Classification of Nonverbal Human-Produced Audio Events

by

Philippe CHABOT

THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT OF A MASTER'S DEGREE
WITH THESIS IN ELECTRICAL ENGINEERING
M.A.Sc.

MONTREAL, OCTOBER 8TH 2020

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

ACKNOWLEDGEMENTS

First, I would like to thank my director Jérémie Voix for his support and guidance during my master's degree. I also want to thank my co-director Patrick Cardinal for sharing his expertise on machine learning by always giving me new ideas and for being available for the biweekly meetings throughout my master's degree. I would like to thank Rachel Bouserhal for supervising me and helping me for this project. I would also like to thank her for making it possible for me to go present a paper I co-wrote at the Interspeech 2018 conference in India.

I would like to thank everyone at the CRITIAS laboratory : Hami for helping me make my detector work on the ARP, Cécile for helping me with Latex. Thanks to Valentin, Thibaut, Jacob, Michel, Sébastien, Gabrielle, Guilhem, Marcos and Vincent for making the two years I spent on this project more enjoyable. Thanks to everyone at the lab who participated to my test. I would like to thank the people at EERS Global Technologies for giving me a place to relax during those coffee breaks.

Thanks to my family and friends for being there for me and encouraging me. And finally, I would like to thank my girlfriend Lisa for supporting me throughout my masters degree.

Classification d’événements audio non verbaux produit par l’humain.

Philippe CHABOT

RÉSUMÉ

La perte d’audition due à une exposition excessive au bruit sur le lieu de travail affecte un nombre croissant de travailleurs et peut être évitée en utilisant correctement des protecteurs auditifs. Les dispositifs de protection se retrouvent sous la forme intra-auriculaire (bouchons) ou circumaural (casque). Lorsque les dispositifs intra-auriculaires sont correctement insérés, ils créent un effet d’occlusion qui amplifie les bruits physiologiques. Ces bruits physiologiques amplifiés peuvent être captés à l’aide d’un microphone placé à l’intérieur du conduit auditif occlus. À l’aide d’un algorithme de détection, ces bruits physiologiques pourraient être utilisés pour de nombreuses applications. Par exemple, l’utilisateur pourrait interagir avec un périphérique portable de manière discrète en claquant des dents ou de la langue; les bruits non verbaux captés pourraient servir à surveiller la santé de l’utilisateur en détectant les événements audio liés à la toux ou aux raclements de la gorge; ou également la dose de bruit reçue des travailleurs pourrait être mieux précisée en supprimant les bruits émis par le porteur lui-même et donc inoffensifs pour l’audition.

Ce projet comportait 3 volets : 1) développer un algorithme de classification d’événements audio non verbaux produits par l’humain, 2) développer un algorithme de détection de signaux non verbaux et 3) valider la performance et la capacité de ces algorithmes à fonctionner en temps réel sur un dispositif portable à faible puissance de calcul.

Dix événements ont été sélectionnés au sein d’une base de données existante constituée d’événements non verbaux enregistrés avec un microphone intra-auriculaire. Au total, 3037 échantillons d’une durée de 400 ms chacun ont été extraits de la base de données afin de développer l’algorithme de classification. Lors du développement de ce dernier, les algorithmes d’apprentissage automatique suivant “Support Vector Machine”, le “Gaussian Mixture Model”, le “Multi-Layer Perceptron”, le “Convolutional Neural Network” et le “Bag-of-Audio-Words” (BoAW) ont été mis en œuvre et évalués. Lors du développement de l’algorithme de détection, trois techniques d’analyse du signal audio trouvées dans la littérature ont également été testés : la technique des “Mel-Frequency Cepstral Coefficients” (MFCC), principalement utilisée pour la reconnaissance vocale, la technique des “Auditory-inspired Amplitude Modulation Features” généralement utilisée pour la vérification de la parole et la “Per-Channel Energy Normalization” (PCEN) souvent utilisée pour la détection de mots-clés dans un champ éloigné.

Les performances optimales ont été trouvées en utilisant l’algorithme d’apprentissage automatique BoAW et les techniques d’analyse MFCC et PCEN avec une sensibilité de 81,5 % et une précision de 83 %. L’algorithme de détection en temps réel a été testé dans un environnement bruyant sur 10 nouveaux sujets. Il a montré une sensibilité de 69,9 % et une précision de 78,9 % dans un environnement calme. Cela en fait un algorithme prometteur à implémenter dans de nouvelles lignes de protecteurs auditifs numériques intelligents capable d’assurer une meilleure dosimétrie sonore des travailleurs tout en offrant de nouvelles fonctionnalités de surveillance de la santé et d’interfaces homme-machine.

Mots-clés: Santé et sécurité au travail, protection auditive, acoustique, hearables, analyse du signal, apprentissage machine, détection et classification d’événements audio

Classification of Nonverbal Human-Produced Audio Events

Philippe CHABOT

ABSTRACT

Noise Induced Hearing Loss due to excessive noise exposure in the workplace is affecting an increasing number of workers and can be reduced by properly using Hearing Protection Devices (HPD). These protection devices are often found in the form of intra-aural (plugs) or circumaural (earmuffs) devices. When intra-aural devices are properly inserted, they create an occlusion effect that amplifies physiological noise and makes it easier to be perceived by the wearer. These amplified physiological noises can be captured using a microphone placed inside the occluded ear canal and with a detection algorithm, these nonverbal audio events could be used for many applications, such as: the user could interact with an audio wearable device in a discreet manner by clicking his teeth or tongue; the user's health could be monitored by detecting coughing or clearing of the throat events and the worker's noise dose exposure calculation could be more accurately measured by the removal of the wearers' own noises, which are harmless to his hearing.

The objective of this project are threefold: 1) build a classification algorithm of nonverbal human-produced audio events 2) build a nonverbal audio event detection algorithm and 3) validate the performances and ensure that these algorithms can run in real-time on a low computational power device.

Ten nonverbal audio events were selected from an existing database featuring nonverbal events recorded using an in-ear microphone. In total, 3037 samples, each 400 ms in length were extracted from the database in order to train the classification algorithm. To create the audio event classifier, several state-of-the-art machine learning algorithms found in the literature such as the Support Vector Machine, Gaussian Mixture Model, Multilayer Perceptron, Convolutional Neural Network and Bag-of-Audio-Words (BoAW) were successively implemented and tested. To provide input for these algorithms, three types of features found in the literature were tested: the Mel-Frequency Ceptral Coeffients (MFCC), mainly used for speech recognition, the Auditory-inspired Amplitude Modulation Features typically used for speaker verification on whispered speech and Per-Channel Energy Normalization (PCEN) features often used for far-field keyword spotting.

Optimal performance of the classification algorithm was found using the BOAW classifier coupled with the MFCC and PCEN features with a sensitivity of 81.5% and a precision of 83%. The real-time detector was tested in a noisy environment on 10 new test subjects. It showed a sensitivity of 69.9% and a precision of 78.9% in a quiet environment. This makes it a promising algorithm to be implemented in new lines of smart digital HPD capable of protecting worker's hearing while opening the way for a new range of health monitoring and human-machine interfaces.

Keywords: Occupational health and safety, Hearing protection, Acoustics, Hearables, Signal processing, Machine learning, Audio event classification and detection

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF ABREVIATIONS

AEC	Audio Event Classification
AED	Audio Event Detection
WID	Wearer Induced Disturbances
HMS	Health Monitoring Systems
NIHL	Noise Induced Hearing Loss
HPD	Hearing Protection Devices
IEM	In-Ear Microphone
STFT	Short Time Fourier Transform
MFCC	Mel Frequency Cepstral Coefficients
ARP	Auditory Research Platform
SVM	Support Vector Machine
GMM	Gaussian Mixture Model
BoAW	Bag-of-Audio-Words
MLP	Multi-Layer Perceptron
CNN	Convolutional Neural Network
HMM	Hidden Markov Model
AAMF	Auditory-inspired Amplitude Modulation Features
PCEN	Per Channel Energy Normalisation
PCA	Principal Component Analysis
LDA	Linear Discriminant Analysis

LIST OF SYMBOLS AND UNITS OF MEASUREMENTS

ms	milliseconds
kHz	kilohertz

INTRODUCTION

Context

Every year, millions of workers are affected by Noise Induced Hearing Loss (NIHL) due to excessive noise exposure in the workplace. One way to reduce the number of people affected by NIHL is to have workers wear hearing protection devices (HPD). These HPDs can be circumaural (earmuffs) or intra-aural (foam earplugs, pre-molded earplugs, custom molded earplugs, etc.). When the user correctly inserts an earplug inside the ear canal, this ensues in an occlusion effect, that is, the seal created at the entrance of the ear canal by the earplug boosts low frequency signals and amplifies physiological noises making them more easily perceptible by the wearer. Sounds such as the heartbeat, respiration, movement of the saliva and even forceful blinking of the eyes become amplified and easier to hear and detect when the occlusion effect is present. While usually a nuisance for the wearer, the occlusion effect could make it possible to record the audio signals inside the occluded earcanal. The audio events recorded could be detected, classified and used by an external processing device, which happens to be the aim of this project.

Applications

Several applications of nonverbal audio event detection could be envisioned, and three are of particular interest for the research conducted within the NSERC-EERS Industrial Research Chair in In-Ear Technologies (CRITIAS) for industrial workplaces. The first application is to monitor the health status of a worker by monitoring health-related physiological noises such as coughing or clearing of the throat: these events could be automatically reported to a remote monitoring system, making it possible for an occupational health and safety manager to assess whether a worker is in a dangerous work situation or has a medical condition. A second application deals with the creation of a handsfree human-computer interface that relies solely on the sounds emitted by the wearer, namely: voluntary noises such as the clicking of the teeth, or the tongue or by forceful blinking; users could use these events as triggers to interact with their digital hearing protection device. For example, they could change the level of attenuation of their HPD or trigger a voice prompt and interact with a remote system in a handsfree manner. The third application deals with the monitoring of the noise exposure of workers: the noises produced by the wearers' themselves, like a loud sneeze, can create a large peak value in the sound pressure level measured inside the occluded earcanal even though it is not dangerous for one's hearing, as the human ear has several mechanisms to protect itself against these loud events. Having the ability to detect such noises produced by the wearers themselves and to remove such artifacts, later referred to as "Wearer-Induced Disturbance (WID)", when calculating the noise dose received by the workers would help improve the accuracy of in-ear noise dosimeters.

This nonverbal audio event detection algorithm could also be implemented in hearables to be used by the general public and not necessarily be used only for industrial workers. Hearables are in-ear wearables that contains sensors such as microphones to capture and monitor signals of interest. The consumer market has recently seen the arrival of new audio devices (containing in-ear microphones), such as the Galaxy Buds (Samsung, Seoul, South Korea) and the AirPods Pro (Apple, Cupertino, USA), which use an in-ear microphone to transmit the voice when they detect the user is in a noisy environment. Using the in-ear microphone, a device could detect nonverbal cues from the user to interact with the phone. A click of the tongue could pause or resume a song while clacking of the teeth could select the next track. This would prevent the user from having to manually press a button, or talk out loud to their phone to perform small tasks creating a handsfree silent interface. The device could also deduce the emotional state of the wearer via involuntary audio cues such as grinding of the teeth which is linked to stress and anxiety (Sutin *et al.*, 2010).

Objectives

The main objective of this project is to detect and classify human-produced nonverbal events. To achieve this, three sub-goals have to be achieved:

1. Build a classification algorithm of nonverbal human-produced audio events;
2. Build a nonverbal audio event detection algorithm;
3. Validate the performances and ensure that these algorithms can run in real-time on a low computational power device.

The methods used are described in detail in Chapter 2.

Research Outcomes

From a scientific perspective, the main merit of this research is to explore a new range of audio events: nonverbal human sounds captured inside the ear. Much research has been conducted to classify audio events, yet none have used audio captured from inside an occluded ear. A conference article was published on the preliminary work on the classification of nonverbal events in the proceedings of the "InterSpeech 2018" conference and a journal article on the classification and detection system developed has been submitted and accepted for publication in the scientific journal "Applied Acoustics".
From an occupational health and safety perspective, the proposed algorithm makes it possible to remove WIDs from the measurement of the noise dose. This allows for a better estimation of the noise levels the workers are subjected to and prevents false calculations accounting for loud events that are not necessarily dangerous for the human ear (such as a loud sneeze that actually triggers a stapedial reflex in the middle-ear muscles that greatly attenuates the sound

in the ear of the human producing it). The detection of other health related audio events is also considered a contribution to the health community.

From an industrial perspective, a functioning prototype of a new type of "smart" HPD has been delivered to EERS Global Technologies Inc., the industrial partner of the NSERC-EERS industrial chair in in-ear technologies. This prototype features a real-time detection and classification algorithm and was built on the prototyping platform dubbed, the Auditory Research Platform (ARP), featuring the limited computational power of the envisioned commercial product.

Structure of the Master's thesis

The structure of this Master's thesis is as follows: The literature review, in Chapter 1 contains the review on the scientific literature and state of the art of multiple fields related to this project: hearables, extractions of features from audio signals and the machine learning classifiers used for audio signals. Chapter 2: “Methods”, describes the methods used to find the best parameters and algorithms used for the classifier. Chapter 3: “Results”, contains the results of the different techniques used to improve the classifier. Chapter 4 describes the final real-time audio detection system developed for this project, how it was tested and its performance. Finally, the Conclusion chapter presents the conclusions of this work and proposes recommendations for future research.

CHAPTER 1

LITERATURE REVIEW

This research project involves 3 main research fields as seen in Figure 1.1. The acoustics field represents the audio signal to classify, how that signal is transmitted to the ear canal and how the transmission medium affects the signal. The signal processing field helps explain and calculate the characteristics of the signal and is used to extract the relevant properties to properly classify the signal. Finally, machine learning is used to do the classification of the audio signals by training an algorithm to distinguish different audio events.

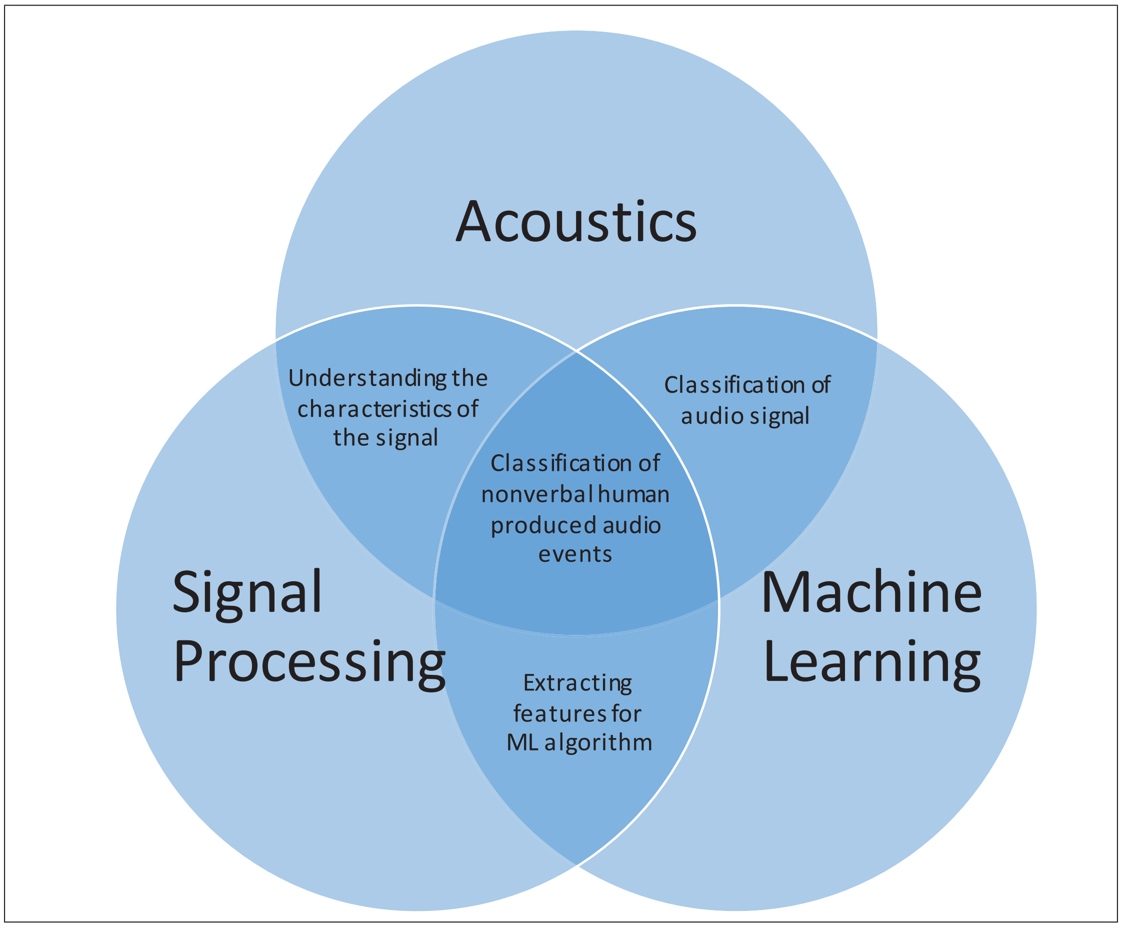

Acoustics
Understanding the characteristics of the signal
Classification of audio signal
Classification of nonverbal human produced audio events
Signal Processing
Machine Learning
Extracting features for ML algorithm

Figure 1.1 Venn diagram of the three main research fields of this project

The literature review is divided into three sections. Section 1.1 reviews the state of the art on hearables. Section 1.2 presents available feature extraction techniques used to analyze audio signals. Finally, the machine learning algorithms used in the current literature for audio event classification (AEC) and audio event detection (AED) are presented in section 1.3.

1.1 Hearables

Wearables are becoming more popular in the consumer market. Devices such as smart watches or smart glasses are being released by many big technology companies. One particular kind of wearable that is becoming more popular is the intra-aural wearable and is dubbed Hearable or Earable. These devices can be placed in and around the user's ear and contain various sensors, including microphones, to capture and track a multitude of signals of interest. Hearables could have health monitoring capabilities, as well as fitness tracking and communication. For this reason, they are becoming more prevalent in the present market. The market for hearable devices is growing rapidly and expected to reach up 85 billion USD by 2021 (MarketResearch.biz, 2018). In the report by Nick Hunn (2016), he separates the hearable market into seven distinct fields. The first is Audio Playback, which is simply the playback of audio by earbuds but hearables are capable of much more than simply audio playback. The hearable can also become a link between the user and his device. Internet of Voice or Human-Computer Interface involves voice recognition technology, smart home control and automatic translation between languages. Another potential capability of hearables is Audio Curation and Augmented Hearing which involves the manipulation of what is heard by the wearer by isolating certain sounds, through noise cancellation or by manipulating the frequency response of what is heard. Hearables can also act as a Health Monitoring System and can monitor biological signals such as heart rate and breathing rate. They can also act as Active Protection or Advanced Hearing Protection and protect the hearing of industrial workers or users in a noisy environment. They could measure the noise dose that the user is submitted to or allow transparent hearing when in a quiet environment. Finally, hearables could also be used as Hearing Aids and could offer the same functionality as current hearing aids for a fraction of the price.

When properly placed, a hearable can create an acoustical seal with the ear canal and induce the occlusion effect. The occlusion effect amplifies the sound transmitted to the ear canal by bone and tissue conduction in a frequency range up to approximately 2 kHz (Brummund *et al.*, 2014). Those amplified sounds mostly come from inside the user's body such as heartbeat, saliva noises or clicking of the teeth. Having a hearable device that detects those nonverbal sounds could be useful for many applications.

Nonverbal audio event detection could improve three of the fields previously listed: Advanced Hearing Protection, Human-Computer Interface and Health Monitoring Systems. Advanced Hearing Protection, presented in section 1.1.1, can be improved by allowing for accurate in-ear dosimetry through the detection of WIDs, such as nonverbal events, that could be removed from the user's noise dose. Human-Computer Interface, discussed in detail in section 1.1.2, can be achieved in several ways including the use of one's voice. However, voluntary nonverbal events such as clicking of the teeth could be used by people with speech pathologies or for more subtle ways of interacting with a device where barely audible signals could be used. Lastly, Health Monitoring Systems, discussed in section 1.1.3, use nonverbal events such as coughing and clearing of the throat to monitor and assess the user's health.

