Recognizing North Atlantic right whale up-calls using Gaussian Mixture Models and Hidden Markov Models

Master's Thesis by

Stine Back Larsen and Morten Albeck Nielsen

Supervisor: Thomas Dyhre Nielsen

Department of Computer Science

Aalborg University

Spring 2013

Summary in English

The modern society is dependent on the transportation by cargo ships, but the cargo ships can cause a negative effect on the marine wild life. The North Atlantic right whale is an endangered species which are especially threatened by cargo ships collisions. The right whales frequently emits a characteristic sound known as an *up-call*. This can be used for detecting when a right whale is in a particular area. A system using hydrophones for detecting whether a right whale is in a particular area has therefore been constructed by Cornell University's Bioacoustic Research Program [5]. In connection to this, a classification system, which can recognize when an audio recording of ocean sounds contains an up-call, is desired. For this purpose audio files containing ocean sounds, recorded by this system, have been provided. Each of these has been annotated with a label telling whether it contain an up-call or not [3]. In this thesis we make a classification system which can classify whether an audio file contains an up-call or not.

In order to make the classification system, the audio files must first be preprocessed into data which describe the source of the audio file content. We use the Mel-Frequency Cepstral Coefficients (MFCCs) as data which have been used often for speech recognition [23, 26] but also for recognizing whale sounds [11, 29]. In order to get the MFCCs, the digital signal is extracted form the audio files, and several transformations are made on the signal to get the features. In this process the signal is divided into overlapping frames. The result of the prepossessing is a feature vector for each frame of each audio file which consists of the MFCCs for that frame. The first parts of the process makes it possible to construct a spectrogram of the audio file which can be used for visualizing the frequencies of an audio file.

The classification system contains two models: A positive model which represents the feature vectors of the audio files that contain an up-call, and a negative model which represents the feature vectors of the audio files which do not contain an up-call.The system classifies an audio file by calculating the ratio between the probability that the feature data for the audio file were generated by the positive model, and the probability that the feature data for the audio file were generated by the negative model. The result is compared with an threshold, and if it is higher than the threshold, the audio files is classified as containing an up-call.

Three different model types are compared in order to investigate which performs best when

used in the classification system. The first model type uses a Gaussian Mixture Model (GMM), and does not divide the audio files into frames. There is therefore only one feature vector per audio file. The second model uses several GMMs. The audio files are divided into frames, and each frame is considered as being generated by a GMM. The third model uses a Hidden Markov Model (HMM) where each state of the underlying Markov process has an associated GMM.

The Expectation-Maximization (EM)-algorithm is used for learning the models. A general description of the EM-algorithm is given, and how the two steps of the algorithm can be derived for GMM and HMM is described, where especially the E-step has been emphasized. The models, and the EM-algorithm for GMMs and HMMs has been implemented, and specifications of the implementations are described.

We then compare how the three model types perform when used in the classification system. The comparison is made by finding the Area Under Curve (AUC) of the Receiver Operating Characteristic (ROC) curves, precision, recall, accuracy, and F_1-measure. For all the measures, the model type using one GMM, where the audio files are not divided into frames, scores the highest values. The model type using several frames and a GMM for each frame scores the second highest values, and the model type which uses an HMM scores the lowest values. A confusion matrix is then constructed for each model type for the best threshold on the ROC curve, and it turns out, that all three model types classifies a high number of audio files as containing an up-call even though they do not contain an up-call. We had expected that the HMM would score the highest values, so we investigated this model further. This was done by looking at the spectrogram for some of the audio files which contains an up-call. Thereafter the most probable path through the state space of a positive HMM for these audio files is found. This is used to investigate whether there is a connection between the state of the HMM to a given frame and the content of the frame. It turns out that the HMM, to some degree, can detect the placement of the up-call in an audio file.

Stine Back Larsen

Morten Albeck Nielsen

Resumé på dansk

Det moderne samfund er afhængig af fragtskibe, men fragtskibene kan have negativ effekt på marinelivet. Nordkaperen er en truet hvalart, og det ønskes derfor at minimere fragtskibes negative indflydelse på denne hvalart. Nordkaperen udsender ofte en karakteristisk lyd kaldet et *up-call*, som kan bruges til at detektere, om en Nordkaper hval er i et bestemt område. Et system, som ved hjælp af hydrofoner kan detektere, om en Nordkaper er i et bestemt område, er blevet konstrueret af Cornell University's Bioacoustic Research Program [5]. I den forbindelse ønskes et klassificeringssystem, som kan genkende up-calls i lydoptagelser fra andre havlyde. Til dette formål er der stillet lydfiler af havlyde til rådighed. Disse er blevet annoteret med en label, der angiver om lydfilen indeholder et up-call eller ej. I dette speciale laves der et klassificeringssystem til at afgøre om en lydfil indeholder et up-call eller ej.

