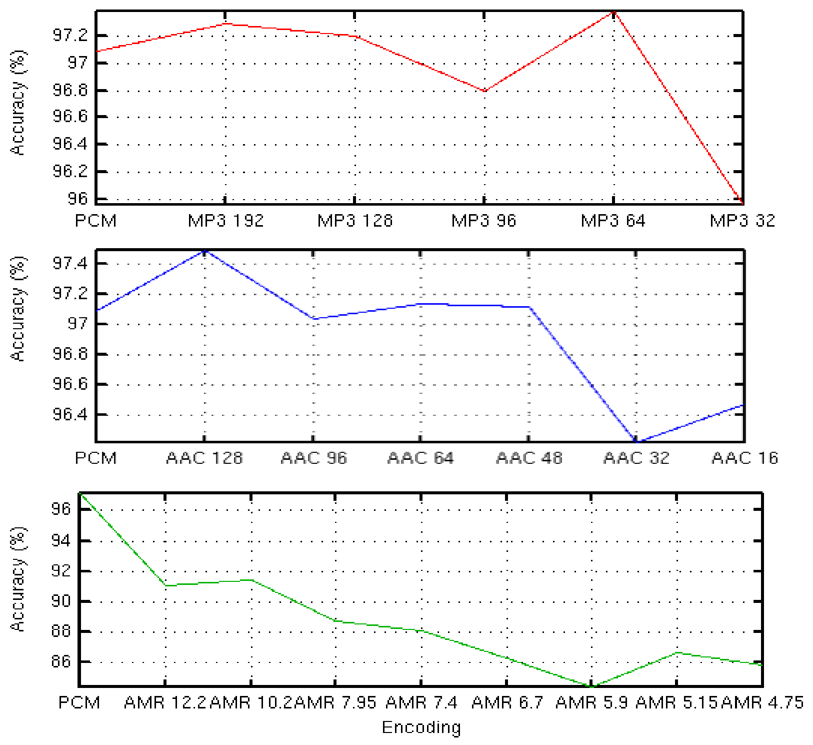


Figure 5.7: Inter-Device Classification - Full Test Set: Accuracy Vs Encoding

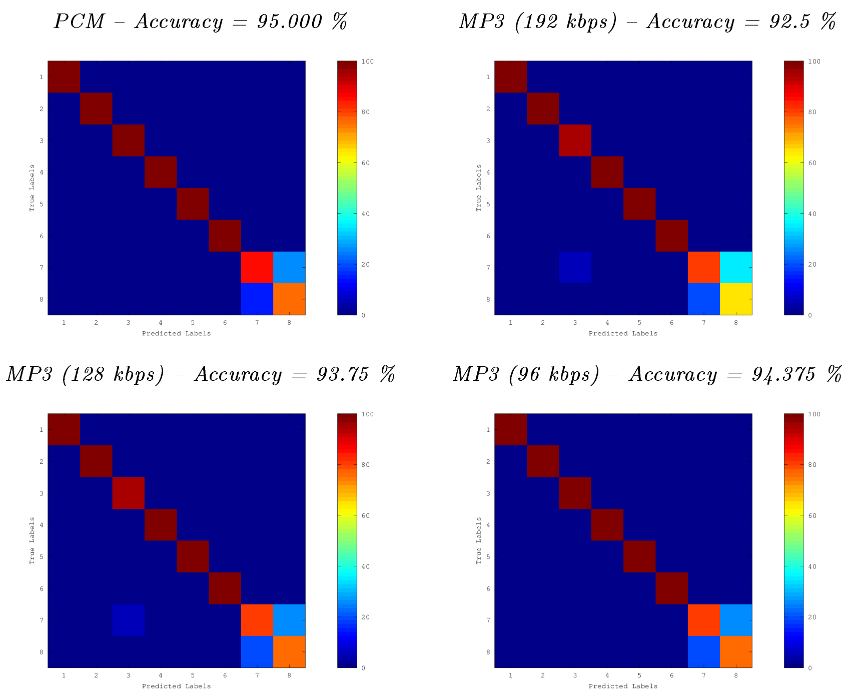
From the figures above we can notice that the dependency between the accuracy and the bitrate of the encoding algorithm is unclear: for the AMR compression a general decreasing trend is present - as we would have expected - but both the MP3-encoded and the AAC encoded test sets sometimes have an higher score that the one obtained by using the PCM encoded test set, which is counterintuitive.

5.3.1.2 Built-in Test Set

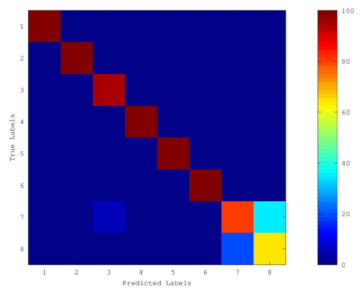
The built-in test set is composed of 8 several devices, that will be labeled numerically as follows:

Label	Device	Class
1	$Dell_1$, with the *built-in* microphone	$Dell_1^{built-in}$
2	$Dell_2$, with the *built-in* microphone	$Dell_2^{built-in}$
3	$GooglePhone_1$, with the *built-in* microphone	$GooglePhone_1^{built-in}$
4	$GooglePhone_2$,with the *built-in* microphone	$GooglePhone_2^{built-in}$
5	$iPhone_1$, with the *built-in* microphone	$iPhone_1^{built-in}$
6	$iPhone_2$,with the *built-in* microphone	$iPhone_2^{built-in}$
7	$GalaxyS2_1$, with the *built-in* microphone	$GalaxyS2_1^{built-in}$
8	$GalaxyS2_2$,with the *built-in* microphone	$GalaxyS2_2^{built-in}$

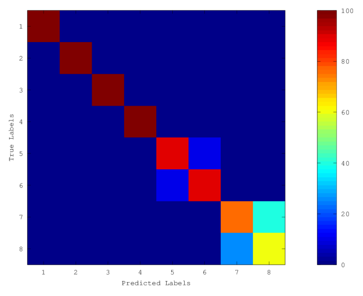
As for the full test set, we performed an intra-device classification at first. A graphical representation of the confusion matrices of the intra-device classification is shown in Figure 5.8.



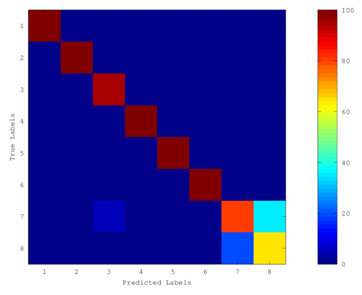
MP3 (64 kbps) – Accuracy = 92.5 %



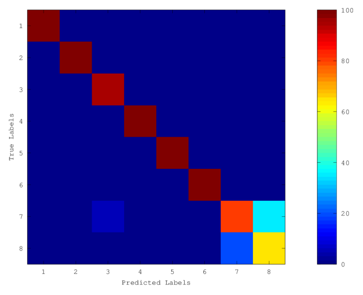
MP3 (32 kbps) – Accuracy = 89.375 %



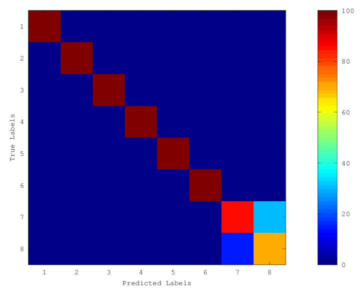
AAC (128 kbps) – Accuracy = 92.5 %



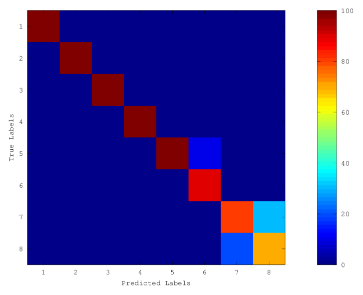
AAC (96 kbps) – Accuracy = 92.5 %



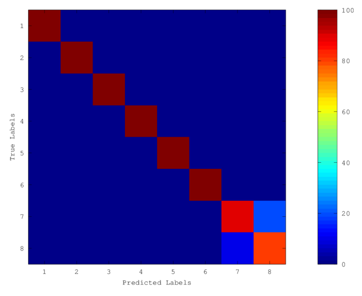
AAC (64 kbps) – Accuracy = 94.375 %



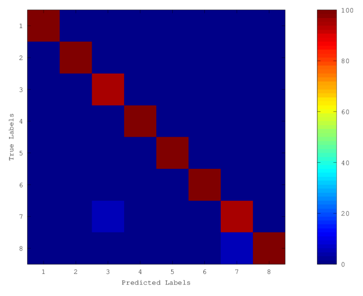
AAC (48 kbps) – Accuracy = 92.5 %



AAC (32 kbps) – Accuracy = 96.25 %



AAC (16 kbps) – Accuracy = 98.75 %



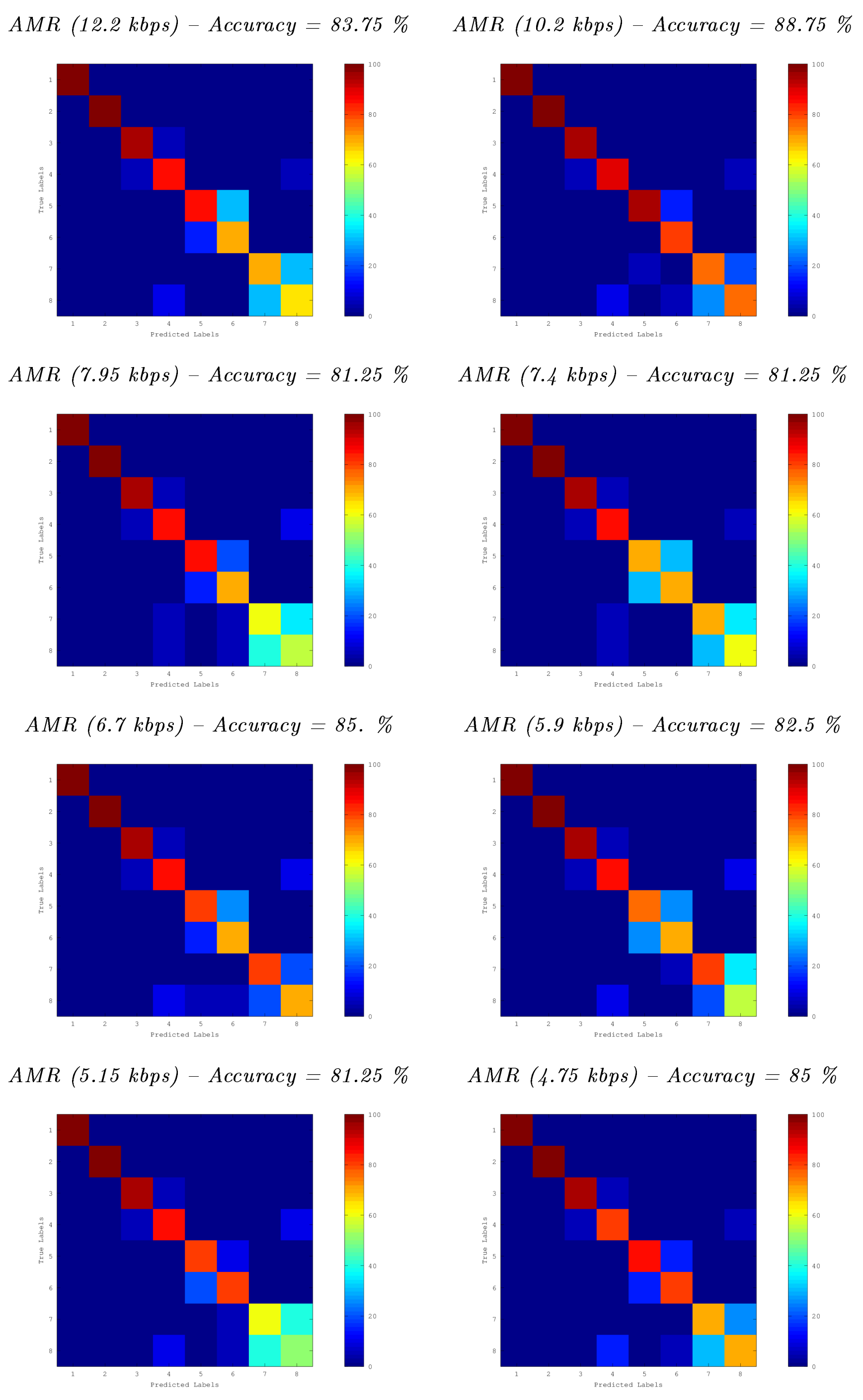
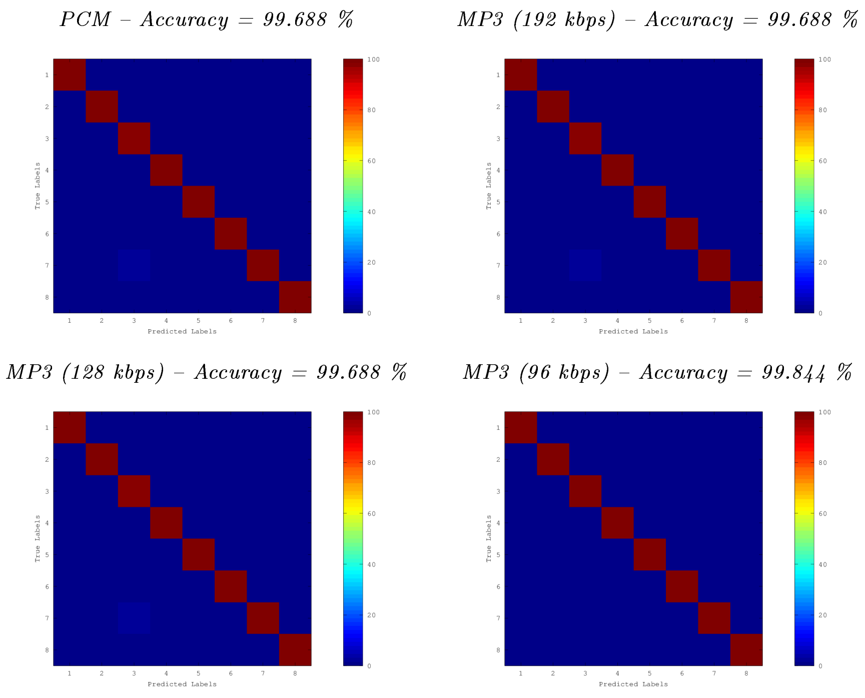


Figure 5.8: Intra-Device Classification - Built-in Test Set

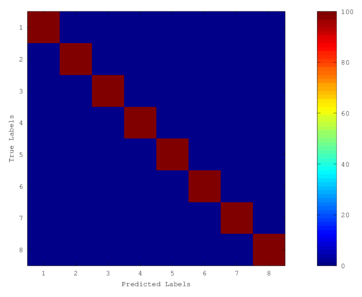
As we can see, the correlation between namely identical devices is clearly exposed: in particular, is extremely difficult to differentiate between the two Samsung Galaxy S2 even from PCM encoded recordings. This problem is addressed, like before, by performing an inter-device classification; the test configurations are selected as follows: $\forall\,(a,b,c,d) \in \{1,2\}^4$

$$\begin{aligned} L = \{l \;:\; & \left(l = Dell_a^{built-in}\right) \vee \left(l = GooglePhone_d^{built-in}\right) \vee \ldots \\ & \left(l = iPhone_f^{built-in}\right) \vee \left(l = GalaxyS2_h^{built-in}\right) \qquad \} \end{aligned}$$

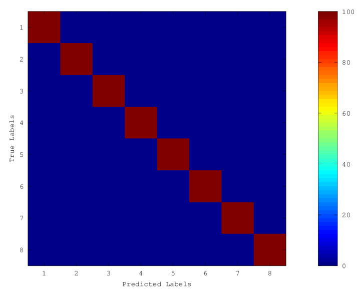
The cross-validation results from the possible 16 configurations are aggregated together in a single confusion matrix, and the whole procedure is repeated for all the available compression algorithms. A graphical representation of the confusion matrices resulting from the inter-device classification is shown in Figure 5.9.



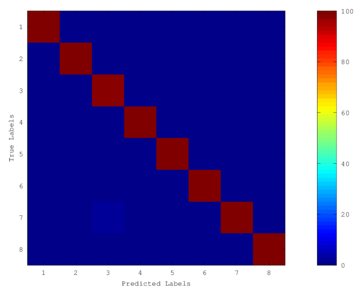
MP3 (64 kbps) – Accuracy = 99.844 %



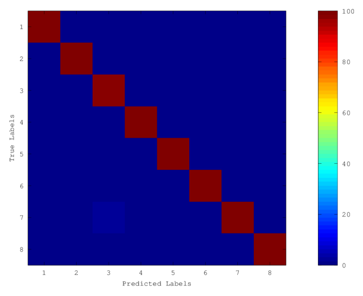
MP3 (32 kbps) – Accuracy = 100.000 %



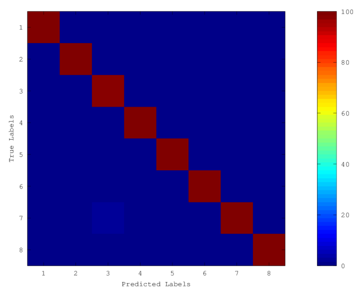
AAC (128 kbps) – Accuracy = 99.688 %



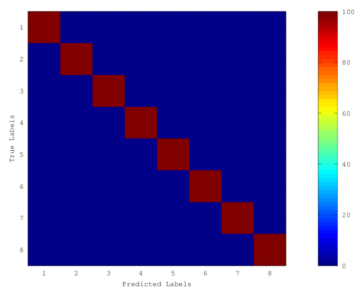
AAC (96 kbps) – Accuracy = 99.688 %



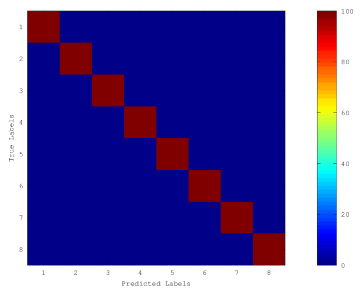
AAC (64 kbps) – Accuracy = 99.688 %



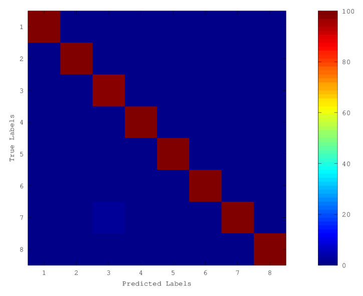
AAC (48 kbps) – Accuracy = 99.844 %



AAC (32 kbps) – Accuracy = 100 %



AAC (16 kbps) – Accuracy = 99.766 %



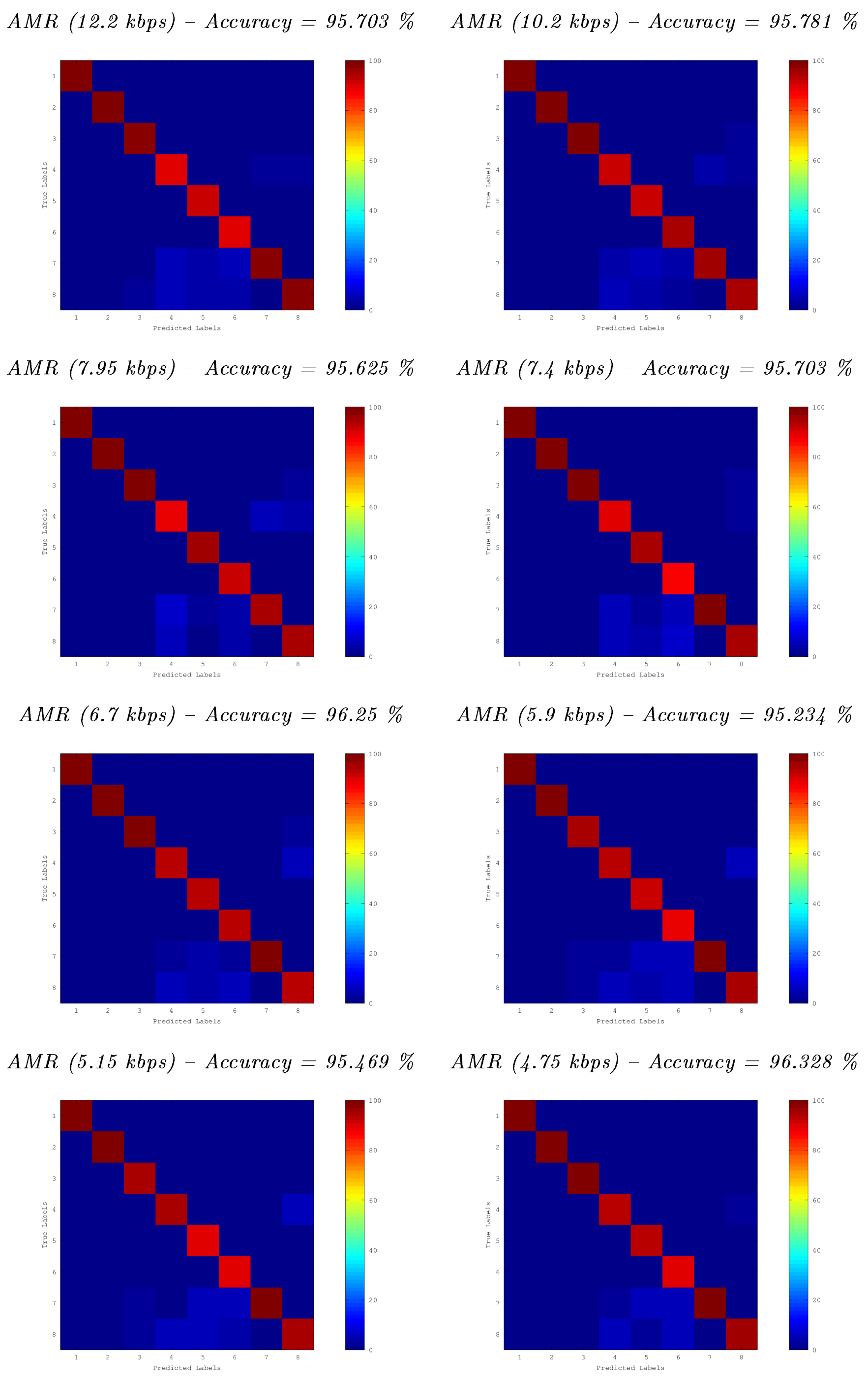


Figure 5.9: Intra-Device Classification - Built-in Test Set

The results of the inter-device classification clearly show that in the built-in test-set is definitely is possible to differentiate between unrelated devices. Figure 5.10 exposes an irregular dependency of the accuracy as a function of the encoding bit-rate during the intra-device classification task:

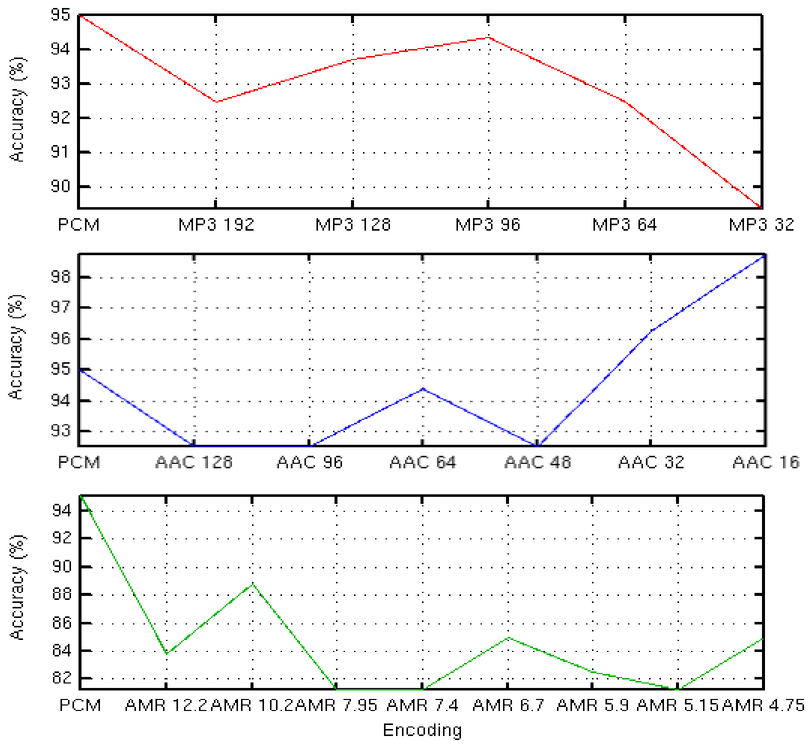


Figure 5.10: Intra-Device Classification - Built-in Test Set: Accuracy Vs Encoding

Figure 5.11, on the other end, shows that for the inter-device classification both the MP3 and the AAC encoding seem not to influence - apart for a really slight increase of the accuracy on the lower bitrates - the results: is possible to achieve a near to perfect classification, like when using PCM encoded recordings.