1.1.1 Advanced Hearing Protection

When workers are in noisy environments, hearing protection devices are often necessary to prevent noise induced hearing loss. According to OSHA and Statistics Canada, the number of workers being subjected to excessive workplace noise in the United States and Canada are 22.4 and 11 million respectively (Tak *et al.*, 2009; Ramage-Morin & Gosselin, 2018). HPDs could either be passive (without any electronics) or active (with electronics) (Berger, 2003). In the past few years, active and advanced hearing protection devices have infiltrated the hearing protection market due to their advanced capabilities. For example, the BlastPLG (Etymotic, Elk Grove Village, USA) is an active hearing protector that reduces the effect of impulse noise such as gunshots using analog technology due to its faster response rate compared to digital. Other devices such as the Quietpro (Honeywell, Charlotte, USA) and SonX (EERS Global

Technologies, Montréal, Canada) use digital signal processing to enhance user experience, namely, communication. For communication in noise, many advanced HPDs use a directional boom microphone to pick up speech in front of the mouth. Directional boom microphones could capture speech with a relatively high SNR, if they are properly placed directly in front of the mouth. However, slight discrepancies in the placement of the microphone significantly degrades the quality of the speech capture. In addition, boom microphones are not compatible with other personal protection equipment.

To address these issues, other advanced HPDs use an in-ear microphone to capture the wearer's voice and transmit it to another worker. Since the audio is captured inside the protected ear, the SNR is relatively high. However, due to the limited bandwidth of bone and tissue conduction some signal processing for speech enhancement is needed to increase the quality and intelligibility of the in-ear microphone speech signal (Bouserhal *et al.*, 2017, 2015).

One of the current limitations with the use of HPDs is that the individual's noise dose is still unknown. Shoulder dosimeters do not factor in the attenuation of the earplug and are, thus, an inaccurate representation of the noise dose. Recently, an effort has been made towards in-ear dosimetry integrated to intra-aural advanced HPDs (Mazur & Voix, 2013). However, for the individual noise dose to be accurate, in-ear dosimetry must be able to identify and reject WIDs from the calculations of the noise dose (Nadon *et al.*, 2020). The event detection system described in this thesis, could detect and reject such WIDs from the individual noise dose and aid in making in-ear dosimetry more accurate.

1.1.2 Human-Computer Interface

Interfaces between computers and humans have evolved over the years, from the keyboard and mouse to the touchscreen to finally being able to give voice commands directly to a machine without having to touch anything. Assistants such as Alexa (Amazon, Seattle, USA), Siri (Apple, Cupertino, USA) and Google Assistant (Google, Mountain View, USA) are becoming increasingly powerful and are able to understand and execute various voice commands. With hearables however, since it is not always convenient to speak up in public spaces where privacy is desirable, most commercial earbuds use either text command through the phone, or taps and buttons to the side of the device in order to communicate.

The Bragi Dash (Bragi, Munich, Germany) contains the Kinetic User Interface that transforms your body in movement into an interface using its 27 different sensors. The Kinetic Interface can understand head movements such as nodding up and down or sideways in order to interface with the hearable. It also contains a virtual 4D menu that can give the option to the user to move its head in specific directions in order to play or pause a track. Instead of tapping on the device itself to interface with it, the user can simply double tap the cheek which can be useful when wearing a hat.

In the literature, hands-free and silent interfaces integrated into hearables are being developed and researched. Taniguchi *et al.* (2018), for example, detect the movement of the tongue using an infrared sensor inside the wearer's ear. This allows the user to interact with the computer by pushing their tongue against their palate.

Another form of silent interface could be through the use of nonverbal human produced audio events such as clicking of the teeth and clicking of the tongue. This is advantageous because it does not require extra sensors on the device, utilizing only the existing in-ear microphone, and limits exaggerated and obvious head movements by the user.

1.1.3 Health Monitoring Systems

Health monitoring systems in hearables are becoming more popular on the market (Nick Hunn, 2016). Table 1.1 presents a list of hearables that are currently available to purchase as well as their features.

While heartbeat is the most common biosignal measured by hearables, some of them are capable of measuring many more signals. The Bragi Dash, with its 27 sensors, can measure heart rate, breathing rate, steps, cadence while cycling and laps while swimming. Most of them use

Table 1.1 List of commercially available hearables with health monitoring features

Name	**Measurements**
Jabra Sport Pulse (Jabra, Copenhagen, Denmark)	Heart Rate
Bose SoundSport Pulse (Bose, Framingham, USA)	Heart Rate
SMS Audio Biosport (SMS Audio, Delray Beach, USA)	Heart Rate
Cosinuss One (Cosinuss, Munich, Germany)	Heart Rate, Body Temperature
iRiver On (iRiver, Seoul, South Korea)	Heart Rate, Distance, V02 Max
Jabra Elite Sport (Jabra, Copenhagen, Denmark)	Heart Rate, Rep Count, V02 Max
Kuaiwear Kuai (KuaiFit Guangdong, China)	Heart Rate, Distance, V02 Max, Speed, Cadence
Bragi Dash (Bragi, Munich, Germany)	Heart Rate, Breathing Rate, Pace, Steps, Cadence, Laps, Oxygen Saturation

sensors that are in contact with the skin and none of them use in-ear microphones as sensors. Physical sensors need a good physical contact with the user and add complexity to the circuit of the hearable.

In-ear microphones could replace some of the aforementioned sensors and reduce the complexity of the integrated circuits. Martin & Voix (2017) have already shown that heartbeat and breathing rate can be captured and detected with an intra-aural device containing an in-ear microphone. Other non-verbal cues that could give some information about the user's physical health such as coughing or clearing of the throat as eye-blinks or excessive saliva noise can be captured with an in-ear microphone. Excessive grinding of the teeth could indicate that someone is stressed or anxious as seen in Sutin *et al.* (2010), while swallowing saliva at a higher rate could indicate a user's level of emotional arousal as seen in Cuevas *et al.* (1995). This expands the potential of health monitoring to include emotional health as well as physical.

In summary, nonverbal audio event detection could be useful in hearable technologies for advanced hearing protection, human-computer interface and health monitoring systems. The next section describes the various machine learning tools currently in the literature that could be used to characterize and classify these signals.

1.2 Feature Extraction

Before inputting the audio in the machine learning system, signal processing is used to reduce the size of the input and to extract valuable information. By knowing the characteristics of the signals of interest, more relevant information with a lower dimensionality can be extracted. For the audio signals of this project, it is known that, because of bone and tissue conduction and the occlusion effect, the physiological noises coming from the user's body will be amplified between 125 Hz and 2 kHz. Therefore, the feature extraction algorithm can have its parameters changed to ignore all the information over 2 kHz to extract a smaller feature vector containing information that represents the signal in a more efficient way.

In this section, three different feature extraction techniques are presented as well as a summary of 2D features used for Convolutional Neural Networks (CNN) and dimensionality reduction techniques to further reduce the size of the input vector.

1.2.1 Mel Frequency Cepstral Coefficients

The Mel Frequency Cepstral Coefficients (MFCC) are one of the most popular features to use as input for classifying sounds and have been used by Rabaoui *et al.* (2008); Portelo *et al.* (2009); Phan *et al.* (2016a) to classify both verbal and nonverbal events. These features are mainly used for voice recognition because the MFCC takes into account how the ear perceives different frequencies.

The process used to calculate the coefficients is presented in Fig. 1.2. First, a pre-emphasis filter that amplifies the high frequencies is applied to the signal in order to avoid numerical errors during the FFT calculation to equalize energy level across the different frequency bands and to

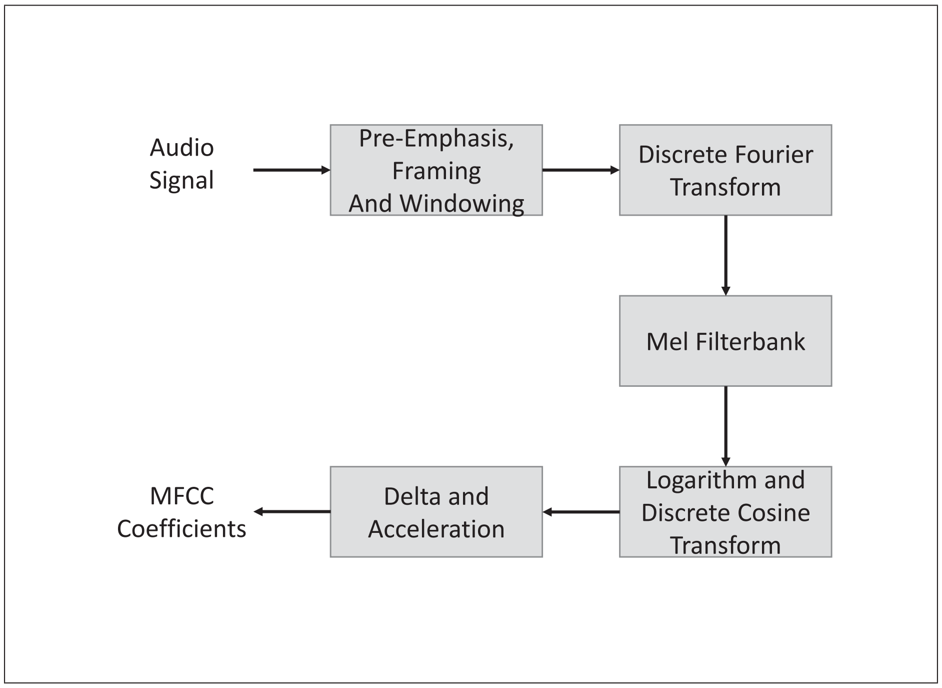


Figure 1.2 How to extract MFCC

increase the SNR. Then, the audio is cut into frames, and a hamming function is applied to it in order to reduce spectral leakage. The spectrogram is calculated using the Discrete Fourier Transform. The triangular MFCC filterbank seen in figure 1.3 is then applied to the spectrogram, resulting in 40 coefficients. The filters all have a response of 1 at the central frequency and they decrease linearly towards 0 until they reach the center frequencies of neighboring filters. The filters have better resolution in the lower frequencies compared to the high frequency to mimic how humans perceive the sounds. The logarithm of those coefficients is then applied followed by the Discrete Cosine Transform (DCT) on the coefficients to decorrelate the information and to get a compressed representation of the coefficients. The first 13 coefficients are then chosen to be the MFCCs, the first coeffiecient representing the energy of the windowed signal.

The delta and acceleration are often calculated and added to the feature vector in order to add dynamic information on the variation of the cepstral coefficients. The delta is calculated using the following formula:

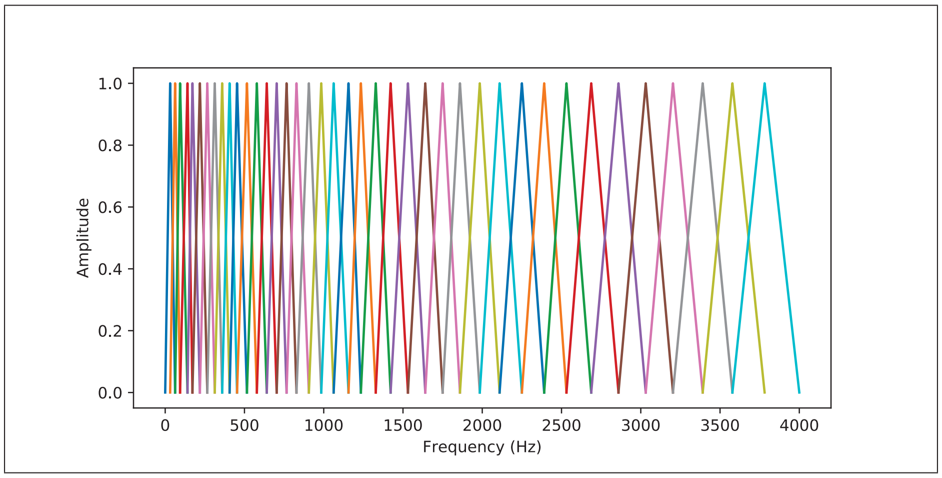


Figure 1.3 MFCC Triangular Filterbank

$$d_t = \frac{\sum_{x=1}^{X} (c_{t+x} - c_{t-x})}{2\sum_{x=1}^{X} x^2} \tag{1.1}$$

where c represents the MFCC coefficient at time t. In the majority of the cases X is equal to 2 as used in the library by Koržinek (2019). The acceleration is calculated the same way as the delta, but instead of c being the MFCC, it is the delta coefficients. In most of the experiments in the literature, the MFCCs are mainly used with speech but many experiments working with nonspeech audio also use MFCCs for classification such as Portelo *et al.* (2009).

Since MFCC features are often used for the classification of both verbal and non-speech audio signals and emulates roughly the way the human ear perceive the different frequencies, it could be a good feature to use for this project.

1.2.2 Auditory-inspired Amplitude Modulation Features

Auditory-inspired amplitude modulation features (AAMF) are first proposed by Sarria-Paja & Falk (2017) for the task of speaker verification using whispered speech. AAMFs are created by first calculating the spectrogram of the audio signal. A Fast Fourrier Transform is then applied to a sliding window of a spectrogram which calculates the modulation of each frequency. This results in a three dimensional vector with the three axes representing time, acoustic frequencies and modulation frequencies. This process can be seen in figure 1.4. Since the resulting vector is big, a dimensionality reduction algorithm is applied to the vector to reduce the number of variables.

When using AAMFs, it is assumed that the audio signal is created by multiplying a low-frequency modulator with a higher frequency carrier. The AAMF can find the amplitude of those low frequency modulators that can give us more information about the signal. In the case of speaker recognition, the modulation frequencies contain information about speaking rate and other speaker identification attributes. Therefore, AAMFs should contain more pertinent information than standard MFCC in audio containing slow varying amplitudes.

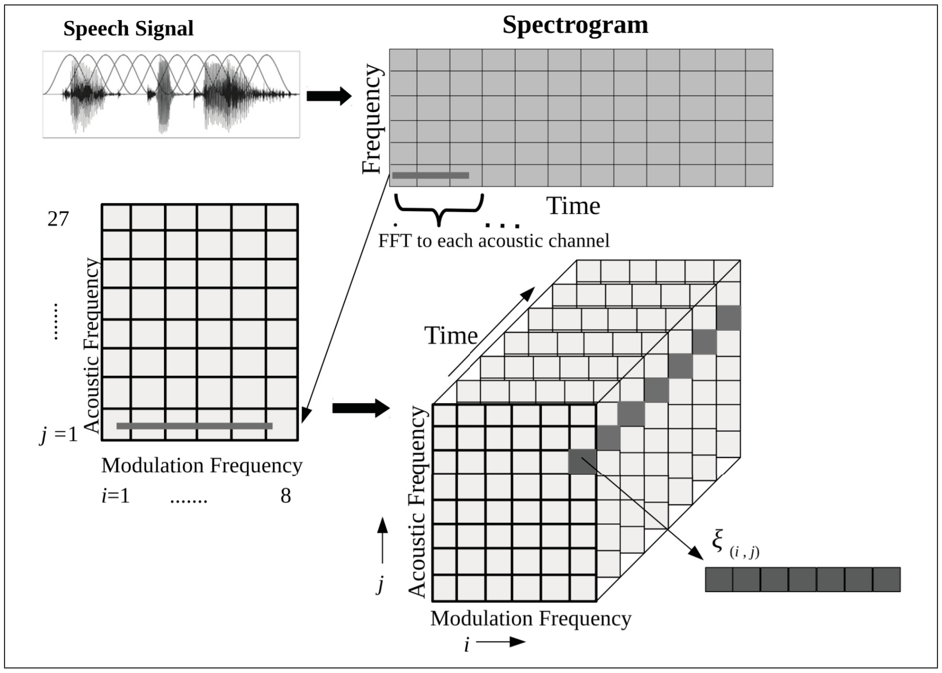


Figure 1.4 How to extract AAMF
Taken From Sarria-Paja & Falk (2017)

For the purpose of this work, AAMF features might be useful to discriminate some of the nonverbal classes that are longer and contain slow varying amplitudes such as saliva noise, grinding of the teeth or closing of the eyes forcefully.

1.2.3 Per Channel Energy Normalisation

Per Channel Energy Normalisation (PCEN) was used in Wang *et al.* (2017) for far-field keyword spotting. One of the main drawbacks of the log compression used in the MFCC is that it uses a lot of its range for low amplitude signals. Low signal amplitudes that are close to zero are not very useful for classification because they are usually the least interesting part of the signal.

Instead of using log compression, PCEN uses dynamic compression. This compression technique stabilizes signal levels for each channel of the melspectrogram and gives more of the dynamic range to the high amplitude signals that are more important. To extract the PCEN, the formula found in Wang *et al.* (2017) is used:

$$PCEN(t,f) = \left(\frac{E(t,f)}{(\varepsilon + M(t,f))^{\alpha}} + \delta\right)^{r} - \delta^{r} \tag{1.2}$$

where $E(t,f)$ is the melspectrogram of the signal and $M(t,f)$ represents the first order IIR filter:

$$M(t,f) = (1-s)M(t-1,f) + sE(t,f) \tag{1.3}$$

The PCEN is interesting for non-verbal events because of the large difference in amplitude between some of the events found in this project such as coughing and blinking of the eyes forcefully.

1.2.4 Spectrograms

If the classifier chosen for the AEC task is a two dimensional CNN, another type of feature is needed. The CNN relies on the convolution of filters over an image or a 2D matrix to do the classification, therefore an input matrix is needed. For audio signals, their spectrograms can be used. Techniques such as the Short Time Fourier Transform (STFT) or the continuous wavelet transform can be used to extract the spectrogram from the audio signal to result in a two dimensional matrix that represents the power of a certain point in frequency and time. MFCCs can also be used as 2D input when all the frames are concatenated together resulting in information on the power of each Mel filter in the signal over time.

Since CNNs are one of the machine learning algorithms used in this project, the spectrogram is going to be used as input for this deep learning algorithm.

1.2.5 Dimensionality Reduction

The goal of the feature extraction is to process the signal in a way that the resulting features have a low dimensionality and characterize the signal in a meaningful way so that it can be properly classified. Having low dimensionality is important because the time needed to run a machine learning algorithm is correlated with the length of its feature vector. It is important for this project since the audio classifier needs to run in real-time on a low-power processor. A problematic phenomenon called the curse of dimensionality might also occur if the length of the feature vector is too high. This problem happens because it is much harder and longer to find the perfect solution to a classification problem when the dimensionality is high because the amount of possible solutions increases exponentially with the number of elements in the input vector. Techniques such as Principal Component Analysis (PCA) as found in Jolliffe (2011) and Linear Discriminant Analysis (LDA) as found in Mika *et al.* (1999) can be used to reduce the dimensionality of the feature vector while keeping the most relevant information. To do the PCA, the covariance matrix of the feature vector is first calculated. The eigenvectors are then calculated with their respective eigenvalue which represents the amount of variance for each of the eigenvectors. The top N eigenvectors are then selected based on their eigenvalue with N being the final length of the feature vector. The amount of information in the PCA can be chosen by increasing or decreasing the amount of eigenvectors that are kept. At the end, it results in a matrix capable of transforming the initial feature vector into a smaller feature vector in another feature space. A visual example of the PCA being applied to a 2 dimentional Gaussian distribution can be seen in Fig. 1.5.

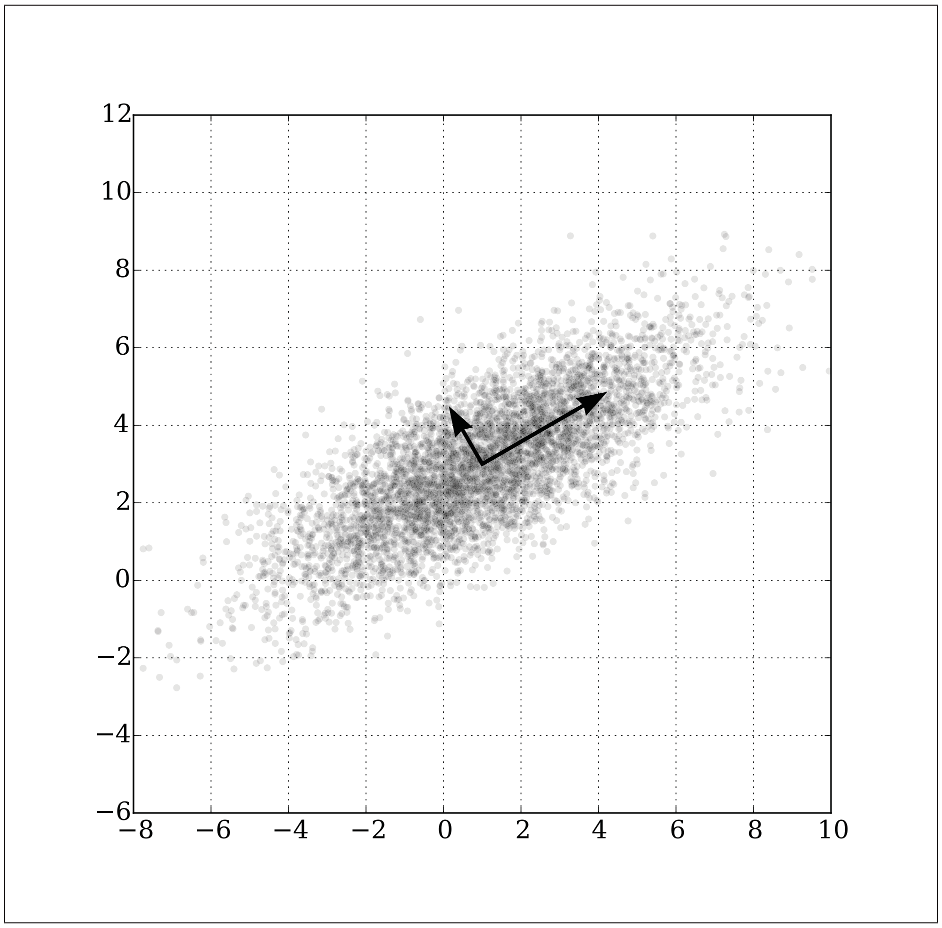


Figure 1.5 Example of eigenvector extracted from a 2D gaussian distribution
https://upload.wikimedia.org
Consulted on July 10th (2019)

The LDA is a supervised feature reduction technique, meaning that the class of each feature vector must be known prior to reducing the dimensionality. The LDA finds a new feature space by maximizing the distance between each class and minimizing the spread within each class. However, having too few elements in the vector makes it harder to classify the signal since not enough information is given for the algorithm to learn. An example of the LDA can be seen in Fig. 1.6 showing a PCA and an LDA on a dataset containing two classes (red and blue).