For at lave det omtalte klassificeringssystem skal lydfilerne først præprocesseres til data, som beskriver lydkilden til indholdet af lydfilen så godt som muligt. Som data bruger vi MFCCs, som er blevet brugt ofte til talegenkendelse [23, 26] men også til genkendelse af hvallyde [11, 29]. For at få MFCC udtrækkes der et digital signal fra hver lydfil, hvor på der laves adskillige transformationer. I denne proces deles signalet op i overlappende tidsintervaller. Resultatet for en lydfil er en datavektor for hvert tidsinterval, der består af MFCCs for det pågældende tidsinterval. Desuden gør den første del af processen det muligt at lave et spektrogram over lydfilen, som kan bruges til at visualisere hvilke frekvenser som lydfilen indeholder.

Klassificeringssystemet består af to modeller: En positiv model der repræsenterer datavektorerne for lydfiler, som indeholder et up-call, og en negativ model der repræsenterer datavektorerne for de lydfiler, som ikke indeholder et up-call. Systemet klassificerer en lydfil ved at udregne forholdet mellem, hvor sandsynligt det er at datavektorerne fra lydfilen er generet af den positive model, og hvor sandsynligt det er, at de er generet af den negative model. Resultatet sammenlignes med en tærskelværdi, og hvis forholdet er større end tærskelværdien klassificeres lydfilen som indeholdende et up-call.

Tre forskellige modeltyper sammenlignes, for at undersøge hvilken der fungerer bedst, når den bruges i vores klassificeringssystem. Den første modeltype benytter en GMM, og opdeler ikke lydfilen i tidsintervaller. Der er derfor kun en datavektor per lydfil. I den anden modeltype

benyttes der flere GMMs. Her deles lydfilerne op i tidsintervaller, og hvert tidsinterval betragtes som værende genereret af hver sin GMM. Den tredje modeltype bruger en HMM, hvor der til hver tilstand af den underliggende Markov proces er associeret en GMM.

EM-algoritmen bruges til at lære modellerne. Der bliver givet en generel beskrivelse af EM-algoritmen, og hvordan de to skridt i algoritmen udledes for GMM og HMM, hvor der især er lagt vægt på E-skridtet. Modellerne og EM-algoritmen for GMMs og HMMs er blevet implementeret, og specifikation omkring implementeringen er beskrevet.

Vi sammenligner derefter de tre modeltype ved at bruge dem i klassificeringssystemet. De sammenlignes ved at finde arealet under ROC kurven, præcision (precision) og genkaldelse (recall), samt nøjagtighed (accuracy) og F_1-mål. For alle mål får modeltypen med GMM, hvor lydfilens ikke opdeles i tidsintervaller, højst værdier, derefter kommer modeltypen, hvor der er en GMM for hvert tidsinterval, og til sidst modeltypen der bruger HMM. En forvirrings (confusion) matrice for hver modeltype konstrueres derefter for det bedste punkt på ROC kurven, og det viser sig, at alle tre modeltyper har et højt antal lydfiler, modellerne klassificerer til at indeholde et up-call, selvom filerne faktisk ikke indeholder et up-call. Da vi havde forventet at HMM ville få de højeste værdier, kigger vi nærmere på denne model. Dette gøres ved at kigge på spektrogrammet for nogle lydfiler som indeholder et up-call. Derefter finder vi for en positiv HMM den mest sandsynlige vej igennem tilstandsrummet for disse lydfiler, og det ses om tidsintervallerne, der dækker up-callene, er i nogle bestemt tilstande, og tidsintervaller udenom er i andre tilstande, eller om det er tilfældigt. Det viser sig, at HMMen, til en vis grad, er i stand til at detektere placeringen af et up-call i en lyd fil.