The AMR encoding, instead, creates a decrease of the accuracy - due to the extremely low bitrates - that remains nearly constant despite the decrease of the bitrate.

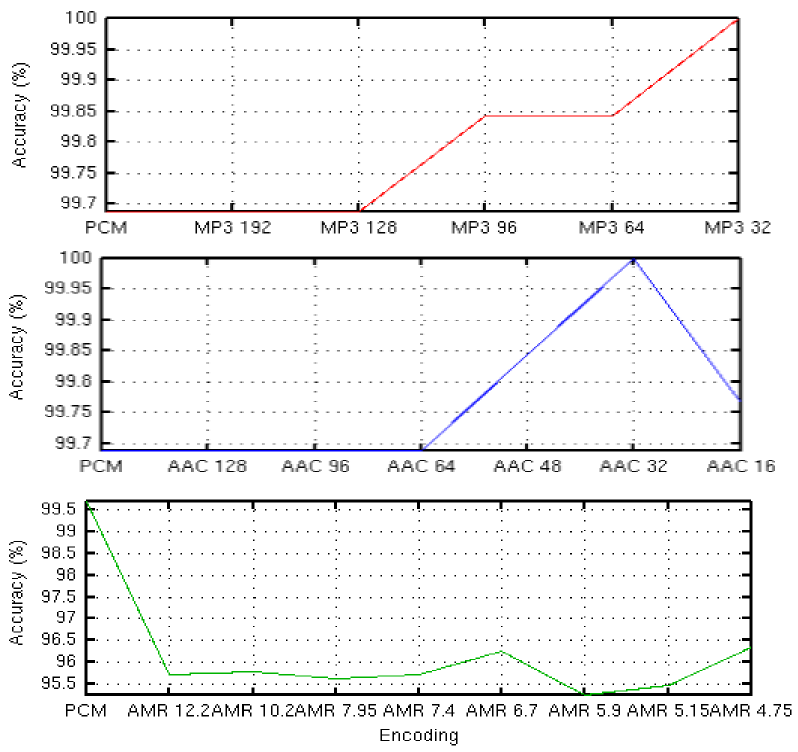


Figure 5.11: Inter-Device Classification - Built-in Test Set: Accuracy Vs Encoding

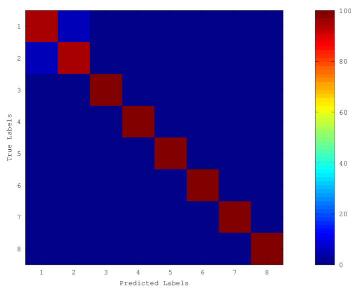
5.3.1.3 Headset Test Set

The headset test set is composed of 8 several devices, that will be labeled numerically as follows:

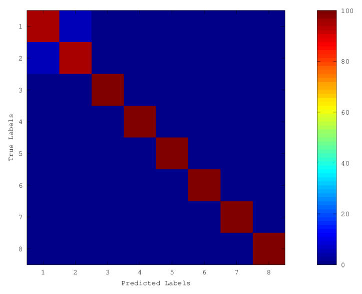
Label	Device	Class
1	$Dell_1$, with the *headset* microphone	$Dell_1^{headset}$
2	$Dell_2$, with the *headset* microphone	$Dell_2^{headset}$
3	$GooglePhone_1$, with the *headset* microphone	$GooglePhone_1^{headset}$
4	$GooglePhone_2$,with the *headset* microphone	$GooglePhone_2^{headset}$
5	$iPhone_1$, with the *headset* microphone	$iPhone_1^{headset}$
6	$iPhone_2$,with the *headset* microphone	$iPhone_2^{headset}$
7	$GalaxyS2_1$, with the *headset* microphone	$GalaxyS2_1^{headset}$
8	$GalaxyS2_2$,with the *headset* microphone	$GalaxyS2_2^{headset}$

The first step was the inter-device classification; a graphical representation of the confusion matrices of the intra-device classification is shown in Figure5.12.

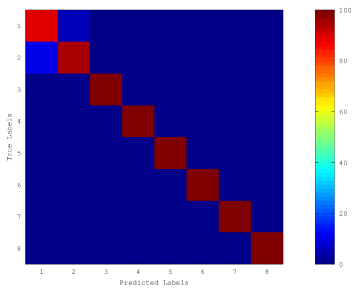
PCM – Accuracy = 98.750 %



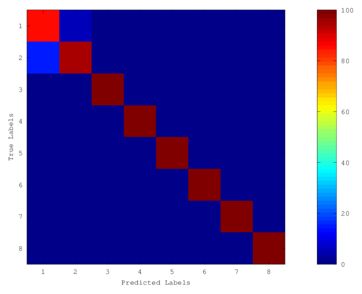
MP3 (192 kbps) – Accuracy = 98.750 %



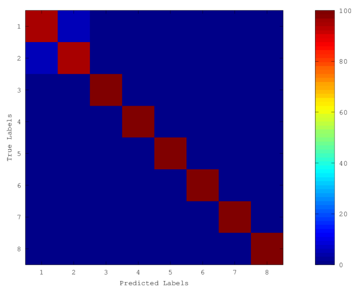
MP3 (128 kbps) – Accuracy = 98.125 %



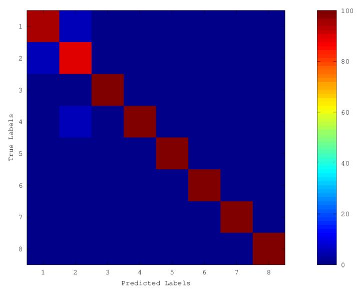
MP3 (96 kbps) – Accuracy = 97.500 %



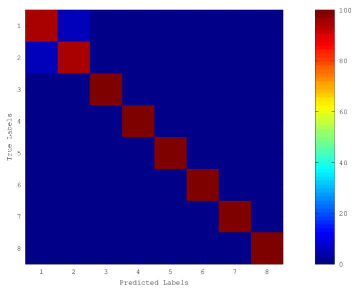
MP3 (64 kbps) – Accuracy = 98.750 %



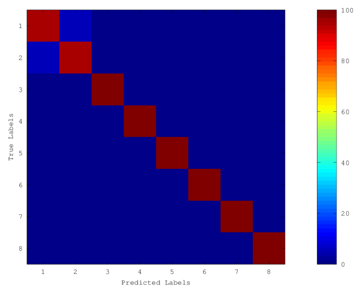
MP3 (32 kbps) – Accuracy = 98.125 %

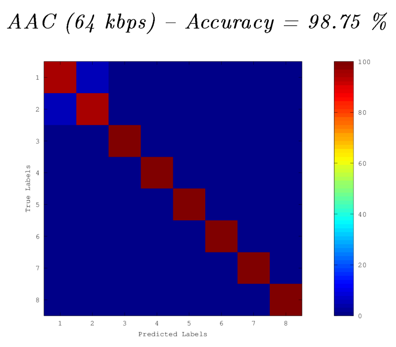


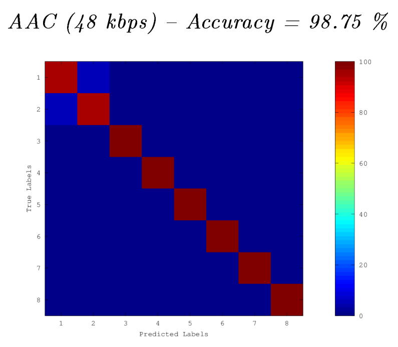
AAC (128 kbps) – Accuracy = 98.75 %

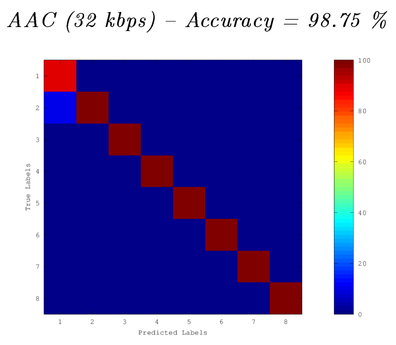


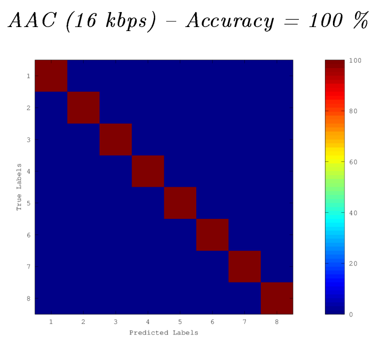
AAC (96 kbps) – Accuracy = 98.75 %

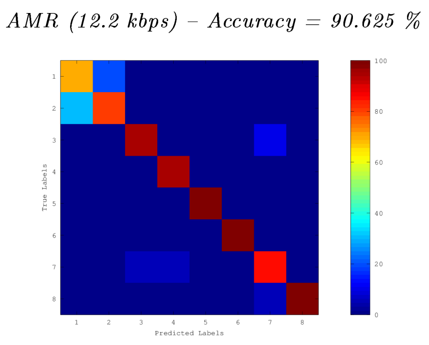


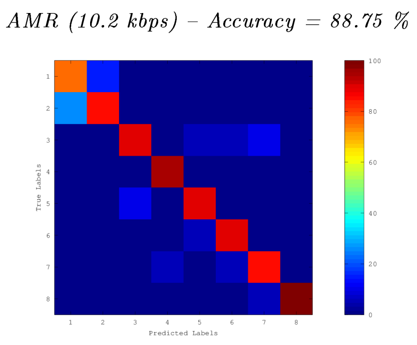

AAC (64 kbps) – Accuracy = 98.75 %
True Labels
Predicted Labels

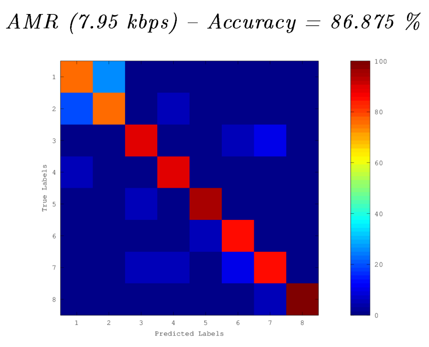

AAC (48 kbps) – Accuracy = 98.75 %
True Labels
Predicted Labels

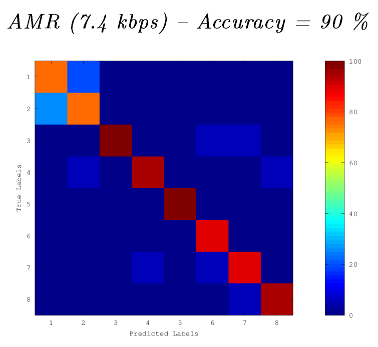

AAC (32 kbps) – Accuracy = 98.75 %
True Labels
Predicted Labels


AAC (16 kbps) – Accuracy = 100 %
True Labels
Predicted Labels


AMR (12.2 kbps) – Accuracy = 90.625 %
True Labels
Predicted Labels


AMR (10.2 kbps) – Accuracy = 88.75 %
True Labels
Predicted Labels


AMR (7.95 kbps) – Accuracy = 86.875 %
True Labels
Predicted Labels


AMR (7.4 kbps) – Accuracy = 90 %
True Labels
Predicted Labels

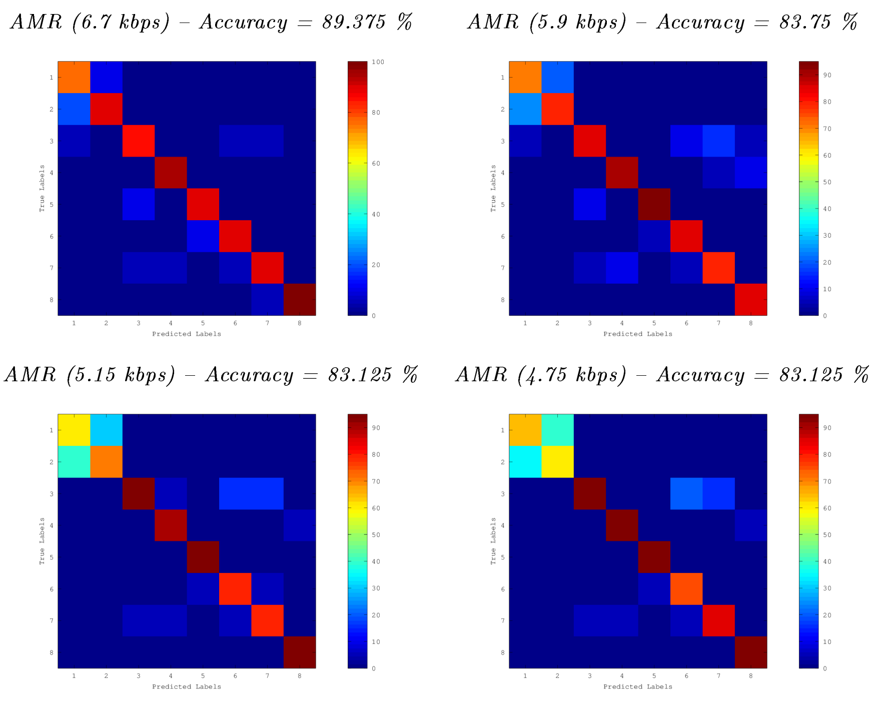
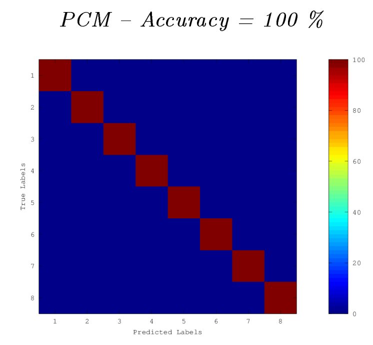


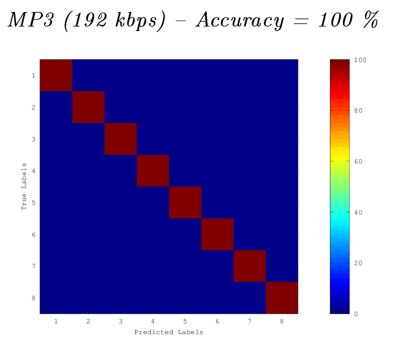
Figure 5.12: Intra-Device Classification - Headset Test Set

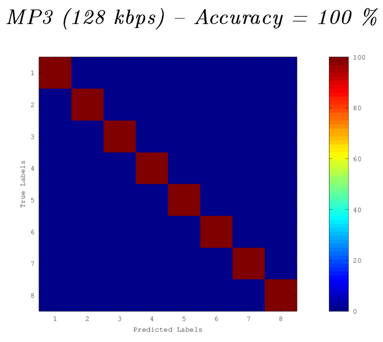
Once again, the correlation between namely identical devices is exposed; however, differently from the built-in device, the most difficult couple is represented by the headset microphones used with the two Dell laptops. The test configurations for the following inter-device classification were selected as follows: $\forall\,(a, b, c, d) \in \{1, 2\}^4$

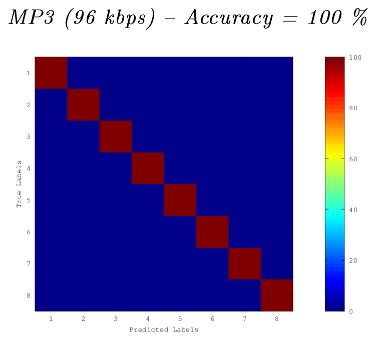
$$\begin{aligned} L = \{l \ : \quad & \left(l = Dell_a^{headset}\right) \vee \left(l = GooglePhone_d^{headset}\right) \vee \ldots \\ & \left(l = iPhone_f^{headset}\right) \vee \left(l = GalaxyS2_h^{headset}\right) \qquad \} \end{aligned}$$

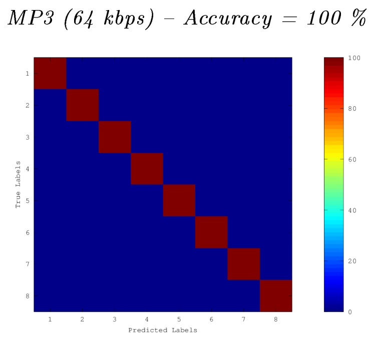
The cross-validation results from the possible 16 configurations are aggregated together in a single confusion matrix, and the whole procedure is repeated for all the available compression algorithms. A graphical representation of the confusion matrices resulting from the inter-device classification is shown in Figure 5.13.

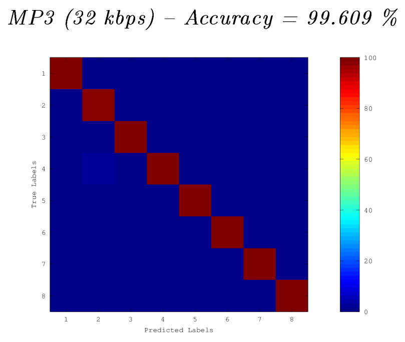

PCM – Accuracy = 100 %
True Labels
Predicted Labels

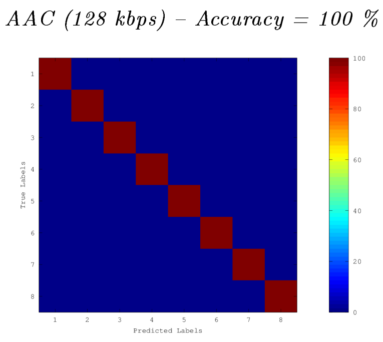

MP3 (192 kbps) – Accuracy = 100 %
True Labels
Predicted Labels

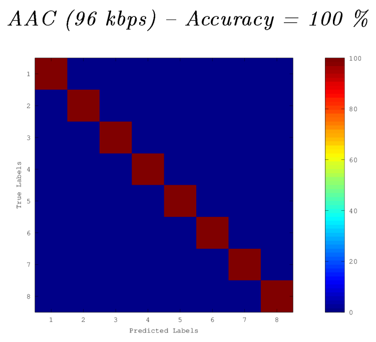

MP3 (128 kbps) – Accuracy = 100 %
True Labels
Predicted Labels


MP3 (96 kbps) – Accuracy = 100 %
True Labels
Predicted Labels

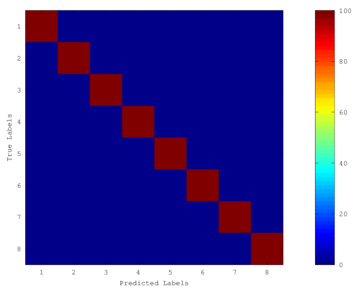

MP3 (64 kbps) – Accuracy = 100 %
True Labels
Predicted Labels


MP3 (32 kbps) – Accuracy = 99.609 %
True Labels
Predicted Labels

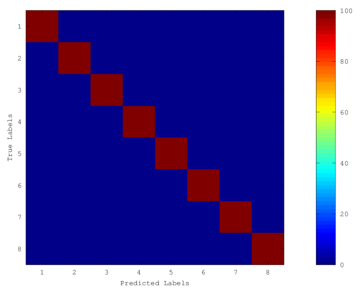

AAC (128 kbps) – Accuracy = 100 %
True Labels
Predicted Labels


AAC (96 kbps) – Accuracy = 100 %
True Labels
Predicted Labels

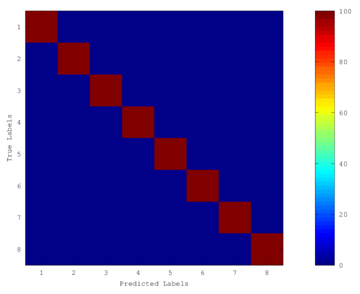
AAC (64 kbps) – Accuracy = 100 %



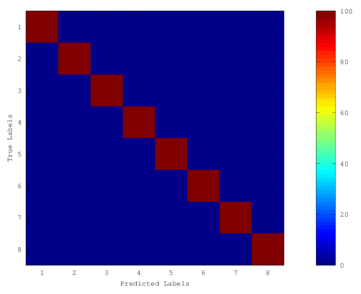
AAC (48 kbps) – Accuracy = 100 %



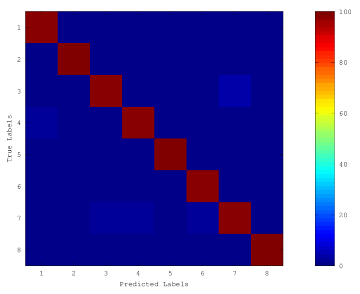
AAC (32 kbps) – Accuracy = 100 %



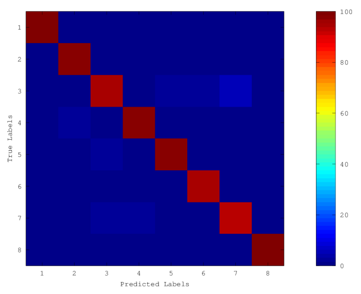
AAC (16 kbps) – Accuracy = 100 %



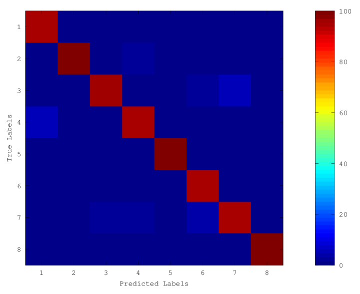
AMR (12.2 kbps) – Accuracy = 98.281 %



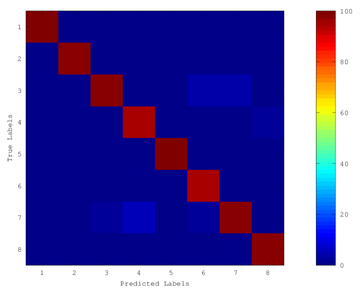
AMR (10.2 kbps) – Accuracy = 96.641 %



AMR (7.95 kbps) – Accuracy = 96.562 %



AMR (7.4 kbps) – Accuracy = 97.422 %



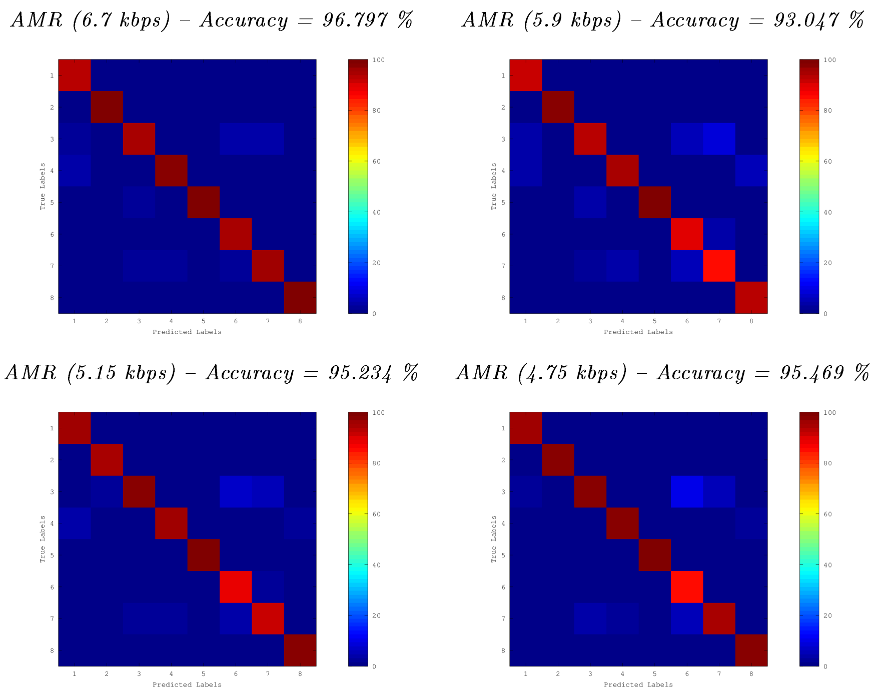


Figure 5.13: Intra-Device Classification - Headset Test Set

The results of the inter-device classification clearly show that also in the headset test-set is definitely is possible to differentiate between unrelated devices. Figure 5.14 exposes a - still existing - irregular dependency of the accuracy as a function of the encoding bit-rate during the intra-device classification task; Figure 5.15,instead, shows that also in the headset test-set is true that for the inter-device classification both the MP3 and the AAC encoding seem not to influence - apart for a really slight increase of the accuracy on the lower bitrates - the results: really often is possible to achieve a perfect classification, like when using PCM encoded recordings.

The AMR encoding creates a decrease of the accuracy - due to the extremely low bitrates - that remains nearly constant until 6.7 kHz, but drops suddenly at lower bitrates.

5.3.1.4 Final Considerations

The results prove that our channel estimation based algorithm works with an high accuracy, that can be raised dramatically when avoiding the presence of highly related devices inside the test set.

Intra-Device Classification: The intra-device classification of the full test set both for PCM, MP3 and AAC encoded audio files achieves an average accuracy higher then 90%. In particular, the average accuracy is equal to 92.812%

with PCM encoded audio files, 91.438% with MP3 encoded audio files and 92.135 with AAC encoded audio files. The average accuracy with AMR encoded audio files is equal to 76.719% if we consider all the available bitrates, however, if we consider only the bitrates higher or equal to 7.4 kbps, i.e. the nominal toll quality bitrate for clean speech only recordings[6], the accuracy rises to 78.906%.