Since the LDA takes into account the different classes when calculating the new feature space, it results in a smaller overlap between the classes.

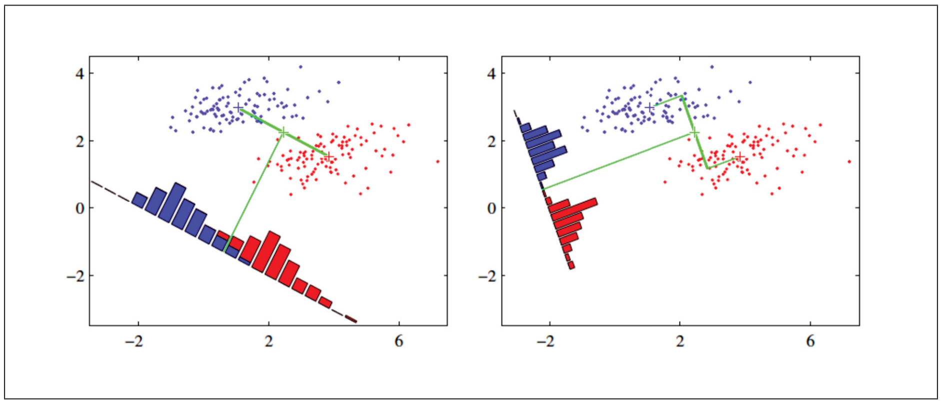


Figure 1.6 Comparison of PCA and LDA on a 2D dataset
Taken from Li (2014)

Dimensionality reduction techniques such as PCA are useful when using feature extraction techniques that output a vector with a high dimensionality. For this work, AAMF outputs a high dimension vector that requires a dimensionality reduction before feeding it to the machine learning algorithm.

1.2.6 Data Augmentation

In addition to the dimensionality reduction seen in the previous subsection, data augmentation can be used to improve the performance of the machine learning system. The performance of a machine learning implementation is correlated to the quality of its database. The larger the database is and the more varied it is, the better the machine learning system will be to successfully classify the audio.

In order to increase the number of audio samples inside the database data augmentation can be used. This method is often used for image classification. It creates new images by rotating, cropping, zooming, warping or flipping the original images (Perez & Wang, 2017).

For audio samples, the transformations used are a bit different. In Salamon & Bello (2017), they compare time shift, pitch shift, dynamic range compression, background noise addition, and all those augmentations combined. The new audio samples need to still be realistic so that they could be encountered in the classification scenario. If those augmentations do not produce realistic sounding audio, the performance of the classifier could be affected.

This technique has mostly been used for deep learning machine learning classifiers since they require the database to be big and varied in order to have a good performance.

1.3 Classifiers

In order to be able to use nonverbal audio events in hearable technologies, both audio event classification (AEC) and audio event detection (AED) must be achieved. Phan *et al.* (2016b) explain thoroughly the difference between the two tasks and why AED is harder than AEC. AED needs to be able to detect a class in an audio stream by first detecting when an event occurs, either by detecting a change in the energy and then classifying the event or by continuously classifying the data in the audio stream. After the detection is made, it needs to correctly identify which class the current sample belongs to. Audio classification on the other hand needs only to classify a given audio sample. The training of a classifier is therefore one of the first steps of AED.

For classification, two kinds of machine learning algorithms exist, supervised and unsupervised. The difference between supervised and unsupervised machine learning algorithms is that supervised algorithms learn by having both the input data and the classes tagged to the data. Unsupervised algorithms only need the input data and they create groups that have similar characteristics within the given data.

In the following sections, the different classifiers that are used in the AEC and AED literature are presented with a short explanation of how they work and a few examples of their use in the literature.

1.3.1 Support Vector Machine

The Support Vector Machine (SVM) is a supervised machine learning algorithm that tries to separate the data using a hyperplane in the dimensional space. For the hyperplane there are multiple types of kernel functions available to choose from such as Linear, Radial Basis Function and Polynomial. The hyperplane is chosen to maximize the distance between two classes. An example of a linear SVM can be seen in figure 1.7.

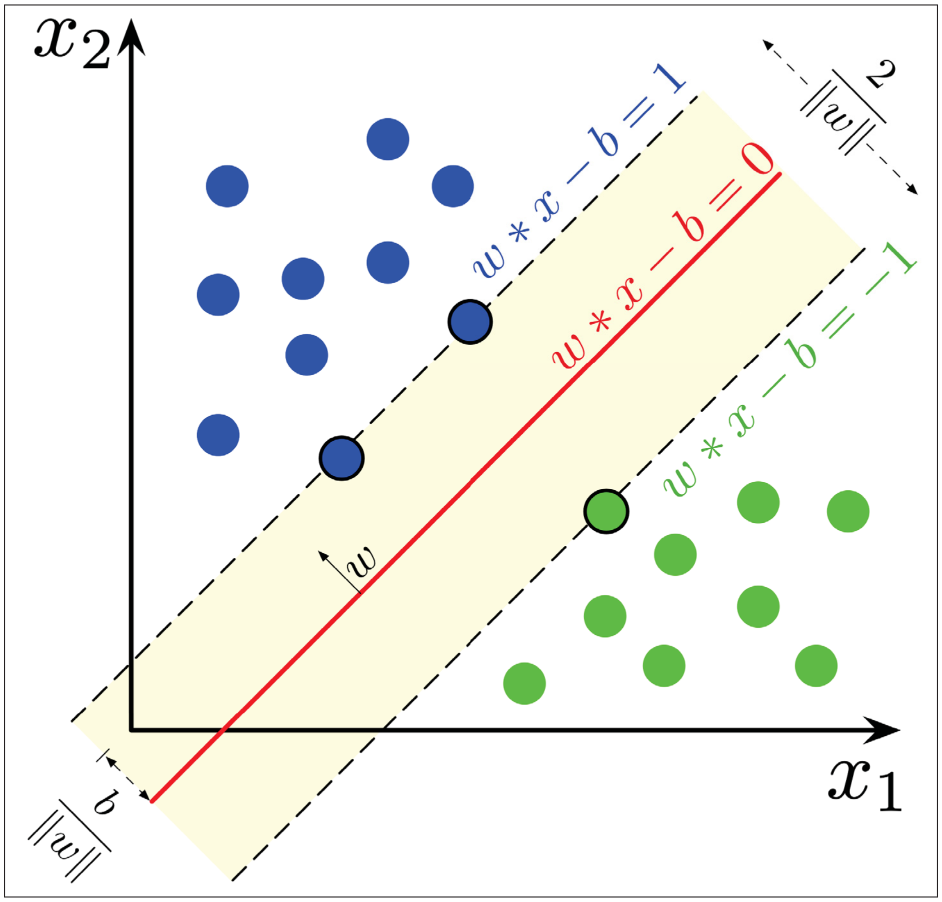


Figure 1.7 SVM classifier using the linear kernel
https://upload.wikimedia.org
Consulted on July 10th (2019)

By adjusting the parameter C, it is possible to either optimize for a hyperplane with smaller margins that separates the training dataset more precisely, or a larger margin hyperplane that looks for a broader trend in the limit between the two classes but might misclassify some data points in the training set. The best value of the parameter depends on the training and testing dataset. To use the SVM algorithm in multi-class classification, there are two approaches mainly used in the literature. The first is the one-vs-one approach which creates classifiers that compare only two classes at a time and choose the final class with a decision function. The total number of classifiers with that approach is: $\frac{n(n-1)}{2}$, where n is the number of classes. The other approach is called one-vs-rest where each class is compared to the rest of the classes. This approach requires only one classifier per class. The SVM algorithm is often used in AEC as seen in Zhuang *et al.* (2008); Phan *et al.* (2016a); Portelo *et al.* (2009).

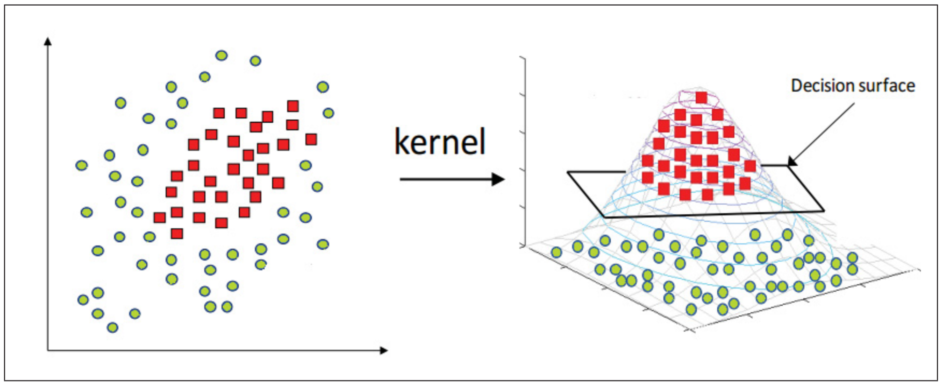


Figure 1.8 Effect of the Kernel on the datapoints
https://www.medium.com
Consulted on July 10th (2019)

Basically, a SVM is a linear classifier. However, most interesting problems are not linearly separable. To circumvent that, SVM use the kernel trick, which projects the data points in a higher dimensionality space on which the data will be linearly separable. This process is called the kernel trick. Figure 1.8 shows an example of this process. The data in the left part of the image is not linearly separable but when a kernel is applied, a linear decision surface appears. In this specific case, a second degree polynomial kernel has been used. Although SVMs are very good at separating classes linearly, when the separation is not linear, the optimal choice of the kernels and parameters related to those kernels becomes more difficult. In addition, this algorithm takes a lot of time to train when working with big datasets.

1.3.2 Gaussian Mixture Model

The Gaussian Mixture Model (GMM) is also a very commonly used classifier. It has been used to classify audio events such as steps, phone ringing, laughter and even gunshots (Zhuang *et al.*, 2008; Geiger & Helwani, 2015). The GMM is an unsupervised clustering method that can be used in a supervised manner as a classifier. To classify the classes, we calculate the score s by taking the probability p of the Gaussian multiplied by its weight w for each of the N Gaussians in the mixture associated with the class as seen in equation 1.4.

$$s = \sum_{i=1}^{N} p_i w_i \tag{1.4}$$

The GMM represents the given data as a mixture of multiple Gaussian distributions and returns the means, variances, and weight of the Gaussians. To find those values, it uses the maximum-likelihood estimation which uses the expectation-maximization algorithm to converge into local maximums. The Gaussian distribution is determined by its covariance matrix and its mean vector. The covariance matrix represents the size and direction of the Gaussians. Four different kinds of covariance matrices can be used as seen in figure 1.9. The first one is Full, where each component of the GMM has its own covariance matrix, this means that each Gaussian is independent and can have any position and shape. In the Tied covariance, the components share the same general covariance matrix which means that each Gaussian has the same shape. In the Diagonal covariance, each component has its own diagonal covariance matrix which makes the Gaussians oriented along the coordinate axes but the shape can vary between them. Finally, the Spherical covariance makes the Gaussians have the same length in every axes, but their size can vary. The best covariance would be Full, but it requires much more data and time to compute than the other types of covariance matrices since each Gaussian is independent and more parameters need to be calculated.

After separating the data by class, a GMM is calculated for each class and a final GMM, with the means and covariance of each GMM, is created. By remembering which Gaussian belongs

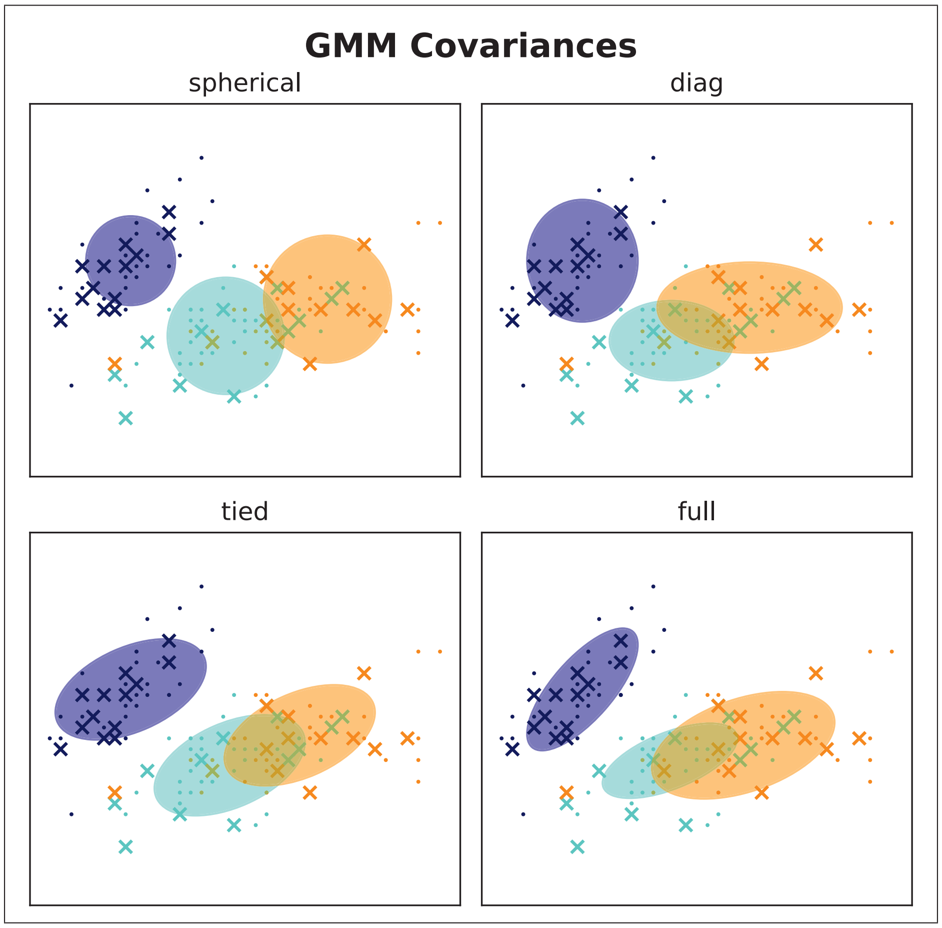


Figure 1.9 Example of various GMM covariances

to which class, it is able to properly classify any sample. This technique turns the GMM into a supervised machine learning algorithm. The GMMs are often used for AEC in the literature, and they are one of the algorithms that produce the best results (Geiger & Helwani, 2015). They are either used alone, or in combination with other machine learning algorithms (Zhuang *et al.*, 2010).

1.3.3 Bag-of-Audio-Words

The Bag-of-Audio-Words (BoAW) or Bag-of-features technique has been prominent in the audio classification and detection literature (Pancoast & Akbacak, 2012; Plinge *et al.*, 2014; Schmitt *et al.*, 2016). The technique Bag-of-Words, which inspired BoAW, has previously been used for Natural Language Processing tasks but it has been successfully modified to work with audio signals for the classification of audio events such as walking, doors closing and typing on a keyboard (Plinge *et al.*, 2014).

Using a big database, certain words that are deemed important for the task are put in a list called a dictionary. Those words can be chosen by taking some of the most reoccurring words in a text database and excluding stop words such as *him*, *the* and *it* because they do not contain a lot of information on the meaning or sentiment of a text. To classify a text, a histogram of all the dictionary words is built, showing the frequency of each word in the dictionary contained in the text. This histogram is then used to classify the text. To use this technique with audio signals, instead of text the audio signal is first divided into multiple frames. A clustering algorithm is then used to find "audio words" or patterns that are used to represent the various classes in every frame. By creating a histogram and finding which "audio word" every frame of the sample belongs to, it is possible to classify the audio signal. The classification is done by a clustering algorithm to learn what the histogram of each class is supposed to be like. This technique removes the temporal information in the classification but there are some techniques, such as the temporal pyramid (Plinge *et al.*, 2014), that gives some temporal information about the event which could be useful when classifying longer samples. A modification done to the BoAW can be seen in Plinge *et al.* (2014), where they improve the performance by using a super-dictionary. The super-dictionary is built by creating a dictionary of size N for each of the C classes, and then clustering each class separately instead of creating a general dictionary by disregarding the labels and using the clustering algorithm on the whole dataset.

1.3.4 Multi-Layer Perceptron

A perceptron is a supervised machine learning technique that does a binary linear classification. A number n of inputs are connected to the perceptron via weights w. The sum of all inputs multiplied by their weight creates the output of the perceptron. During the learning phase, the weights are learned to predict the correct output given a number of samples. This is the essential building block for neural networks and Multi-Layer Perceptrons (MLP).

When multiple perceptrons are combined together, as in figure 1.10, it is called a multilayer perceptron. It contains at least three layers: the input layers, one or more hidden layers and the output layer. As the number of hidden layers increases, the network becomes more complex and capable of solving harder classification problems. However, the size of the training database and the time it takes to train and classify increases as well. To train the MLP, back propagation is used that updates the weights of the network based on the predicted output and the real output using the gradient descent optimisation method presented in Hecht-nielsen (1992). To do the back propagation, first, a forward pass is done that tries to find the output with the current weights given an input. For each output neuron, the error is calculated by calculating the squared error between the true output and the output given by the network. Then, the backward pass is done to update the weights of the neurons inside the network. To do so, the partial derivatives of the total error with respect to each weight are calculated from the last layer to the first layer. The partial derivatives multiplied by a learning rate are then subtracted from the previous weights, which results in new updated weights.

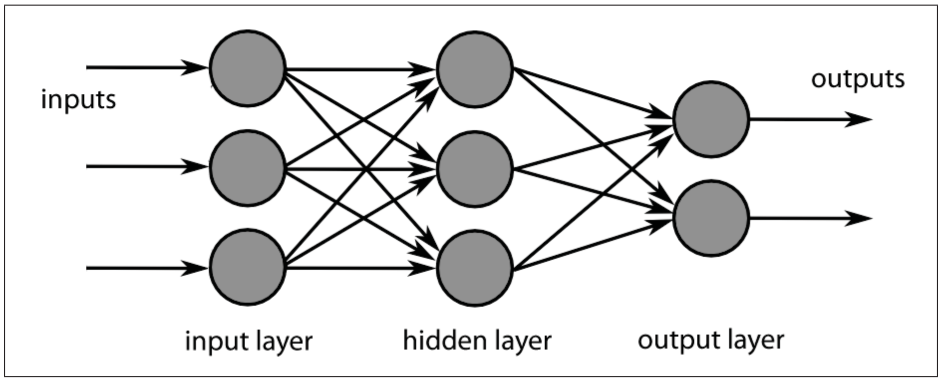


Figure 1.10 MLP architecture
https://upload.wikimedia.org
Consulted on July 10th (2019)

While the MLP is not as popular as other deep learning algorithms discussed in the recent literature for audio classification, it is still useful for its reduced complexity and lower demand for large datasets.

1.3.5 Convolutional Neural Network

The CNN is a deep learning technique that uses neural networks and the convolution of learned filters to classify images or matrices. It was used to classify environmental sounds by Salamon & Bello (2017), acoustic scenes in Phan *et al.* (2016a) and various sound events by McLoughlin *et al.* (2017). The CNN is a supervised machine learning algorithm where the weights of the neural network and the values of the filters are learned using back-propagation and an optimization function such as the gradient descent algorithm or the Adam optimizer algorithm.

The visualisation of the filters learned in different layers can be seen in Fig. 1.11. In the first few layers, the CNN recognizes basic shapes such as lines. The more it advances, the more complex the filters become, recognizing more details such as part of a car wheel or more complex textures.