Contents

List of Figures

List of Tables

Listings

Acronyms

AUC Area Under Curve

DAG Directed Acyclic Graph

DCT Discrete Cosine transform

DFT Discrete Fourier transform

EM Expectation-Maximization

FFT Fast Fourier transform

FN False Negative

FP False Positive

FPR False Positive Rate

GMM Gaussian Mixture Model

HMM Hidden Markov Model

i.i.d independently and identically distributed

MFCC Mel-Frequency Cepstral Coefficient

ROC Receiver Operating Characteristic

TN True Negative

TP True Positive

TPR True Positive Rate

List of symbols

R	Number of audio files
T	Number of frames for an audio file/number of time slices for an HMM
$\mathbf{x}$	Feature vector consiting of MFCCs
$\mathbf{x}_t$	Feature vector for time slice/frame t
$\mathbf{x}_{t:t'}$	Feature vector for frame t to frame t'
$\mathbf{x}_t^{(r)}$	Feature vector for audio file r and time slice/frame t
$\mathbf{X}_t$	Variable for feature vector for time slice/frame t
$\mathcal{X}$	All T feature vectors for all R audio files
D	Number of MFCCs for each feature vector, i.e. the size of the feature vectors
λ	Parameters for a model
K	Number of components for a GMM
c_k	Component k
w_k	Weight for component k
$\boldsymbol{\mu}_k$	Mean vector for component k
$\boldsymbol{\Sigma}_k$	Covariance matrix for component k
N	Number of states for an HMM
S_t	Variable for a state of an HMM at time slice t
C_t	Variable for component at time slice t for a HMM
w_{nk}	Weight for component k of the GMM to state n of an HMM
$\boldsymbol{\mu}_{nk}$	Mean vector for component k of the GMM to state n of an HMM
$\boldsymbol{\Sigma}_{nk}$	Covariance matrix for component k of the GMM to state n of an HMM
$Q(\lambda, \lambda^{(i-1)})$	Expected value of $\log P(\mathcal{X}, \mathcal{Y}\|\lambda)$ given $\mathcal{X}$ and $\lambda^{(i-1)}$
α_t	Forward message to time slice t
β_t	Backward message to time slice t
γ_t	Smoothing variable to time slice t
c_t	Scaling factor for forward and backward message to time slice t
f_t	Scaled forward message to time slice t
b_t	Scaled backward message to time slice t
A	Matrix for state transition probabilities of an HMM
π	Vector for the intial state distribution
E_t	Vector for the emision message at time slice t

1 Introduction

The modern society depends on the shipping industry's ability to transport goods across long distances. This transport is often done by cargo ships because of their large shipping capacity, and because they can transport goods across oceans. However the ships' movements can have a negative effect on the marine wild life. The North Atlantic right whale is particularly endangered by the movement of cargo ships close to shore [3]. The goal of this thesis is to develop a system which detects a call made by the North Atlantic right whale in audio files containing underwater recordings, in order to prevent cargo ships from harming the whales. The system will be based on machine intelligence methods.

1.1 Problem Statement

The world's last 350 North Atlantic right whales live along the North American East Coast, and the movement of ships in that region poses a deadly hazard for them. Colliding with a cargo ship is one of the extremely serious hazards for the whales because a whale is not likely to survive a collision with a cargo ship. However, if the ships slow down and post extra lookouts it reduces the risk of collisions, but making all cargo ships slow down is not a sustainable solution [5].

In order to prevent these collisions while achieving commercial sustainability, Cornell University's Bioacoustic Research Program [1], in cooperation with the company Marinexplore [6], is

developing a system that can detect the presences of right whales in a region and alert ships heading for that region. During the development of this system a buoy network was deployed in Massachusetts Bay [5]. The buoys are listening for a certain type of call known as an *up-call* which is a contact call that the right whales use to let other whales know that they are nearby. The system exploits the up-call because the whales are making these up-calls often, and the sound of the up-call is characteristic. The buoys have an embedded system installed which record sounds and determine if the recordings possibly contains an up-call based on some very simple features i.e. the length of sounds. If the embedded system determine that the recorded sound perhaps contain an up-call it is transmitted to a server at Cornell University for further analysis. Human experts at Cornell University then decide whether the recorded sound really does contain the sound of a right whale up-call. If this is the case, the ships in that region are alerted otherwise no further actions are taken. When a ship is alerted it can slow down below 10 knots and post a lookout in order to avoid colliding with a right whale. This system cannot determine the position of the right whales, but it can detect when they are in the region near a buoy. The embedded system installed in the buoys are able to determine whether a recorded sound is possibly a right whale up-call. However, verification by experts is necessary before ships can be alerted in order to reduce the number of false alerts. The classification procedure performed by the buoy's embedded system is very simple, its purpose is to limit the amount of audio recordings that should be transmitted from the buoy's embedded system to the server at Cornell University [31].