The intra-device classification of the built-in test set both for PCM, MP3 and AAC encoded audio files also achieves an average accuracy higher then 90%. In particular, the average accuracy is equal to 95% with PCM encoded audio files, 92.5% with MP3 encoded audio files and 94.479 with AAC encoded audio files. The average accuracy with AMR encoded audio files is equal to 83.594% if we consider all the available bitrates, and rises to 83.75% if we consider only the bitrates higher or equal to 7.4 kbps.

The intra-device classification of the full test set both for PCM, MP3 and AAC encoded audio files achieve an average accuracy higher then 90%. In particular, the average accuracy is equal to 98.75% with PCM encoded audio files, 98.25% with MP3 encoded audio files and 98.958% with AAC encoded audio files. The average accuracy with AMR encoded audio files is equal to 86.953% if we consider all the available bitrates, and rises to 89.062% if we consider only the bitrates higher or equal to 7.4 kbps.

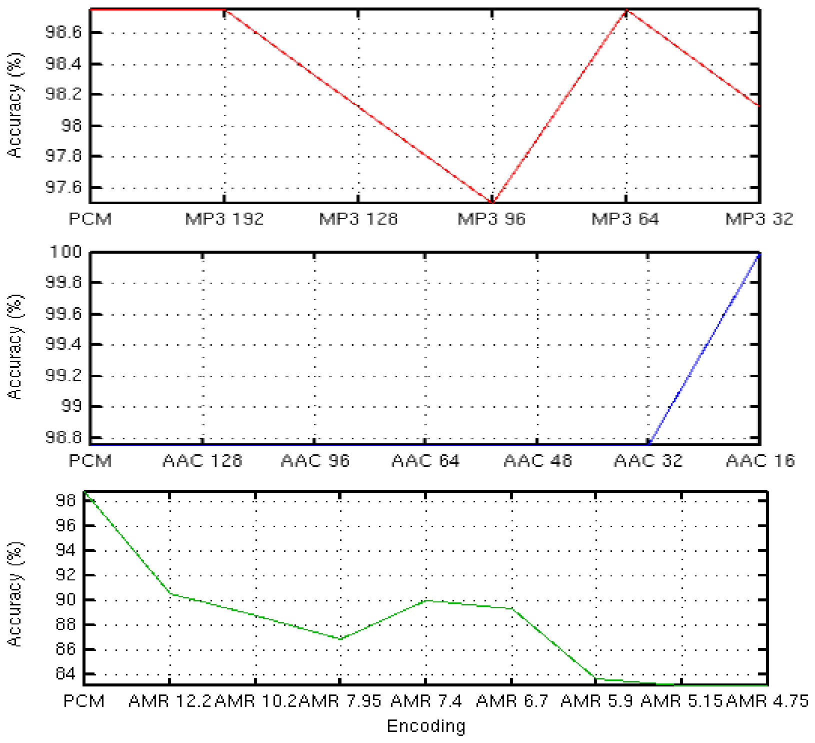


Figure 5.14: Intra-Device Classification - Headset Test Set: Accuracy Vs Encoding

[6]which is not the test content tested used during this work, see Section 5.1.2

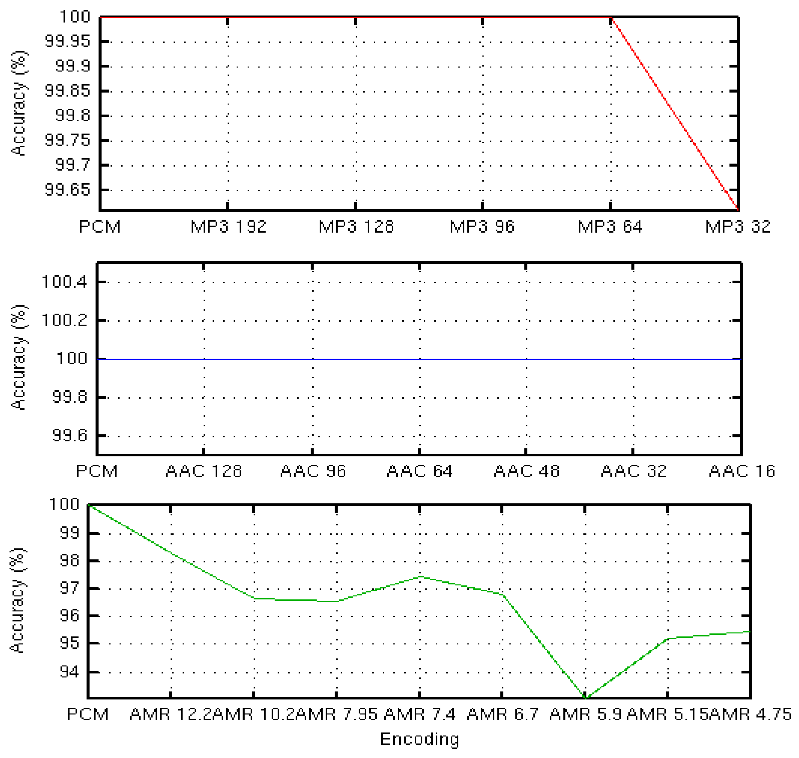


Figure 5.15: Inter-Device Classification - Headset Test Set: Accuracy Vs Encoding

The only research so far that dealt with inter-device classification one by Kraetzer [4]: two intra-device classification task were performed on two different sets of four microphones, and the best accuracy was 75.88% for Røde microphones and of 82.51% for Beyer microphones. These results were achieved on PCM encoded audio files and represent the current state-of-the-art on intra-device classification. If we consider our worst-case scenario, i.e. the classification of the full test set, we can see that our algorithm seems to outperform the previous one on PCM, MP3 or AAC encoded audio files. Moreover, the average accuracy achieved on the AMR encoded test set - 76.719% or 78.906% - is in the same range of the results in [4], even if obtained on bitrates equal to 12.2 kbps or lower. If we reduce the number of devices involved in the test-set, i.e. we consider the built-in test set or the headset test set with only 8 devices involved, our algorithm achieves an higher result with every tested bitrate, despite the number of devise involved is doubled.

Inter-Device Classification: The inter-device classification of the full test set both for PCM, MP3 and AAC encoded audio files achieves an average accuracy higher then 95%. In particular, the average accuracy is equal to 97.090% with PCM encoded audio files, 96.924% with MP3 encoded audio files and 96.915 with AAC encoded audio files. The average accuracy with AMR encoded audio files is equal to 87.829% if we consider all the available bitrates, and rises to 89.847% if we consider only the bitrates higher or equal to 7.4 kbps.

The inter-device classification of the built-in test set both for PCM, MP3 and AAC encoded audio files achieves an average accuracy higher then 99%. In particular, the average accuracy is equal to 99.688% with PCM encoded audio files, 99.912% with MP3 encoded audio files and 99.779 with AAC encoded audio files. The average accuracy with AMR encoded audio files is equal to 95.762% if we consider all the available bitrates, but drops to 95.703% if we consider only the bitrates higher or equal to 7.4 kbps.

Also the inter-device classification of the full test set both for PCM, MP3 and AAC encoded audio files achieve an average accuracy higher then 99%. In particular, the average accuracy is equal to 100% with PCM encoded audio files, 99.992% with MP3 encoded audio files and again 100% with AAC encoded audio files. The average accuracy with AMR encoded audio files is equal to 96.182% if we consider all the available bitrates, and rises to 96.182% if we consider only the bitrates higher or equal to 7.4 kbps.

The research that represents the state-of-the-art on inter-device classification is the one by Romero and Wilson [9]: they performed an inter-device classification on two different sets of 8 devices, achieving an accuracy of 99.0% on the ICSI subset of the NIST 2006 Speaker Recognition Evaluation database, and of 93.2% on half of the LLHDB subset of the HTIMIT database. The devices included in the ICSI subset are totally uncorrelated between them, both regarding the model and the quality of the recording device, e.g. an ear-wrap microphone, a condenser microphone, a built-in microphone from a laptop, an hand recorder and so on. On the other hand, the devices included in the LLHDB can be considered to have all the same quality - they are all landlines telephone handset - but different construction properties; the only possibly shared characteristic is the transducer class - electrect or carbon-button. Our test set is composed each time by 8 devices that are from different models, that can considered to share the same quality - since they are all microphones from mobile devices - and that pairwise show a correlation, as mentioned several times. Both on PCM, MP3 or AAC encoded audio files we have an accuracy score slightly higher or slightly lower then 97%, that is unfortunately difficult to compare with the tests performed in [9]. In terms of pure accuracy the results on the ICSI subset are higher, but this accuracy drops dramatically when the system is applied on the LLHDB subset. We believe that our classification algorithm could be considered on par with the one from Romero and Wilson, since the average accuracy also with MP3 or AAC encoded audio files drops less then 0.2% - a property that is not granted by their approach. Moreover, when the devices are not correlated, i.e. the inter-device classification task on the built-in test set and on the headset test set, PCM ,MP3 and AAC encoded audio file are classified with an average accuracy higher then 99%, and AMR encoded audio files with an average accuracy higher then 95%, despite the low bitrates involved.

5.3.2 Model Classification

During the model classification evaluation we put in the same class namely identical devices: as a consequence we don't need anymore the concept of intra-device or inter-device classification. We expect as the main source of misclassification the presence of headsets built from the same manufacturer of the mobile device they are meant to be used with; moreover, it's also possible that our parametrization of the GMM, as well as the feature selection, are not the optimal ones for this task, since both were set in order to maximize the accuracy of the device classification.

We will address this problem as we did for the device classification evaluation, i.e. by performing the test not only on the full test set, but also on the built-in test set and on the headset test set, where this correlation should vanish.

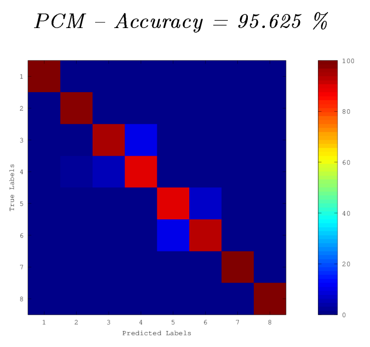
All the following results were achieved with RBF kernel functions with parameters $(c_{RBF}, \gamma_{RBF}) = (2^8, 2^{-9})$, with the feature selection $J^{select} = J^{select}_{MP3}$, and with a dimension normalization between -1 and 1.

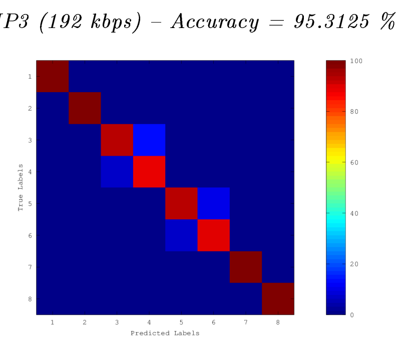
5.3.2.1 Full Test Set

The full test set is composed of 8 classes, that will be labeled numerically as follows:

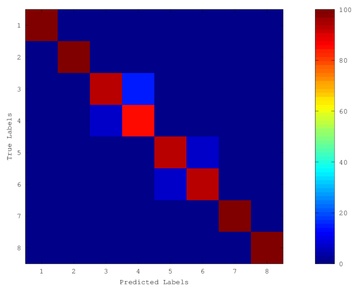
Label	Devices	Class
1	$Dell_1$ and $Dell_2$, with the *built-in* microphone	$Dell^{built-in}$
2	$Dell_1$ and $Dell_2$, with the *headset* microphone	$Dell^{headset}$
3	$GooglePhone_1$ and $GooglePhone_2$, with the *built-in* microphone	$GooglePhone^{built-in}$
4	$GooglePhone_1$ and $GooglePhone_2$, with the *headset* microphone	$GooglePhone^{headset}$
5	$iPhone_1$ and $iPhone_2$, with the *built-in* microphone	$iPhone^{built-in}$
6	$iPhone_1$ and $iPhone_2$, with the *headset* microphone	$iPhone^{headset}$
7	$GalaxyS2_1$ and $GalaxyS2_2$, with the *built-in* microphone	$GalaxyS2^{built-in}$
8	$GalaxyS2_1$ and $GalaxyS2_2$, with the *headset* microphone	$GalaxyS2^{headset}$

Again, for each compression algorithm we had several confusion matrices, one per bitrate. A graphical representation of the confusion matrices of the intra-device classification is shown in Figure 5.16.

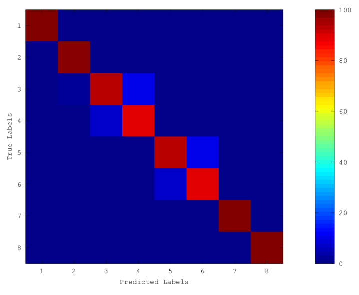




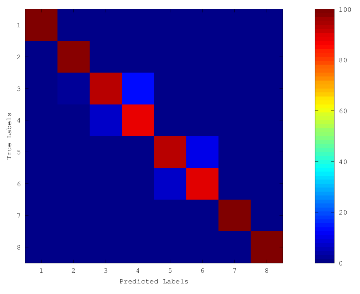
MP3 (128 kbps) – Accuracy = 95.3125 %



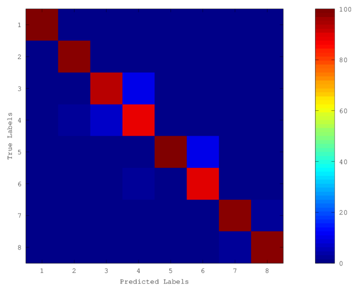
MP3 (96 kbps) – Accuracy = 95.3125 %



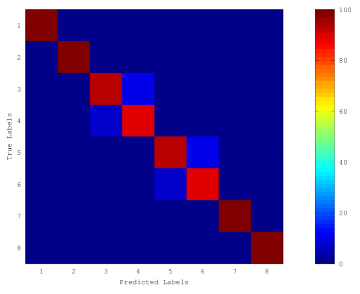
MP3 (64 kbps) – Accuracy = 95 %



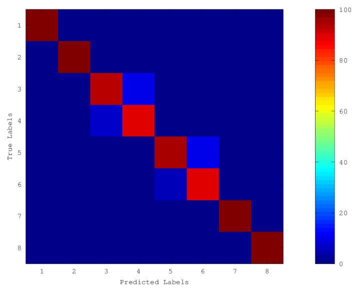
MP3 (32 kbps) – Accuracy = 95.3125 %



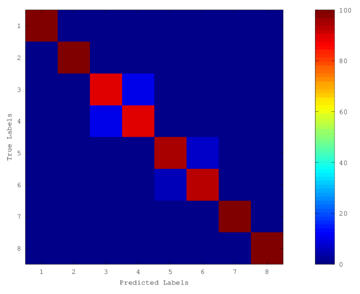
AAC (128 kbps) – Accuracy = 95.625 %



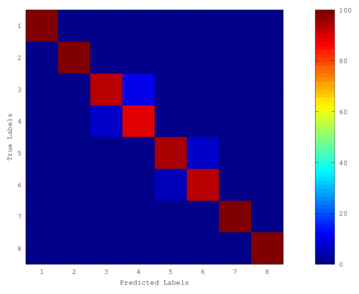
AAC (96 kbps) – Accuracy = 95.9375 %



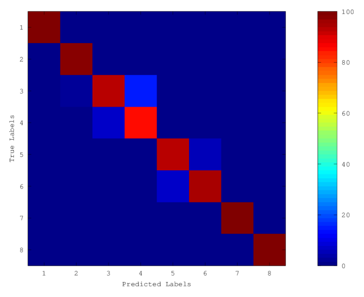
AAC (64 kbps) – Accuracy = 95.9375 %



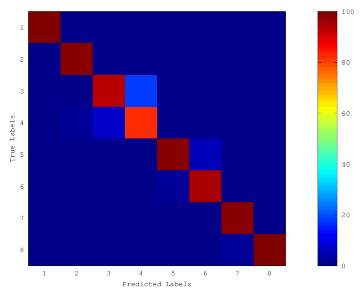
AAC (48 kbps) – Accuracy = 96.25 %



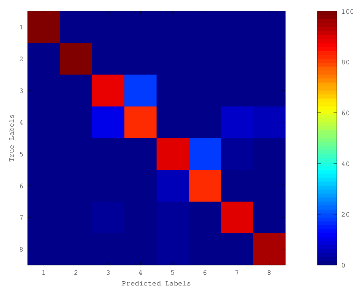
AAC (32 kbps) – Accuracy = 95.3125 %



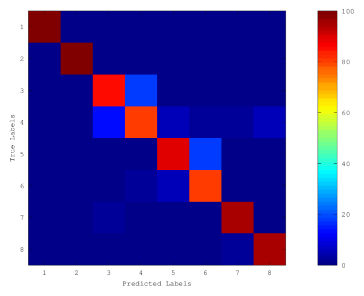
AAC (16 kbps) – Accuracy = 95.3125 %



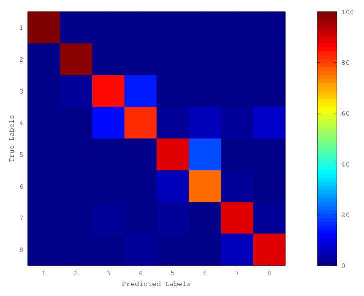
AMR (12.2 kbps) – Accuracy = 90.9375 %



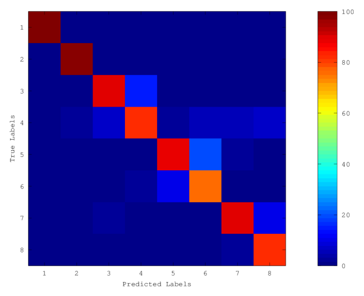
AMR (10.2 kbps) – Accuracy = 90.625 %



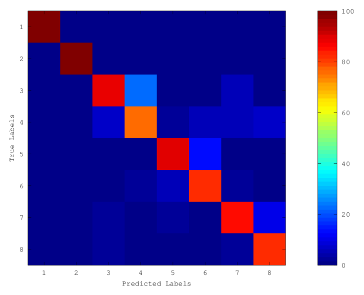
AMR (7.95 kbps) – Accuracy = 88.75 %



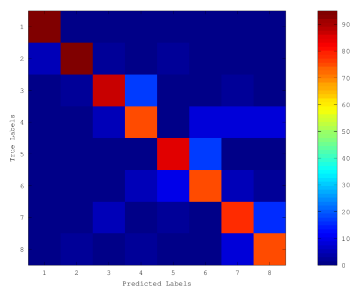
AMR (7.4 kbps) – Accuracy = 88.125 %



AMR (6.7 kbps) – Accuracy = 87.8125 %



AMR (5.9 kbps) – Accuracy = 83.125 %



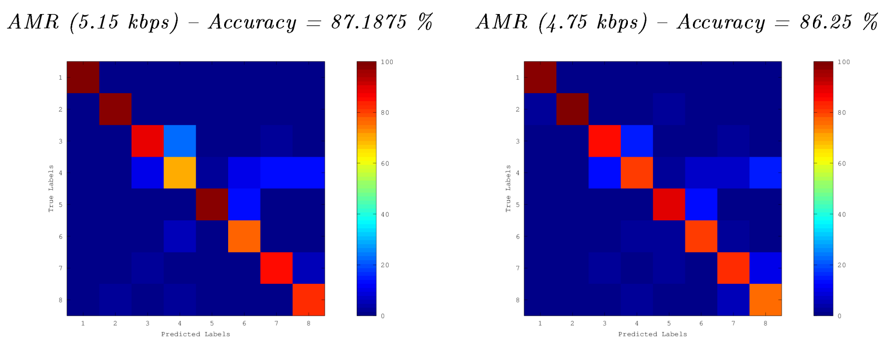


Figure 5.16: Model Classification - Full Test Set

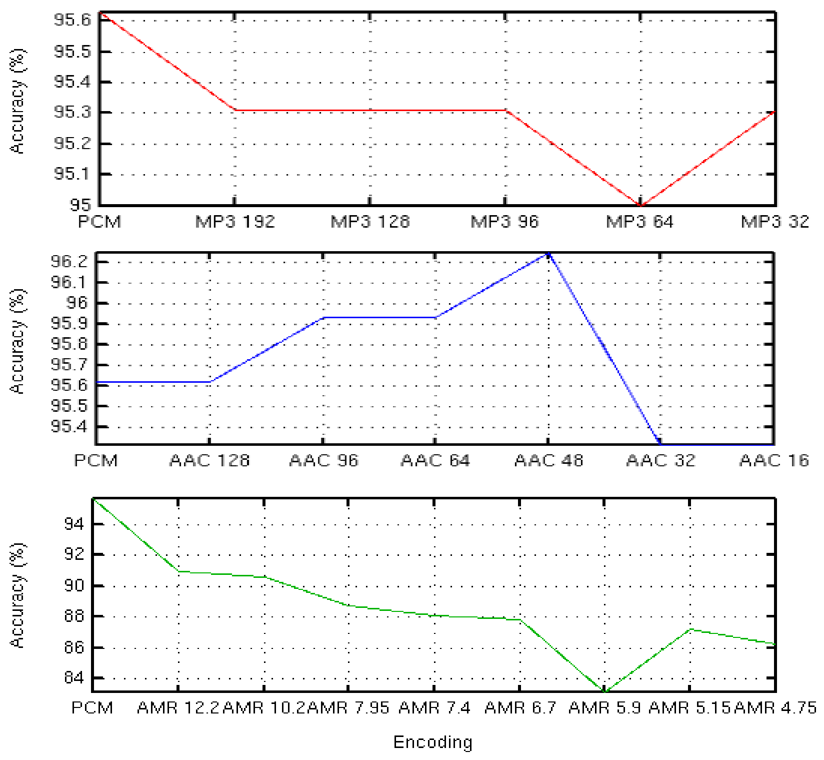


Figure 5.17: Model Classification - Full Test Set: Accuracy Vs Encoding

The confusion matrices resemble square block matrices with 2×2 blocks, highlighting a correlation between the pairs $(1, 2), (3, 4), (5, 6), (7, 8)$. The existence of these pairs, from the labeling stated herebefore, corresponds to a correlation between the classes *($Dell^{built-in}$, $Dell^{headset}$), ($GooglePhone^{built-in}$,*

$GooglePhone^{headset}$), ($iPhone^{built-in}$, $iPhone^{headset}$), ($SamsungGalaxyS2^{built-in}$, $SamsungGalaxyS2^{headset}$).

This confirms that our assumption of an existing relationship between the built-in microphones and the corresponding headsets built from the same manufacturer of the mobile device they are meant to be used with holds.

Also in the model classification is straightforward to notice that the encoding affects the accuracy of the classification. It's interesting to notice that the result of the compression with respect to the PCM encoding is not a monotonically decrease of the accuracy, as we would expect; moreover, with the AAC compression sometimes the algorithm performs better on compressed audio files than on the PCM encoded ones, which is counterintuitive. The AMR compression is definitely the one which creates the highest drop of accuracy, both due to it's encoding algorithm and due to the extremely low bitrates, it does not cause a monotonically decrease of the accuracy, as for MP3 and AAC.

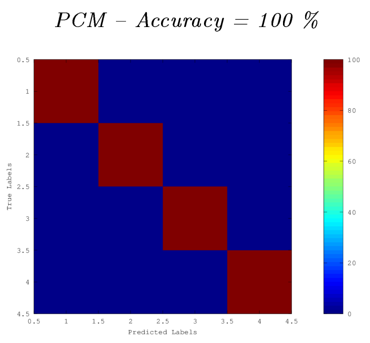
In Figure 5.17 it's possible to find a graphical representation of these result. Unfortunately, we are still not able to completely understand the relationship between the AAC compression and such a behavior: hopefully in the future it would be possible to exploit its characteristic in order to enhance the accuracy of the classification algorithm.