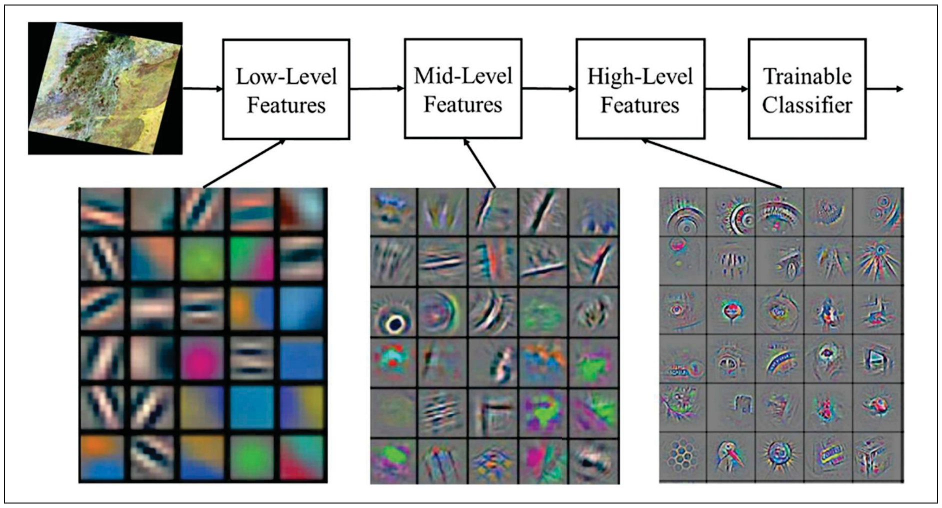


Figure 1.11 CNN filters visualisation
Taken from Traore *et al.* (2018)

The components of a CNN architecture are the convolutional layers, the pooling layers and the fully connected layers. The convolutional layer uses multiple filters that do a convolution over the image. These filters are learned during the training phase and represent patterns that the

CNN looks for to do the classification. The pooling layer, or subsampling layer, reduces the dimensionality using a pooling function such as max pooling or average pooling. For example, if a 4 by 4 matrix needs to be reduced to a 2 by 2 matrix, the original matrix is divided into 4 quadrants with 4 values each. When using the max pooling, the first values of the final matrix are equal to the maximum values of each quadrant while with the average pooling, they are equal to the average of each quadrant. Before the fully connected layers, the matrices are flattened to give a 1D vector that will be used as an input to the fully connected layers. The drawbacks of deep learning algorithms in general, including CNNs, is that training requires a lot of data and a lot of time, which might not always be available.

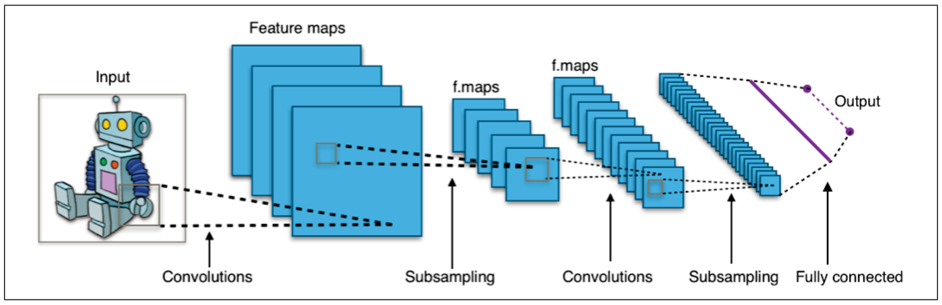


Figure 1.12 Basic CNN architecture
https://upload.wikimedia.org
Consulted on July 10th (2019)

1.3.6 Hidden Markov Model

The Hidden Markov Model (HMM) is mainly used for speech recognition (Rabiner, 1989) but it also works well for AEC because it is made to give the probability of a state or a class, given a sequence of observation in time, which in the case of AEC is the sequence of features extracted for every frame of the audio sample. It works by finding the parameters of the model such as the hidden states, the probability matrix and the start probability vector. To find those, the expectation-maximisation algorithm is used. The model will compute the probability of each class given an input using the Viterbi algorithm that finds the most probable sequence of hidden states that returns the sequence of observed events. The HMMs are used often for

AEC and are part of the state-of-the-art for speaker verification and audio scene classification (Zhuang *et al.*, 2008; Rabaoui *et al.*, 2008; Temko *et al.*, 2009).

Fig. 1.13 represents a basic HMM architecture where *X* represents the states, *y* represents the possible observations (classes), *a* represents the state transition probabilities and *b* the output probabilities.

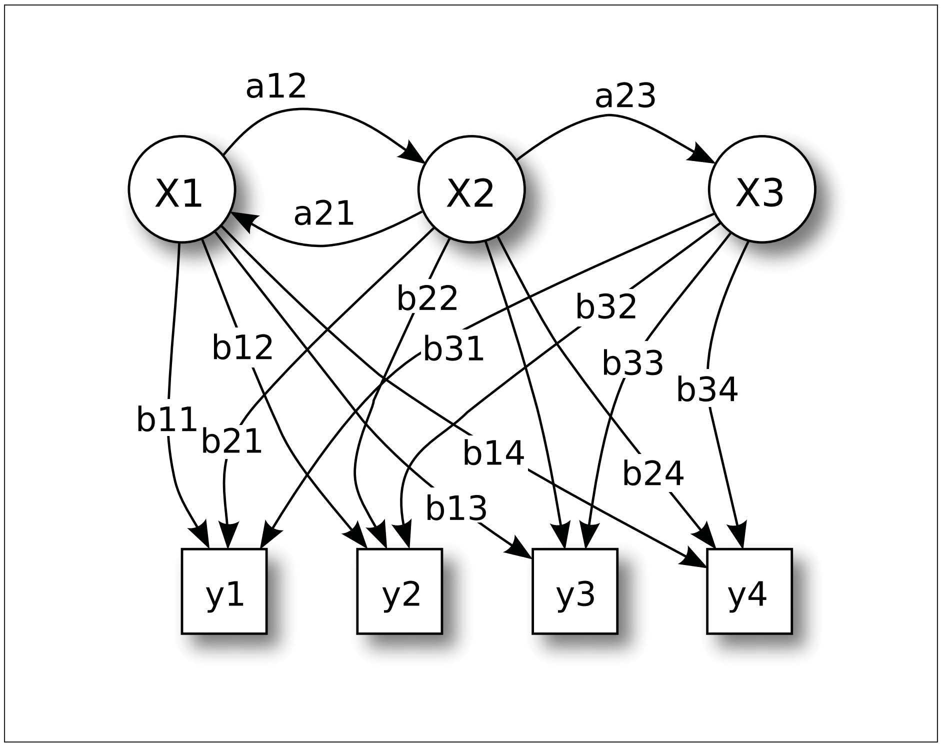


Figure 1.13 Basic HMM architecture where *X* represents the states, *y* represents the possible observations, *a* represents the state transition probabilities and *b* the output probabilities
https://upload.wikimedia.org
Consulted on July 10th (2019)

While long, structured audio can be classified using HMMs easily, it is more difficult to do the classification on short, impulsive audio events due to the lack of high level structures found in speech and music. This is why HMMs are not used for the classification of nonverbal events.

1.4 Conclusion

In this literature review, the state of the art on hearables, signal processing techniques for feature extraction and audio event classification machine learning algorithms are presented. Following the review, it is clear that the classification of nonverbal events could be useful in many ways for the hearable market. The knowledge on signal analysis for these kinds of signals is not available but multiple feature extraction techniques that could help discriminate these events are identified.

CHAPTER 2

METHODOLOGY

In this chapter, methods used to build a nonverbal event classifier are presented. This methodology is exploratory, multiple kinds of feature extraction techniques and classification algorithms are implemented in order to try to find the combination that would provide the best results. In section 2.1, the database used is described. Section 2.2 shows the feature extraction techniques and how they are used. The implementation of the machine learning classifiers are presented in 2.5. The various tests and the validation used for the classifiers are presented in section 2.6.

2.1 iBad Database and Samples Extraction

The database used in this work is created by Alexis Martin and is described in detail in Martin & Voix (2017). In summary, participants are equipped with an occluding intra-aural device containing in-ear microphones as shown in figure 2.1. The audio is captured using the in-ear microphones in each of the participants' ears. For the purpose of this Master's work, only the audio files containing the non-verbal audio events are used. In total there are 24 files containing on average 4 minutes of audio from each unique participant. The participants are asked to do approximately 10 seconds of each of the following events : move the body and head, clear the throat, talk, swallow their saliva, click their tongue, grind their teeth, click their teeth softly and then loudly, close their eyes softly and forcefully, blink their eyes softly and forcefully and yawn. Of these events, 11 are chosen for this task as presented in table 2.1.

Audio samples each containing one iteration of an event are extracted from the audio files. The length of the sample is chosen to be 400 ms to accommodate for the length of some longer events such as talking, clearing the throat and saliva noise. The open-source audio editing software Audacity is used to do the manual segmentation. Each of the events are segmented, tagged and extracted manually.

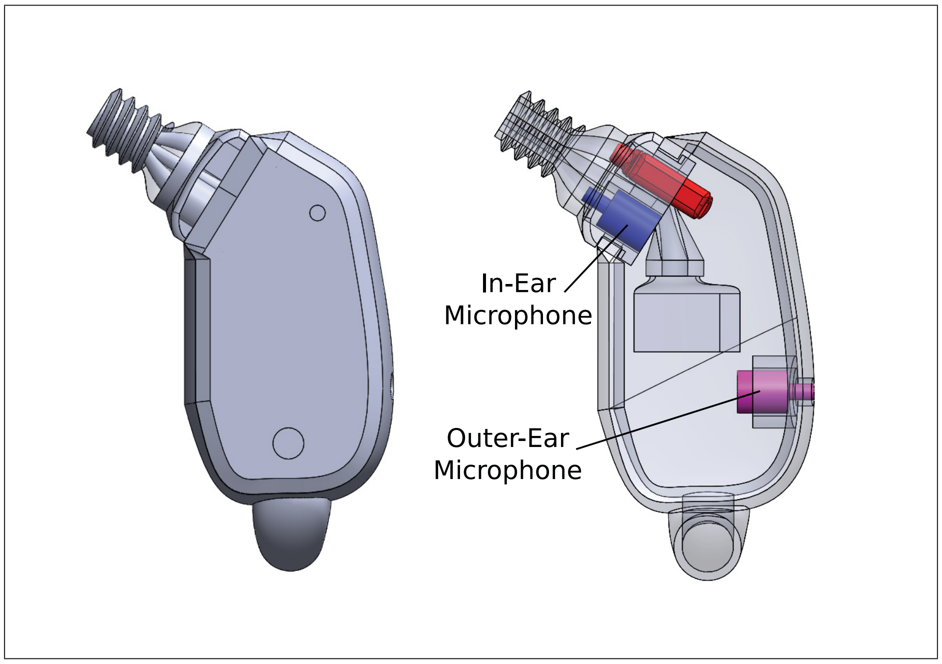


Figure 2.1 The occluding intra-aural device used to collect the database containing an in-ear microphone and an outer-ear microphone

Table 2.1 The total of 400 ms samples for each class

Event	**Number of samples**
Clicking of teeth softly (cts)	246
Clicking of teeth loudly (ctl)	304
Tongue clicking (cl)	364
Blinking forcefully (bf)	207
Closing the eyes (ce)	286
Closing the eyes forcefully (cef)	329
Grinding the teeth (gt)	170
Clearing the throat (clt)	163
Saliva noise (sn)	213
Coughing (c)	219
Talking (t)	526

Certain nonverbal events used in the current work can be categorized by the nature and source of the captured signal. Some audio events are short impulses produced with the mouth such as clicking of the teeth and tongue, while other events come from eye movements, and are of very low amplitude such as blinking forcefully, closing the eyes and closing the eyes forcefully.

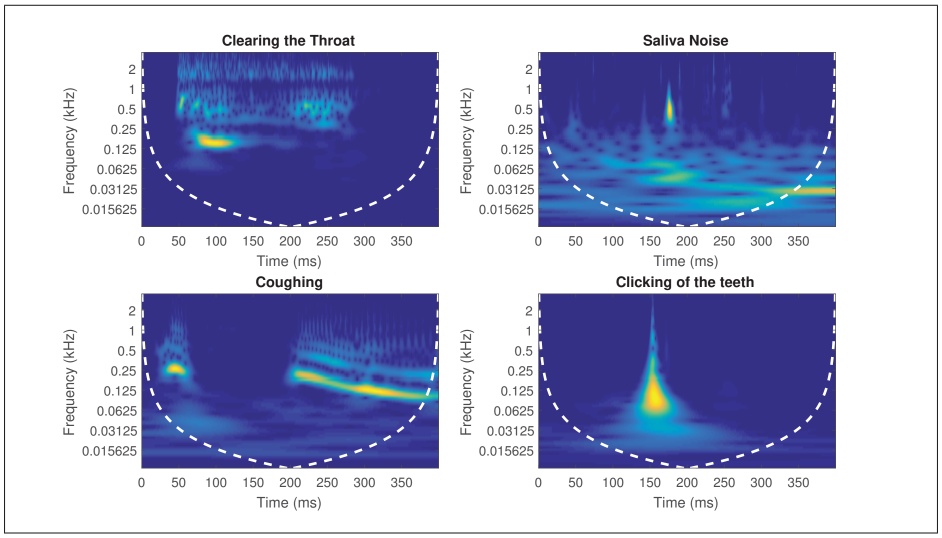


Figure 2.2 Wavelet spectrogram of four randomly selected samples of the following audio events: *clt*, *sn*,*c* and*ctl*

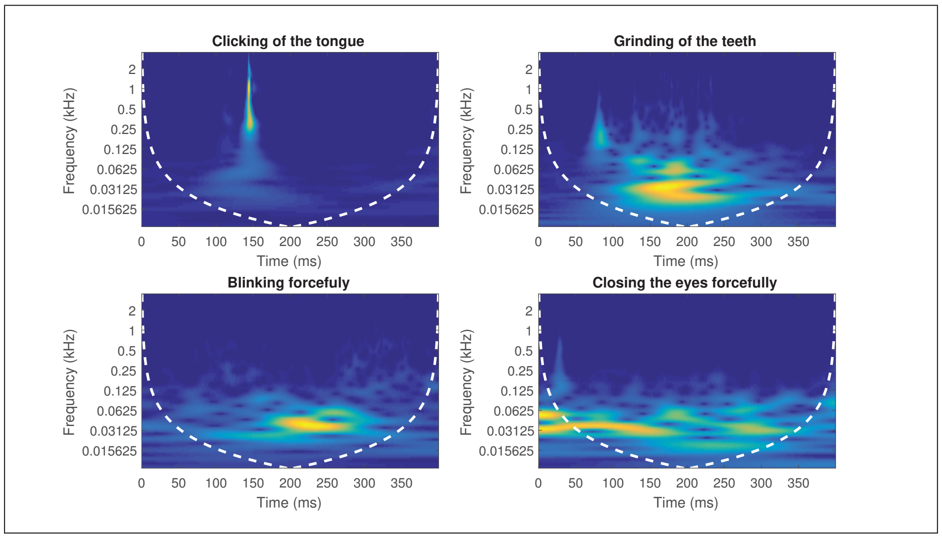


Figure 2.3 Wavelet spectrogram of four randomly selected samples of the following audio events: *ct*, *gt*,*bf* and*cef*

The various audio events shown in this paper have not often been investigated in the literature. The nature of the events being classified differ greatly from one another. This becomes clear when looking at the spectrograms in Fig. 2.2 and Fig. 2.3. Eight different wavelet spectrograms are presented for eight randomly selected samples from different events: Clearing of the throat (*clt*), Saliva noise (*sn*), Coughing (*c*), Clicking of the teeth loudly (*ctl*), Clicking of the tongue (*ct*), Grinding of the teeth (*gt*), Blinking forcefully (*bf*) and Closing the eyes forcefully (*cef*). The spectrogram of the cough signal shows harmonics at the very end of the event as some people tend to voice the end of their cough. Saliva noise event contains low frequency signals with some sporadic high frequency occurrences and has some similarity with grinding of the teeth. Clearing of the throat contains more high frequency content, and some harmonics can be detected as often people voice this event as well. Clicking of the teeth and clicking of the tongue are both a clear impulse containing a large range of frequencies at a precise moment in time. The difference between those two classes is hard to find when looking at their spectrograms. Both blinking forcefully and closing the eyes forcefully contain mostly low frequencies with blinking having more energy concentrated in one point in time. This large variation in amplitude and spectral content requires a feature extraction tool that can account for such wide differences while properly representing each signal.

2.2 Implementation of Feature Extraction Algorithms

To use the samples in the classifier, features must be extracted from the audio signal. By knowing the nature of the signal, the dimensions of the features extracted can be reduced, without losing valuable information. In this case, the audio signals are captured inside an occluded ear and are propagated through bone and soft tissue conduction. The frequency range of this type of conduction is limited to 2 kHz. Respecting the Nyquist frequency, the signal can be downsampled from 48 kHz to 8 kHz without risking the loss of any important information in the audio signal.

Considering that most other feature extraction techniques utilize frequency changes in time, extracting the spectrogram is often one of the first steps. The spectrogram itself is rarely used alone in traditional machine learning algorithms because of its high dimensionality. However, some deep learning techniques such as the CNN, discussed in Section 1.3, are able to use the whole spectrogram without having to reduce the amount of data. To extract the spectrogram, a hamming window is applied to the signal before the short-time Fourier transform (STFT) is calculated, using either MATLAB (The MathWorks Inc., Natick, Massachusetts, United States) or python functions.

In the following subsections, the technique used to extract the MFCC and AAMF features are presented.

2.2.1 Mel Frequency Cepstral Coefficients

The MFCC features are implemented in Python with the help of the pyHTK script (Koržinek, 2019) which emulates the HTK software. HTK (Hidden Markov Model Toolkit) is a popular software in the scientific field used for the training of HMM for speech recognition. It also contains tools to extract features such as MFCC that are used for other audio recognition tasks (Young, 1994). For this reason, this software is used for the feature extraction of MFCC.

First, the window and overlap size used to create the spectrogram need to be chosen. Three different pairs of values are tested based on what is used in the literature for AEC :

- 50 ms window, 25 ms overlap,
- 25 ms window, 12.5 ms overlap,
- 25 ms window, 10 ms overlap.

The MFCC filterbank containing 40 triangular filters going from 33 Hz to 4000 Hz is then applied to the spectrogram of the signal. After the Direct Cosine Transform (DCT), only the first 13 coefficients are taken with their deltas and acceleration, resulting in a feature vector of length 39. The Zero Crossing Rate (ZCR) is also added to the feature vector to give information about the average frequency of the frame by counting the time the audio signal crosses zero, increasing the length of the input vector to 40.

This feature is chosen because it is the most popular feature and has proved itself in the literature to be a good feature for the classification of audio events.

2.2.2 Auditory-inspired Amplitude Modulation Features

The AAMF features are extracted using a MATLAB script created by Milton Sarria Paja who collaborated on part of this work. As found in Sarria-Paja & Falk (2017), windows of 200 ms are used in order to properly capture low frequency information. This results in a matrix containing 216 elements in total (27 acoustic bands $\times$ 8 modulation bands $=$ 216 dimensions). This matrix is then collapsed into a smaller vector of 40 using the PCA technique. Even when reducing the dimensionality, 98.7% of the cumulative variance still remains.

This feature extraction technique is chosen due to its performance in whispered speech. This feature is not used a lot in the literature but looked promising due to the slow varying nature of some of the classes of interest in this work.

2.2.3 Per-Channel Energy Normalisation

The PCEN features are extracted using the librosa.core.pcen function from the librosa library (McFee *et al.*, 2015). This function takes the spectrogram as input and outputs the PCEN representation of it. The following parameters are chosen empirically and found to give the best results for the classification problem at hand: time constant $s = 0.025$, gain $\alpha = 0.75$, bias $\delta = 2$, $\varepsilon = 1e-06$ and power $r = 0.05$.

This feature is chosen because it builds on what the MFCCs are doing but is shown to work better in situation where the signal to noise ratio is lower, which could be the case for some classes related to the movement of the eyelids.

2.3 Input Vector Style

After extracting the features from the samples, they need to be shaped into 1D input vectors to the various classifier algorithms. Three styles of input vectors have been identified and tested: Framed, Contextual and Total.

The Framed style separates the features that are in a time-frequency matrix into multiple time-based vectors. All the vectors are classified and the whole sample is classified using a majority vote.

The second input vector style is the Contextual style. Using the Framed vector, the current vector is concatenated with the m previous and the m next vectors. Thus, the length of one Contextual vector is $(2*m)+1$ the size of one Framed vector. This style would also require a majority vote in order to classify the whole sample.

The last input vector style, Total, concatenates all framed vectors together and contains one input vector per sample. With this style a majority vote is not needed since there is only one vector to classify.

Figure 2.4, presents a visualization of the different types of input vectors frames. Note that 40 represents the size of the MFCC feature vector and N represents the number of frames extracted. The number of frames depends on the window and overlap size.