The full system should be working continuously such that ships are alerted in nearly real-time, and since the final verification of the transmitted audio recordings are performed by human experts, it is necessary to always have the system manned. In order to free the human experts from their duties, an automatic verification system is desired. This system should be able to classify an audio recording, transmitted from a buoy, as either containing or not containing a right whale up-call, and thus it can be addressed as a classification problem.

To find an appropriate solution to this classification problem, a competition was held on the *Kaggle* website [2] where participants were encouraged to present a method for performing this classification. The final evaluation of this competition was based on the Area Under Curve (AUC) of Receiver Operating Characteristic (ROC) curves [1] which were calculated from test data. The labels for the test data were not published and thus hidden for the participants. The participants' submission should contain a real value for each audio file in this data set such that a low value indicated that the audio file did not contain a right whale up-call, and a high value indicated that the audio file did contain a right whale up-call. In addition to the test data, Kaggle provided a data set for model learning where each audio file was labeled as containing or not containing a right whale up-call.

[1] A description of ROC curves can be found in Section 6.1.1.

This thesis addresses the classification problem from the Kaggle competition by using methods from the well studied problem of speech recognition where Hidden Markov Models (HMMs) have been used extensively in the past [24, 26, 30]. We will investigate how Gaussian Mixture Models (GMMs) and HMMs can be used for recognizing right whale up-calls, and what issues that occur when implementing such a solution. The main focus of this thesis is on the machine learning, and we will thus not investigate different types of features which are considered to be in the domain of digital signal processing. To learn the parameters of each type of model, the Expectation-Maximization (EM)-algorithm can be used. We will present an description of the general theory behind the EM-algorithm and describe how the formulas for updating the model parameters for GMMs and HMMs can be derived. Further we will investigate how well the different model types perform when used in the classification system, and what they have problem capturing.

1.2 Related Work

The task of recognizing marine animal sounds has been addressed by several articles in the past using various methods.

Weisburn et al. [34] investigated two different methods for detecting bowhead whale calls in audio recordings which were recorded in the Arctic. Besides bowhead calls they contained noise, and, possibly interferences made by other animals, or by ice that was cracking. The two different methods, that they used, were an HMM and a matched filter. The feature data for the HMM was the three largest peaks in the frequency spectrum for each time frame. The HMM had 18 states, and for each of these it had a Gaussian distribution over the feature data. The matched filter was determined from 40 recordings that contained only whale calls and no interferences. These recordings were also used to learn the HMM. In order to detect whale calls in the recorded signals, they computed a score and compared it to a threshold. For the HMM the score was the likelihood found by the Viterbi algorithm, and for the matched filter it was the correlation between the signal and the filter. Weisburn et al. found that their HMM method performed better than the method using a matched filter, but both methods identified a high portion wrongly.

Mellinger and Clark [22] compared several methods for recognizing bowhead whale calls. They suggested a method using spectrogram correlation and compared this to three other methods, which used an HMM, a matched filter and a neural network, respectively. The HMM method is similar to the one used by Weisburn et al. The input layer of the neural network was an 11×22 array computed from the spectrogram. The hidden layer contained four units, and the output layer contained a single unit. Each of the method returned a score which were compared to a threshold for determining whether a call was detected. The first data set was used for comparing the spectrogram correlation method to the method using an HMM and the

method using a matched filter while the second data set was used for comparing the spectrogram correlation method to the method using a neural network and the method using a matched filter. Mellinger and Clark found that the spectrogram correlation method performed marginally better than the method using an HMM, and that the method using a neural network performed even better. However they also found that the neural network requires a relatively large data set for learning. Further they found that the match filter performed poorly, and they concluded that the matched filter method is not appropriate because the noise in the recordings were not Gaussian distributed, and the bowhead whale calls were too divergent from each other.

Datta and Sturtivant [13] used HMMs to identify three different groups of dolphin whistles. Their HMMs represented the contour of the shape of the dolphin whistle when drawn as a spectrogram. For each of their audio recordings, the part that contained a dolphin whistle was identified in the preprocessing, and a spectrogram representation of this was constructed. Then a contour following algorithm was applied on the spectrogram to find the shape of whistle sound. An HMM was learned for each whistle class. These were then used for classifying future whistles by calculating the likelihood that a recorded whistle belongs to each class.