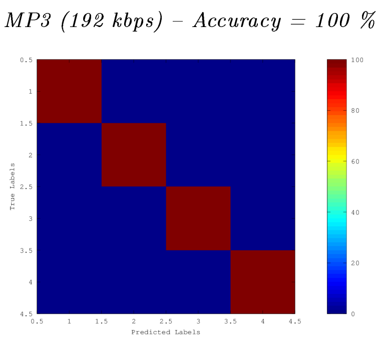
5.3.2.2 Built-in Test Set

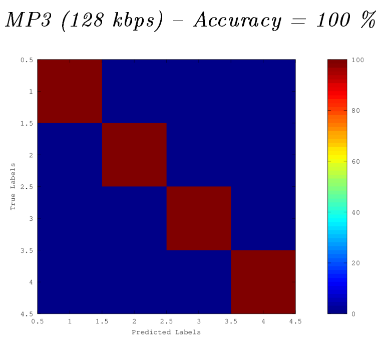
The built-in test set is composed of 4 classes, that will be labeled numerically as follows:

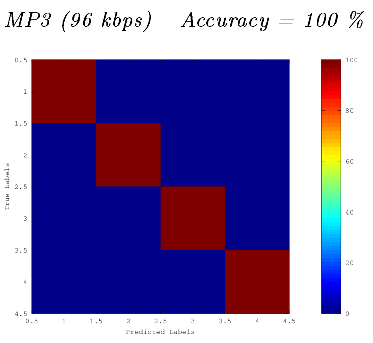
Label	Devices	Class
1	$Dell_1$ and $Dell_2$, with the *built-in* microphone	$Dell^{built-in}$
2	$GooglePhone_1$ and $GooglePhone_2$, with the *built-in* microphone	$GooglePhone^{built-in}$
3	$iPhone_1$ and $iPhone_2$, with the *built-in* microphone	$iPhone^{built-in}$
4	$GalaxyS2_1$ and $GalaxyS2_2$, with the *built-in* microphone	$GalaxyS2^{built-in}$

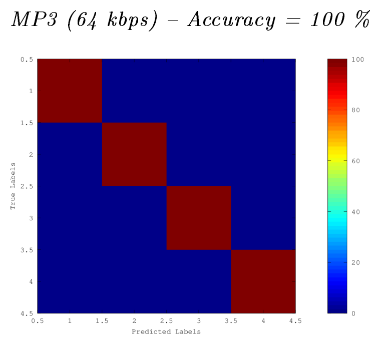
The confusion matrices of the built-in test set are plotted in figure 5.18:

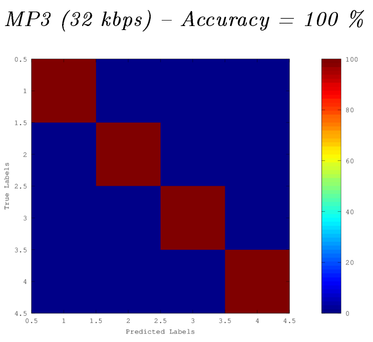


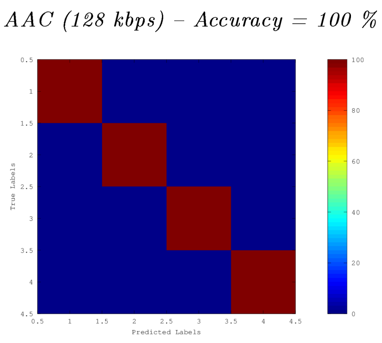


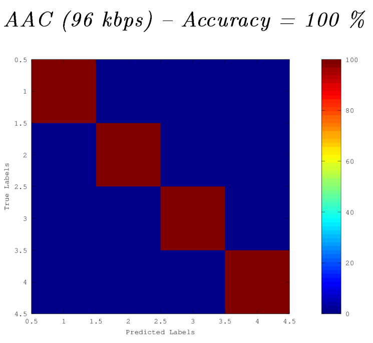

MP3 (128 kbps) – Accuracy = 100 %
True Labels
Predicted Labels

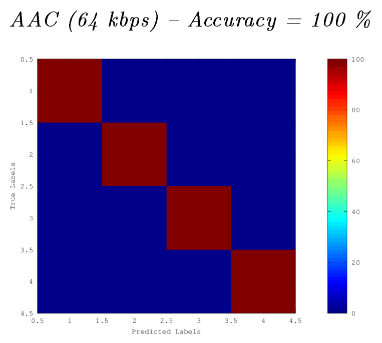

MP3 (96 kbps) – Accuracy = 100 %
True Labels
Predicted Labels

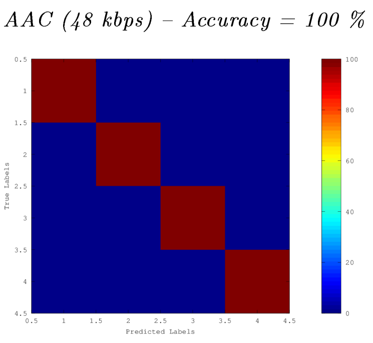

MP3 (64 kbps) – Accuracy = 100 %
True Labels
Predicted Labels


MP3 (32 kbps) – Accuracy = 100 %
True Labels
Predicted Labels

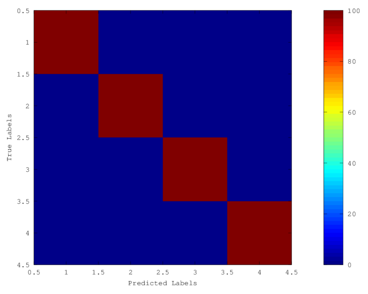

AAC (128 kbps) – Accuracy = 100 %
True Labels
Predicted Labels


AAC (96 kbps) – Accuracy = 100 %
True Labels
Predicted Labels

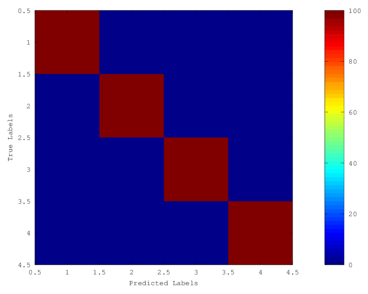

AAC (64 kbps) – Accuracy = 100 %
True Labels
Predicted Labels


AAC (48 kbps) – Accuracy = 100 %
True Labels
Predicted Labels

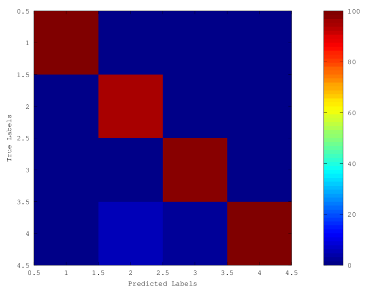
AAC (32 kbps) – Accuracy = 100 %



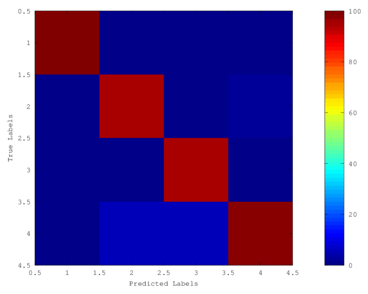
AAC (16 kbps) – Accuracy = 100 %



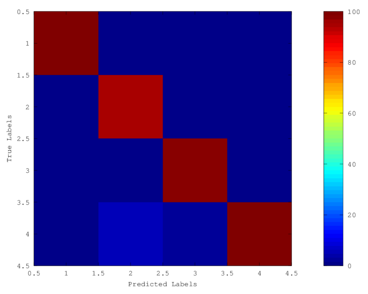
AMR (12.2 kbps) – Accuracy = 98.125 %



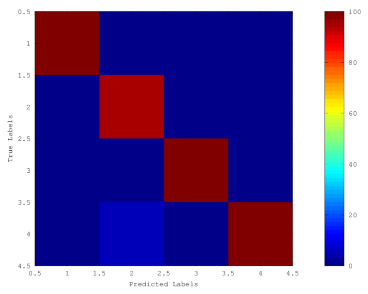
AMR (10.2 kbps) – Accuracy = 96.875 %



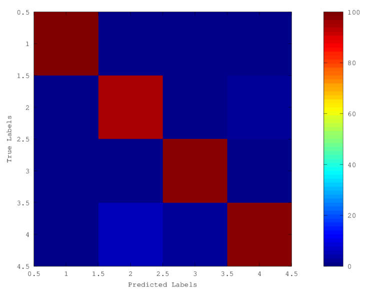
AMR (7.95 kbps) – Accuracy = 98.125 %



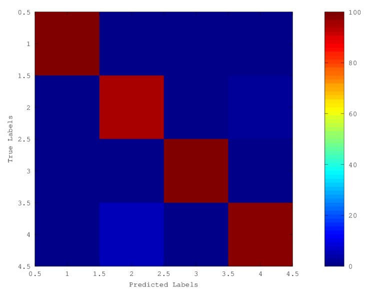
AMR (7.4 kbps) – Accuracy = 98.75 %



AMR (6.7 kbps) – Accuracy = 97.5 %



AMR (5.9 kbps) – Accuracy = 98.125 %



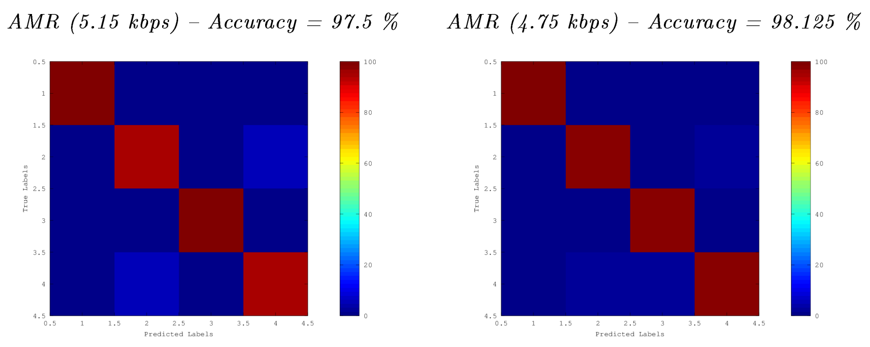


Figure 5.18: Model Classification - Built-in Test Set

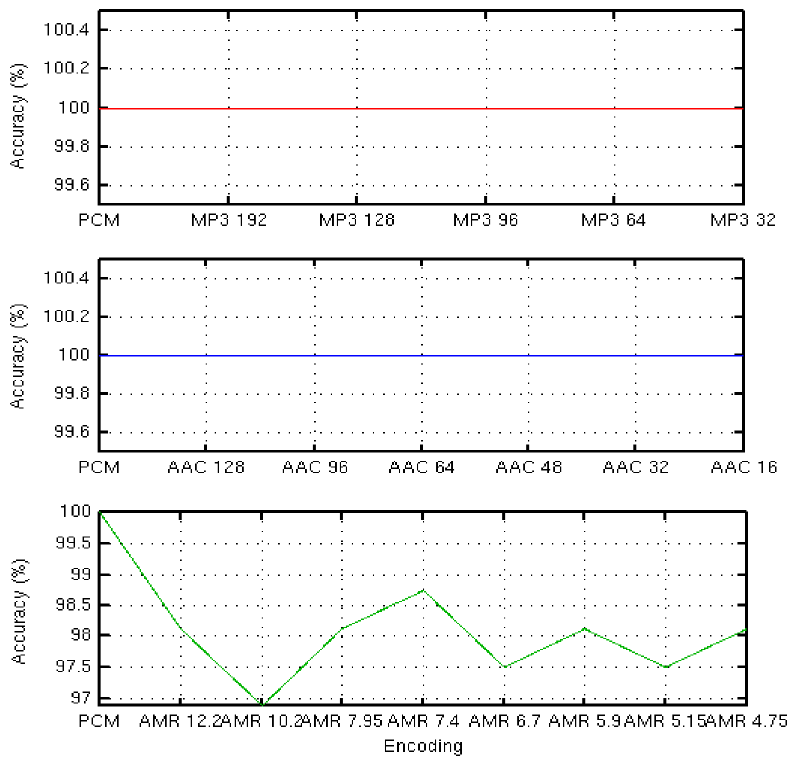


Figure 5.19: Model Classification - Built-in Test Set: Accuracy Vs Encoding

On the built-in test set the model classification achieves high performances: we have no more the correlation between coupled built-in microphones and headset microphones and it seems that, thanks to our channel estimate based feature vector, the peculiar characteristic of each class were modeled in an extremely good way.

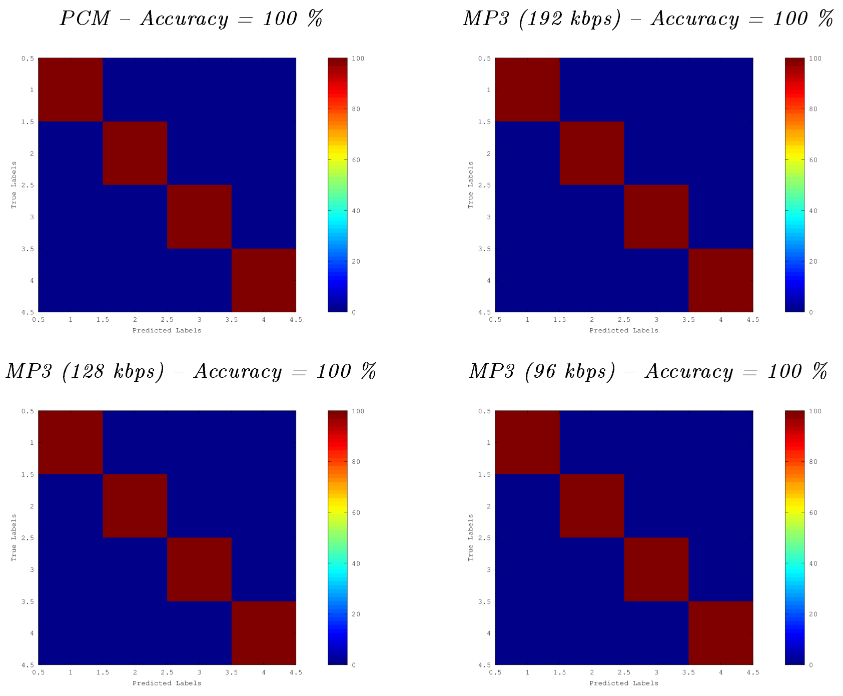
Moreover, in this scenario our algorithm proved to be strong against compression: both PCM, MP3 and AAC encoded audio files are classified with a 100% accuracy independently from the bitrate. The AMR encoding creates a slight decrease of the accuracy, that we can tolerate since the extremely low target bitrates of this encoding algorithm. A visual summary of the encoding effect can be found in figure 5.19

5.3.2.3 Headset Test Set

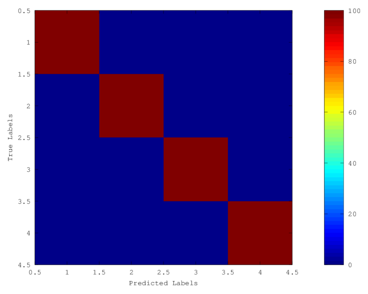
The headset test set is also composed of 4 classes, that will be labeled numerically as follows:

Label	Devices	Class
1	$Dell_1$ and $Dell_2$, with the *headset* microphone	$Dell^{headset}$
2	$GooglePhone_1$ and $GooglePhone_2$, with the *headset* microphone	$GooglePhone^{headset}$
3	$iPhone_1$ and $iPhone_2$, with the *headset* microphone	$iPhone^{headset}$
4	$GalaxyS2_1$ and $GalaxyS2_2$, with the *headset* microphone	$GalaxyS2^{headset}$

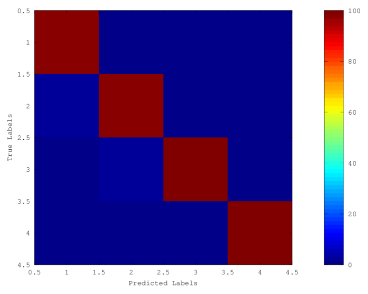
The confusion matrices of the headset test set are plotted in figure 5.20.



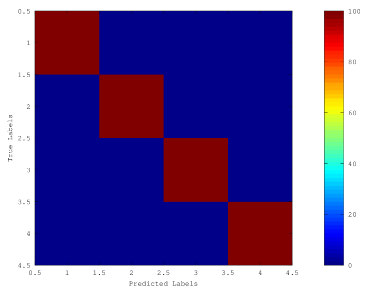
MP3 (64 kbps) – Accuracy = 100 %



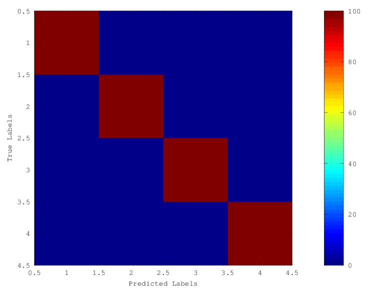
MP3 (32 kbps) – Accuracy = 98.75 %



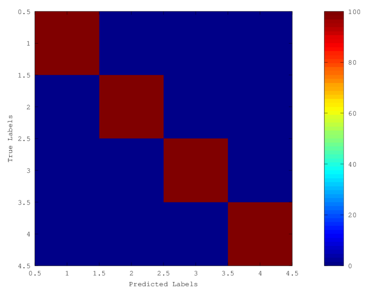
AAC (128 kbps) – Accuracy = 100 %



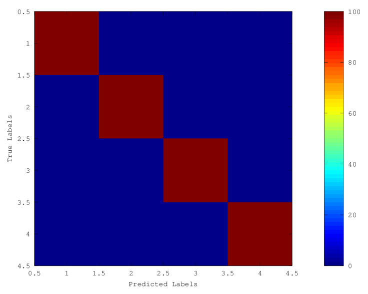
AAC (96 kbps) – Accuracy = 100 %



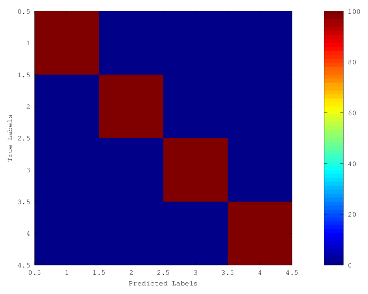
AAC (64 kbps) – Accuracy = 100 %



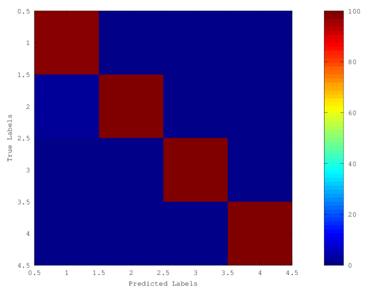
AAC (48 kbps) – Accuracy = 100 %



AAC (32 kbps) – Accuracy = 100 %



AAC (16 kbps) – Accuracy = 99.375 %



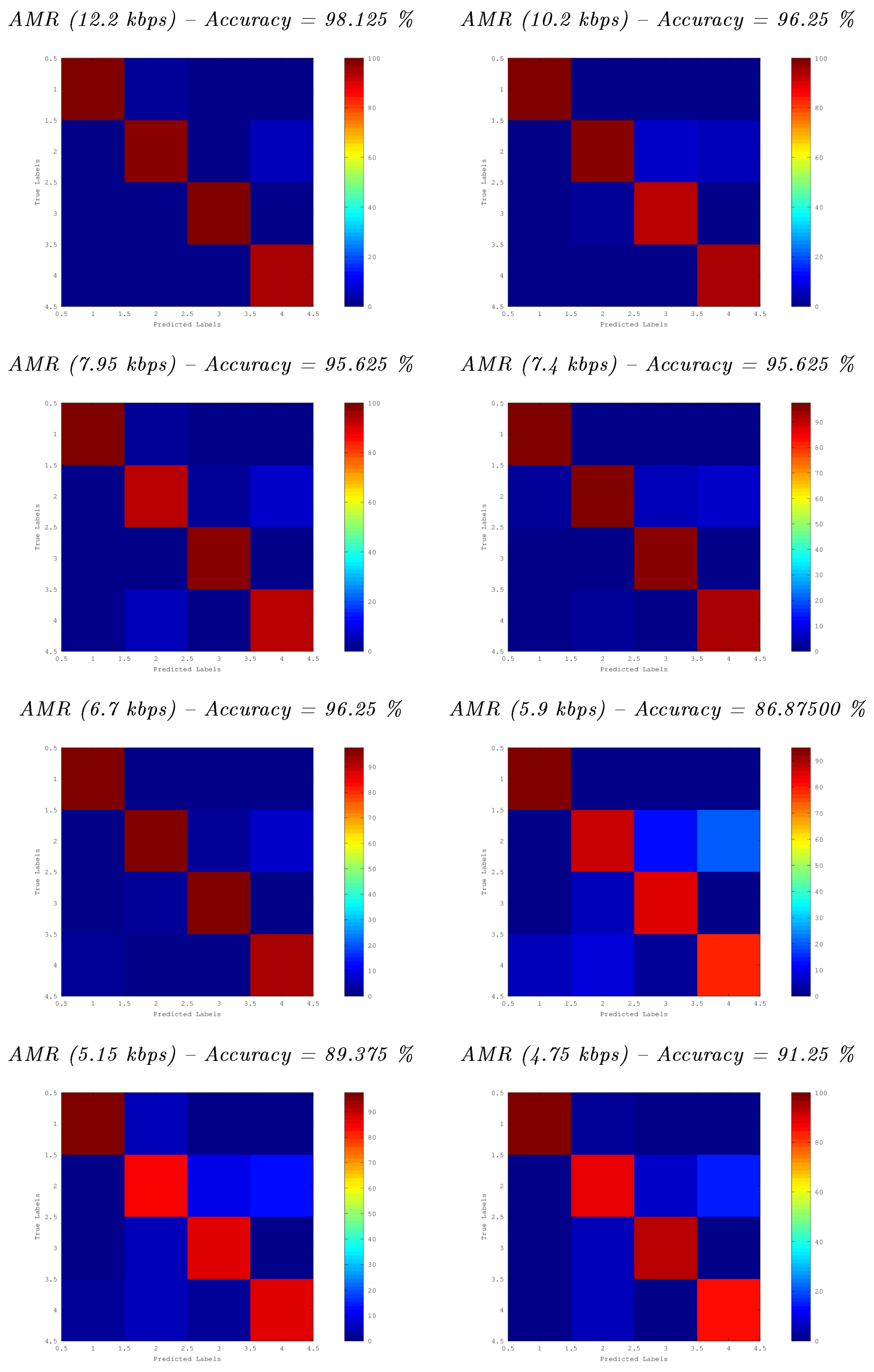


Figure 5.20: Model Classification - Headset Test Set

Also with the headset subset the model classification works extremely good: we achieved a 100 % accuracy on PCM encoded audio files, on all MP3 encoded audio files but those encoded at 32 kbps, and on all AAC encoded audio files but those encoded at 16 kbps. The AMR encoded audio files are still identified correctly with an high accuracy, apart from bitrates lower than 6 kbps.

If we compare these results with those from the built-in test set we can't avoid to notice than the drop in accuracy that we suffer due to the AMR encoding is much more relevant with the headset devices. This should be mainly due to the noise found on the channel estimate of the magnitude response of the channel computed on the headset: from the start, the estimates of the headset devices were not as compact as those from the built-in devices, and it's possible that this reduces the robustness of out model to the AMR lossy encoding.

A graphical representation of the accuracy as a function of the encoding can be found in figure 5.21.

5.3.2.4 Final Considerations

Even if our microphone classification proposal was not meant to carry out model identification, the results prove that - even with some limitations - our channel estimation based algorithm works with an extremely high accuracy on sets with no strong correlations between its member, when applied to model classification.

For the full test set both PCM, MP3 and AAC encoded audio files are classified with an accuracy higher than 90%: with the PCM encoding we have a 95.625% accuracy; on MP3 encoded audio files we have an average accuracy of 95.25 %; on AAC encoded audio files we have an average accuracy of 95.729 %. The average accuracy for the AMR encoded full test set was 87.852%, but if we consider only the bitrates higher or equal to 7.4 kbps, i.e. the nominal toll quality bitrate for clean speech only recordings[7], the accuracy rises to 89.609 %, that we can consider to be high due to the harshness of the AMR encoding algorithm.

For the built-in test set both PCM, MP3 and AAC encoded audio files are classified with an accuracy of 100%. The AMR average accuracy across all the available bitrates is 97.891 %, and it's equal to 97.969 % if we consider only the toll quality bitrates.

For the headset test set PCM encoded audio files are classified with an accuracy of 100%, together with all the MP3 or AAC encoded audio files with bitrates greater than 32 kbps: the average accuracy on MP3 encoded audio files is 99.75 %, and for AAC encoded audio files is 99.896 %. The AMR average accuracy across all the available bitrates is 93.672%, and it's equal to 96.406 % if we consider only the toll quality bitrates.