2.4 Data Augmentation

The performances of classifiers are directly related to the number of samples in the training dataset and the diversity in the different samples. To increase the number of samples in a dataset, it is possible to do what is called Data augmentation (Salamon & Bello, 2017). The audio samples are taken and modified to create new data that are different than the original yet still realistic and could be encountered in the classification scenario. In this project, various types of data augmentation are tested such as binaural augmentation and additive noise.

2.4.1 Binaural Data

The data from the iBad database contains two channel audio representing signals captured from each ear. Usually, in audio classification, either one channel is chosen, or the average of both channels since the information in both channels is similar. However, in the case of this data,

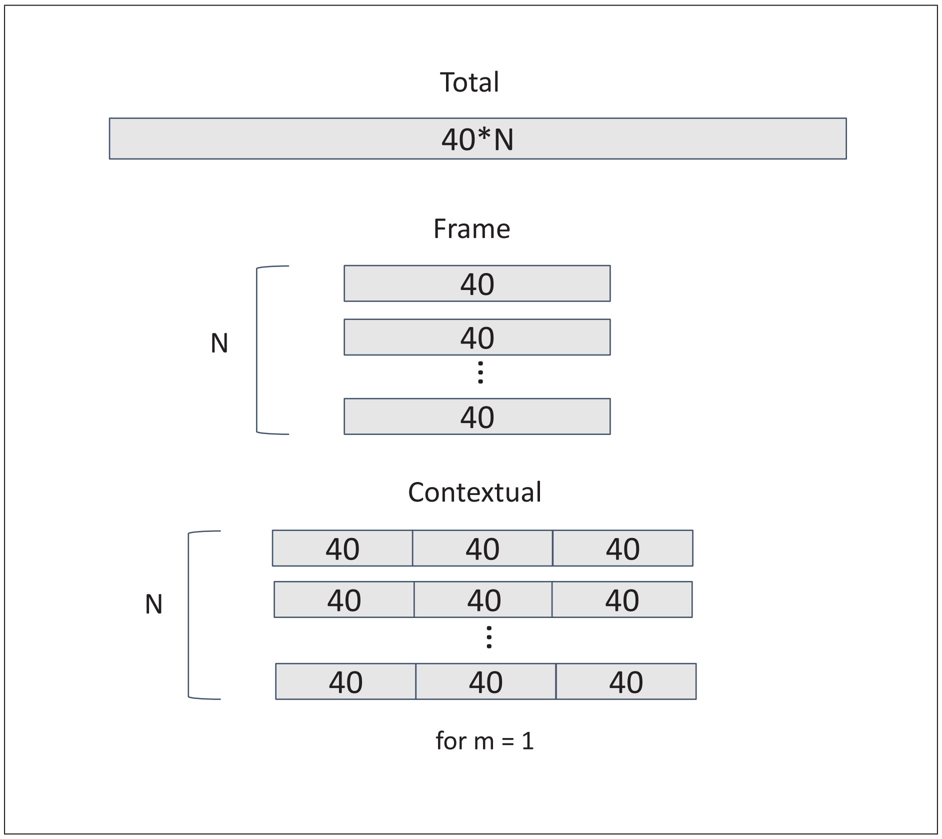


Figure 2.4 Visualization of the different types of input vector frames

since the microphones are situated in both ears, the difference in the fit of the earplug and the difference in geometry of one's earcanal is enough to create a difference in the audio signal. This makes using both channels a natural and robust form of data augmentation.

2.4.2 Additive Noise

The other kind of data augmentation used is additive noise to the samples. In theory, this could make the classifier more resilient against noisy data. Since one of the applications of this work involved noisy industrial environments, the classifier needs to be robust to noisy input. The signal-to-noise ratios (SNR) tested is chosen to be 10dB because it adequately represents the noise level of the factory environment under the hearing protection of the worker. To do this, a MATLAB script is used that adds the noise onto the already existing audio events and adjusts the power of the added background noise as a function of the specified SNR. Factory noise from the NOISEX database (Varga & Steeneken, 1993) is used to simulate a noisy industrial setting. Samples taken from the noise audio clip are chosen at random as to not over-fit on a particular segment of the time-varying audio clip.

2.5 Implementation of Machine Learning Algorithms

All the classifiers are implemented in Python 3.5. The open-source python programming language is chosen because of its large amount of machine learning libraries, its great potential for fast prototyping, and the large support community surrounding it. With Python, it is easy and fast to create machine learning algorithms, train them and test them.

Four different classifying algorithms are tested for the audio event classification task: the SVM, GMM, MLP and CNN.

2.5.1 Support Vector Machine

To program the SVM algorithm, the open-source library Scikit-learn (Pedregosa *et al.* (2011)) is used with the package LinearSVC and OneVSAll. The Linear kernel is used to calculate the hyperplane. Other kernels are briefly tested but the linear option is the fastest and most accurate. The OneVSAll makes the SVM into a multiclass classifier with 11 hyperplanes. The optimal *C* constant used for the SVM is empirically found to be 0.01.

This classifier is chosen as it is popular in the literature and works well with high dimentionality input vector compared to some of the other classifiers, even if the size of the database is small. It is also an algorithm that does not require a lot of computational power.

2.5.2 Gaussian Mixture Model

For the GMM algorithm, Scikit-learn is used with the package GMM. The diagonal covariance is used to calculate the Gaussian mixtures with 15 gaussians per class. Since the GMM is usually an unsupervised algorithm, the pseudocode for training it in a supervised way by training each class separately is presented in algorithm 2.1.

Algorithm 2.1 Pseudocode of the Training of the GMM classifier

```
Input: Feature Vector X_Train, Truth Vector Y_Train of classes.
Output: Trained Final GMM Classifier
for every class $C \in$ Y_Train do
    ClassData = X_train[ Where Y_Train = C ]
    PatrialGMM = TrainGMM(ClassData)
    Means.append( PartialGMM.means )
    Weight.append( PartialGMM.weight )
    Covariance.append( PartialGMM.covariance )
end
FinalGMM.means = Means
FinalGMM.weights = Weights
FinalGMM.covariance = Covariance
```

This algorithm is chosen as it is also popular in the literature, having usually better performance than the SVM for audio event classification. In addition, it can be used as a clustering algorithm for the BoAW.

2.5.3 Multi-Layer Perceptron

The MLP is programmed using the library Tensorflow (Abadi *et al.* (2015)) which is one of the most popular libraries for programming deep learning algorithms. Even though there are libraries that are much simpler for programming small MLPs, this library is used to better understand Tensorflow as it is used later to train the CNN.

The MLP is built using two or three hidden layers depending on the input vector used with a rectified linear activation function and a linear activation function for the output layer. The network is trained using the cross entropy loss function and the Adam method of optimization as described in Kingma & Ba (2014).

This algorithm is chosen to see if a simple neural network would perform well compared to a complex CNN due to the small size of the database.

2.5.4 Convolutional Neural Network

Similarly to how the MLP is trained, the CNN deep learning algorithm is implemented using the Library Tensorflow. The architecture of the CNN is based on the one found in McLoughlin *et al.* (2017) since it is used to classify audio events. It contains a convolutional layer with 24 5x5 filters, a 2x2 pooling layer, another convolutional layer with 48 5x5 filters, a 2x2 pooling layer, followed by a fully connected layer with 64 hidden units and 11 outputs. The input used for the CNN is a spectrogram calculated with a window of 50 ms with 25 ms overlap.

The CNN is chosen because it has become popular recently in audio event classification. This algorithm works well with bigger dataset but experimentation is still done to see if it could still perform well when using data augmentation to make the database bigger.

2.5.5 Bag Of Audio Words

For the BoAW classifier, a clustering algorithm and classification algorithm need to be trained. The clustering is trained on the dataset and the classifier is trained on the histogram outputted by the clustering algorithm.

Many clustering methods can be used, however, for the purposes of this work, the GMM clustering method is chosen since it has already shown good performance in previous experimentation and is compared to the K-Means method using the Lloyd's algorithm (Kanungo *et al.*, 2002), which are both clustering algorithms that are often used for BoAW.

For the classification task, the two algorithms compared are the SVM and the Random Forest. The SVM is programmed using the Scikit-Learn library on Python using the OneVSAll and SVC modules (Pedregosa *et al.*, 2011). A polynomial kernel of second degree is used with a

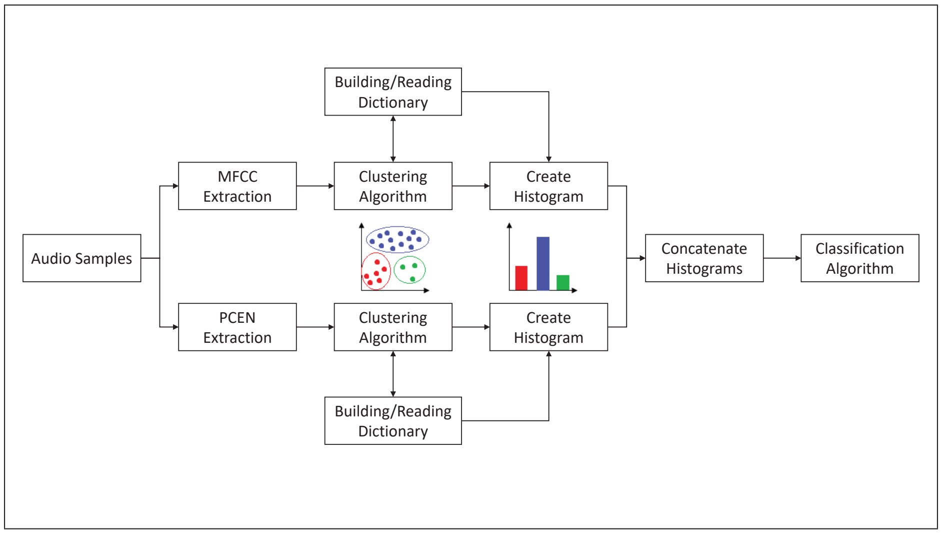


Figure 2.5 Diagram of the BoAW algorithm using fusion at the histogram level with MFCC and PCEN features

constant $C = 0.01$. The Random Forest is programmed using the Scikit-Learn library with 500 decision trees in total.

A modification is made to the classic BoAW method. Instead of doing the fusion of different features at the input level, the fusion is done at the histogram level. For each feature used, one dictionary is created and one histogram is extracted, and the histograms are subsequently concatenated together. For the N features used, N different dictionaries are created. This modification gives the possibility of using different lengths of frames and overlap to the different features. The previous method requires that all feature extraction techniques output the same number of frames to concatenate the frames of each technique. When the fusion is at the histogram level, the number of frames can differ between each feature extraction technique. The structure of this technique can be seen in Fig. 2.5.

The BoAW is chosen since it is becoming increasingly popular in the literature and is performing better than SVM and GMM in some cases while not requiring a bigger database.

2.6 Tests and Validation

2.6.1 Testing of the Classification Algorithm

In order to calculate the performance of the machine learning algorithms on the whole database, the classification is done using the 10-fold cross validation technique. This method separates the database into 10 batches of samples of equal size. In order to make sure that the classifiers are not speaker dependant, all the different participants are separated in different folds while making sure the each fold had the same number of samples per class. By training on 9 folds, doing the verification on one, and repeating 10 times until all the folds have been tested, the average performance over the whole database can be calculated.

To evaluate the performance of the classifier, the sensitivity is used which uses the formula 2.1

$$SN = \frac{TP}{TP+FN}, \tag{2.1}$$

The precision is also used in the later tests which uses the formula 2.2

$$PREC = \frac{TP}{TP+FP}, \tag{2.2}$$

In those formulas, TP represents the number of true positive, FN the number of false negative and FP the number of false positives. These metrics are chosen because they do not contain the true negative rate (TN), which would not be representative since the number of TN is, on average, ten times greater than the number of TP due to the high number of classes.

The sensitivity and performance of each class is calculated separately. The performance of the whole classifier is then calculated by taking the average of the sensitivity of each classes. This prevents that a class with more samples would have a bigger weight in the calculation of the score due to the bigger amount of TP or FN of a given class.

2.7 Conclusion

This chapter presented the various tools and methods used in this project:

- the iBad database,
- the different combinations of types of feature vectors ,
- the machine learning approaches that have been tested ,
- the metrics used to compare results from the various experiments.

The various combinations of the aforementioned existing tools and techniques allows for the exploration and better understanding of what works with these rarely studied nonverbal events. The results of the experiments are presented in detail in the next chapter.

CHAPTER 3

EXPERIMENTATION AND RESULTS

In this section, the experimentation to find the best classifier is described chronologically. The results of these tests are then presented and analyzed. Each experiment and its results are shown in their respective subsection which is followed by an experiment that aims to improve on what was previously done.

3.1 Classifier and Input Vector Style

The GMM, MLP and SVM algorithms are first chosen due to their popularity in the literature and the fact that they could work well on databases of smaller size compared to deep learning.

The three different kinds of input vectors described in section 2.3 are also tested for each of the classifiers to see which one is the most appropriate for the task of non-verbal audio event classification.

Figure 3.1 represents the average sensitivity over all classes for the different tested conditions. The three classifiers GMM, MLP and SVM are compared for different input vectors and are represented with a suffix in the figure: Contextual (-C), Framed (-F) and Total (no suffix).

The data used for the training had the binaural data and the MFCC features extracted with windows of 50 ms and 25 ms overlap. The average is represented by the diamonds and the exact numerical values are displayed for each box plot. The boxplot shows that the GMM with the contextual feature vector has the best performance with 75.5% compared to the other classifiers. The MLP and SVM have considerably worse performance with all of the feature vector types. In contrast to the GMM, both of the classifiers had the best performance with the Total feature vector and worst with the Framed. The bad performance of the MLP could be due to the small dataset while the reason why the SVM did not perform well is unknown.

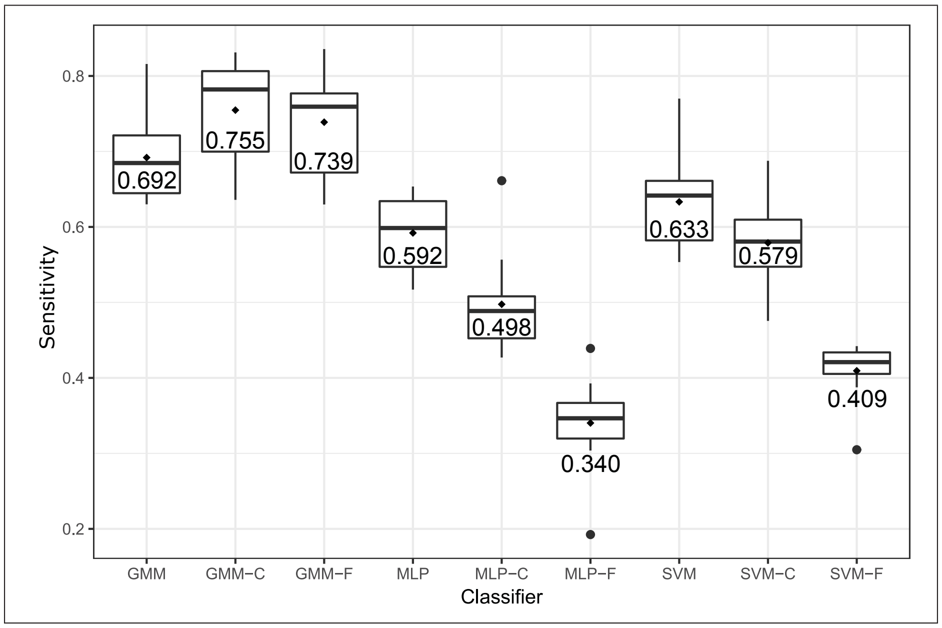


Figure 3.1 Boxplot of the various classifiers and input vectors with 50 ms window and 25 ms overlap

Since the GMM-C classifier had the best performance in sensitivity, the remainder of the tests are performed using the GMM classifier using the contextual input vector.

In table 3.1, the performance obtained with the GMM-C algorithm using different window and overlap time is presented. The best results are achieved using the 50 ms windows with the 25 ms overlap with a sensitivity of 75.5% compared to 74.1% for 25 ms window with 17.5 ms overlap and 72.4% with the 25 ms window with 10 ms overlap.

Table 3.1 Performance of the GMM-C classifier using different window length and overlap

Window		
Length	**Overlap**	**Sensitivity**
50 ms	25 ms	0.755
25 ms	17.5 ms	0.741
25 ms	10 ms	0.724

3.2 Number of Contextual Frames and Binaural Dataset

The performances of classifiers are directly related to the number of samples in the training dataset and the diversity in the different samples. As mentioned in section 2.4.1, the audio in the database contains the recording from both ears. This can help the performance by doubling the size of the database by using both the right and left channel recordings.

Table 3.2 Overall mean for the GMM classifier with contextual features (GMM-C) using monaural versus binaural datasets

Dataset	**Left**	**Right**	**Binaural**
Overall Mean	0.750	0.733	0.755

In table 3.2, the right and left stereo tracks are compared to using both channels to train the GMM-C classifier. While the previous results in Fig. 3.1 already contained the binaural dataset, this table shows that there is a benefit in using the binaural dataset rather than either the right or left channel of the samples inside the dataset. The binaural dataset has 75.5% of sensitivity compared to 75% and 73.3% for the left and right channels respectively. As suspected , including both channels in the feature vector served as data augmentation and increases the sensitivity of the classifier. The difference between the right and left channel could be attributed to the difference in fit between the right and left ear of the test subjects as well the earcanal geometry of each ear, resulting in varying levels of occlusion effect and thus creating differences between signals of the left and right channel. In order to try and improve the performance, both channels are used going forward in this project.

In table 3.3, the performance of the GMM-C is presented in relation to the number m of frames used to build the contextual input vector. The value of m varies from 1 to 5 and the best performance is achieved using a $m = 4$ resulting in a sensitivity of 75.5% while both neighboring numbers of frames had a lower sensitivity of 75.3%. For that reason, 4 frames of context are used for the remainder of the tests and are also used for the previous tests.

Table 3.3 Performance of the GMM-C classifier using different number of frames *m* to create the contextual input vector

Nb *m* of frames	**Sensitivity**
1	0.742
2	0.752
3	0.753
4	0.755
5	0.753

3.3 AAMF and Noisy Dataset

Due to the success of data augmentation when using the binaural dataset and to have the classifier more resilient to noise, a noisy dataset is created. The methods used to create this dataset is discussed in section 2.4.2.

To try to increase the performance of the classifier, AAMF features are added to the MFCC feature vector. The noise dataset is also added to the training and testing datasets to look at the performance of the classifier under noisy conditions. The result of these tests can be seen in table 3.4. When trained on only the clean dataset, the performance in noise for the MFCC features decreases from 75.5% to 24.3%. When training on the clean and the noisy dataset, the performance in noise increases to 70.5% but the performance in quiet decreases to 73.1%. When adding the AAMF features and training only on the clean dataset, it is much more sensitive in noise at 32.9% while having the same performance in quiet. When training the MFCC+AAMF features on both clean and noisy datasets, the performance is higher than when using only MFCC in both the clean and noisy condition at 73.5% and 72.8% respectively. Adding the AAMF to the feature vector and adding noisy samples to the database help the GMM algorithm be more resistant to noise and does not decrease the sensitivity when classifying clean events.