Roch et al. [29] used GMMs to determine the species of recorded dolphin whistles. The recorded signal was split up into time frames from which the cepstral coefficients was calculated. These were then used as feature data for the GMMs. A GMM was learned from the whistles for each species. When the species for a recorded whistle was determined, the likelihood for each GMM representing the feature data was calculated. The dolphin that made the whistle was then assumed to belong to the species whose GMM had returned the highest likelihood. The number of components of the GMMs was 64, 128, 256, and 512. The best results were found using GMMs with 256 mixtures.

Brown and Smaragdis [11] classified calls from killer whales into seven different call types. They investigated the use of GMMs and HMMs where the HMMs had a GMM for each state. Their data set consisted of 75 recorded calls which each contained one and only one of the seven call types. As feature data the Mel-Frequency Cepstral Coefficients (MFCCs) and their temporal derivatives were used. These were calculated using the program *melcepst* from the Matlab toolbox VOICEBOX [7]. Testing was performed using the *leave-one-out* method where each recording from the data set in turn was classified while the remaining were used for learning the models. To measure performance the percentage agreement was used. The GMMs were learned with 1 to 6 components and 8 to 30 features. The best result was 92% agreement which was obtained using GMMs with two components and 30 features. The HMMs was learned with 5 to 17 states, 1 to 4 components, and 8 to 42 features. The best results was 95% agreement, which was obtained using HMMs with 24 to 30 features, 13 to 17 states, and one component.

Recognizing marine animal sounds is a problem that has great similarity to speech recognition. For both problems we are trying to classify audio signals by the source which generated them. Thus it is the particular source, that we are trying to recognize, which distinguishes the problems. For speech recognition we know that the source is a human vocal tract, and we are trying to recognize the setting of this vocal tract. For the problem addressed by this project, the source could have been a right whale which emitted an up-call. Otherwise it could also be some other source e.g. other marine animals. For both problems we must extract feature data which carry information about the process that generated the signal, and from this learn models which capture the process that generated the signal. Speech recognition is a problem that has been extensively studied in the past [24, 26, 30], and because of its similarity to our problem it is reasonable to investigate how methods for speech recognition can be applied to recognizing up calls. Roch et al. [29] and Brown and Smaragdis [11] used an approach very similar to the one that was proposed for speech recognition by Rabiner in 1989 [26]. They also used the cepstral coefficients which are used often in speech recognition because it carries much information about the vocal tract [23].

1.3 Approach

The data set for this project consists of audio files which were recorded using hydrophones. These can be divided into two classes: The positive class which are the audio files containing up-calls, and the negative class which are the audio files not containing up-calls. The task is therefore to make a classifier for the audio files in order to recognize which class each audio file belongs to. In order to solve this, we take the approach described in the introduction chapter of [15]. First feature data must be extracted from the audio files. How the audio files are interpreted and the feature data are extracted is explained in Chapter 2.

Then we must decide on a type of model for representing the features. In this thesis we limit ourselves to use the same type of model for both classes, and we can therefore compare how well different model types perform, when they are used for making a classifier. The different model types which are used in this project are based on GMMs and HMMs where the HMMs have a GMM for each state. Thus we assume that the observed features are generated by some hidden process, and the distribution of these can be described using Gaussian mixtures. For the HMM, we further model the development of this process through time. We are trying to recognize whether an up-call is present in an audio file, and by using GMMs or HMMs, we are trying to model this source with the GMM components and the Markov process of the HMMs. The different model types are described in Chapter 3.

Besides its structure, a model is also defined by some parameters λ. For the models used in this project, these parameters are learned from annotated data. This we will refer to as *learning a model*. The available data are divided into to three disjoint data sets. The first data set is used

for learning a model for representing the files in each class. How the models are learned is explained in Chapter 4, and the implementation of the models and the learning procedure is explained in Chapter 5.

When a model for both the positive and negative class have been learned they can be combined to form a classification system. The classification procedure used here is inspired by the those explained by Rabiner [26] and Roch [29]. The architecture for the classification system used in this thesis is shown in Figure 1.1. The procedure for classifying an audio file is the following. First the digital signal is read from the audio file. Then the feature data are extracted from the signal returning an observation sequence. For both classes, we then compute the probability of this observation sequence given the model for the class. To combine the two probabilities values into a single scalar, we compute the ratio between the probability returned by the model for the files in the positive class, and the probability returned by the model for the files in the negative class. This returns a positive real number which is large when it is likely that the audio file contain an up-call and small otherwise. In order to make the final classification, the ratio is then compared to a threshold. By adjusting the threshold, we can change likelihood of classifying an audio file as positive or negative.