[7]which is not the test content tested used during this work, see Section 5.1.2

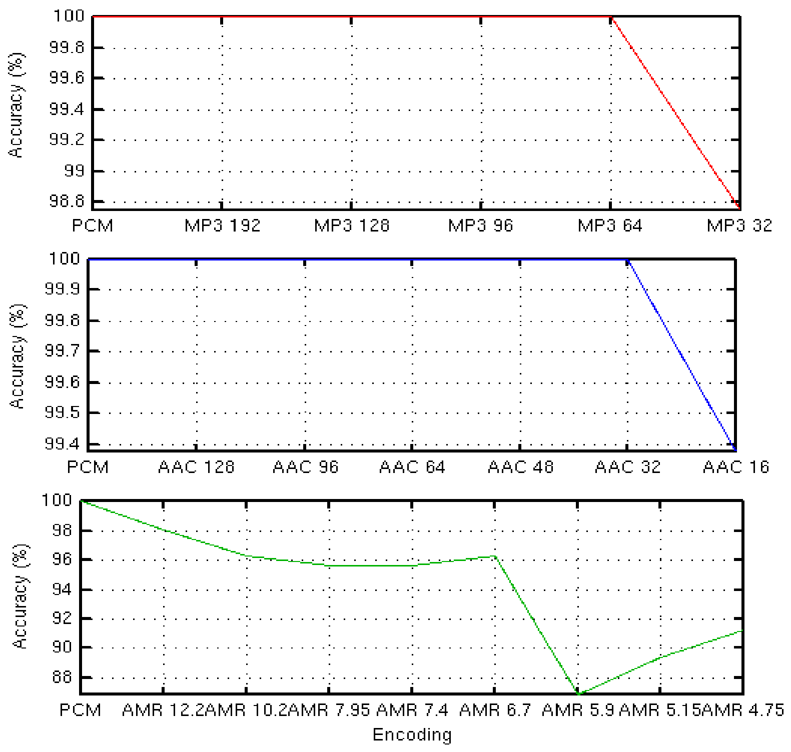


Figure 5.21: Model Classification - Headset Test Set: Accuracy Vs Encoding

In conclusion, we provided a model classification algorithm reliable both with PCM encoded audio files and with MP3 and AAC encoded audio files; the model classification with AMR encoded audio files is possible, but still needs a further work due to the peculiarity of their compression scheme.

5.4 Feature Vector Influence on the Classification

In order to evaluate the influence of the feature vector introduced in Section 4.1, we tried to perform the same evaluations considering a baseline framework, where the channel estimate computed starting from the results by Gaubitch et al. [13] is used directly as the feature vector for the classification.

The test were performed including intra-device classification, inter-device classification and model classification using all the three subsets previously defined. in Figure 5.22 we can visually compare the difference for the intra-device classification of the full test; the normal framework is plotted in red, and the baseline framework in green.

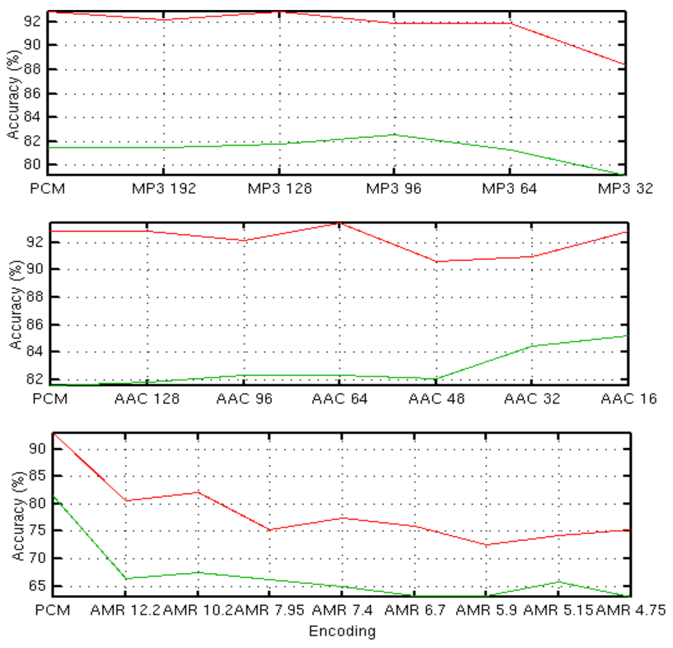


Figure 5.22: Baseline Framework - Intra-Device Classification

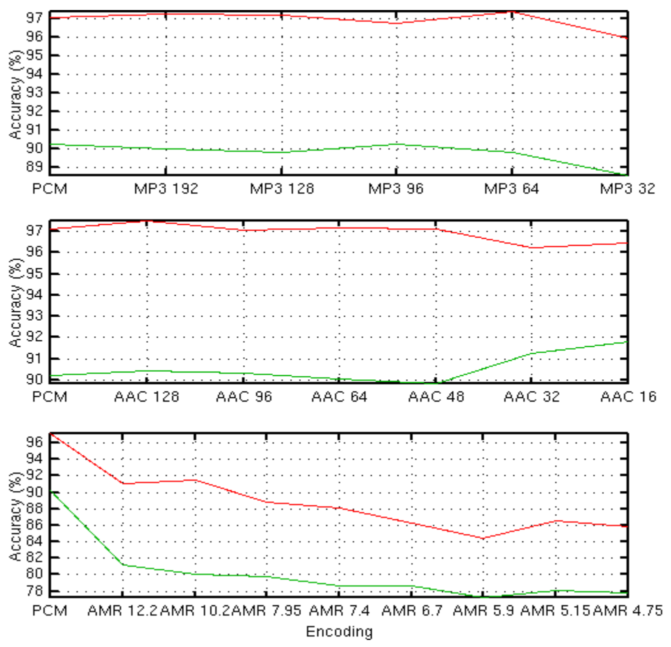


Figure 5.23: Baseline Framework - Intra-Device Classification

We can see how, independently of the encoding, the average difference in terms of accuracy is constantly around 10%. The same happens if we consider the inter-device classification on the full test sets, plotted in Figure 5.23.

The baseline framework never performed better then the normal one, so we won't discuss it further in this chapter. Readers interested can find the complete results of the best performing baseline framework in Appendix C.

5.5 Skype Processing Considerations

In order to correctly evaluate the influence of the Skype processing on the channel estimates we need to correlate somehow the bandwidth available to the perceived Quality of Service (QoS) of the Skype calls: intuitively, if the bandwidth available is low the source encoding performed by Skype should target a lower bitrate, thus overcoming the low condition of the network.

The only research investigating the Skype QoS, written by Baset and Schulzrinne [30] , dates back to 2006: the two authors observed that uplink and downlink bandwidth of 2 kB/s each were necessary for bare minimum audible call quality, and that the voice was almost unintelligible at an uplink and downlink bandwidth of 1.5 kB/s.

The main change with respect to 2006 according to the official information form Skype is the introduction of a new audio codec (SILK), whose minimum bit-rate (6 kbps) is lower than the one (10 kbps) involved during the cited research (the iSAC codec). When only simple calls are involved, the minimum speed requirements by Skype are uplink and downlink bandwidth of at least 30 kbps, i.e. 3.75 kB/s each.

We chose to test the two device previously labeled as $Dell_1^{built-in}$ and $Dell_2^{built-in}$. For each device we recorded four short sentences with different bandwidth, i.e. 10 kB/s, 5 kB/s, 2.5 kB/s, 1 kB/s and 0.5 kB/s. The first difference that we noticed with the research from Baset and Schulzrinne is that, even with an uplink and downlink bandwidth of 0.5 kB/s each, we were still able to understand the speech, as long as only one person is talking: if both the speakers talk at once with a bandwidth of 10 kB/s is already possible to experience packet drops and severe degradation of the intelligibility.

All the recordings were made by using the software Callburner 1.0.0.83: this software allows the user to collect in separate files the speech from the primary speaker, the speech from the secondary speaker and the mixed version[8]. Thus, we were able to acquire each recording before and after the transmission on the network; and to perform a classification using the same training content and the same SVM parameters selection of the previous sections.

Unfortunately the outcome of the classification was extremely disappointing, since only 8 recordings out of 80 were correctly classified. This is understandable: when Skype is involved and Callburner is used, the audio source from the primary speaker is recorded only after the SILK compression of the source, thus violating the process flow stated in Section 2.2.2; we can suppose that the SILK encoding target bitrates is somehow driven by the overall bandwidth availability. As for the AMR compression the SILK encoder influences a lot the signal, to the point that - unfortunately for our purposes - the channel estimate can't be classified with an SVM trained on PCM encoded audio file. However, we

[8]See the process flow of the Skype interview in figure 2.2.

believe that, if the SILK encoder influence is considered from the start like with the MP3, AAC and AMR compression algorithm, is indeed possible to achieve good results also when using speech recordings from Skype.

Chapter 6

Summary and Future Work

The proposed method for microphone classification proved to be extremely reliable both on PCM encoded audio files and on audio files compressed with the most popular compression algorithms, i.e. both the MP3 and the AAC. Moreover, as extensively reported in Chapter 5, this reliability holds also when the audio files are compressed with the lowest bitrates, despite the perceivable loss of information that these operations mean. If the AMR compression scheme is involved the system accuracy unfortunately decreases consistently, both due to the bitrates involved, ranging from a maximum of 12.2 kbit/s to a minimum of 4.75 kbit/s, and because of the underlying Code-Excited Linear Predictive (CELP) coding model, which provides a reliable method for speech coding, but consistently reduces the information available about the microphone channel.

The results provided in Section 5.3.1 and 5.3.2 clearly shows that, when the extremely low bitrates available with the AMR compression algorithm are involved, the classification of the device is still possible up to the nominal toll quality bitrate of 7.4 kbit/s - with an accuracy higher then 95% - only if in the test set the devices are not correlated between each other. The same happens if we are not interested in the single device, but only on its general model. When considering test sets with strong relations between the recording devices AMR compressed audio files should be avoided, while both MP3, AAC and PCM encoded audio files can be classified with an high consistency of the result, up to be considered on par with the current state of the art on microphone classification.

As far as we know lossy compressed audio files were never involved in research papers about microphone classification: due to the overwhelming presence of encoded audio files on the network and in contest when the authenticity the content is an issue - e.g. when the audio file is presented in a courtyard - the robustness of the algorithm can be considered an highly desirable feature and represents a precedent.

This framework relies on the recent work by Gupta et al. [13], that provides an algorithm for the blind estimation of the magnitude response of a channel using the observations of a single microphone. Their research can be considered still in its early stages, and every enhancement on the single-microphone blind channel estimation will positively influence the accuracy of our proposal. Another possible improvement could derive from a pre-processing of the input audio files with a de-reverberation algorithm: this operation in theory would lessen the influence of the environment on the recording, thus enhancing the discriminant power of the magnitude response of the microphone channel and consequently of our algorithm.

Finally, it's possible to combine our proposal with a tampering/editing detection algorithm in order to enhance its reliability: at the present ENF based techniques and local noise level estimation based technique are able to perform splicing detection, but can only highlight possible suspect regions by the means of their borders. This strong hint can be double-checked by our algorithm by searching for inconsistencies regarding the device involved in the suspect regions.

Glossary

AAC	Advanced Audio Coding
AMR	Adaptive Multi-Rate Coding
AWGN	Additive White Gaussian Noise
ENF	Electric Network Frequency
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
FMFCCs	Filtered Mel-Frequency Cepstral Coefficients
GMM	Gaussian Mixture Model
GSV	Gaussian Super Vector
HE-AAC	High-Efficiency Advanced Audio Coding
HTK	Hidden Markov Model Toolkit
IIR	Infinite Impulse Response
LFCCs	Linear-Frequency Cepstral Coefficients
LMSCs	Logarithmic Mel-Spectral Coefficients
LTASS	Long Term Average Speech Spectrum
MFCCs	Mel-Frequency Cepstral Coefficients
RASTA	RelAtive SpeTrAl
RBF	Radial Basis Function
SNR	Signal to Noise Ratio
STFT	Short Time Fourier Transform
SVM	Support Vector Machine
UBM	Universal Background Model
WGN	White Gaussian Noise

Bibliography

[1] C. Kraetzer et al. *Digital Audio Forensics: A First Practical Evaluation on Microphone and Environment Classification.* Int'l Conf. Multimedia Conference, Proc. 9th Workshop Multimedia & Security, ACM Press, 2007, pp. 6374.

[2] Gupta, Cho, Kuo. *Current Developments and Future Trends in Audio Authentication.* Multimedia in Forensics, Security and Intelligence, p. 50-59, 2012.

[3] REWIND Consortium. *Deliverable D3.1: State-of-the-art on Multimedia Footprint Detection.* 2011.

[4] C. Kraetzer, K. Qian, M. Schott, J. Dittmann. *A Context Model for Microphone Forensics and Its Application in Evaluations.* SPIE Conference on Media Watermarking, Security, and Forensics, 2011.

[5] H. Malik and H. Farid. *Audio Forensics From Acoustic Reverberation.* Proveedings of the IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP), no. iid, 2010, pp. 1710-1713.

[6] H. Malik and H. Zhao. *Recording Environment Identification Using Acoustic Reverberation.* IEEE International Conference on Acoustic Speech and Signal Processing (ICASSP), 2012.

[7] A. Oermann. *Verifier-Tuple for Audio-Forensic to Determine Speaker Environment.* MM&Sec '05: Proceedings of the 7th workshop on Multimedia and security, New York, NY, USA, 2005, pp. 57-62.

[8] R. Buchholz, C. Kraetzer, J. Dittman. *Microphone Classification Using Fourier Coefficients.* Information Hiding, LNCS 5806, Springer, 2009, pp. 235.246.

[9] D. Garcia-Romero and C. Y. Espy-Wilson. *Automatic Acquisition Device Identification from Speech Recordings.* IEEE International Conference on Acoustic Speech and Signal Processing (ICASSP), 2010, pp. 1806-1809.

[10] D. Nicolalde and J. Apolinario *Evaluating digital audio authenticity with spectral distances and ENF phase change.* IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP), 2009.

[11] X. Pan, X. Zhang, S. Lyu. *Detecting Splicing in Digital Audios Using Local Noise Level Estimation.* IEEE International Conference on Acoustic Speech and Signal Processing (ICASSP), 2012.

[12] Nikolay D. Gaubitch, Mike Brookes, Patrick A. Naylor. *Blind Channel Identification in Speech Using the Long-Term Average Speech Spectrum.* Proc. European Signal Processing Conference (EUSIPCO), Glasgow, Aug. 2009.

[13] Nikolay D. Gaubitch, Mike Brookes, Patrick A. Naylor, Dushyant Sharma. *Single-Microphone Blind Channel Identification in Speech using Spectrum Classification.* Proc. 17th European Signal Processing Conf. (EUSIPCO-11), Barcelona, Spain, Aug., 2011.

[14] Y. Hu and P. Loizou. *Subjective evaluation and comparison of speech enhancement algorithms.* Speech Communication, 49, 588-601, 2007.

[15] *Hidden Markov Model Toolkit.* Official website: http://htk.eng.cam.ac.uk/.

[16] Kamil Wojcicki. *HTK MFCCs.* Sep. 2011. Avaible online at http://www.mathworks.com/matlabcentral/fileexchange/authors/39321

[17] H. Hermansky and N. Morgan. *RASTA processing of speech.* IEEE Trans. Speech Audio Process., vol. 2, no. 4, pp. 578–589, Oct. 1994.

[18] R Rifkin and A Klautau. *In defense of one-vs-all classication.* Journal of Machine Learning Research Vol. 5, pp- 101-141.

[19] M. Slaney. *Auditory Toolbox v2.* Avaible online at https://engineering.purdue.edu/~malcolm/interval/1998-010/.

[20] U. von Luxburg. *A Tutorial on Spectral Clusterin*g. Statistics and Computing Vol. 17 Issue 4, pp. 395-416, December 2007.

[21] *GNU Octave.* Sofware available online at http://www.gnu.org/software/octave/index.html.

[22] Sylvain Calinon. *Gaussian Mixture Model (GMM) - Gaussian Mixture Regression (GMR).* 2009. Avaible online at http://www.mathworks.com/matlabcentral/fileexchange/authors/30869.

[23] Chang and Lin. *LIBSVM : a library for support vector machines.* ACM Transactions on Intelligent Systems and Technology Vol. 2 Issue 3, 2011. Software available online at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[24] *LAME Ain't an MP3 Encoder (LAME).* Software available online at http://lame.sourceforge.net/index.php.

[25] *Nero AAC Codec.* Software available online at http://www.nero.com/enu/technologies-aac-codec.html.

[26] *OpenCORE Adaptive Multi Rate (AMR) Speech Codec.* Software available online at http://lame.sourceforge.net/index.php.

[27] Naoki Shibata. *Shibatch Audio Tools - Quality Audio Tools and More.* Software available online at http://shibatch.sourceforge.net/.

[28] Y.W. Chen and C.J. Lin. *Combining SVMs with various feature selection strategies.* 2005.

[29] Skype Developer website. *SILK: Super Wideband Audio Codec.* 2009. Documentation available online at https://developer.skype.com/silk.

[30] S.A. Baset and H.G. Schulzrinne. *An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol.* INFOCOM, 2006.

[31] 3GPP. *3GPP TS 26.090 - Mandatory Speech Codec speech processing functions; Adaptive Multi-Rate (AMR) speech codec; Transcoding functions.* 2010.

[32] 3GPP. *3GPP TS 26.071 - Mandatory speech CODEC speech processing functions; AMR speech Codec; General description.* 2010.

Appendix A

Implementation Framework

A.1 Octave

The algorithm has been completely implemented under GNU Octave 3.4.3 [21], installed on a machine running the Fedora 16 x86-64 GNU/Linux OS.

Our work also includes some external routines, namely

- Kamil Wojcicki's algorithm for the computation of the HTK MFCCs [16].
- Malcolm Slaney's implementation of the RASTA filtering, part of the Auditory Toolbox v2 [19].
- Sylvain Calinon's implementation of the GMM training [22].
- LIBSVM library for the SVM classification [23].

A.2 Mp3 Encoder

The MP3 compression involved the LAME Mp3 Encoder v 3.99.5 [24]. For the encoding we used the following parameter selection:

```
./lame -b bitrate input_file.wav output_file.mp3
```

where *lame* is the encoder executable and *bitrate* is the target bit-rate in kbps of the mp3 file. The lame encoder with this setting tries to achieve a constant bit-rate with its own default algorithm: we avoid a thin control of the encoder since, in principle, we don't know the settings used during an on-the-fly encoding that could occur on a mobile device recording, that are the target of our algorithm.

A.3 AAC Encoder

The AAC compression involved the Nero AAC Encoder 1.5.1.0 [25], mainly because it implements also the High-Efficiency Advanced Audio Coding (HE-AAC) compression scheme, i.e. has the possibility to reach a bit-rate of 16 kbps.

The encoding uses the following parameter selection:

```
./neroAacEnc -br bitrate -if input_file.wav -of output_file.aac
```

where *neroAacEnc* is the encoder executable and *bitrate* is the target bit-rate in bps of the AAC file. Again, we tried to control the encoder as less as possible.

A.4 AMR Encoder

The AMR compression involved the OpenCORE Adaptive Multi Rate (AMR) Speech Codec [26], an implementation of the 3GPP TS 26.073 specification for the AMR encoding. The encoder doesn't accept every audio files, but only those compliant to the input specification, i.e. a sampling frequency of 8000 kHz and a bit depth of 16 bit. In order to ensure both of them, we resampled each input audio file by using the Shibatch Sampling Rate Converter [27] just before the encoding:

```
./ssrc --rate 8000 input_file.wav --bits 16 input_file_resampled.↩
    wav
./amrnb-enc -r bitrate input_file_resampled.wav output_file.amr
```

where *ssrc* is the resampler executable, *amrnb-enc* is the AMR encoder executable and *bitrate* is one of the 8 target bit-rate in bps allowed by the 3GPP-specifications.

Appendix B

Clustering Results

B.1 Test Content

The clustering results refer to the following test content:

*** Count (from 1 to 5) , English
1. male, 2. female

*** Count (from 1 to 5) , German
1. male, 2. female

*** Dialogue (English)
/w small pauses between the sentences
1. male, 2. female

1. In the play's climactic scene, all three men reveal their secrets, and each spy attempts to convince Möbius to come with him.
2. It is he, however, who is successful in convincing them.
1. He persuades them that the secrets he has discovered are too terrible for man to know and assures them that their efforts are in vain because he recently burned all the papers that he developed during his time in the sanatorium.
2. After much debate, the three men finally agree that they are content to protect mankind by living out the rest of their lives in captivity, while furthering and serving physics.
1. However, these noble plans are thwarted by the play's final plot twist.
2. Fräulein Doktor Mathilde von Zahnd, head of Les Cerisiers, enters the drawing room and reveals to the three men that she has eavesdropped on their entire conversation.
1. Furthermore, she has known about Möbius for years and has been secretly copying his documents and using his scientific discoveries to

construct an international empire.
2. She believes that King Solomon is speaking to her, and she believes that with his guidance and Möbius' discoveries she can become the most powerful woman on earth.
1. The story ends with a sense of impending doom.
2. Möbius, "Newton", and "Einstein" have been outmaneuvered and trapped.
1. "Those things which were thought can never be unthought."
2. The play ends with each of the three men speaking directly and pitiably to the audience, emphasizing their plight and the plight of all mankind.

*** Dialogue German)
/w small pauses between the sentences
1. male, 2. female

1. Johann Wilhelm Möbius
1. Er hat als Physiker mehrere große Entdeckungen gemacht und "das System aller möglichen Erfindungen", die einheitliche Feldtheorie als Weltformel entwickelt.
2. Da er sich der fatalen Folgen seiner Erfindungen bewusst ist und die Verantwortung dafür nicht übernehmen kann, stellt er sich wahnsinnig und lässt sich ins Irrenhaus einliefern, um die Menschheit nicht zu gefährden.
1. Er gibt vor, seine Erfindungen von Salomo offenbart zu bekommen, der sich für ihn vom ehemals weisen Psalm-Dichter des Hohenliedes zum "armen König der Wahrheit" gewandelt hat und "nackt und stinkend in seinem Zimmer kauert".
2. Der Psalm, den Möbius in einem umgedrehten Tisch hockend vorträgt, zeichnet ein düsteres Bild von den möglichen apokalyptischen Folgen wissenschaftlicher Erkenntnis.
1. Beim Abschiedsbesuch seiner Ex-Frau Lina (die nun mit dem Missionar Rose verheiratet ist) gibt Möbius vor, sie und die drei gemeinsamen Söhne nicht zu erkennen, um es ihnen dadurch zu erleichtern, ihn zu vergessen.
2. Wie sehr Möbius sich zur Rettung der Menschheit aufopfert, wird auch darin deutlich, dass er das Heiratsgesuch von Schwester Monika ablehnt, die sein Spiel durchschaut hat.
1. Obwohl er sie ebenfalls liebt, bringt er sie um, um nicht in "Freiheit" zu kommen und somit die Menschheit zu retten. 2. Da er fürchtet, von verschiedenen Mächten ausspioniert zu werden, verbrennt er seine wissenschaftlichen Manuskripte, ohne zu ahnen, dass die Anstaltsleiterin Fräulein Doktor von Zahnd bereits heimlich Kopien davon angefertigt hat.
1. Herbert Georg Beutler, genannt Newton
1.Auch er ist Physiker und gibt vor, verrückt zu sein.
2. Später stellt sich heraus, dass er zugleich Agent eines nicht näher benannten westlichen Geheimdienstes ist.

1. Um Möbius bespitzeln zu können, musste er eigens Deutsch lernen und sich als Sir Isaac Newton ausgeben.
2. Er versucht Möbius zu überreden, für die Landesverteidigung seines westlichen Staates zu arbeiten.
1. Er verspricht ihm den Nobelpreis und mahnt ihn an seine Pflicht, seine Entdeckungen der Menschheit zu übergeben.
2. Eine Verantwortung des Wissenschaftlers für seine Entdeckungen lehnt er ab, stattdessen schiebt er die Verantwortung der Allgemeinheit zu.

*** Voice (English) Male

The story is set in the drawing room of a sanatorium, an idyllic home for the mentally ill, run by famed psychiatrist Mathilde von Zahnd. This drawing room connects to three sick rooms each of which is inhabited by a single mentally ill patient. These three men, all physicists by trade, are permitted use of the drawing room, where they are periodically monitored by the female nurses that are charged with their care. The first patient is Herbert Georg Beutler, and he believes that he is Sir Isaac Newton. The second patient is Ernst Heinrich Ernesti, who believes himself to be Albert Einstein. The third patient is Johann Wilhelm Möbius, and he believes that he is regularly visited by the biblical King Solomon. When the play begins, "Einstein" has just killed one of his nurses, and the police are examining the scene. It is revealed through their discussion that this is the second slaying of a nurse by one of these three patients in just three months, the first having been committed by "Newton".

*** Voice (English) Female

The motive behind these two murders becomes clear in the play's second Act, when it is revealed with startling abruptness that none of the three patients are mad. They are all only faking insanity. Möbius is actually an incredibly brilliant physicist whose discoveries include such fabled results as a solution to the problem of gravitation, a "Unitary Theory of Elementary Particles", and the "Principle of Universal Discovery". Fearing what mankind could do with these powerful discoveries, Möbius chose not to reveal his work. He instead feigned madness, that he might be committed to a sanatorium and thus protected along with his knowledge. Möbius failed to avoid attention. "Einstein" and "Newton" are both spies, representatives of two different countries, and they have penetrated Les Cerisiers in order to secure Möbius' documents and, if possible, the man himself. Each spy murdered a nurse to protect his secrets and to strengthen his simulation of madness.