Considering that when using the MFCC and MFCC+AAMF features, the performance is the same when testing and training with the clean dataset, it is of interest to compare their respec-

Table 3.4 Overall mean for the GMM classifier with MFCC and MFCC+AAMF contextual features (GMM-C) using Clean, Noisy and Clean & Noisy datasets for training and testing

		Test Dataset	
Features	**Train Dataset**	**Clean**	**Noisy**
MFCC	Clean	0.755	0.243
MFCC	Clean&Noisy	0.731	0.705
MFCC+AAMF	Clean	0.755	0.329
MFCC+AAMF	Clean & Noisy	0.735	0.728

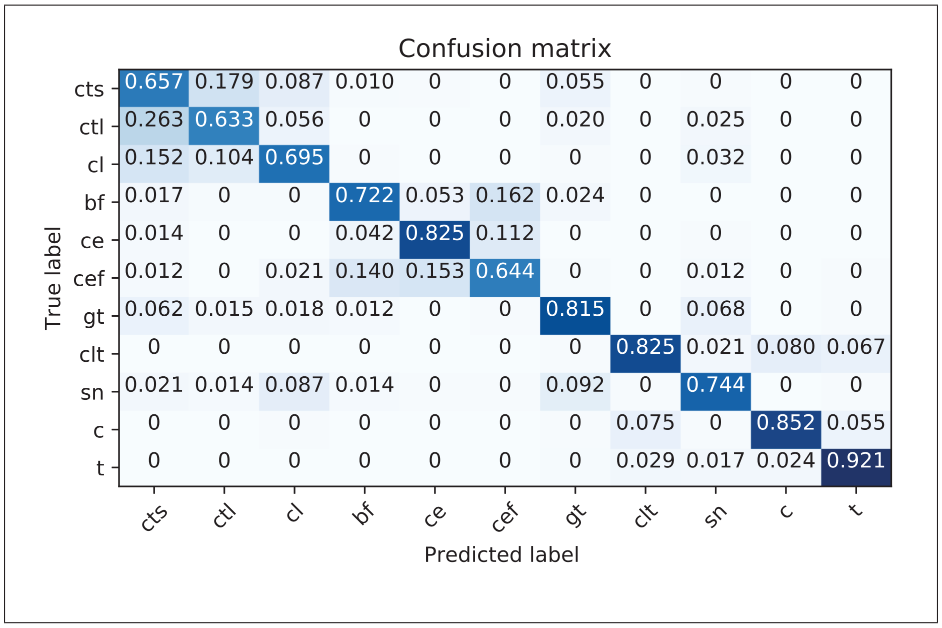
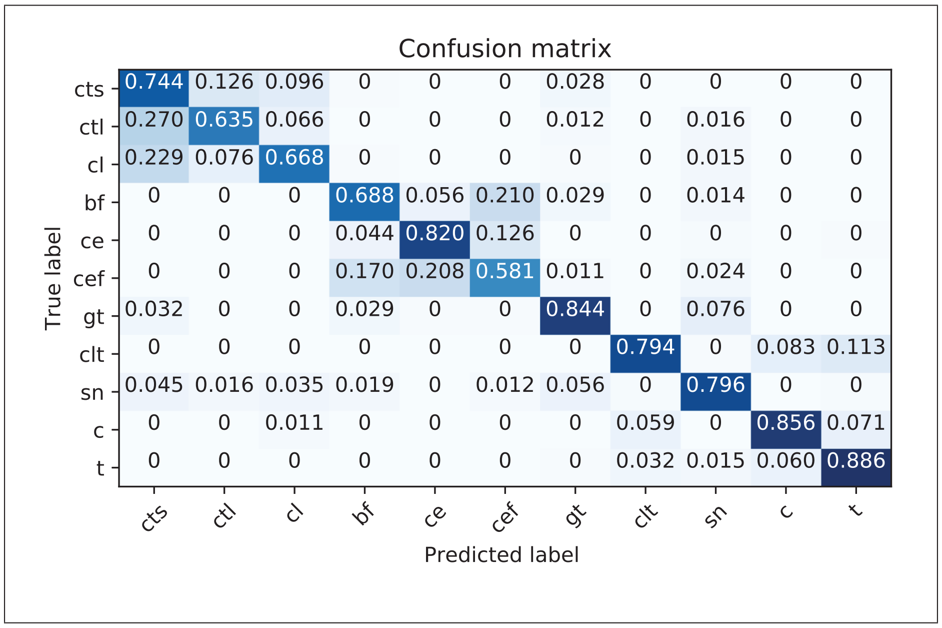


Figure 3.2 Confusion matrix with MFCC feature tested and trained on the clean dataset

tive confusion matrices shown in figure 3.2 and figure 3.3 respectively, to better understand the effects of the feature selection.

For the MFCC features, the most accurate class is speech (*t*) with a sensitivity of 92.1% followed by coughing (*c*) at 85.5% and closing the eyes (*ce*) and clearing the throat (*clt*) at 82.5% sensitivity each. The classes with the lowest performance are clicking the teeth loudly (*ctl*) and closing the eyes forcefully (*cef*) at 63.3% and 64.4%. When looking at the confusion matrix, there are two clear groups of events that are confused together. The first group has all the events


Confusion matrix
True label
Predicted label

True label \ Predicted label	cts	ctl	cl	bf	ce	cef	gt	clt	sn	c	t
cts	0.744	0.126	0.096	0	0	0	0.028	0	0	0	0
ctl	0.270	0.635	0.066	0	0	0	0.012	0	0.016	0	0
cl	0.229	0.076	0.668	0	0	0	0	0	0.015	0	0
bf	0	0	0	0.688	0.056	0.210	0.029	0	0.014	0	0
ce	0	0	0	0.044	0.820	0.126	0	0	0	0	0
cef	0	0	0	0.170	0.208	0.581	0.011	0	0.024	0	0
gt	0.032	0	0	0.029	0	0	0.844	0	0.076	0	0
clt	0	0	0	0	0	0	0	0.794	0	0.083	0.113
sn	0.045	0.016	0.035	0.019	0	0.012	0.056	0	0.796	0	0
c	0	0	0.011	0	0	0	0	0.059	0	0.856	0.071
t	0	0	0	0	0	0	0	0.032	0.015	0.060	0.886

Figure 3.3 Confusion matrix with MFCC and AAMF feature tested and trained on the clean dataset

related to clicking events such as clicking of the teeth softly (*cts*), clicking of the teeth loudly (*ctl*) and clicking of the tongue (*cl*). The second group contains events with very subtle sounds coming from the eyes such as blinking forcefully (*bf*), closing the eyes (*ce*) and closing the eyes forcefully (*cef*).

When comparing the two confusion matrices, the AAMF+MFCC features are better than the MFCC features in 4 different events: clicking of the teeth softly (*cts*) with a difference of 8.7% , clicking of the teeth loudly (*ctl*) with a difference of 0.2%, grinding the teeth (*gt*) with a difference of 2.9% and saliva noise (*sn*) with a difference of 5.2%. The 7 other events had better sensitivity when using only the MFCC features.

The increase in the sensitivity for the classes *gt* and *sn* is indicative of the nature of those classes. Specifically, it can be inferred that these classes are better categorized by changes in their slow-varying amplitude envelope.

3.4 Binary Classifier

In order to try to improve the accuracy, the events are separated into two distinct groups depending on if they are better classified using the MFCC or AAMF+MFCC features. The two groups can be seen in table 3.5.

To build the binary classifier, the linear SVM is used with a constant $c = 0.01$. The features used for the binary classifier are AAMF+MFCC. Two different GMMs are trained and tested using the MFCC or the AAMF+MFCC. Depending on the ouput of the binary classifier, the input is sent to one of those two GMMs for their final classification.

Table 3.5 Groups of events for the binary classifier

AAMF+MFCC	**MFCC**
Clicking of the teeth softly (cts)	Closing the eyes (ce)
Clicking of the teeth loudly (ctl)	Closing the eyes forcefully (cef)
Clicking of the tongue (ct)	Blinking forcefully (bf)
Grinding the teeth (gt)	Clearing the throat (clt)
Saliva Noise (sn)	Coughing (c)
	Talking (t)

The average sensitivity of this classifier is 73.7%. The performance of the binary classifier is 93.0% while the average performance between the classifier using AAMF+MFCC and MFCC is 79.3%.

The classifiers has a better performance than the previous results, but due to the fact that the binary classifier is not perfect, the final sensitivity is lower than the 75.5% previously achieved. To obtain a performance higher than 75.5%, the performance of the binary classifier would need to be higher than 97.7% or the performance of the classifiers would need to increase to from, 79.3% to at least 84%. In future work, other features or classifier could be used in order to improve the binary classifier.

3.5 Convolutional Neural Network

Now that more data are available due to both the binaural and noisy dataset the CNN classifier could have a better performance than the previously tested classification algorithm. The CNN is then tested using the architecture described in section 2.5.4.

In table 3.6, the results of the CNN algorithm using different types of augmentation are presented. The results show that the augmentation increases the sensitivity of the classifier and it is even higher when using both binaural and noise augmentation. In section 3.3 when using the GMM classifier, augmenting the data with noise decreases the overall sensitivity but increases the performance when classifying noisy samples. When adding a noisy dataset for the CNN, it increases the overall sensitivity. However, even with the increase in sensitivity from the augmentation, the performance is not good enough to surpass the performance of the GMM classifier. The lack of sensitivity is due to the low number of samples per class in the database since there are on average only 275 samples per class. When using the augmentations, it quadruples the number of samples, however, those samples are not entirely new samples and contain information already present in the original database. The CNN also would require too much processing power to be able to run in real time on a low computational power computer but is still tested in case of increased computing power in the future.

Table 3.6 Performance of the CNN classifier using different methods of augmentation

Augmentation	**Sensitivity**
None	0.581
Binaural	0.584
Noise	0.588
Both	0.591

3.6 Bag of Audio Words and Merging of the teeth clicking events

Since the CNN algorithm did not improve the accuracy, the BoAW algorithm is tested. This algorithm is chosen since it can work by using the current best algorithm and proposes a different way to combine the GMM output of the different frames. By using a histogram created by the output of the GMM of each frame instead of using the "winner takes all" technique, more information is kept to make the final decision but it requires another classifier.

As seen in table 3.7 the BoAW algorithm using the MFCC & AAMF features improves the performance from a sensitivity of 75.5% when using the GMM classifier to a sensitivity of 78.4%, an increase of 2.9% absolute. When using BoAW, the impact of the addition of the AAMF to the feature vector is more evident as there is a significant increase of the sensitivity even when using only the clean dataset.

Table 3.7 Sensitivity of the BoAW classifier when using different features and dataset

	Dataset	
Features Used	**Clean**	**Clean & Noisy**
MFCC	0.772	0.756
MFCC & AAMF	0.784	0.764

Fig. 3.2 shows that the most confused events are clicking of the teeth softly (*cts*) and clicking of the teeth loudly (*ctl*). Further investigation into this discrepancy revealed that the participants' interpretation of these classes is too broad. The main difference between the two classes is the intensity of the impulses. As seen in Fig. 3.4, the RMS value of the signal generated by clicking of the teeth varies greatly. When placing an upper threshold at an RMS value of 0.945, 14% of *cts* are misclassified as *ctl* and 29.1% of *ctl* are misclassified as *cts*. These values match the false discovery rate (rate of misclassification) between those two classes calculated from the confusion matrix found in Fig. 3.2 which are respectively 14.4% $(0.126/(0.744+0.126))$ and 29.8%. Therefore, to resolve participant variability and inconsistency, clicking of the teeth softly (*cts*) and clicking of the teeth loudly (*ctl*) are grouped into one event simply dubbed clicking of the teeth (*ct*).

Table 3.8 shows the difference in sensitivity due to the merge. Without the merge, the sensitivity is 78.4%. When simulating the merge using the same test data, a sensitivity of 80.3% is found. When training and testing after merging both *ctl* and *cts* classes into one named *ct*, the sensitivity becomes 80.2%. While there is an increase in sensitivity due to the merge, it is

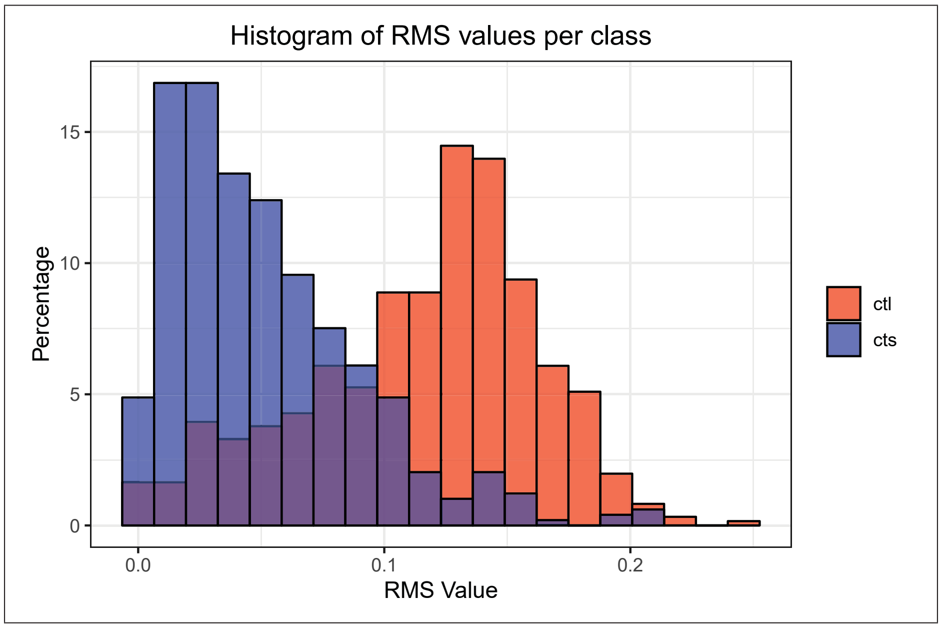


Figure 3.4 Comparison of RMS values of the samples from the cts and ctl classes

only due to the high confusion between those two classes that is now gone. From now on, both classes will be merged into one class named Clicking of the teeth (*ct*)

Table 3.8 Sensitivity due to the merge of the *ctl* and *cts* classes

	Sensitivity
Without merge	0.784
With simulated merge	0.803
With real merge	0.802

3.7 Feature Fusion Techniques and Per Channel Energy Normalisation

To improve the performance of the classifier, the fusion at the histogram level is tested to reduce the dimentionality at the input of the GMM and pass it along to the SVM classifying the histogram. The PCEN feature extraction technique is also tested due to its performance on low-amplitude audio in the literature that could help for some of the low-amplitude classes such as blinking or closing the eyes.

The results of the comparison between the two fusion techniques is presented in Table 3.9. The fusion at the histogram level has a sensitivity of 81.3% and a precision of 82.6% compared to 80.2% and 81.3% respectively for the fusion at the input vector level.

Table 3.9 Comparison of the type of features fusion for the BoAW technique with MFCC and AAMF features

Technique	**Sensitivity**	**Precision**
Fusion at Input Vector Level	80.2	81.3
Fusion at Histogram Level	81.3	82.6

The results of using different features when fusing at the histogram level can be seen in Table 3.10. For this test, four cases are tested: MFCC, MFCC & AAMF, MFCC & PCEN and MFCC & AAMF & PCEN. Fusion without the MFCC features is not tested given that previous attempts already showed poor results. The algorithms used for the clustering and classifying tasks of this test are the GMM and SVM. In order to help compare the performance of the models, the standard deviation of the 10 folds are now added to the tables.

The best results are achieved using the MFCC & PCEN features with a sensitivity of 81.5% with a standard deviation of 4.4 and the worst performance is achieved using only the MFCC features at 78.4% sensitivity with a standard deviation of 5.4. The precision follows the same trend as the sensitivity. Their percentages are presented in Table 3.10.

Table 3.10 Comparison of the features used with the BoAW algorithm with the histogram level fusion with the GMM & SVM algorithms

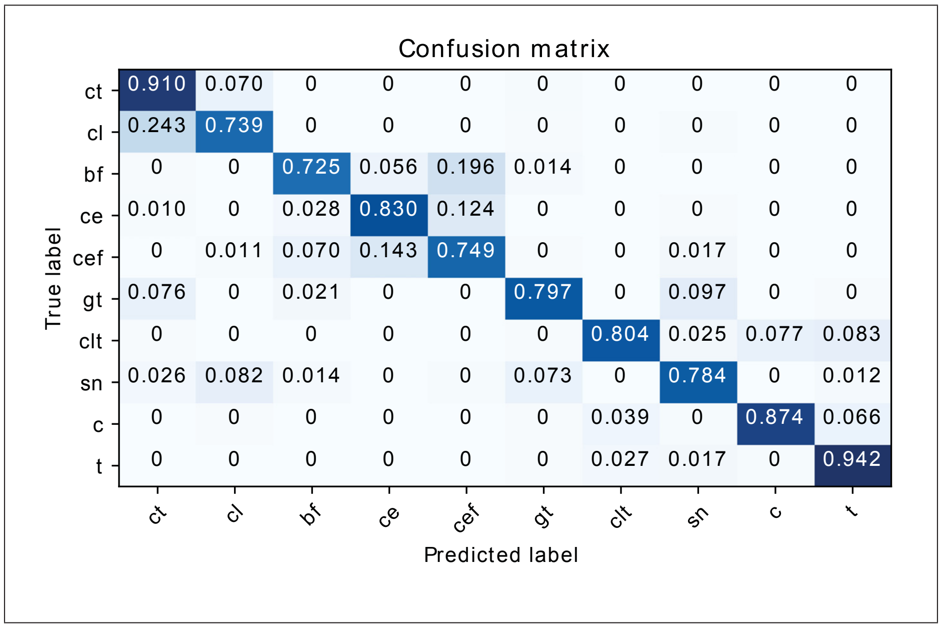
Features used	**Sensitivity**	**Precision**
MFCC	78.4 ± 5.4	80.1 ± 5.1
MFCC AAMF	81.3 ± 5.2	82.6 ± 4.8
MFCC PCEN	81.5 ± 4.4	83.0 ± 4.6
MFCC AAMF PCEN	80.7 ± 4.5	82.0 ± 4.6

In table 3.11, the results of the comparison between the clustering and classifying algorithm can be seen. Using the K-Means algorithm for the clustering method decreases the accuracy to 78% when using the SVM classifier and 78.8% when using the Random Forest classifier. When using the GMM clustering algorithm, both the SVM and Random Forest classifiers achieve a sensitivity of 81.5%. However, the SVM has a standard deviation of 4.4 while the Random forest had a standard deviation of 5.5.

Table 3.11 Comparison of the BOAW clustering and classification method with the MFCC and PCEN features

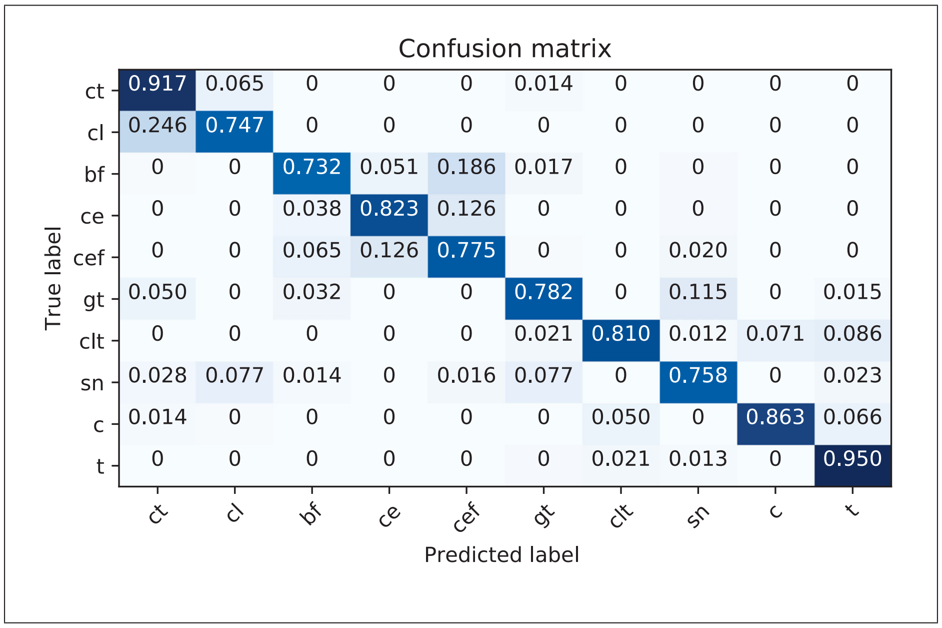
Clustering	**Classification**	**Sensitivity**	**Precision**
GMM	SVM	81.5 ± 4.4	83.0 ± 4.6
GMM	Random Forest	81.5 ± 5.5	82.8 ± 4.9
K-Means	SVM	78.0 ± 4.6	79.1 ± 4.7
K-Means	Random Forest	78.8 ± 5.2	79.5 ± 4.6

In Fig. 3.5 and 3.6, the confusion matrices of the best results using the SVM and Random Forest classifying algorithms are presented. Both classifiers achieve an average sensitivity of 81.5% but the sensitivity of certain classes such as *cef* and *sn* varied significantly.



True label \ Predicted label	ct	cl	bf	ce	cef	gt	clt	sn	c	t
ct	0.910	0.070	0	0	0	0	0	0	0	0
cl	0.243	0.739	0	0	0	0	0	0	0	0
bf	0	0	0.725	0.056	0.196	0.014	0	0	0	0
ce	0.010	0	0.028	0.830	0.124	0	0	0	0	0
cef	0	0.011	0.070	0.143	0.749	0	0	0.017	0	0
gt	0.076	0	0.021	0	0	0.797	0	0.097	0	0
clt	0	0	0	0	0	0	0.804	0.025	0.077	0.083
sn	0.026	0.082	0.014	0	0	0.073	0	0.784	0	0.012
c	0	0	0	0	0	0	0.039	0	0.874	0.066
t	0	0	0	0	0	0	0.027	0.017	0	0.942

Figure 3.5 Confusion matrix of the best results with 81.5% performance over all the classes with the SVM classifying algorithm



True label \ Predicted label	ct	cl	bf	ce	cef	gt	clt	sn	c	t
ct	0.917	0.065	0	0	0	0.014	0	0	0	0
cl	0.246	0.747	0	0	0	0	0	0	0	0
bf	0	0	0.732	0.051	0.186	0.017	0	0	0	0
ce	0	0	0.038	0.823	0.126	0	0	0	0	0
cef	0	0	0.065	0.126	0.775	0	0	0.020	0	0
gt	0.050	0	0.032	0	0	0.782	0	0.115	0	0.015
clt	0	0	0	0	0	0.021	0.810	0.012	0.071	0.086
sn	0.028	0.077	0.014	0	0.016	0.077	0	0.758	0	0.023
c	0.014	0	0	0	0	0	0.050	0	0.863	0.066
t	0	0	0	0	0	0	0.021	0.013	0	0.950

Figure 3.6 Confusion matrix of the best results with 81.5% performance over all the classes with the Random Forest classifying algorithm

3.8 Discussion

After experimenting on 5 different classifiers, using three different types of features and trying two types of data augmentation, the best classifier, with the best sensitivity and accuracy, is created using the BoAW algorithm with a GMM clustering method followed by a SVM classifier. MFCC and PCEN are used as the input and both binary and noisy augmentation are used to improve the performance both in clean and noisy scenarios.