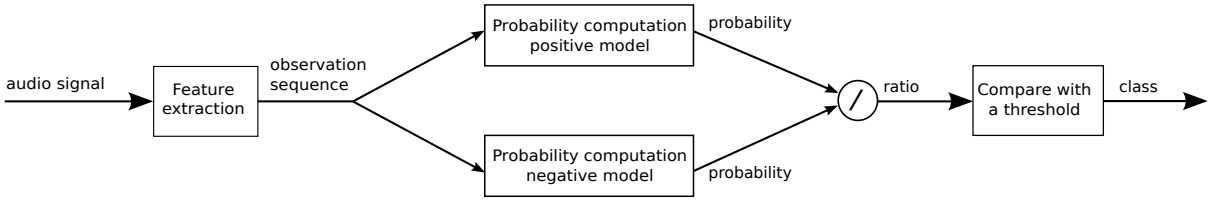


Figure 1.1: Illustration of the classification procedure.

Different models with different number of components and states are learned, and the second data set can then be used to deciding which model for each class that performs best when used in a classifier. The third data set is reserved for testing. Selection of models and tests are explained in Chapter 6. In Chapter 7 we conclude on the conducted work and found results. Further we discuss topics for future work.

2 Data

The available data set consists of 30000 underwater audio recordings which have been annotated. There is a binary label for each recording; Positive if it is believed to contain a whale call of interest i.e. an up-call from a North Atlantic Right whale, otherwise it is labeled negative. For the annotated audio files the labeling where performed by whale experts who are able to determine whether a recording contain an up-call or not by listening to the audio recording and investigate a corresponding spectrogram[1]. In this thesis it is assumed that all the recordings where labeled correct by the whale experts.

From each audio file a *discrete signal* can be extracted. An example of a discrete signal can be seen in Figure 2.1. On the horizontal axis time is plotted, and on the vertical axis the amplitude is plotted. Each point represent a *sample* of the recorded sounds, which is the amplitude at the particular point in time. The number of samples pr. seconds is called the *sampling rate*, and this is determined by the way the audio file was recorded. Let L be the number of samples in an audio file, the signal can then be described as a function $s(l)$ where $l = 1, 2, \ldots, L$.

The available audio files all have a sample rate of 2000 Hz and a duration of two seconds which therefore corresponds to 4000 samples [3]. As feature data for audio signals, the cepstral coefficients have been used in the past both for speech recognition [23, 25, 26] and animal sound classification [11, 27, 29], and therefore we also chose to use the cepstral coefficients as our fea-

[1] Spectrograms are explained in Section 2.1.1.

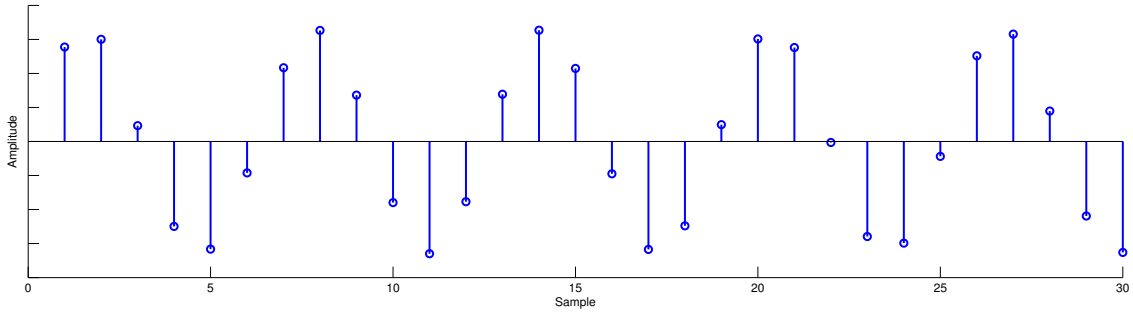


Figure 2.1: Example of a discrete signal.

ture data. This chapter gives a description of the feature data used in this thesis. In Section 2.1 the process of extracting the feature data is described, and in Section 2.2 an analysis of the available audio files, and which considerations that have been made for feature extraction are presented.

2.1 Feature Extraction

In order to get feature data which represent an audio file several transformations on the digital signal are made. The first part is to compute a spectrogram. This is explained in Section 2.1.1. From the spectrogram the Mel-Frequency Cepstral Coefficients can be derived. This is explained in Section 2.1.2.