*** Voice (German) Male

Es war einmal ein junger Mann, der sehnte sich danach, daß sich sein Wunschtraum erfüllte. Obgleich dies kein ungewöhnlicher Anfang für eine Geschichte ist (denn jede Geschichte über einen jungen Mann,ob in der Vergangenheit oder der Zukunft, könnte auf ähnliche Weise beginnen), war an dem jungen Mann und seinen Erlebnissen doch viel Seltsames, das nicht einmal er selbst jemals in vollem Umfang begriff. Die Geschichte begann - wie viele andere Geschichten - in Wall. Der kleine Ort Wall liegt heute wie seit sechshundert Jahren auf einem hohen Granitfelsen mitten in einem kleinen Waldgebiet. Die Häuser von Wall sind alt und robust, aus grauem Stein, mit dunklen Schieferdächern und hohen Schornsteinen. Um auf dem Felsen jeden Zentimeter Platz zu nutzen, kuscheln sie sich eng aneinander, eins dicht an das andere gebaut, mit hier und dort einem Busch oder Baum, der aus einer Gebäudemauer wächst. Aus Wall heraus führt nur eine Straße, ein verschlungener Pfad, der vom Wald her steil ansteigt, gesäumt von Felsbrocken und Steinen. Folgt man ihm weit genug nach Süden, aus dem Wald heraus, wird aus dem Pfad eine richtige asphaltierte Straße; noch ein Stück weiter verbreitert sie sich abermals, und auf ihr drängen sich zu jeder Tages- und Nachtzeit Autos und Lastwagen, die es eilig haben, von einer Stadt zur anderen zu kommen. Irgendwann schließlich gelangt man auf der Straße nach London, aber dafür ist man von Wall aus eine ganze Nacht lang unterwegs.

*** Voice (German) Female

Die Einwohner von Wall sind ein wortkarges Völkchen, das sich grob in zwei Typen unterteilen läßt: zum einen leben hier die Ureinwohner, groß und robust wie der Granit, auf dem ihr Städtchen erbaut wurde, und zum anderen die Zugewanderten, die sich im Lauf der Jahre in Wall niedergelassen haben, samt ihren Nachfahren. Unterhalb von Wall im Westen liegt der Wald; im Süden befindet sich ein trügerisch friedlicher See, gespeist von den Bächen aus den Hügeln im Norden des Dorfes. Auf den Weiden der Hügel grasen Schafe, und im Osten erstreckt sich ebenfalls Wald. Unmittelbar östlich von Wall erhebt sich eine hohe graue Steinmauer, von der das Dorf seinen Namen hat. Diese Mauer ist sehr alt, aus grob behauenen Granitbrocken aufgeschichtet; sie kommt aus dem Wald und führt wieder in ihn zurück. In dieser Mauer gibt es nur eine einzige Lücke: eine knapp zwei Meter breite Öffnung, ein Stückchen nördlich vom Dorf. Durch den Spalt in der Mauer blickt man auf eine große grüne Wiese, hinter der Wiese liegt ein Bach, hinter dem Bach sieht man Bäume. Von Zeit zu Zeit kann man zwischen den Bäumen in der Ferne Gestalten erkennen. Riesige, seltsame Gestalten und kleine, schimmernde Erscheinungen, die aufblitzen und leuchten und dann plötzlich wieder verschwunden sind. Obwohl es hervorragendes Weideland ist, hat noch nie ein Dorfbewohner sein Vieh auf der Wiese jenseits der Mauer grasen lassen. Auch hat niemand sie je als Ackerland benutzt.

*** Music playback
1. English radio Podcast, 2. German radio Podcast , 3. Music-only

*** Silence / Noise
1. Silence in quiet office, 2. Noise (fan, cars passing by)

The test content was recorded twice: the first time by using the built-in microphones of the mobile device; the second time by using the corresponding headset microphones. Afterward, as specified in chapter 5, from the original PCM encoding each recording was lossy-compressed with all the desired MP3 bitrates, i.e.192 kbps, 128 kbps, 96kbps, 64 kbps, 32 kbps. Nor AAC or AMR compression were involved during the clustering evaluation.

B.2 Feature Vector Evolution

Before reporting the numerical results of the clustering algorithm, we are going to briefly recall the evolution of the feature vector used for the clustering.

This evolution, other then revealing the strong correlation between theoretical considerations and evaluations always present during the development phase, can also help to understand the composition of the feature vector adopted during the classification.

B.2.1 Version 1: Standard RASTA-MFCCs

B.2.1.1 Version 1.0

The GMM is trained with 12 RASTA-MFCCs, obtained using the standard approach found in literature. The training set is the whole clean subset of the NOIZEUS speech corpus [14].

We first compute 13 MFCCs and 13 RASTA-MFCCs for each frame of each recording in the training set, then we drop the first RASTA-MFCCs, obtaining the final training set for the GMM., but this choice lead us to a really noisy estimation.

The clustering algorithm is the classic K-means, and for each pair of device the whole speech testing set is given. The final result is the one with the best score within 10 repetition of k-means, each one with 100 replicas.

Results show two main issues:

1. German speech: the recordings in German are not identified correctly. In principle, this could be due to the training set used for the GMM - that contains only English sentences - hence leading to a bad estimate of the clean speech used in order to achieve the channel.

2. Loudspeaker: the recordings of the loudspeaker playback are not identified correctly. Probably, this is due to the loudspeaker's channel, which has a strong impact on the recording, and lead to an estimate too far from those from a real speaker.

B.2.1.2 Version 1.1

The GMM is trained with RASTA-MFCCs, obtained using the standard approaches found in literature. The training set is the whole clean subset of the NOIZEUS speech corpus and some of it's corrupted versions: we used the versions corrupted with bubble-noise, car-noise, street-noise and train-noise, with SNR=0dB.

The different training set was chosen in order to overcome some limitation inherent the blind channel identification algorithm: the method should work only with recordings of clean speech. This is not our case, since in our recordings there are both the original environment noise and the noise introduced by the resampling phase. Hence, the idea to train a more "robust" GMM: environment noise is affected by the microphone's frequency response, so a GMM trained also with noisy speech should be able to exploit also noisy frames in order to achieve a better channel estimate.

Results highlight the following:

1. Loudspeaker: the recordings of the loudspeaker playback are always identified together as a single cluster, hence leading to misclassification.
2. No classification gain: also dropping the recordings from the loudspeaker, there's no gain in the classification accuracy.

Starting from these results, we chose to keep the standard training of the GMM.

B.2.1.3 Version 1.2

Version 1.2 works on the same training set and channel estimates of Version 1.0, but use a different policy regarding the k-means clustering.

Most of the misclassifications in version 1.0 were due to a tiny difference between the wrong result and the correct one, e.g.:

```
--------------------------------------------------------------------------
                                                                     WRONG
IDX = 2    2    2    2    2    1    2    1    1    1
sumD = 1.635        3.0012
                                          |sumD(1)-sumD(2)|=1.3662
totSum = 4.6362
--------------------------------------------------------------------------
                                                                   CORRECT
IDX = 1    1    1    1    1    2    2    2    2    2
sumD = 1.648        3.0018
                                          |sumD(1)-sumD(2)|=1.3538
totSum = 4.6498
--------------------------------------------------------------------------
|totSum(WRONG)-totSum(CORRECT)|=0.0136
--------------------------------------------------------------------------
```

```
                                                            CORRECT
    IDX = 1    1    1    1    1    2    2    2    2    2
    sumD = 2.3046      2.4587
                                          |sumD(1)-sumD(2)|=0.1541
    totSum = 4.7633
    ----------------------------------------------------------------
                                                              WRONG
    IDX = 2    2    2    2    1    1    1    1    1    1
   sumD = 3.276       1.4643
                                          |sumD(1)-sumD(2)|=1.8117
    totSum = 4.7402
    ----------------------------------------------------------------
    |totSum(WRONG)-totSum(CORRECT)|=0.0231
    ----------------------------------------------------------------
```

In the first test case the wrong result has score *totSum* = 4.6362, while the correct one has score *totSum* = 4.6498: if we just follow the k-means algorithm we select the wrong classification, that is smaller by 0.0136.

The same happens in the second test case: the wrong result has score *totSum* = 4.7402, while the correct one has score *totSum* = 4.7633, and we select the wrong one, that is smaller by 0.0231.

In both cases even if *totSum* is decreasing, the difference between the intra-cluster distances is increasing when we select the best - but wrong - result: in the first test case $|sumD(1) - sumD(2)|$ goes from 1.3538 (correct) to 1.3662 (wrong), while in the second test case $|sumD(1) - sumD(2)|$ goes from 0.1541 (correct) to 1.8117 (wrong).

Hence, we chose to change the standard behavior of the k-means, and to consider also the difference between the intra-cluster distances as a function to minimize. Obviously, we couldn't just choose not to consider the total distance, since this would've lead to an incorrect behavior, e.g.:

```
    ----------------------------------------------------------------
                                                              WRONG
    IDX = 2    2    2    2    2    1    1    1    1    2
    sumD = 3.2628      2.9801
                                          |sumD(1)-sumD(2)|=0.2827
    totSum = 6.2429
    ----------------------------------------------------------------
                                                            CORRECT
    IDX = 2    2    2    2    2    1    1    1    1    1
    sumD = 3.7395      2.1079
                                          |sumD(1)-sumD(2)|=1.6316
    totSum = 5.8474
    ----------------------------------------------------------------
    |totSum(WRONG)-totSum(CORRECT)|=0.3955
    ----------------------------------------------------------------
                                                            CORRECT
    IDX = 2    2    2    2    2    2    1    1    1    1    1    1
    sumD = 4.8157      2.9755
                                          |sumD(1)-sumD(2)|=1.8202
    totSum = 7.7911
    ----------------------------------------------------------------
                                                              WRONG
    IDX = 2    2    2    2    2    2    1    2    1    1    1    1
    sumD = 3.6871      4.272
                                          |sumD(1)-sumD(2)|=0.5849
    totSum = 7.9591
    ──────────────────────────────────────────────────────────────────
    |totSum(WRONG)−totSum(CORRECT)|=0.1680
    ──────────────────────────────────────────────────────────────────
```

The final decision was to keep the total sum of distances as the primary function to minimize and to use the difference between cluster distances only when tiny variations occur, in order to validate the new best result.

This change of the standard behavior of the k-means clustering led to the following:

1. A general increment in terms of total number of correct classifications.

2. The loss of some previously correct classification.

The trade-off between the the increment and the loss is driven by the threshold value - set used in order to select when to apply the second object function[1].

B.2.2 Version 2: Spectral Clustering

The really poor results achieved with version 1.x led us to change the classification phase, which in version 2 is performed by a spectral clustering algorithm [20].

Unfortunately, due to the nature of our data-set, this algorithm is unfit. However, the strong reduction in terms of dimension of the data-set allowed us to speed up the prototype testing, both in terms of the GMM training phase and in terms of clustering-features tuning.

Version 2.0 highlighted the following:

1. A naive normalization of the channel estimates makes the data-set much more noisy, hence creating an abrupt decrease in terms of performances.

2. A naive biasing of the channel estimates reduces the inter-cluster distance, hence decreasing the performances.

3. RASTA-HTK-MFCCs overall performances are higher than those obtained by the use of standard RASTA-MFCCs.

4. Increasing the number of dimensions of the GMM training seems not to increase the classification performances

Observation 3. was the starting point for the third version of the prototype, as we'll see.

[1]i.e., the threshold values states which variations are “tiny” and which are not.

B.2.3 Version 3: RASTA-HTK-MFCCs

The GMM is trained with 12 RASTA-HTK MFCCs, obtained with a RASTA filtering in the mel-spectral domain of MFCCs computed using the same approach present in the Hidden Markov Model Toolkit (HTK) [15, 16]. The training set is the whole clean subset of the NOIZEUS speech corpus .

As in version 1.0 we first achieved 13 HTK-MFCCs and 13 RASTA-HTK-MFCCs for each frame of each recording in the training set, then we drop the first RASTA-HTK-MFCCs, obtaining the final training set for the GMM, and this choice lead us to a less noisy estimation.

The clustering algorithm is the classic k-means and for each pair of device the English speech testing set is given. The final result is the one with the best score within 10 repetition of k-means, each one with 100 replicas.

In version 3 we added a strong assumption on the testing phase, that is a previous knowledge of the number of recordings for each device. We also change the clustering feature-set, as we will see in the following section, in order to overcome the noise inherent the channel estimates.

These choices allowed us to achieve, for the PCM test case, an inter-device classification for the headphones subset, and promising results for the built-in microphone subset.

B.2.4 Clustering Feature-Set

Our first clustering attempts were performed on the direct channel estimates, and led us to insufficient clustering results. In order to overcome the noise present in these estimates, which is the source of many incorrect classification, we have to rely also on something else than their amplitude. In the following sections we'll report which choices has been made in order to tune the clustering feature-set.

B.2.4.1 First Derivative

Despite the noise, it's straightforward to notice that all the recordings from the same device share a common pattern between them. We chose to express this common behavior with the first derivative of the channel estimate: first derivatives are much less spread then the channel estimates and are able to clearly identify the position of peculiar spikes in the frequency response of a specific recording device. Starting from these properties, we chose to further investigate higher order derivatives.

B.2.4.2 Second Derivative

The second derivative showed to be meaningful, and to share the same properties of the first derivative: spike detection and small variance. From the third derivative on, however, the small variance property is lost: that's why there's no further derivation of the channel estimate included in the feature vector.

B.2.4.3 Magnitude Response of the Channel Estimate

Due to the spike-detection property of the first and second derivative, we were able to drop all the middle frequency and the high frequency of the magnitude response, preserving only the high-detailed representation of the low frequencies (0-781.25 Hz).

This operation has been done because mid and high frequency magnitude response of different devices often overlaps: apart from peculiar spikes, they seem not to contain any meaningful information. Hence, in order to avoid local minima and to ease the burden of having an high numbers of dimensions, we chose to drop them and we managed to achieve a greater number of correct classifications.

B.2.4.4 Channel-Dependent Gain

Since a normalization was not possible[2] we chose to apply a gain factor to each channel, depending of its power: the goal of this operation was an enhancement of inter-class differences, and it's done both directly on the magnitude response of the channel estimate - in order to affect both the first and the second order derivative - and on the complete clustering feature vector, which is a concatenation of the low-frequency magnitude response, of the first derivative and of the second derivative.

In Fig.B.1 we can see the result of these operations:

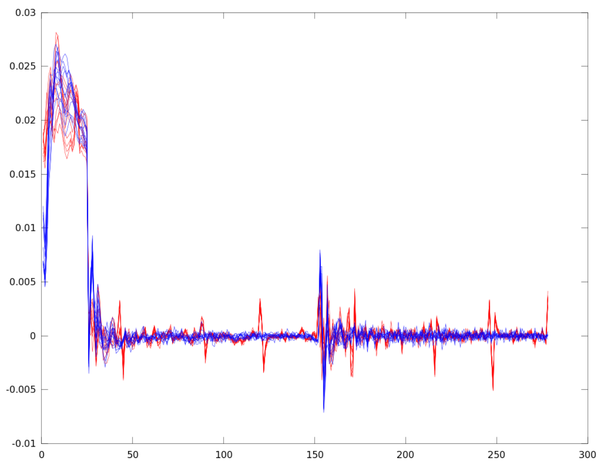


Figure B.1: Discriminating Power of the Feature Vector (1/2)

The red line denotes the feature vector of the built-in microphone of a Dell Laptop, while the blue line denotes the feature vector of the built-in microphone of an iPhone 3gs. It's straightforward to notice how the classification, in this particular test case, is mainly possible due to the presence of the derivatives.

B.2.4.5 Feature Vector Components

In the previous section, we saw how effective the derivatives are in terms of discriminating power: it's important to stress that the magnitude response can't

[2]it cripples the peculiar shape of each device and makes the channel estimates more noisy

allow by itself good results, but also that the derivatives alone aren't powerful enough to discriminate every test case:

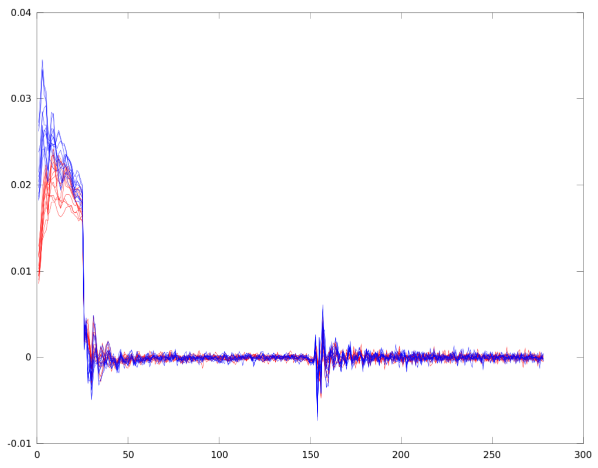


Figure B.2: Discriminating Power of the Feature Vector (2/2)

In Fig.B.2 the red line denotes the feature vector of the built-in microphone of a Samsung Galaxy-S2, while the blue line denotes the feature vector of its headset: it's straightforward to notice how the classification, in this particular test case, can't rely only on the derivatives, which are really close, but needs also the presence of the magnitude response of the channel estimate.

B.3 Complete Results

B.3.1 PCM, MP3 96 kbps

As anticipated in chapter 5, audio files encoded in PCM and in MP3 96 kbps share the same classification outcome. We will now report the direct result for PCM-encoded files:

```
——————————————————— dell_hss/
dell_hss_skype/                    CORRECT     ( 0.8962 , 0.023378 )
dell_sld/                          CORRECT     ( 0.9785 , 0.016558 )
dell_sld_skype/                    CORRECT     ( 1 , 0.022168 )
G2_A_sld/G2_A_direkt/              CORRECT     ( 0.9965 , 0.019373 )
G2_A_sld/G2_A_headset/             CORRECT     ( 0.998 , 0.023089 )
G2_B_sld/G2_B_direct/              CORRECT     ( 1 , 0.020784 )
G2_B_sld/G2_B_headset/             CORRECT     ( 0.9975 , 0.022541 )
iPhone_3gs/iPhone_3gs_direct/      CORRECT     ( 1 , 0.017747 )
iPhone_3gs/iPhone_3gs_headset/     CORRECT     ( 1 , 0.015566 )
iPhone_4s/iPhone_4s_direct/        CORRECT     ( 1 , 0.016475 )
iPhone_4s/iPhone_4s_headset/       CORRECT     ( 1 , 0.018058 )
s2_ath/S2_ath_direct/              CORRECT     ( 0.992 , 0.016846 )
s2_ath/S2_ath_headset/             CORRECT     ( 0.999 , 0.018555 )
s2_kht/S2_kht_direct/              CORRECT     ( 0.989 , 0.017422 )
```

```
s2_kht/S2_kht_headset/              CORRECT      ( 0.997 , 0.020888 )


-------------------- dell_hss_skype/
dell_sld/                           CORRECT      ( 0.9995 , 0.020722 )
dell_sld_skype/                        w         ( 0.9975 , 0.018567 )
G2_A_sld/G2_A_direkt/               CORRECT      ( 0.9265 , 0.023537 )
G2_A_sld/G2_A_headset/              CORRECT      ( 0.987 , 0.027253 )
G2_B_sld/G2_B_direct/               CORRECT      ( 1 , 0.024948 )
G2_B_sld/G2_B_headset/              CORRECT      ( 0.9775 , 0.026706 )
iPhone_3gs/iPhone_3gs_direct/       CORRECT      ( 0.9945 , 0.021912 )
iPhone_3gs/iPhone_3gs_headset/      CORRECT      ( 1 , 0.01973 )
iPhone_4s/iPhone_4s_direct/         CORRECT      ( 0.999 , 0.02064 )
iPhone_4s/iPhone_4s_headset/        CORRECT      ( 0.995 , 0.022223 )
s2_ath/S2_ath_direct/               CORRECT      ( 0.963 , 0.02101 )
s2_ath/S2_ath_headset/              CORRECT      ( 0.996 , 0.022719 )
s2_kht/S2_kht_direct/               CORRECT      ( 0.9975 , 0.021586 )
s2_kht/S2_kht_headset/              CORRECT      ( 0.998 , 0.025053 )


-------------------- dell_sld/
dell_sld_skype/                     CORRECT      ( 1 , 0.019513 )
G2_A_sld/G2_A_direkt/               CORRECT      ( 0.999 , 0.016718 )
G2_A_sld/G2_A_headset/              CORRECT      ( 0.987 , 0.020434 )
G2_B_sld/G2_B_direct/               CORRECT      ( 0.999 , 0.018129 )
G2_B_sld/G2_B_headset/              CORRECT      ( 0.8485 , 0.019886 )
iPhone_3gs/iPhone_3gs_direct/       CORRECT      ( 1 , 0.015092 )
iPhone_3gs/iPhone_3gs_headset/      CORRECT      ( 1 , 0.012911 )
iPhone_4s/iPhone_4s_direct/         CORRECT      ( 1 , 0.01382 )
iPhone_4s/iPhone_4s_headset/        CORRECT      ( 1 , 0.015403 )
s2_ath/S2_ath_direct/               CORRECT      ( 1 , 0.01419 )
s2_ath/S2_ath_headset/              CORRECT      ( 1 , 0.0159 )
s2_kht/S2_kht_direct/               CORRECT      ( 1 , 0.014767 )
s2_kht/S2_kht_headset/              CORRECT      ( 0.9475 , 0.018233 )


-------------------- dell_sld_skype/
G2_A_sld/G2_A_direkt/               CORRECT      ( 0.9945 , 0.022328 )
G2_A_sld/G2_A_headset/              CORRECT      ( 0.9995 , 0.026044 )
G2_B_sld/G2_B_direct/                  w         ( 0.69083 , 0.024447 )
G2_B_sld/G2_B_headset/              CORRECT      ( 0.9995 , 0.025496 )
iPhone_3gs/iPhone_3gs_direct/       CORRECT      ( 1 , 0.020702 )
iPhone_3gs/iPhone_3gs_headset/      CORRECT      ( 1 , 0.018521 )
iPhone_4s/iPhone_4s_direct/         CORRECT      ( 1 , 0.01943 )
iPhone_4s/iPhone_4s_headset/        CORRECT      ( 1 , 0.021013 )
s2_ath/S2_ath_direct/               CORRECT      ( 1 , 0.019801 )
s2_ath/S2_ath_headset/              CORRECT      ( 1 , 0.02151 )
s2_kht/S2_kht_direct/               CORRECT      ( 1 , 0.020377 )
s2_kht/S2_kht_headset/              CORRECT      ( 1 , 0.023843 )


-------------------- G2_A_sld/G2_A_direkt/
G2_A_sld/G2_A_headset/              CORRECT      ( 0.71501 , 0.023249 )
G2_B_sld/G2_B_direct/               CORRECT      ( 1 , 0.020944 )
G2_B_sld/G2_B_headset/              CORRECT      ( 0.9925 , 0.022701 )
iPhone_3gs/iPhone_3gs_direct/       CORRECT      ( 1 , 0.017907 )
iPhone_3gs/iPhone_3gs_headset/      CORRECT      ( 1 , 0.015725 )
iPhone_4s/iPhone_4s_direct/         CORRECT      ( 1 , 0.016635 )
iPhone_4s/iPhone_4s_headset/        CORRECT      ( 0.9925 , 0.018218 )
s2_ath/S2_ath_direct/               CORRECT      ( 1 , 0.017005 )
s2_ath/S2_ath_headset/              CORRECT      ( 0.9915 , 0.018714 )
s2_kht/S2_kht_direct/               CORRECT      ( 1 , 0.017582 )
```

```
s2_kht/S2_kht_headset/            CORRECT     ( 0.9995 , 0.021048 )


——————————————————— G2_A_sld/G2_A_headset/
G2_B_sld/G2_B_direct/             CORRECT     ( 0.9975 , 0.02466 )
G2_B_sld/G2_B_headset/               w        ( 0.7705 , 0.02109 )
iPhone_3gs/iPhone_3gs_direct/     CORRECT     ( 1 , 0.021623 )
iPhone_3gs/iPhone_3gs_headset/    CORRECT     ( 1 , 0.019442 )
iPhone_4s/iPhone_4s_direct/       CORRECT     ( 1 , 0.020351 )
iPhone_4s/iPhone_4s_headset/      CORRECT     ( 1 , 0.021934 )
s2_ath/S2_ath_direct/             CORRECT     ( 1 , 0.020721 )
s2_ath/S2_ath_headset/            CORRECT     ( 0.995 , 0.022431 )
s2_kht/S2_kht_direct/             CORRECT     ( 1 , 0.021298 )
s2_kht/S2_kht_headset/            CORRECT     ( 0.9685 , 0.024764 )


——————————————————— G2_B_sld/G2_B_direct/
G2_B_sld/G2_B_headset/            CORRECT     ( 0.995 , 0.024112 )
iPhone_3gs/iPhone_3gs_direct/     CORRECT     ( 1 , 0.019318 )
iPhone_3gs/iPhone_3gs_headset/    CORRECT     ( 1 , 0.017136 )
iPhone_4s/iPhone_4s_direct/       CORRECT     ( 1 , 0.018046 )
iPhone_4s/iPhone_4s_headset/      CORRECT     ( 1 , 0.019629 )
s2_ath/S2_ath_direct/             CORRECT     ( 1 , 0.018416 )
s2_ath/S2_ath_headset/            CORRECT     ( 1 , 0.020126 )
s2_kht/S2_kht_direct/             CORRECT     ( 1 , 0.018993 )
s2_kht/S2_kht_headset/            CORRECT     ( 1 , 0.022459 )


——————————————————— G2_B_sld/G2_B_headset/
iPhone_3gs/iPhone_3gs_direct/     CORRECT     ( 1 , 0.021075 )
iPhone_3gs/iPhone_3gs_headset/    CORRECT     ( 1 , 0.018894 )
iPhone_4s/iPhone_4s_direct/       CORRECT     ( 1 , 0.019803 )
iPhone_4s/iPhone_4s_headset/      CORRECT     ( 0.999 , 0.021386 )
s2_ath/S2_ath_direct/             CORRECT     ( 1 , 0.020174 )
s2_ath/S2_ath_headset/            CORRECT     ( 0.994 , 0.021883 )
s2_kht/S2_kht_direct/             CORRECT     ( 1 , 0.02075 )
s2_kht/S2_kht_headset/            CORRECT     ( 0.9575 , 0.024216 )


——————————————————— iPhone_3gs/iPhone_3gs_direct/
iPhone_3gs/iPhone_3gs_headset/    CORRECT     ( 1 , 0.0141 )
iPhone_4s/iPhone_4s_direct/          w        ( 0.8102 , 0.012463 )
iPhone_4s/iPhone_4s_headset/         w        ( 0.74843 , 0.013858 )
s2_ath/S2_ath_direct/             CORRECT     ( 1 , 0.01538 )
s2_ath/S2_ath_headset/            CORRECT     ( 1 , 0.017089 )
s2_kht/S2_kht_direct/             CORRECT     ( 1 , 0.015956 )
s2_kht/S2_kht_headset/            CORRECT     ( 1 , 0.019422 )


——————————————————— iPhone_3gs/iPhone_3gs_headset/
iPhone_4s/iPhone_4s_direct/       CORRECT     ( 1 , 0.012828 )
iPhone_4s/iPhone_4s_headset/      CORRECT     ( 1 , 0.014411 )
s2_ath/S2_ath_direct/             CORRECT     ( 1 , 0.013198 )
s2_ath/S2_ath_headset/            CORRECT     ( 1 , 0.014907 )
s2_kht/S2_kht_direct/             CORRECT     ( 1 , 0.013775 )
s2_kht/S2_kht_headset/            CORRECT     ( 1 , 0.017241 )


——————————————————— iPhone_4s/iPhone_4s_direct/
iPhone_4s/iPhone_4s_headset/         w        ( 0.95134 , 0.012811 )
s2_ath/S2_ath_direct/             CORRECT     ( 0.951 , 0.014108 )
s2_ath/S2_ath_headset/            CORRECT     ( 1 , 0.015817 )
```

```
s2_kht/S2_kht_direct/               CORRECT     ( 0.9985 , 0.014684 )
s2_kht/S2_kht_headset/              CORRECT     ( 1 , 0.01815 )


-------------------- iPhone_4s/iPhone_4s_headset/
s2_ath/S2_ath_direct/               CORRECT     ( 0.996 , 0.015691 )
s2_ath/S2_ath_headset/              CORRECT     ( 1 , 0.0174 )
s2_kht/S2_kht_direct/               CORRECT     ( 0.9985 , 0.016267 )
s2_kht/S2_kht_headset/              CORRECT     ( 1 , 0.019733 )


-------------------- s2_ath/S2_ath_direct/
s2_ath/S2_ath_headset/                 w        ( 1 , 0.016816 )
s2_kht/S2_kht_direct/                  w        ( 1 , 0.013504 )
s2_kht/S2_kht_headset/              CORRECT     ( 1 , 0.018521 )


-------------------- s2_ath/S2_ath_headset/
s2_kht/S2_kht_direct/                  w        ( 0.75507 , 0.016776 )
s2_kht/S2_kht_headset/              CORRECT     ( 0.901 , 0.02023 )


-------------------- s2_kht/S2_kht_direct/
s2_kht/S2_kht_headset/              CORRECT     ( 1 , 0.019097 )
```