When comparing the BoAW results, it is clear that the fusion of different features is better when done at the histogram level than at the input level. One of the reasons for the increase in sensitivity and precision might be that the SVM algorithm is better at handling higher dimensionality input than the GMM algorithm. This is also shown in Fig. 3.1 where only the GMM algorithm had a lower performance when using the larger input vector. When reducing the dimensionality at the GMM level, by having separate clustering algorithms for each feature, their outputs are concatenated into one larger histogram. The SVM might lose a bit of sensitivity because of the increase in dimensionality, but the increase in sensitivity due to the lower dimensionality at the GMM level is greater and gives an overall higher performance in the end.

When looking at the performance of the various features used, an increase in both sensitivity and precision by adding a new feature to the MFCC is present. Due to the increase in sensitivity, it can be said that the information contained in the PCEN and AAMF features are complementary to the MFCC features when trying to classify the nonverbal events of interest. However, when combining all the features, there is a decrease in sensitivity and precision compared to using only two features. The information added by the PCEN and AAMF features might be redundant and the increase in performance due to the new information might not be sufficient to counteract the loss of performance caused by the increase in dimensionality.

When comparing the clustering algorithms, the GMM is superior to the K-Means regardless of the classification algorithm used. The Random Forest performs better than the SVM when used with the K-Means algorithm. When using the GMM clustering technique, the Random Forest algorithm has the same sensitivity as the SVM, 81.5%, but a slightly higher precision. When comparing their confusion matrices, the only classes with a difference in sensitivity of more than 1% are closing the eyes forcefully (*cef*), grinding the teeth (*gt*), saliva noise (*sn*) and coughing (*c*), *cef* being the only class that has better sensitivity with the Random Forest algorithm.

When using a 10-fold cross validation and having separate users in separate folds, it is hard to be certain that one model is statistically better than the other. Standard deviation is used in the final classifier results to help compare the models. However, according to Nadeau & Bengio (2003), cross-validaiton techniques can overestimate or underestimate the variance or standard deviation of the models and lead to the wrong conclusion. For the classifier, the BOAW with GMM and SVM algorithm using the MFCC and PCEN features has the best performance, but the other models with similar but lower performance cannot, with certainty, be seen as inferior.

To help increase the performance further, new features that properly characterize the data must be found. Deep learning could be useful to find discriminative features from the spectrogram but due to the small number of samples contained in the database, it is not currently possible to achieve good results with deep learning. The database could be made larger by recording more samples with new participants since only 24 participants are in the original database. To make the classifier more robust to real noise, the new samples could be recorded outside of the audiometric booth. These recordings would contain noises either created by the participant or by their environment. With such recordings, the creation of a new class containing unknown or other noises could be created since not all noises can be classified in a real-world environment. It is not possible to do that for this classifier because the database only contains the nonverbal events.

3.9 Conclusion

This chapter presented the results of the main experiments that have been conducted for this work. Based on these results, the final system is presented in the next chapter. It is the one that has been chosen to be implemented on the device as it has the best performance regarding sensitivity and accuracy of classification. Results from a simulation of a real-world environment are presented and discussed.

CHAPTER 4

FINAL SYSTEM

The final goal of this project is to build a real-time audio event detector using the best classifier found in section 3.

4.1 Description of the Detection System

The in-ear devices are connected to the Auditory Research Platform (ARP) in order to record the signal from the IEM. The ARP used is built using the Yuntab B01 Mini Desktop PC (Schenzhen Wave Multimedia Co.,Ltd, Shenzhen, China). This mini computer contains 2 GB of RAM, and an Atom Z3735F CPU at 1.33 GHz. The microphones are interfaced with the ARP using the CY8CKIT-059 DSP (Cypress Semiconductor Corp, San Jose, USA).

The best performance found during the classification study as discussed in Chapter 3 is used for the real-time detection. Therefore, the BoAW detector is used with the MFCC and PCEN features. The fusion of the two features is done at the histogram level and the clustering and classification algorithm used in the BoAW are the GMM and SVM. The GMM contains 15 Gaussians per class and the SVM has a constant $C = 0.01$.

The detector is programmed in Python on the ARP since the classifier used is already programmed in Python. The BoAW is trained on a desktop using the binaural and noisy database. The resulting GMM and SVM parameters are exported to the ARP to be used in the detector.

The proposed BoAW classification algorithm is implemented to detect and classify the non-verbal events in a continuous audio stream in real-time. The detection by classification technique is used to detect the events of interest. This technique continuously classifies the input audio stream to detect the occurrence of an event. To do so, frames of audio data of 100 ms are recorded and concatenated with the previous chunk to create a complete 400 ms sample. Each new frame of 100 ms is concatenated at the end of the sample and the oldest 100 ms is removed to keep the sample to a constant length of 400 ms, creating a first-in first-out buffer.

This sample is then classified and compared to the three previous samples. If two samples or more out of the four are of the same class, then they are considered part of this class and a detection occurs. Otherwise, no audio event is detected. Due to the low computational power of the device used, only the channel with the better-fitted earpiece is used for classification since the noise isolation is greater and the occlusion effect is more accentuated, increasing the SNR of the audio event. It takes 96 ms for the ARP to classify a 400 ms audio buffer using this BoAW detection algorithm which makes it functional in real-time since a sliding window of 100 ms is used. The speed of the algorithm could be improved by programming the detector in a more efficient language such as C but for this prototype, a Python script is sufficient.

4.2 Testing of the Detection System

In order to test the detection system, the detection algorithm is tested on audio captured from a device similar to the one depicted in Fig. 2.1 worn by participants in various noise environments. Ten participants are asked to produce the events of clicking of the teeth (ct), clicking of the tongue (cl) , blinking forcefully (bf), grinding of the teeth (gt) and clearing the throat (clt) 5 times each and the events closing the eyes (ce), closing the eyes forcefully (cef), saliva noise (sn) and talking (t) for 5 seconds each. The recording took place in an audiometric booth where the participants are asked to repeat the test 3 times under three different noise conditions. In the first condition, the participants performed the events in a quiet environment. For the second and third conditions, the participants are exposed to 70 dBA of factory noise and babble noise consecutively from the NOISEX-92 database (Varga & Steeneken, 1993). The two noisy environments are chosen to assess the performance of the detector in two of the potential use cases of this technology. The first relates to industrial workers in a noisy setting and the second one to the consumer market, such as a cocktail party setting. Before the recording took place the fit of the in-ear device is verified to ensure that the intra-aural device is well-fitted creating an acoustical seal, inducing the occlusion effect, and attenuating the ambient noise.

For the tests where the recording are made in a noisy environment, it is beneficial to denoise the IEM to reduce the degradation by noise on the captured signals. An adaptive filtering technique developed for intra-aural devices similar to that in Fig. 2.1 is used, since it has shown good results when denoising IEM speech using the relationship between the OEM and the IEM (Bouserhal *et al.*, 2017).

It is important to note that the IEM used for the test differed from the one used to record the training data. The IEM used for the test has a different frequency response containing a gain of up to 24 dB at 3 kHz, which the IEM used for the training did not have. This caused a decrease in the performance of the detector. To improve the performance, a notch filter at 3 kHz is used on the incoming audio to reduce the effect of the frequency boost caused by the microphone.

The detection algorithm is applied to the audio tracks and the results are manually compared to the events produced in the audio and compiled into a confusion matrix. An unknown class (denoted by *?*) is also created to represent the occurrences in which the detector does not find two of the same class in the last 3 classifications.

4.3 Results of the Detection System

In Table 4.1 the sensitivity (SN) and precision (PREC) can be seen for every class in the different noise conditions. The average sensitivity over all the classes is 69.9% in quiet. It decreased to 63.5% and 55.6% for the factory and babble environments respectively. For the precision, the average for each class are 78.7%, 72.9% and 64.2% for silent, factory and babble noise respectively.

In Fig. 4.1, the confusion matrix containing the results of the performance of the detector combining every noisy environment.

4.4 Discussion

When in a quiet environment, the performance in terms of sensitivity of the detector compared to the classifier decreases from 81.5% to 69.9%. The difference in performance might be due to the fact that detection is a harder task than classification. Another problem might be that the

Table 4.1 Sensitivity and Precision for each classes under Silent, Factory and Babble environments

	Quiet		Factory		Babble	
Events	**SN**	**PREC**	**SN**	**PREC**	**SN**	**PREC**
ct	0.767	0.667	0.480	0.774	0.196	0.556
cl	0.833	0.641	0.612	0.556	0.686	0.479
bf	0.386	1	0.160	0.800	0.220	1
ce	0.434	0.657	0.348	0.889	0.240	0.75
cef	0.774	0.414	0.88	0.396	0.660	0.407
gt	0.182	1	0.422	0.679	0.212	0.647
clt	0.764	1	0.700	0.946	0.660	0.440
sn	0.911	0.708	0.900	0.556	0.940	0.528
c	0.926	1	0.843	0.977	0.750	1
t	1	0.786	1	0.718	1	0.613
AVG	0.699	0.787	0.635	0.729	0.556	0.642

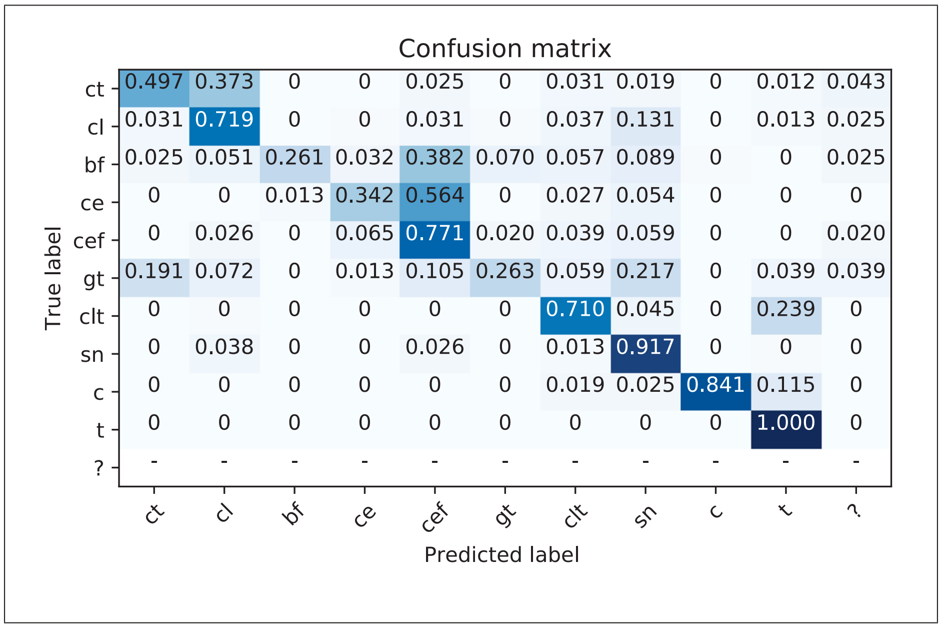


Figure 4.1 Confusion matrix of the real-time detection algorithm

IEM in the detection tests used is different than the ones used to build the training database and they have slightly different frequency responses despite the compensation filters used. For some classes such as blinking forcefully (*bf*), closing the eyes (*ce*) and grinding the teeth (*gt*), the sensitivity decreased significantly, a difference of more than 40%. These are the classes whose audio signals had the lowest energy. Therefore, small disturbances due to background noise or due to the different frequency response of the IEM had more impact on their detection.

The clicking of the teeth (*ct*) class is often classified as clicking of the tongue (*cl*) since both are impulsive sounds, which is why (*cl*) has a good sensitivity but worse precision. The same thing happened between the classes blinking forcefully (*bf*), closing the eyes (*ce*) and closing the eyes forcefully (*cef*) with the first two being confused with the latter, which gave *cef* a precision of around 40% in every noise condition. The considerable decrease in performance between the factory environment and the babble environment might be due to the frequency content of the babble sounds being closer to that of the events than to the factory sounds. The denoising algorithm might also have more difficulty with babble noise since it is built for denoising speech and might not perform as well with speech-like noise. When doing the data augmentation, only factory noise is added, which can be the reason that the algorithm has more difficuly with the babble noise compared to the facory noise. The denoising algorithm itself might have affected the integrity of the audio signal, decreasing the performance of the detector.

To improve the detector, a bigger database should be built containing audio from various IEMs so the detector does not overfit on the frequency response of the microphone. Other features could also be found to better represent the various nonverbal events present in the database.

CONCLUSION

The main goal of this thesis was to build an algorithm to identify nonverbal audio events. This was achieved by building a classification algorithm using the BoAW technique with the MFCC and PCEN features. This classifier had a sensitivity of 81.5% with a precision of 83%. A detection algorithm was also built using the classification algorithm and was able to run in real-time on a low-power device. The detection algorithm was then validated on 10 participants, who are asked to produce the 10 events that are classified in the context of this project. The real-time implementation of the detector achieved a sensitivity of 69.9% with a precision of 78.7% in a quiet environment.

As a result of this project, a nonverbal-audio-event detector has been developed. The code of the detector has been made available to CRITIAS and its industrial partner EERS Global Technologies. A conference paper has been written and a poster presentation has been made for the Interspeech 2018 conference. A journal paper has been accepted to Applied Acoustics. Finally, a patent application, entitled "In-ear nonverbal audio events classification system and method", was submitted to the USPTO on October 29th, 2018 as mentioned in VOIX *et al.* (2019).

The outcome of this project is that the detector developed could be implemented into several embedded systems, such as audio wearables (or "hearables") and used in many different fields of application.

In particular, in the industry, the developed algorithm could benefit occupational health and safety purposes:

- The system could be used in worker remote health monitoring systems to monitor events such as coughing, grinding of the teeth or saliva noise, that can be early indicators of medical conditions. Health problems could be detected by looking for an increased amount of coughing that could be due to a chemical leakage or to problems in the ventilation system.

- Voluntary events such as the clicking of the teeth or of the tongue could be detected and provide a handsfree, thus safer and easier, interface with a machine or computer through a digital hearing protector.
- Finally, the proposed algorithm makes it possible to remove WIDs from the measurement of the noise dose. This makes the worker's noise exposure estimation more accurate by preventing loud events produced by the worker and thus not dangerous for the human ear from being included in the estimation.

This project is also beneficial for society in general. The technology developed could help disabled people to interface easily with their devices without the need to touch a screen. It could also help monitor the health of seniors in real-time from the comfort of their home. A health professional could have access to the information of the patient as to the occurrence of certain events.

5.1 Recommendations and Future Work

Despite the good performance of the detector, some improvements could be done to both the detection algorithm and the database of non verbal audio events.

As for the detection algorithm, different techniques of detection other than a detection by classification could be implemented and experimented on. For instance, the detection could be done by detecting sounds that are captured by the IEM but not present in the outer microphone signal and classifying those events. This detection algorithm would require less computing power than the continuous classification used in this project for detection.

As for the database, the following improvements could be made:

- More audio events recordings could be added to the database, such as sneezing, someone touching his own face or someone touching the side of the earbud, etc. The sneezing event could be a good indicator of health. A worker touching his face produces noise inside the occluded ear and is a type of WID that could be removed from the noise-dose calculation. A user touching his earbud could act as an input replacing the need for physical buttons, thereby offering even more options for new types of human-machine interfaces.

- The nonverbal audio event database could be made larger by recording more samples of the events using different microphones. In order to use deep learning or simply to increase the performance of the BoAW, the larger the database, the better. The database used only contained 11 events recorded from 25 participants.

- Nonverbal audio events could be recorded also outside of the -very silent- audiometric booth. Such recordings would hence contain noise such as the participant moving and walking and would help in the fine-tuning of the algorithm to further increase its immunity to everyday ambient noises and real-world conditions of use.

- Another class could be created containing all unknown or other noises that are not detected so that these noises do not trigger false positives. This could not be done since the database used only contains the nonverbal events in this project.

- The nonverbal audio event database could be populated with audio samples recorded using the exact same device on which the detector will be implemented. This would ensure that the frequency response of the microphones used for the training and later used for the prototype would be the same and avoid any discrepancies in their frequency responses or electroacoustic properties.