2.1.1 Spectrograms

The discrete signal from each audio file is in the time domain where the audio signal has been recorded. This can be transformed to the frequency domain where the signal can be represented and understood, and by dividing the signal into intervals, and transforming each interval to the frequency domain a joint time and frequency representation of the signal is obtained. The result of this is called a spectrogram [19]. The process of going from the discrete signal to a spectrogram is illustrated in Figure 2.2. Each step is described in the following paragraphs.

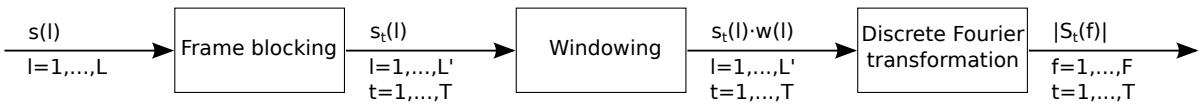


Figure 2.2: The overall procedure for generating a spectrogram.

Frame blocking A signal can change a lot over time, and therefore the first step is to divide the discrete signal up into overlapping parts refereed to as frame. This is illustrated in Figure 2.3. The signal in each frame is then assumed to be static which means that the frequency is not changing or at least not changing too much [19, 23]. This is sometimes refereed to as blocking a signal into frames [26]. Let T be the number of frames, and let L' be the number of samples

in each frame. Then the signal for frame t is $s_t(l)$ where $l = 1, \ldots, L'$ and $t = 1, \ldots, T$. We must therefore decide on the number of frames and the size of the overlap in order to compute the feature data. This is presented in Section 2.2.

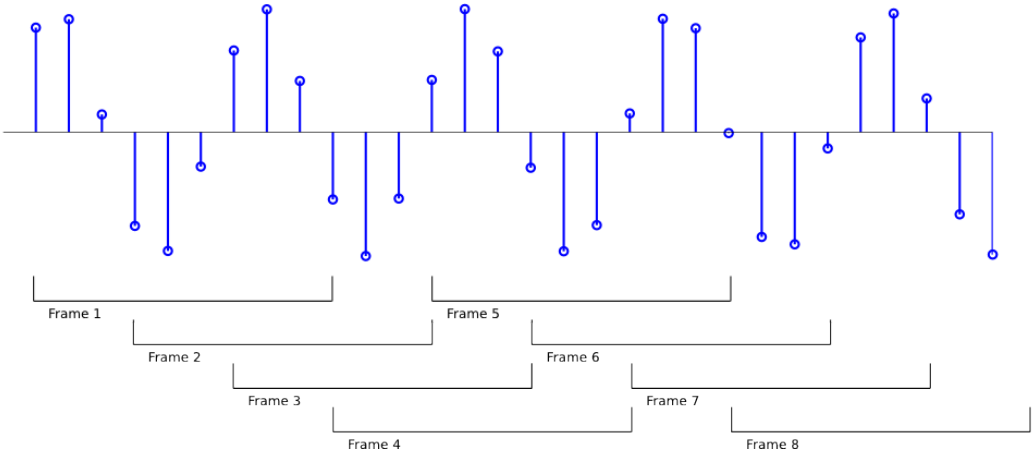


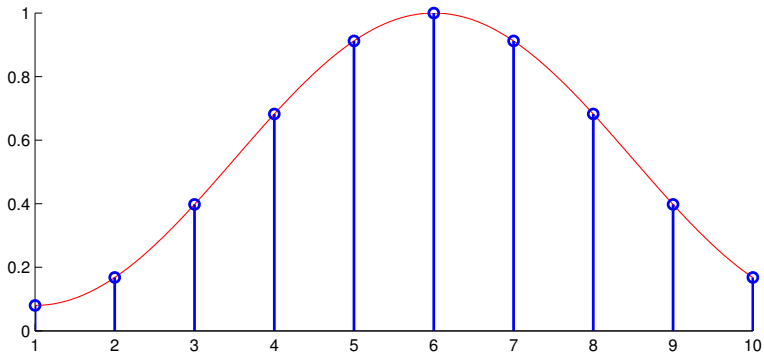
Figure 2.3: Frame blocking of a digital signal.

Windowing The second step is to apply a window to each frame [19, 23]. First a window function must be chosen. The window function used in this project is the Hamming window function [23] which is given in Definition 2.1.1.