Where the number on the left is the total score of the chosen clustering configuration

$$score_{final}(IDX_{BEST}) = \alpha \cdot \frac{votes(IDX_{BEST})}{\sum_k votes(IDX_k)} + (1-\alpha) \cdot \frac{\min_i(score(IDX_i))}{score(IDX_{BEST})}$$

and the number of the right is its k-mean score $score(IDX_{BEST})$. For further information refer back to section 4.1.4 and 4.2.4.

B.3.2 MP3 192 kbps, MP3 128 kbps, MP3 64 kbps, MP3 32 kbps

All the other encoding rates can be represented by the direct result for the MP3 192 kbps-encoded audio files:

```
-------------------- dell_hss/
dell_hss_skype/                     CORRECT     ( 0.881 , 0.02342 )
dell_sld/                           CORRECT     ( 0.9835 , 0.016917 )
dell_sld_skype/                     CORRECT     ( 1 , 0.022663 )
G2_A_sld/G2_A_direkt/               CORRECT     ( 0.9975 , 0.019553 )
G2_A_sld/G2_A_headset/              CORRECT     ( 0.998 , 0.023475 )
G2_B_sld/G2_B_direct/               CORRECT     ( 1 , 0.021423 )
G2_B_sld/G2_B_headset/              CORRECT     ( 0.997 , 0.023232 )
iPhone_3gs/iPhone_3gs_direct/       CORRECT     ( 1 , 0.017991 )
iPhone_3gs/iPhone_3gs_headset/      CORRECT     ( 1 , 0.015614 )
iPhone_4s/iPhone_4s_direct/         CORRECT     ( 1 , 0.016722 )
iPhone_4s/iPhone_4s_headset/        CORRECT     ( 1 , 0.018333 )
s2_ath/S2_ath_direct/               CORRECT     ( 0.992 , 0.01686 )
s2_ath/S2_ath_headset/              CORRECT     ( 0.9985 , 0.018648 )
s2_kht/S2_kht_direct/               CORRECT     ( 0.994 , 0.017449 )
s2_kht/S2_kht_headset/              CORRECT     ( 0.995 , 0.021368 )
```

```


—————————————————— dell_hss_skype/
dell_sld/                          CORRECT     ( 0.9995 , 0.02101 )
dell_sld_skype/                       w        ( 0.9985 , 0.019119 )
G2_A_sld/G2_A_direkt/              CORRECT     ( 0.9355 , 0.023646 )
G2_A_sld/G2_A_headset/             CORRECT     ( 0.984 , 0.027568 )
G2_B_sld/G2_B_direct/              CORRECT     ( 1 , 0.025516 )
G2_B_sld/G2_B_headset/             CORRECT     ( 0.9875 , 0.027325 )
iPhone_3gs/iPhone_3gs_direct/      CORRECT     ( 0.994 , 0.022084 )
iPhone_3gs/iPhone_3gs_headset/     CORRECT     ( 1 , 0.019707 )
iPhone_4s/iPhone_4s_direct/        CORRECT     ( 0.9995 , 0.020815 )
iPhone_4s/iPhone_4s_headset/       CORRECT     ( 0.9935 , 0.022426 )
s2_ath/S2_ath_direct/              CORRECT     ( 0.9675 , 0.020953 )
s2_ath/S2_ath_headset/             CORRECT     ( 0.9985 , 0.022742 )
s2_kht/S2_kht_direct/              CORRECT     ( 0.9965 , 0.021542 )
s2_kht/S2_kht_headset/             CORRECT     ( 1 , 0.025461 )


—————————————————— dell_sld/
dell_sld_skype/                    CORRECT     ( 1 , 0.020253 )
G2_A_sld/G2_A_direkt/              CORRECT     ( 0.9975 , 0.017143 )
G2_A_sld/G2_A_headset/             CORRECT     ( 0.959 , 0.021065 )
G2_B_sld/G2_B_direct/              CORRECT     ( 0.9975 , 0.019013 )
G2_B_sld/G2_B_headset/             CORRECT     ( 0.77 , 0.020822 )
iPhone_3gs/iPhone_3gs_direct/      CORRECT     ( 1 , 0.015581 )
iPhone_3gs/iPhone_3gs_headset/     CORRECT     ( 1 , 0.013204 )
iPhone_4s/iPhone_4s_direct/        CORRECT     ( 1 , 0.014312 )
iPhone_4s/iPhone_4s_headset/       CORRECT     ( 1 , 0.015923 )
s2_ath/S2_ath_direct/              CORRECT     ( 1 , 0.01445 )
s2_ath/S2_ath_headset/             CORRECT     ( 0.9955 , 0.016239 )
s2_kht/S2_kht_direct/              CORRECT     ( 1 , 0.015039 )
s2_kht/S2_kht_headset/             CORRECT     ( 0.928 , 0.018958 )


—————————————————— dell_sld_skype/
G2_A_sld/G2_A_direkt/              CORRECT     ( 0.9965 , 0.022888 )
G2_A_sld/G2_A_headset/             CORRECT     ( 1 , 0.02681 )
G2_B_sld/G2_B_direct/                 w        ( 0.74879 , 0.025471 )
G2_B_sld/G2_B_headset/             CORRECT     ( 0.9975 , 0.026567 )
iPhone_3gs/iPhone_3gs_direct/      CORRECT     ( 1 , 0.021327 )
iPhone_3gs/iPhone_3gs_headset/     CORRECT     ( 0.9995 , 0.018949 )
iPhone_4s/iPhone_4s_direct/        CORRECT     ( 1 , 0.020057 )
iPhone_4s/iPhone_4s_headset/       CORRECT     ( 0.997 , 0.021668 )
s2_ath/S2_ath_direct/              CORRECT     ( 1 , 0.020196 )
s2_ath/S2_ath_headset/             CORRECT     ( 1 , 0.021984 )
s2_kht/S2_kht_direct/              CORRECT     ( 1 , 0.020785 )
s2_kht/S2_kht_headset/             CORRECT     ( 1 , 0.024704 )


—————————————————— G2_A_sld/G2_A_direkt/
G2_A_sld/G2_A_headset/                w        ( 0.7105 , 0.023088 )
G2_B_sld/G2_B_direct/              CORRECT     ( 1 , 0.021649 )
G2_B_sld/G2_B_headset/             CORRECT     ( 0.9915 , 0.023457 )
iPhone_3gs/iPhone_3gs_direct/      CORRECT     ( 1 , 0.018217 )
iPhone_3gs/iPhone_3gs_headset/     CORRECT     ( 0.999 , 0.015839 )
iPhone_4s/iPhone_4s_direct/        CORRECT     ( 1 , 0.016947 )
iPhone_4s/iPhone_4s_headset/       CORRECT     ( 0.9925 , 0.018559 )
s2_ath/S2_ath_direct/              CORRECT     ( 0.82829 , 0.017086 )
s2_ath/S2_ath_headset/             CORRECT     ( 0.9925 , 0.018874 )
s2_kht/S2_kht_direct/              CORRECT     ( 0.97557 , 0.017675 )
s2_kht/S2_kht_headset/             CORRECT     ( 1 , 0.021594 )
```

```


-------------------- G2_A_sld/G2_A_headset/
G2_B_sld/G2_B_direct/              CORRECT     ( 0.992 , 0.025571 )
G2_B_sld/G2_B_headset/                w        ( 0.768 , 0.021717 )
iPhone_3gs/iPhone_3gs_direct/      CORRECT     ( 1 , 0.022139 )
iPhone_3gs/iPhone_3gs_headset/     CORRECT     ( 1 , 0.019761 )
iPhone_4s/iPhone_4s_direct/        CORRECT     ( 1 , 0.020869 )
iPhone_4s/iPhone_4s_headset/       CORRECT     ( 1 , 0.02248 )
s2_ath/S2_ath_direct/              CORRECT     ( 1 , 0.021008 )
s2_ath/S2_ath_headset/             CORRECT     ( 0.995 , 0.022796 )
s2_kht/S2_kht_direct/              CORRECT     ( 1 , 0.021597 )
s2_kht/S2_kht_headset/             CORRECT     ( 0.963 , 0.025516 )


-------------------- G2_B_sld/G2_B_direct/
G2_B_sld/G2_B_headset/             CORRECT     ( 0.9865 , 0.025328 )
iPhone_3gs/iPhone_3gs_direct/      CORRECT     ( 1 , 0.020087 )
iPhone_3gs/iPhone_3gs_headset/     CORRECT     ( 1 , 0.01771 )
iPhone_4s/iPhone_4s_direct/        CORRECT     ( 1 , 0.018818 )
iPhone_4s/iPhone_4s_headset/       CORRECT     ( 1 , 0.020429 )
s2_ath/S2_ath_direct/              CORRECT     ( 1 , 0.018956 )
s2_ath/S2_ath_headset/             CORRECT     ( 1 , 0.020745 )
s2_kht/S2_kht_direct/              CORRECT     ( 1 , 0.019545 )
s2_kht/S2_kht_headset/             CORRECT     ( 0.9995 , 0.023464 )


-------------------- G2_B_sld/G2_B_headset/
iPhone_3gs/iPhone_3gs_direct/      CORRECT     ( 1 , 0.021896 )
iPhone_3gs/iPhone_3gs_headset/     CORRECT     ( 1 , 0.019518 )
iPhone_4s/iPhone_4s_direct/        CORRECT     ( 1 , 0.020626 )
iPhone_4s/iPhone_4s_headset/       CORRECT     ( 0.9975 , 0.022237 )
s2_ath/S2_ath_direct/              CORRECT     ( 1 , 0.020765 )
s2_ath/S2_ath_headset/             CORRECT     ( 0.9925 , 0.022553 )
s2_kht/S2_kht_direct/              CORRECT     ( 1 , 0.021354 )
s2_kht/S2_kht_headset/             CORRECT     ( 0.937 , 0.025273 )


-------------------- iPhone_3gs/iPhone_3gs_direct/
iPhone_3gs/iPhone_3gs_headset/     CORRECT     ( 0.9995 , 0.014278 )
iPhone_4s/iPhone_4s_direct/           w        ( 0.81126 , 0.012683 )
iPhone_4s/iPhone_4s_headset/          w        ( 0.80498 , 0.014107 )
s2_ath/S2_ath_direct/              CORRECT     ( 1 , 0.015524 )
s2_ath/S2_ath_headset/             CORRECT     ( 1 , 0.017313 )
s2_kht/S2_kht_direct/              CORRECT     ( 1 , 0.016113 )
s2_kht/S2_kht_headset/             CORRECT     ( 1 , 0.020032 )


-------------------- iPhone_3gs/iPhone_3gs_headset/
iPhone_4s/iPhone_4s_direct/        CORRECT     ( 1 , 0.013008 )
iPhone_4s/iPhone_4s_headset/       CORRECT     ( 1 , 0.014619 )
s2_ath/S2_ath_direct/              CORRECT     ( 1 , 0.013147 )
s2_ath/S2_ath_headset/             CORRECT     ( 1 , 0.014935 )
s2_kht/S2_kht_direct/              CORRECT     ( 1 , 0.013736 )
s2_kht/S2_kht_headset/             CORRECT     ( 1 , 0.017655 )


-------------------- iPhone_4s/iPhone_4s_direct/
iPhone_4s/iPhone_4s_headset/          w        ( 0.93985 , 0.013168 )
s2_ath/S2_ath_direct/              CORRECT     ( 0.925 , 0.014255 )
s2_ath/S2_ath_headset/             CORRECT     ( 1 , 0.016043 )
s2_kht/S2_kht_direct/              CORRECT     ( 0.998 , 0.014844 )
```

```
s2_kht/S2_kht_headset/              CORRECT     ( 1 , 0.018763 )


—————————————————— iPhone_4s/iPhone_4s_headset/
s2_ath/S2_ath_direct/               CORRECT     ( 0.9915 , 0.015866 )
s2_ath/S2_ath_headset/              CORRECT     ( 1 , 0.017654 )
s2_kht/S2_kht_direct/               CORRECT     ( 0.9925 , 0.016455 )
s2_kht/S2_kht_headset/              CORRECT     ( 1 , 0.020374 )


—————————————————— s2_ath/S2_ath_direct/
s2_ath/S2_ath_headset/            *    w        ( 0.83929 , 0.0022892 )
s2_kht/S2_kht_direct/                  w        ( 1 , 0.013431 )
s2_kht/S2_kht_headset/              CORRECT     ( 1 , 0.018901 )


—————————————————— s2_ath/S2_ath_headset/
s2_kht/S2_kht_direct/                  w        ( 0.75 , 0.016606 )
s2_kht/S2_kht_headset/              CORRECT     ( 0.9025 , 0.02069 )


—————————————————— s2_kht/S2_kht_direct/
s2_kht/S2_kht_headset/              CORRECT     ( 1 , 0.01949 )
```

Appendix C

Classification Results

C.1 Test Content

The classification results refer to all the test content involved in the clustering results, plus the following one:

*** Count (from 6 to 10) , English
1. male, 2. female

*** Count (from 6 to 10) , German
1. male, 2. female

*** Dialogue (English)
/w small pauses between the sentences
1. male, 2. female

1. You're late.
2. Sorry?
1. I said, you're late.
2. Sorry if I held you up, you in a hurry?
1. Time is certainly of the essence, but no, I am not in a hurry. I was merely concerned that you would not make the plane.
2. That was kind of you.
1. Kind my ass. I've got a job for you, Shadow. Aren't you going to ask me what kind of job?
2. How do you know who I am?
1. Oh, it's the easiest thing in the world to know what people call themselves. A little thought, a little luck, a little memory. Ask me what kind of job.
2. No.
1. Why not?
2. I'm going home. I've got a job waiting for me there. I don't want

any other job.
1. You don't have a job waiting for you at home,You have nothing waiting for you there. Meanwhile, I am offering you a perfectly legal job - good money, limited security, remarkable fringe benefits. Hell, if you live that long, I could throw in a pension plan. You think maybe you'd like one of them?
2. You could have seen my name on my boarding pass. Or on the side of my bag. Whoever you are, you couldn't have known I was going to be on this plane. I didn't know I was going to be on this plane, and if my plane hadn't been diverted to St. Louis, I wouldn't have been. My guess is you're a practical joker. Maybe you're hustling something. But I think maybe we'll have a better time if we end this conversation here.

*** Dialogue German)
/w small pauses between the sentences
1. male, 2. female

1. Zwölf Jahre vergangen, die ganze Welt durchstreift und doch nirgends echte Freude, nirgends wahrer Genuss. Wenn ich meinte, es wäre Gold, so war es billiger Flitter. Der schäumende Becher der Lust hatte bittere Hefe. Und für solchen leeren Schein hab ich die ewige Seligkeit verscherzt. Wie glücklich war ich hier in diesem Städtchen, als ich ein Kind war und noch glauben und beten konnte.
2. Was ist mit Euch? Seid Ihr krank? Oder wollt Ihr Mönch werden.
1. Störe mich nicht. Ich habe nichts mehr zu schaffen mit dir.
2. Zu spät, Faust. Lass das Beten, es hilft nichts, du hast dem, zu dem du betest, längst abgeschworen. Oder hast du unseren Vertrag vergessen? Habe ich dir nicht treu gedient?
1. Pah! Alle Genüsse, die du mir verschafft hast, waren flüchtig, leer und nichtig. Betrug war alles!
2. Du Narr! Was hast du vom Teufel, von dem jedes Kind weiß, dass er der Vater aller Lügen ist, anderes erwartet? Aber du bist noch weit mehr betrogen, als du denkst. Deine Zeit ist um. Schlag Mitternacht bist du mein.
1. Was sagst du? Mir bleiben doch noch zwölf Jahre!
2. Du Narr! Wer mit der Hölle Verträge schließt, muss höllisch gut rechnen können. Habe ich dir nicht auch die Nächte hindurch gedient, und du willst nur die Tage zählen? In zwölf Jahren hast du die vierundzwanzig Jahre meines Dienstes verbraucht.
1. Noch sind wir auf Erden, da gilt euer teuflisches Höllenrecht nicht.
2. Du Narr! Willst du vom Teufel Gerechtigkeit verlangen? Unrecht, Lug und Trug - das ist des Teufels Handwerk!
1. O Himmel hilf!
2. Gott verschworen, Ewig verloren!

*** Voice (English) Male

There was a boy who was miserable at home, although they did not beat him. He did not fit well, not his family, his town, nor even his life. He had two older brothers, who were twins, older than he was, and who hurt him or ignored him, and were popular. They played football: some games one twin would score more and be the hero, and some games the other would. Their little brother did not play football. They had a name for their brother. They called him the Runt.They had called him the Runt since he was a baby, and at first their mother and father had chided them for it. The twins said, "But he is the runt of the litter. Look at him. Look at us." The boys were six when they said this. Their parents thought it was cute. A name like the Runt can be infectious, so pretty soon the only person who called him Donald was his grandmother, when she telephoned him on his birthday, and people who did not know him. Now, perhaps because names have power, he was a runt: skinny and small and nervous. He had been born with a runny nose, and it had not stopped running in a decade. At mealtimes, if the twins liked the food, they would steal his; if they did not, they would contrive to place their food on his plate and he would find himself in trouble for leaving good food uneaten.

*** Voice (English) Female

Their father never missed a football game, and would buy an ice cream afterward for the twin who had scored the most, and a consolation ice cream for the other twin, who hadn't. Their mother described herself as a newspaperwoman, although she mostly sold advertising space and subscriptions: she had gone back to work full-time once the twins were capable of taking care of themselves. The other kids in the boy's class admired the twins. They had called him Donald for several weeks in first grade, until the word trickled down that his brothers called him the Runt. His teachers rarely called him anything at all, although among themselves they could sometimes be heard to say that it was a pity the youngest Covay boy didn't have the pluck or the imagination or the life of his brothers. The Runt could not have told you when he first decided to run away, nor when his daydreams crossed the border and became plans. By the time that he admitted to himself he was leaving he had a large Tupperware container hidden beneath a plastic sheet behind the garage containing three Mars bars, two Milky Ways, a bag of nuts, a small bag of licorice, a flashlight, several comics, an unopened packet of beef jerky, and thirty-seven dollars, most of it in quarters. He did not like the taste of beef jerky, but he had read that explorers had survived for weeks on nothing else; and it was when he put the packet of beef jerky into the Tupperware box and pressed the lid down with a pop that he knew he was going to have to run away.

*** Voice (German) Male

Es war einmal ein junger Mann, der sehnte sich danach, daß sich sein Wunschtraum erfüllte. Obgleich dies kein ungewöhnlicher Anfang für eine Geschichte ist (denn jede Geschichte über einen jungen Mann,ob in der Vergangenheit oder der Zukunft, könnte auf ähnliche Weise beginnen), war an dem jungen Mann und seinen Erlebnissen doch viel Seltsames, das nicht einmal er selbst jemals in vollem Umfang begriff. Die Geschichte begann - wie viele andere Geschichten - in Wall. Der kleine Ort Wall liegt heute wie seit sechshundert Jahren auf einem hohen Granitfelsen mitten in einem kleinen Waldgebiet. Die Häuser von Wall sind alt und robust, aus grauem Stein, mit dunklen Schieferdächern und hohen Schornsteinen. Um auf dem Felsen jeden Zentimeter Platz zu nutzen, kuscheln sie sich eng aneinander, eins dicht an das andere gebaut, mit hier und dort einem Busch oder Baum, der aus einer Gebäudemauer wächst. Aus Wall heraus führt nur eine Straße, ein verschlungener Pfad, der vom Wald her steil ansteigt, gesäumt von Felsbrocken und Steinen. Folgt man ihm weit genug nach Süden, aus dem Wald heraus, wird aus dem Pfad eine richtige asphaltierte Straße; noch ein Stück weiter verbreitert sie sich abermals, und auf ihr drängen sich zu jeder Tages- und Nachtzeit Autos und Lastwagen, die es eilig haben, von einer Stadt zur anderen zu kommen. Irgendwann schließlich gelangt man auf der Straße nach London, aber dafür ist man von Wall aus eine ganze Nacht lang unterwegs.