APPENDIX I

CODE FOR TRAINING OF BOAW CLASSIFIER

```
import numpy as np
from sklearn.mixture import GMM
import pickle
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
from sklearn.metrics import confusion_matrix
import h5py
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC



def Weighted_Accuracy(training,result, nb_c):
    average = 0
    for i in range(nb_c):
        indexes = np.where(training == i)[0]
        average += np.mean(result[indexes] == i)
    return average/nb_c

def Result_From_Frame_Most(array, nb_f, nb_c):

    result = []

    for i in range(len(array)//nb_f):
        sub_array = array[i*nb_f:(i+1)*nb_f]
        best_class = 0
        best_nb = 0
        for n in range(nb_c):
            nb = 0
            for m in sub_array:
                if m == n:
                    nb+=1

            if nb > best_nb:
                best_nb = nb
                best_class = n

        result.append(best_class)

    return np.array(result)

def Contextualize(array, nb_context, nb_frame, len_ceps):

    new_array = np.empty((0,len_ceps))

    for i in range(nb_frame):

        new_line = np.empty(0)

        if i < nb_context:

            for n in range(nb_context):
                if n < nb_context-i:
                    new_line =  np.hstack([new_line,array[0]])
                else:
                    new_line =  np.hstack([new_line,array[n-(⌋
                     ↪  nb_context-i)]])

            for n in range(i,i+nb_context+1):
                new_line =  np.hstack([new_line,array[n]])

        elif (i > nb_frame-(nb_context+1)):

            for n in range(i-nb_context,i+1):
                new_line =  np.hstack([new_line,array[n]])

            for n in range(1,nb_context+1):
                if i+n < nb_frame-1:
                    new_line =
                     ↪  np.hstack([new_line,array[i+n]])
                else:
                    new_line =  np.hstack([new_line,array[nb_⌋
                     ↪  frame-1]])

        else:

            for n in range(i-nb_context,i+nb_context+1):
                new_line =  np.hstack([new_line,array[n]])


        new_array = np.vstack([new_array,new_line])

    return new_array
```

```
def read_h5_milton(read_file):

    f = h5py.File(read_file, 'r')
    key = list(f.keys())

    data = np.array(f[key[0]])
    data = data.transpose()



    return data

def read_sample(read_file,aamf_path):



    mfcc = np.load(read_file)
    mfcc = np.repeat(mfcc,2,0)

    aamf = np.load(aamf_path.replace("h5","npy")).transpose()

    return (mfcc,aamf)

def Result_Hist(array, nb_f, size):

    result = []

    for i in range(len(array)//nb_f):
        hist = [0]*size
        sub_array = array[i*nb_f:(i+1)*nb_f]
        for m in sub_array:
            hist[m] +=1

        result.append(hist)

    return np.array(result)

def Result_Hist_proba(array, nb_f, size):

    result = []

    for i in range(len(array)//nb_f):
        hist = [0]*size
        sub_array = array[i*nb_f:(i+1)*nb_f]


        for m in sub_array:
            hist += m



        result.append(hist)

    return np.array(result)

GetData = True
if GetData:
```

```
info_path = "/media/pchabot/XCELLON-02/Stage_Chabot_⌋
  ↪ Philippe/Signaux_non_vocaux/labeled_data/_tags_⌋
  ↪ nb2.txt"
samples_path =
  ↪ "/media/pchabot/XCELLON-02/Masters/FRAME_LR_DS_50-50/"
samples_path_noisy =
  ↪ ["/media/pchabot/XCELLON-02/Masters/FRAME_LR_DS_10DB_⌋
  ↪ F_50-50/"]#["/media/pchabot/XCELLON-02/Masters/FRAME_⌋
  ↪ LR_DS_10DB_F_⌋
  ↪ 50-50/"]#,"/media/pchabot/XCELLON-02/Masters/FRAME_⌋
  ↪ Factory_10_2400-1200/"]

aamf_path =
  ↪ "/media/pchabot/XCELLON-02/Masters/FRAME_LR_PCEN/"
aamf_path_noisy =
  ↪ '/media/pchabot/XCELLON-02/Masters/FRAME_LR_PCEN_⌋
  ↪ 10DB/'

samples_list = []
class_list = []
samples_nb = []
sample_name = ""
nb_context = [4,4 ]

#On lis le fichier nous disant combien il y a de samples
  ↪ de chaque sons
info_file = open(info_path,"r")

for line in info_file:
    [class_name,nb] = line[:-1].split('-')

    class_list.append(class_name)
    samples_nb.append(int(nb))


sample_name = class_list[0]
ceps = read_sample(samples_path+sample_name+"_
 ↪ L.npy",aamf_path+sample_name+"_L.h5")
nb_batch = 10
nb_classes = len(class_list)
nb_frame = []
len_ceps = []
X_batches = []
Y_batches = []
X_batches_n = []
Y_batches_n = []

for f in range(len(ceps)):
    nb_frame.append(len(ceps[f]))
    len_ceps.append(len(ceps[f][0])*(nb_context[f]*2+1))

    X_batch = [np.empty((0,len_ceps[f]), float),]*nb_batch

    Y_batch = [np.empty((0,1), float),]*nb_batch

    X_batch_n = [np.empty((0,len_ceps[f]),
     ↪ float),]*nb_batch
```

```
        Y_batch_n = [np.empty((0,1), float),]*nb_batch

        resultscsv = np.empty((0,3),)

        class_test = []

        X_batches.append(X_batch)
        Y_batches.append(Y_batch)
        X_batches_n.append(X_batch_n)
        Y_batches_n.append(Y_batch_n)

        #On lis chaque sample et ot les mets dans deux
          ↪ tableau, un pour le training, un pour le test.
    for i in range(nb_classes):

        index = 0
        nb_vals = samples_nb[i]//nb_batch
        over_vals = samples_nb[i]%nb_batch
        if i == 0:
            cls_nb = 0
        elif i == 1:
            cls_nb = 0
        else:
            cls_nb = i-1

        for n in range(nb_batch):

            nb_val_batch = nb_vals
            if n < over_vals:
```

```
        nb_val_batch += 1

    for m in range(nb_val_batch):

        sample_name = class_list[i]
        if index>0:
            sample_name+="-"+str(index+1)

        for suffix in ["_L","_R"]:

            ceps = read_sample(samples_path+sample_
             ↪  name+suffix+".npy",aamf_path+sample_
             ↪  name+suffix+".h5")
            for f in range(len(ceps)):
                ceps_con = Contextualize(ceps[f],nb_
                 ↪  context[f],nb_frame[f],len_
                 ↪  ceps[f])



                X_batches[f][n] =
                 ↪  np.vstack([X_batches[f][n],
                 ↪  ceps_con])
                Y_batches[f][n] =
                 ↪  np.vstack([Y_batches[f][n],
                 ↪  np.full((nb_frame[f], 1),
                 ↪  cls_nb)])

            for path in samples_path_noisy:
```

```
                        ceps = read_sample(path+sample_⌋
                         ↪ name+suffix+".npy",aamf_path_⌋
                         ↪ noisy+sample_name+suffix+".h5")
                        for f in range(len(ceps)):
                            ceps_con = Contextualize(⌋
                             ↪ ceps[f],nb_context[f],nb_⌋
                             ↪ frame[f],len_ceps[f])

                            X_batches_n[f][n] =
                             ↪ np.vstack([X_batches_n[f][n],
                             ↪ ceps_con])
                            Y_batches_n[f][n] =
                             ↪ np.vstack([Y_batches_n[f][n],
                             ↪ np.full((nb_frame[f], 1),
                             ↪ cls_nb)])

                index += 1
    print("Finished getting data")

nb_classes = 10
nb_gaussian = [15,5]
c = 0.01
avg = 0
cm = np.zeros([nb_classes,nb_classes])
acc_batch_R = []
for n in range(nb_batch):

    for f in range(len(nb_gaussian)):
```

```
X_train = np.empty((0,len_ceps[f]), float)
X_test = np.empty((0,len_ceps[f]), float)

Y_train = np.empty((0), int)
Y_test = np.empty((0), int)

for m in range(nb_batch):

    if m == n :
        X_test = np.vstack([X_test, X_batches[f][m]])
        Y_test = np.append(Y_test, Y_batches[f][m])
        X_test = np.vstack([X_test,
         ↪ X_batches_n[f][m]])
        Y_test = np.append(Y_test, Y_batches_n[f][m])
    else:
        X_train = np.vstack([X_train,
         ↪ X_batches[f][m]])
        Y_train = np.append(Y_train, Y_batches[f][m])
        X_train = np.vstack([X_train,
         ↪ X_batches_n[f][m]])

covar = 'diag'

for i in range(0,nb_classes):
    data = X_train[np.where(Y_train == i)]
    clf = GMM(n_components=nb_gaussian[f],
     ↪ covariance_type=covar, n_iter=50,)

    clf.fit(data)
```

```
        if i == 0:

            means = np.array(clf.means_)
            weights = np.array(clf.weights_)
            covars = np.array(clf.covars_)

        else:

            means = np.vstack([means, clf.means_])
            weights = np.hstack([weights, clf.weights_])
            covars = np.vstack([covars, clf.covars_])

    clf = GMM(n_components=nb_gaussian[f]*nb_classes,
      ↪ covariance_type=covar, init_params='',
      ↪ n_iter=200,)
    clf.means_ = means
    clf.weights_ = weights/nb_classes
    clf.covars_ = covars

    res = clf.predict(X_train)

    gmm_test = clf.predict(X_test)
    if f == 0:
        hist_train = Result_Hist(res,nb_frame[f],
          ↪ nb_classes*nb_gaussian[f])
        hist_test = Result_Hist(gmm_test,nb_frame[f],
          ↪ nb_classes*nb_gaussian[f])
    else:
```

```
                hist_train = np.hstack([hist_train,
                  ↪  Result_Hist(res,nb_frame[f],
                  ↪  nb_classes*nb_gaussian[f])])
                hist_test = np.hstack([hist_test,
                  ↪  Result_Hist(gmm_test,nb_frame[f],
                  ↪  nb_classes*nb_gaussian[f])])

        svm = OneVsRestClassifier(SVC(C = c, kernel = 'poly',
          ↪  degree = 2))

        Y_train_SVM = Result_From_Frame_Most(Y_train,nb_frame[f],
          ↪  nb_classes)
        svm.fit(hist_train,Y_train_SVM)

        result_test = svm.predict(hist_test)
        Y_test = Result_From_Frame_Most(Y_test,nb_frame[f],
          ↪  nb_classes)
        accuracy =
          ↪  Weighted_Accuracy(Y_test,result_test,nb_classes)

        cm+=confusion_matrix(Y_test,result_test)

        avg += accuracy
        print(accuracy)
    print("Average")
    print(avg/nb_batch)
```

APPENDIX II

CODE FOR DETECTION ALGORITHM

```
def detect(audio_array):

    #On specifie le sample rate, le sample length et le window
     ↪ shift length
    fs = 8000

    with open('clustererMFCC_N.plk', 'rb') as fin:
      GMM_MFCC = pickle.load(fin)

    with open('clustererPCEN_N.plk', 'rb') as fin:
      GMM_PCEN = pickle.load(fin)

    with open('classifier_SVM_N.plk', 'rb') as fin:
      SVM = pickle.load(fin)

    data = []

    sample = []

    #Create new sample and shift len  while taking in account
     ↪ the downsampling

    sample_len = int(fs*0.4) #longur d'un sample de 0.4s
    shift_len = int(sample_len/4)
    frame_size = int(0.05*fs)
    frame_shift = int(0.025*fs)
```

```
nb_context_mfcc = 4
nb_context_pcen = 4

nb_gmm_pcen = 5
nb_gmm_mfcc = 15

nb_classes = 10

flag = 1

mfcc =
 ↪  MFCC_HTK(filter_file="filter.csv",win_len=frame_size,
 ↪  win_shift=frame_shift,samp_freq=8000)

classif_res = []
classify = 1
frst = 1

prev_res = [-1,-1,-1,-1]
nb_sep = int(len(audio_array)/shift_len)

result = []
for i in range(nb_sep):
    data = audio_array[i*shift_len:(i+1)*shift_len]

    #To adapt the sensitivity of this microphone compared
     ↪  to the one used for training
    data = data/6
```

```
if classify == 1:
    if len(sample) == 0:
        sample = data

    else:

        sample = np.hstack((sample[-(sample_↩
         ↪ len-shift_len):],data))

        if len(sample) == sample_len:

            sig = sample

            pcen_sig =
             ↪ extract_pcen(sig,fs).transpose()
            mfcc_sig = extract_mfcc(sig.astype(↩
             ↪ float),mfcc,frame_size,frame_shift)

            if flag:
                flag = 0
                nb_frame_pcen = len(pcen_sig)
                nb_frame_mfcc = len(mfcc_sig)
                len_frame_pcen = len(pcen_sig[0])*(↩
                 ↪ nb_context_pcen*2+1)
                len_frame_mfcc = len(mfcc_sig[0])*(↩
                 ↪ nb_context_mfcc*2+1)
```

```
pcen_sig_contex =
 ↪  Contextualize(pcen_sig,nb_context_⌋
 ↪  pcen,nb_frame_pcen,len_frame_pcen)
mfcc_sig_contex =
 ↪  Contextualize(mfcc_sig,nb_context_⌋
 ↪  mfcc,nb_frame_mfcc,len_frame_mfcc)

res_pcen =
 ↪  GMM_PCEN.predict(pcen_sig_contex)
res_mfcc =
 ↪  GMM_MFCC.predict(mfcc_sig_contex)

hist_pcen = Result_Hist(res_pcen,
 ↪  nb_frame_pcen,nb_classes*nb_gmm_pcen)
hist_mfcc = Result_Hist(res_mfcc,
 ↪  nb_frame_mfcc,nb_classes*nb_gmm_mfcc)

hist = np.hstack([hist_mfcc,hist_pcen])

res = SVM.predict(hist)
res = res[0]
result.append(res)

result_LR = res

prev_res.pop(0)
prev_res.append(result_LR)

if prev_res.count(result_LR) >= 2:
```

```
                        if frst == 1:
                            classif_res.pop()
                            if len(classif_res) > 1:
                                classif_res.pop()
                                classif_res.append(result_LR)
                            classif_res.append(result_LR)
                            frst = 0
#                        print(class_list[result_LR])
                        classif_res.append(result_LR)

                    else:
                        classif_res.append(-1)
                        frst = 1

        else:
            if len(sample) == 0:
                sample = data
            else:

                sample = np.vstack((sample,data))

    return [classif_res[0]]*3+classif_res
```

BIBLIOGRAPHY

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. & Zheng, X. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. Software available from tensorflow.org, Consulted at http://tensorflow.org/.

Berger, E. H. (2003). *The Noise Manual*. AIHA.

Bouserhal, R. E., Falk, T. H. & Voix, J. (2015). On the potential for artificial bandwidth extension of bone and tissue conducted speech: A mutual information study. *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5108–5112.

Bouserhal, R. E., Falk, T. H. & Voix, J. (2017). In-ear microphone speech quality enhancement via adaptive filtering and artificial bandwidth extension. *The Journal of the Acoustical Society of America*, 141(3), 1321–1331.

Brummund, M. K., Sgard, F., Petit, Y. & Laville, F. (2014). Three-dimensional finite element modeling of the human external ear: Simulation study of the bone conduction occlusion effect. *The Journal of the Acoustical Society of America*, 135(3), 1433–1444. doi: 10.1121/1.4864484.

Cuevas, J. L., Cook, E. W., Richter, J. E., McCutcheon, M. & Taub, E. (1995). Spontaneous swallowing rate and emotional state: Possible mechanism for stress-related gastrointestinal disorders. *Digestive Diseases and Sciences*, 40(2), 282–286. doi: 10.1007/BF02065410.

Geiger, J. T. & Helwani, K. (2015, 8). Improving event detection for audio surveillance using Gabor filterbank features. *2015 23rd European Signal Processing Conference (EUSIPCO)*, pp. 714–718. doi: 10.1109/EUSIPCO.2015.7362476.

Hecht-nielsen, R. (1992). III.3 - Theory of the Backpropagation Neural Network**Based on “nonindent” by Robert Hecht-Nielsen, which appeared in Proceedings of the International Joint Conference on Neural Networks 1, 593–611, June 1989. © 1989 IEEE. In Wechsler, H. (Ed.), *Neural Networks for Perception* (pp. 65–93). Academic Press. doi: 10.1016/B978-0-12-741252-8.50010-8.

Jolliffe, I. (2011). Principal Component Analysis. In Lovric, M. (Ed.), *International Encyclopedia of Statistical Science* (pp. 1094–1096). Berlin, Heidelberg: Springer Berlin Heidelberg. doi: 10.1007/978-3-642-04898-2_455.

Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C. D., Silverman, R. & Wu, A. Y. (2002). An efficient k-means clustering algorithm: analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7), 881–892. doi: 10.1109/TPAMI.2002.1017616.

Kingma, D. P. & Ba, J. (2014). Adam: A Method for Stochastic Optimization. Consulted at http://arxiv.org/abs/1412.6980.

Koržinek, D. (2019). HTK features in Python. original-date: 2015-12-18T12:00:42Z, Consulted at https://github.com/danijel3/PyHTK.

Li, C. (2014). Fisher Linear Discriminant Analysis.

MarketResearch.biz. (2018). Global Hearable Devices Market Revenue is Projected to be Over US$ 85 Billion by 2021. Consulted at https://www.prnewswire.com/news-releases/global-hearable-devices-market-revenue-is-projected-to-be-over-us-85-billion-by-2021-685929661.html.

Martin, A. & Voix, J. (2017). In-Ear Audio Wearable: Measurement of Heart and Breathing Rates for Health and Safety Monitoring. *IEEE Transactions on Biomedical Engineering*, 1–1. doi: 10.1109/TBME.2017.2720463.

Mazur, K. & Voix, J. (2013). Implementing 24-hour in-ear dosimetry with recovery. *Proceedings of Meetings on Acoustics ICA2013*, 19(1), 040016.

McFee, B., Raffel, C., Liang, D., Ellis, D. P. W., McVicar, M., Battenberg, E. & Nieto, O. (2015). librosa: Audio and Music Signal Analysis in Python. doi: 10.25080/majora-7b98e3ed-003.

McLoughlin, I., Zhang, H., Xie, Z., Song, Y., Xiao, W. & Phan, H. (2017). Continuous robust sound event classification using time-frequency features and deep learning. *PLOS ONE*, 12(9), e0182309. doi: 10.1371/journal.pone.0182309.

Mika, S., Ratsch, G., Weston, J., Scholkopf, B. & Mullers, K.-R. (1999). Fisher discriminant analysis with kernels. *Neural networks for signal processing IX: Proceedings of the 1999 IEEE signal processing society workshop (cat. no. 98th8468)*, pp. 41–48.

Nadeau, C. & Bengio, Y. (2003). Inference for the Generalization Error. *Machine Learning*, 52(3), 239–281. doi: 10.1023/A:1024068626366.

Nadon, V., Bonnet, F., Bouserhal, R. E., Bernier, A. & Voix, J. (2020). Method for protected noise exposure level assessment under an in-ear hearing protection device: a pilot study. *International Journal of Audiology*, 1–10.

Nick Hunn. (2016). *The Market for Hearable Devices 2016-2020.* Consulted at http://www.nickhunn.com/wp-content/uploads/downloads/2016/11/The-Market-for-Hearable-Devices-2016-2020.pdf.

Pancoast, S. & Akbacak, M. (2012). Bag-of-Audio-Words Approach for Multimedia Event Classification. *INTERSPEECH*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.

Perez, L. & Wang, J. (2017). The Effectiveness of Data Augmentation in Image Classification using Deep Learning. 239–281.

Phan, H., Hertel, L., Maass, M. & Mertins, A. (2016a, 4). Robust Audio Event Recognition with 1-Max Pooling Convolutional Neural Networks. Consulted at http://arxiv.org/abs/1604.06338.

Phan, H., Koch, P., Katzberg, F., Maass, M., Mazur, R., McLoughlin, I. & Mertins, A. (2016b, 12). What Makes Audio Event Detection Harder than Classification? Consulted at http://arxiv.org/abs/1612.09089.

Plinge, A., Grzeszick, R. & Fink, G. A. (2014, 5). A Bag-of-Features approach to acoustic event detection. *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3704–3708. doi: 10.1109/ICASSP.2014.6854293.

Portelo, J., Bugalho, M., Trancoso, I., Neto, J., Abad, A. & Serralheiro, A. (2009, 4). Non-speech audio event detection. *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 1973–1976. doi: 10.1109/ICASSP.2009.4959998.

Rabaoui, A., Davy, M., Rossignol, S. & Ellouze, N. (2008). Using One-Class SVMs and Wavelets for Audio Surveillance. *IEEE Transactions on Information Forensics and Security*, 3(4), 763–775. doi: 10.1109/TIFS.2008.2008216. 00082.

Rabiner, L. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2), 257–286. doi: 10.1109/5.18626.

Ramage-Morin, P. L. & Gosselin, M. (2018). Canadians vulnerable to workplace noise. *Health reports*, 29(8), 9–17.

Salamon, J. & Bello, J. P. (2017). Deep Convolutional Neural Networks and Data Augmentation for Environmental Sound Classification. *IEEE Signal Processing Letters*, 24(3), 279–283. doi: 10.1109/LSP.2017.2657381.

Sarria-Paja, M. & Falk, T. H. (2017). Fusion of auditory inspired amplitude modulation spectrum and cepstral features for whispered and normal speech speaker verification. *Computer Speech & Language*, 45, 437–456.

Schmitt, M., Janott, C., Pandit, V., Qian, K., Heiser, C., Hemmert, W. & Schuller, B. (2016, 10). A Bag-of-Audio-Words Approach for Snore Sounds' Excitation Localisation. *Speech Communication; 12. ITG Symposium*, pp. 1–5.

Sutin, A. R., Terracciano, A., Ferrucci, L. & Costa, P. T. (2010). Teeth grinding: Is Emotional Stability related to bruxism? *Journal of Research in Personality*, 44(3), 402–405. doi: 10.1016/j.jrp.2010.03.006.

Tak, S., Davis, R. R. & Calvert, G. M. (2009). Exposure to hazardous workplace noise and use of hearing protection devices among US workers–NHANES, 1999-2004. *American Journal of Industrial Medicine*, 52(5), 358–371. doi: 10.1002/ajim.20690.

Taniguchi, K., Kondo, H., Kurosawa, M. & Nishikawa, A. (2018). Earable TEMPO: A Novel, Hands-Free Input Device that Uses the Movement of the Tongue Measured with a Wearable Ear Sensor. *Sensors*, 18(3), 733. doi: 10.3390/s18030733.

Temko, A., Nadeu, C., Macho, D., Malkin, R., Zieger, C. & Omologo, M. (2009). Acoustic Event Detection and Classification. In *Computers in the Human Interaction Loop* (pp. 61). Consulted at http://adsabs.harvard.edu/abs/2009chil.book...61T.

Traore, B. B., Kamsu-Foguem, B. & Tangara, F. (2018). Deep convolution neural network for image recognition. *Ecological Informatics*, 48, 257–268. doi: 10.1016/j.ecoinf.2018.10.002.

Varga, A. & Steeneken, H. J. (1993). Assessment for automatic speech recognition: II. NOISEX-92: A database and an experiment to study the effect of additive noise on speech recognition systems. *Speech communication*, 12(3), 247–251.

VOIX, J., Monsarrat-Chanon, H., SERHAL, R. B., Cardinal, P. & Chabot, P. (2019). In-ear nonverbal audio events classification system and method. Consulted at https://patents.google.com/patent/WO2019079909A1/fr?q=voix&inventor=philippe+chabot&oq=philippe+chabot+voix.

Wang, Y., Getreuer, P., Hughes, T., Lyon, R. F. & Saurous, R. A. (2017, 3). Trainable frontend for robust and far-field keyword spotting. *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5670–5674. doi: 10.1109/ICASSP.2017.7953242.

Young, S. (1994). The HTK Hidden Markov Model Toolkit: Design and Philosophy. *Entropic Cambridge Research Laboratory, Ltd*, 2, 2–44.

Zhuang, X., Zhou, X., Huang, T. S. & Hasegawa-Johnson, M. (2008, 3). Feature analysis and selection for acoustic event detection. *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 17–20. doi: 10.1109/ICASSP.2008.4517535.

Zhuang, X., Zhou, X., Hasegawa-Johnson, M. A. & Huang, T. S. (2010). Real-world acoustic event detection. *Pattern Recognition Letters*, 31(12), 1543–1551. doi: 10.1016/j.patrec.2010.02.005. 00113.