*** Voice (German) Female

Die Einwohner von Wall sind ein wortkarges Völkchen, das sich grob in zwei Typen unterteilen läßt: zum einen leben hier die Ureinwohner, groß und robust wie der Granit, auf dem ihr Städtchen erbaut wurde, und zum anderen die Zugewanderten, die sich im Lauf der Jahre in Wall niedergelassen haben, samt ihren Nachfahren. Unterhalb von Wall im Westen liegt der Wald; im Süden befindet sich ein trügerisch friedlicher See, gespeist von den Bächen aus den Hügeln im Norden des Dorfes. Auf den Weiden der Hügel grasen Schafe, und im Osten erstreckt sich ebenfalls Wald. Unmittelbar östlich von Wall erhebt sich eine hohe graue Steinmauer, von der das Dorf seinen Namen hat. Diese Mauer ist sehr alt, aus grob behauenen Granitbrocken aufgeschichtet; sie kommt aus dem Wald und führt wieder in ihn zurück. In dieser Mauer gibt es nur eine einzige Lücke: eine knapp zwei Meter breite Öffnung, ein Stückchen nördlich vom Dorf. Durch den Spalt in der Mauer blickt man auf eine große grüne Wiese, hinter der Wiese liegt ein Bach, hinter dem Bach sieht man Bäume. Von Zeit zu Zeit kann man zwischen den Bäumen in der Ferne Gestalten erkennen. Riesige, seltsame Gestalten und kleine, schimmernde Erscheinungen, die aufblitzen und leuchten und dann plötzlich wieder verschwunden sind. Obwohl es hervorragendes Weideland ist, hat noch nie ein Dorfbewohner sein Vieh auf der Wiese jenseits der Mauer grasen lassen. Auch hat niemand sie je als Ackerland benutzt.

*** Music playback
1. English radio Podcast, 2. German radio Podcast

The test content was recorded twice: the first time by using the built-in microphones of the mobile device; the second time by using the corresponding headset microphones. Afterward, as specified in chapter 5, from the original PCM encoding each recording was lossy-compressed with all the desired bitrates and coding, i.e. MP3 192 kbps, 128 kbps, 96kbps, 64 kbps, 32 kbps, AAC 128 kbps, 96kbps, 64 kbps, 48 kbps, 32 kbps, 16 kbps, and AMR 12.20 kbps, 10.20 kbps, 7.95 kbps, 7.4 kbps, 6.7 kbps, 5.9 kbps, 5.15 kbps, 4.75 kbps.

C.2 Complete Results

We are now going to present all the result of the classification via SVM. In order to be both exhaustive and concise, as made when presenting the clustering result we will provide only the summary results. In particular, we will provide the summary results while varying the feature selection J^{select} discussed in section 4.1.5 and 4.2.5.

All the result reported were achieved with the best performing SVM tuning $(c_{RBF}, \gamma_{RBF}) = (2^8, 2^{-9})$ and with feature normalization between -1 and 1. We will split in two sections results concerning device classification and results concerning model classification.

C.2.1 Device Classification

The results for the device classification present at once the outcome of the intra-device classification and of the inter-device classification, As a remainder, in the inter-device classification we avoid test sets where two nominally identical devices are present.

C.2.1.1 Feature Selection J^{select}_{AAC}

```
---------------------------------------------------------------------------PCM

A_full              91.25
A_inter             96.816
A_built-in_full     91.875
A_built-in_inter    99.375
A_headset_full      99.375
A_headset_inter       100


---------------------------------------------------------------------------AMR

                    12.2        10.2        7.95        7.4

A_full                 80       79.688         75       76.562
A_inter             90.583      89.893      88.848      87.769
A_built-in_full       82.5        87.5         80        81.25
A_built-in_inter    95.625      95.234      95.312      95.547
A_headset_full       91.25       86.25      88.125       88.75
A_headset_inter     98.516      96.719      96.328      96.797
```

```




                     6.7         5.9         5.15        4.75

A_full               74.375      72.812      73.125      73.125
A_inter              86.426      84.485      85.518      85.547
A_built-in_full      85.625      83.125        82.5      83.125
A_built-in_inter     96.172      95.156      95.469      96.172
A_headset_full        86.25       83.75        82.5       81.25
A_headset_inter      96.172      93.359      95.547      94.453


----------------------------------------------------------------------------------MP3

                     192         128         96          64          32

A_full               90.938      91.562      92.188      90.938      85.938
A_inter              96.763      97.002      96.848      96.917      95.718
A_built-in_full       91.25        92.5        92.5      93.125        87.5
A_built-in_inter     99.453      99.375      99.688      99.609      99.844
A_headset_full       99.375       98.75        97.5      98.125        97.5
A_headset_inter         100         100      99.844         100      99.766


----------------------------------------------------------------------------------AAC

                     128         96          64          48          32          16

A_full               90.312       91.25      92.188          90      90.625      94.062
A_inter              96.497       96.76      96.855      96.914      95.938      97.009
A_built-in_full      91.875       91.25        92.5        92.5       96.25       98.75
A_built-in_inter     99.375      99.375      99.531      99.844         100      99.766
A_headset_full       99.375      99.375      99.375       98.75      98.125         100
A_headset_inter         100         100         100         100         100         100
```

C.2.1.2 Feature Selection J_{AMR}^{select}

```
----------------------------------------------------------------------------------PCM

A_full               89.062
A_inter              96.631
A_built-in_full        92.5
A_built-in_inter     99.688
A_headset_full         97.5
A_headset_inter      99.844


----------------------------------------------------------------------------------AMR

                     12.2        10.2        7.95        7.4

A_full               81.562      79.062      74.688      80.625
A_inter              90.046      89.678       87.92      89.299
A_built-in_full       83.75      85.625      78.125      81.875
A_built-in_inter     94.844      95.703      93.828      94.766
A_headset_full       93.125          90      89.375      91.875
A_headset_inter      98.438      96.484      97.031      96.797




                     6.7         5.9         5.15        4.75

A_full                78.75       73.75      74.688      73.438
A_inter              87.034      85.168      86.228      87.097
A_built-in_full      86.875       83.75        82.5      83.125
A_built-in_inter     95.312      95.469      94.531      94.922
A_headset_full        91.25      88.125       91.25          85
A_headset_inter      96.406      94.922      96.953      94.141


----------------------------------------------------------------------------------MP3

                     192         128         96          64          32

A_full               90.312          90      90.625      89.375      88.125
A_inter              96.707      96.829      96.863      96.907      95.925
A_built-in_full        92.5      93.125       93.75       93.75          90
A_built-in_inter     99.688      99.688      99.766         100         100
A_headset_full         97.5        97.5      95.625      95.625      98.125
A_headset_inter         100         100      99.688      99.844      99.453


----------------------------------------------------------------------------------AAC

                     128         96          64          48          32          16

A_full                91.25      89.062      89.062      90.312      90.938      92.188
A_inter              96.785      96.704      96.274      97.068      96.621      96.277
A_built-in_full      93.125      93.125        92.5       91.25      95.625      98.125
```

```
A_built-in_inter     99.688      99.688      99.766         100         100      99.844
A_headset_full       98.125      98.125      98.125      98.125      98.125      98.125
A_headset_inter         100         100         100         100      99.688      99.297
```

C.2.1.3 Feature Selection $J^{select}_{DEFAULT}$

```
---------------------------------------------------------------------------------PCM

A_full               94.375
A_inter              97.732
A_built-in_full      93.125
A_built-in_inter     99.844
A_headset_full       99.375
A_headset_inter         100


---------------------------------------------------------------------------------AMR

                     12.2        10.2        7.95        7.4

A_full                 77.5      76.875      72.812      76.562
A_inter              89.426      88.982      88.313      88.262
A_built-in_full        77.5      84.375          80          80
A_built-in_inter     95.859      95.859      95.703      95.078
A_headset_full        88.75      88.125      84.375        87.5
A_headset_inter      97.812      96.328      96.562      97.344




                     6.7         5.9         5.15        4.75

A_full               72.188      69.688      69.062       68.75
A_inter              86.089      83.491      84.211      84.299
A_built-in_full      80.625      80.625          80      81.875
A_built-in_inter     95.938          95      94.688      96.016
A_headset_full           85       81.25        77.5       78.75
A_headset_inter      96.484      92.812      93.203      93.281


---------------------------------------------------------------------------------MP3

                     192         128         96          64          32

A_full               94.375      92.812      92.188        92.5      85.312
A_inter              97.632      97.109      97.251      97.725      95.623
A_built-in_full       93.75       93.75      91.875       91.25      86.875
A_built-in_inter     99.844      99.844      99.844         100         100
A_headset_full       99.375      99.375       98.75       98.75        97.5
A_headset_inter         100         100         100         100      99.766


---------------------------------------------------------------------------------AAC

                     128         96          64          48          32          16

A_full               93.125      94.062      92.812        92.5       91.25      91.875
A_inter              97.361       97.49      97.458      96.917      97.063      96.619
A_built-in_full       93.75       93.75          90      91.875          95      99.375
A_built-in_inter     99.844      99.688      99.844         100         100         100
A_headset_full       99.375      99.375       98.75       98.75       98.75      99.375
A_headset_inter         100         100         100         100      99.766      99.844
```

C.2.1.4 Feature Selection J^{select}_{MERGE}

```
---------------------------------------------------------------------------------PCM

A_full                 92.5
A_inter               97.08
A_built-in_full      94.375
A_built-in_inter     99.688
A_headset_full       98.125
A_headset_inter      99.688


---------------------------------------------------------------------------------AMR

                     12.2        10.2        7.95        7.4

A_full               79.688       81.25      73.438      76.562
A_inter               90.42      90.188      88.232      88.237
A_built-in_full      83.125        87.5      80.625       81.25
A_built-in_inter     95.703      95.156      95.469      95.156
A_headset_full        91.25       86.25       86.25       86.25
A_headset_inter      98.359      96.562      96.016      97.344
```

```




                     6.7        5.9        5.15       4.75

A_full               73.438     71.562     72.188      73.75
A_inter              86.277     83.992      85.64      85.72
A_built-in_full      83.125     83.125      81.25     84.375
A_built-in_inter     95.938     94.844     95.234     96.328
A_headset_full           85      81.25         80     80.625
A_headset_inter      96.484     93.047     94.844     94.297


----------------------------------------------------------------------------MP3

                     192        128        96         64         32

A_full               93.125     92.188     91.875     92.812     88.438
A_inter              97.141     97.063     97.217     97.366     96.184
A_built-in_full      93.125     94.375       92.5      93.75     88.125
A_built-in_inter     99.766     99.766     99.844     99.844     99.844
A_headset_full        98.75     99.375     98.125     98.125      98.75
A_headset_inter         100        100        100        100        100


---------------------------------------------------------------------------AAC

                     128        96         64         48         32         16

A_full                 92.5     91.875     92.812      91.25     91.875      93.75
A_inter              96.926     96.912     97.134      97.18     96.768     96.843
A_built-in_full       93.75      93.75         95     93.125     95.625      98.75
A_built-in_inter     99.531     99.375     99.531        100        100     99.844
A_headset_full       99.375     99.375     99.375      98.75      98.75        100
A_headset_inter         100        100        100        100     99.844     99.844
```

C.2.1.5 Feature Selection J_{MP3}^{select}

```
----------------------------------------------------------------------------PCM

A_full               93.125
A_inter               97.28
A_built-in_full      93.125
A_built-in_inter     99.688
A_headset_full       98.125
A_headset_inter      99.922


---------------------------------------------------------------------------AMR

                     12.2       10.2       7.95       7.4

A_full               79.688     81.875     75.312     77.188
A_inter               90.73     90.723      88.96     87.988
A_built-in_full       83.75     88.125       82.5         80
A_built-in_inter     95.938     95.312     95.703     95.625
A_headset_full       90.625       87.5     86.875     89.375
A_headset_inter      98.359     96.719     96.797     96.406




                     6.7        5.9        5.15       4.75

A_full               77.188     71.562     74.062     74.375
A_inter              86.299     84.766     86.436     85.415
A_built-in_full       83.75       82.5     79.375     83.125
A_built-in_inter     95.938     95.391     95.469     96.016
A_headset_full       88.125       82.5     83.125     80.625
A_headset_inter      96.172     93.438     95.234     93.672


---------------------------------------------------------------------------MP3

                     192        128        96         64         32

A_full               91.562       92.5         90     92.188     87.812
A_inter              97.151     97.388     96.672     97.146     96.501
A_built-in_full      93.125     94.375     93.125     94.375      88.75
A_built-in_inter     99.688     99.688     99.688     99.844     99.844
A_headset_full        98.75      98.75       97.5     98.125     98.125
A_headset_inter         100        100        100        100     99.609


---------------------------------------------------------------------------AAC

                     128        96         64         48         32         16

A_full                91.25     92.812     92.188     90.938      91.25     92.812
A_inter              97.402     97.429     97.217     96.956     96.531     96.377
A_built-in_full        92.5     91.875     94.375     94.375     96.875      98.75
```

```
A_built-in_inter    99.688      99.609      99.531      99.922         100      99.766
A_headset_full       98.75       98.75      99.375      99.375      98.125         100
A_headset_inter        100         100         100         100         100         100
```

C.2.1.6 Feature Selection J_{PCM}^{select}

```
--------------------------------------------------------------------------------PCM

A_full              92.812
A_inter              97.09
A_built-in_full         95
A_built-in_inter    99.688
A_headset_full       98.75
A_headset_inter        100


--------------------------------------------------------------------------------AMR

                    12.2        10.2        7.95        7.4

A_full              80.625      82.188      75.312        77.5
A_inter             91.077      91.433      88.743      88.135
A_built-in_full      83.75       88.75       81.25       81.25
A_built-in_inter    95.703      95.781      95.625      95.703
A_headset_full      90.625       88.75      86.875          90
A_headset_inter     98.281      96.641      96.562      97.422




                    6.7         5.9         5.15        4.75

A_full              75.938        72.5      74.375      75.312
A_inter             86.318      84.407      86.633      85.884
A_built-in_full         85        82.5       81.25          85
A_built-in_inter     96.25      95.234      95.469      96.328
A_headset_full      89.375       83.75      83.125      83.125
A_headset_inter     96.797      93.047      95.234      95.469


--------------------------------------------------------------------------------MP3

                    192         128         96          64          32

A_full              92.188      92.812      91.875      91.875      88.438
A_inter             97.285      97.197      96.797      97.375      95.964
A_built-in_full       92.5       93.75      94.375        92.5      89.375
A_built-in_inter    99.688      99.688      99.844      99.844         100
A_headset_full       98.75      98.125        97.5       98.75      98.125
A_headset_inter        100         100         100         100      99.609


--------------------------------------------------------------------------------AAC

                    128         96          64          48          32          16

A_full              92.812      92.188      93.438      90.625      90.938      92.812
A_inter             97.495      97.041      97.139      97.124      96.216      96.475
A_built-in_full       92.5        92.5      94.375        92.5       96.25       98.75
A_built-in_inter    99.688      99.688      99.688      99.844         100      99.766
A_headset_full       98.75       98.75       98.75       98.75       98.75         100
A_headset_inter        100         100         100         100         100         100
```

C.2.2 Model Classification

In the model classification nominally identical devices are considered part of the same class.

C.2.2.1 Feature Selection J_{AAC}^{select}

```
------------------------------------------------------------------------PCM

A_full       94.062
A_built-in      100
A_headset       100


------------------------------------------------------------------------AMR

               12.2       10.2        7.95        7.4

A_full       90.312     90.625      88.125     87.812
A_built-in     97.5     98.125        97.5     98.125
A_headset      97.5       97.5       93.75     95.625




                6.7        5.9        5.15       4.75

A_full       87.812     84.688      87.188     84.062
A_built-in   98.125     96.875        97.5       97.5
A_headset    93.125     89.375      89.375     89.375


------------------------------------------------------------------------MP3

                192        128          96         64        32

A_full       95.625     96.562      94.062         95    93.125
A_built-in      100        100         100        100       100
A_headset       100        100         100        100    99.375


------------------------------------------------------------------------AAC

                128         96          64         48        32        16

A_full       94.375     94.688          95     95.625    94.688        95
A_built-in      100        100         100        100       100       100
A_headset       100        100         100        100    99.375    99.375
```

C.2.2.2 Feature Selection J_{AMR}^{select}

```
------------------------------------------------------------------------PCM

A_full       94.375
A_built-in      100
A_headset    99.375


------------------------------------------------------------------------AMR

               12.2       10.2        7.95        7.4

A_full        88.75     89.688      86.562     90.625
A_built-in   96.875     96.875        97.5      98.75
A_headset    98.125      96.25       93.75     94.375




                6.7        5.9        5.15       4.75

A_full       86.875     83.125      85.625     85.312
A_built-in     97.5       97.5      98.125      98.75
A_headset        95       92.5      90.625      91.25


------------------------------------------------------------------------MP3

                192        128          96         64        32

A_full       95.625     94.688      95.938     94.688    93.438
A_built-in      100        100         100        100       100
A_headset    99.375        100         100        100    99.375

```

```

---------------------------------------------------------------------------AAC

                 128         96          64          48          32          16

A_full            95     95.312      95.312      95.312      95.312       93.75
A_built-in       100        100         100         100         100         100
A_headset        100     99.375         100         100      99.375       98.75
```

C.2.2.3 Feature Selection $J^{select}_{DEFAULT}$

```
---------------------------------------------------------------------------PCM

A_full        96.875
A_built-in       100
A_headset        100


---------------------------------------------------------------------------AMR

                12.2       10.2        7.95         7.4

A_full        89.062     87.812        87.5      89.062
A_built-in      97.5     98.125      96.875      96.875
A_headset     98.125     94.375       93.75       96.25




                 6.7        5.9        5.15        4.75

A_full        87.812     82.812      84.062      83.438
A_built-in     96.25      96.25      95.625       98.75
A_headset     95.625     85.625          85       86.25


---------------------------------------------------------------------------MP3

                 192        128          96          64          32

A_full        96.562     95.938      95.938      95.938       93.75
A_built-in       100        100         100         100         100
A_headset        100        100         100         100      99.375


---------------------------------------------------------------------------AAC

                 128         96          64          48          32          16

A_full         96.25     96.875      95.312        97.5      94.688      93.438
A_built-in       100        100         100         100         100         100
A_headset        100        100         100         100      99.375       98.75
```

C.2.2.4 Feature Selection J^{select}_{MERGE}

```
---------------------------------------------------------------------------PCM

A_full         96.25
A_built-in       100
A_headset     99.375


---------------------------------------------------------------------------AMR

                12.2       10.2        7.95         7.4

A_full        89.375      88.75       88.75          90
A_built-in    98.125     96.875      98.125      98.125
A_headset      98.75      93.75          95      95.625




                 6.7        5.9        5.15        4.75

A_full        88.438     84.688        87.5      84.062
A_built-in      97.5     96.875        97.5      98.125
A_headset         95     86.875        87.5       88.75


---------------------------------------------------------------------------MP3

                 192        128          96          64          32

A_full        95.625     95.625      95.625      95.312      94.375
```

```
A_built-in        100         100         100         100         100
A_headset         100         100         100         100      99.375


-----------------------------------------------------------------------------AAC

                  128          96          64          48          32          16

A_full             95          95      94.688      95.312      94.688      94.688
A_built-in        100         100         100         100         100         100
A_headset         100         100         100         100         100      99.375
```

C.2.2.5 Feature Selection J_{MP3}^{select}

```
-----------------------------------------------------------------------------PCM

A_full         95.625
A_built-in        100
A_headset         100


-----------------------------------------------------------------------------AMR

                 12.2        10.2        7.95         7.4

A_full         90.938      90.625       88.75      88.125
A_built-in     98.125      96.875      98.125       98.75
A_headset      98.125       96.25      95.625      95.625




                  6.7         5.9        5.15        4.75

A_full         87.812      83.125      87.188       86.25
A_built-in       97.5      98.125        97.5      98.125
A_headset       96.25      86.875      89.375       91.25


-----------------------------------------------------------------------------MP3

                  192         128          96          64          32

A_full         95.312      95.312      95.312          95      95.312
A_built-in        100         100         100         100         100
A_headset         100         100         100         100       98.75


-----------------------------------------------------------------------------AAC

                  128          96          64          48          32          16

A_full         95.625      95.938      95.938       96.25      95.312      95.312
A_built-in        100         100         100         100         100         100
A_headset         100         100         100         100         100      99.375
```

C.2.2.6 Feature Selection J_{PCM}^{select}

```
-----------------------------------------------------------------------------PCM

A_full         95.312
A_built-in        100
A_headset         100


-----------------------------------------------------------------------------AMR

                 12.2        10.2        7.95         7.4

A_full         90.312          90      88.125       88.75
A_built-in      98.75      98.125      98.125      98.125
A_headset      98.125      95.625      95.625          95




                  6.7         5.9        5.15        4.75

A_full           87.5          85      85.312      85.938
A_built-in     98.125      98.125        97.5      98.125
A_headset      94.375      88.125      88.125      90.625


-----------------------------------------------------------------------------MP3

```

```
                 192         128          96          64          32

A_full        95.312      95.625      95.312      95.625       93.75
A_built-in       100         100         100         100         100
A_headset        100         100         100         100       98.75


------------------------------------------------------------------------AAC

                 128          96          64          48          32          16

A_full        95.625          95      95.938      96.562      94.375          95
A_built-in       100         100         100         100         100         100
A_headset        100         100         100         100         100      99.375
```

C.3 Baseline Framework

In this section we are going to present the result of the classification via SVM by using directly the channel estimate as the feature vector, instead of the one defined in Chapter 4. Once again, we will split in two sections results concerning device classification and results concerning model classification.

C.3.1 Device Classification

The results for the device classification present at once the outcome of the intra-device classification and of the inter-device classification, As a remainder, in the inter-device classification we avoid test sets where two nominally identical devices are present.

```
------------------------------------------------------------------------------PCM

A_full                 81.51
A_inter               90.212
A_built-in_full       89.062
A_built-in_inter      99.023
A_headset_full        93.229
A_headset_inter        97.07


------------------------------------------------------------------------------AMR

                      12.2        10.2        7.95        7.4

A_full                66.406      67.448      66.146      64.844
A_inter                81.23      80.042      79.732      78.642
A_built-in_full       77.604      77.604      73.438      76.042
A_built-in_inter      92.578      92.969      92.448      91.471
A_headset_full        78.125      76.042      71.875      76.042
A_headset_inter       92.578      90.625      91.667      90.885




                      6.7         5.9         5.15        4.75

A_full                63.281      63.281      65.885      63.021
A_inter               78.658      77.236      78.158      77.842
A_built-in_full       72.396      71.354      72.396      72.396
A_built-in_inter      91.732      91.341      89.909      89.909
A_headset_full        72.396      76.042      73.438          75
A_headset_inter       89.714      89.974      89.714      89.779


------------------------------------------------------------------------------MP3

                      192         128         96          64          32

A_full                 81.51      81.771      82.552       81.25      79.167
A_inter               90.019      89.777      90.263      89.807      88.552
A_built-in_full       88.021      88.021        87.5      88.542      82.812
A_built-in_inter      98.828      98.958      98.698      98.828      98.828
A_headset_full         93.75      94.271      94.271      94.792      93.229
A_headset_inter       97.005      96.549      96.615      96.289      97.461


------------------------------------------------------------------------------AAC

                      128         96          64          48          32          16

A_full                81.771      82.292      82.292      82.031      84.375      85.156
```

```
A_inter               90.47     90.312    90.059    89.832    91.258    91.801
A_built-in_full       88.021    89.583    89.062    89.062    92.708    95.312
A_built-in_inter      98.958    98.893    98.828    99.219    99.089    98.958
A_headset_full        93.75     93.75     94.792    94.792    93.75     93.75
A_headset_inter       97.07     96.81     96.745    96.81     97.591    97.07
```

C.3.2 Model Classification

In the model classification nominally identical devices are considered part of the same class.

```
--------------------------------------------------------------------------------PCM

A_full        90.365
A_built-in       100
A_headset     97.917


--------------------------------------------------------------------------------AMR

              12.2        10.2        7.95        7.4

A_full        83.333      87.5        82.292      83.594
A_built-in    98.958      98.958      98.438      97.396
A_headset     89.583      91.146      88.542      89.062




              6.7         5.9         5.15        4.75

A_full        82.031      80.208      79.427      79.167
A_built-in    97.396      97.396      98.438      96.875
A_headset     88.542      84.896      85.417        87.5


--------------------------------------------------------------------------------MP3

              192         128         96          64          32

A_full        91.406      90.104      90.885      90.365      88.802
A_built-in       100         100         100         100         100
A_headset     99.479      97.396      97.396      97.396      96.354


--------------------------------------------------------------------------------AAC

              128         96          64          48          32          16

A_full        91.667      90.625      90.365      89.062      90.104      89.583
A_built-in       100         100         100      99.479         100         100
A_headset     97.917      97.396      96.875      97.396      98.438      96.875
```