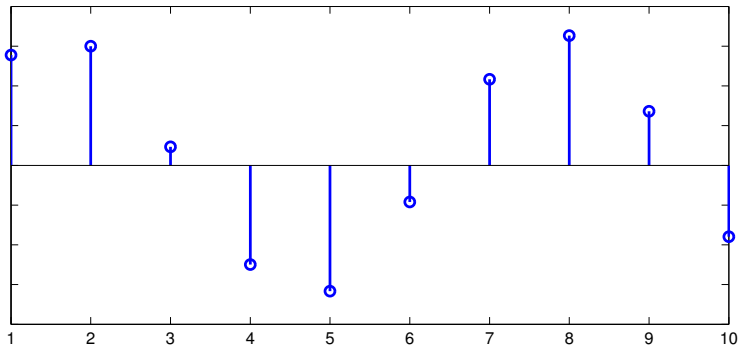
Definition 2.1.1 (Hamming window function)**.**

$$w(l) = \begin{cases} 0.54 - 0.46\cos(\frac{2\pi l}{L'-1}) & \textit{if } 0 \leq n \leq L' - 1 \\ 0 & \textit{otherwise} \end{cases}$$

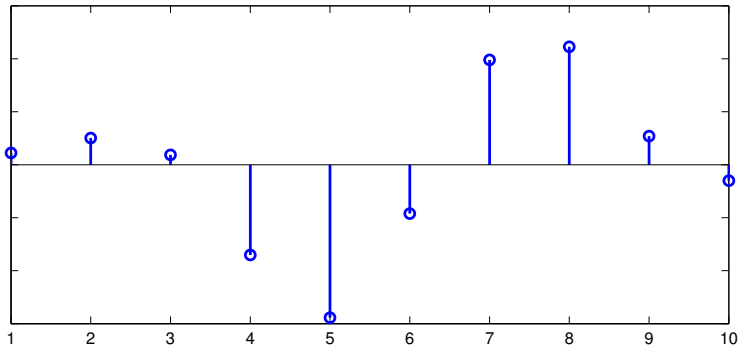
In order to apply the window to a frame, $s_t(l) \cdot w(l)$ is calculated for $l = 1, \ldots, L'$. The effect of windowing is illustrated in Figure 2.4. In the top figure the Hamming window function $w(l)$ for each sample position l is illustrated. In the middle figure a signal for one frame is illustrated. In the bottom figure the signal after the window has been applied is illustrated. Windowing is done to reduce the adverse effect of chopping out the frame from the original complete signal [23, 26]. When blocking the signal into frames, the sample at the ends of the frame may be high, so the signal goes from zero to a high amplitude. By applying the window, the signal of one frame becomes more realistic. This can been seen by comparing the middle and bottom figure. In the latter, the amplitude of samples in the ends of the frame have been decreased after windowing. If the window had not been applied it could introduce high frequencies that was not in the signal of the frame.



Window function $w(l)$.



Frame before windowing $s_t(l)$.



Frame after windowing $w(l) \cdot s_t(l)$.

Figure 2.4: Applying a Hamming window $w(l)$ to signal $s_t(l)$.

Discrete Fourier transformation The third step is to map the windowed signal for each frame to the frequency domain which gives L' coefficients for each frame. Each coefficient represents the magnitude of a correlation to a signal with a certain frequency. This is done by computing the absolute value of the Discrete Fourier transform (DFT)[2] which is given by [23]:

$$S_t(f) = \sum_{l=1}^{L'} s_t(l) e^{-i \cdot \frac{2\pi}{L'} \cdot f \cdot l} \text{ for } f = 1, \dots L' \tag{2.1}$$

Here i is the imaginary unit. Because of conjugate symmetry there are two coefficients for each frequency [23]. Thus half of the coefficients are redundant and can therefore be discarded. Let the number of coefficients for a frame after the discard be F, i.e. $F = \frac{L'}{2} + 1$, the coefficients for frame t is then denoted as $|S_t(f)|$ where $f = 1, \dots, F$. A spectrum for each frame can then be drawn from the found magnitudes. This is shown in Figure 2.5. The top figure illustrates a windowed signal of a frame. The middle figure illustrates the magnitude for each frequency found by computing the absolute value of the DFT. The bottom figure shows the same as the middle figure but as a spectrum. The frequency is plotted horizontal. The second dimension is plotted as a color gradient. This represent the magnitude of a frequency. In this thesis we use blue color as low magnitudes and red as high magnitudes.

Result: Spectrogram When a spectrum has been computed for all the frames of the audio signal, the spectrums can be combined into a spectrogram by drawing each spectrum on the horizontal axis chronologically, i.e. in the order that their corresponding frame appear in the signal. An example of a spectrogram is shown in Figure 2.6. The horizontal axis represent time while the vertical axis represent frequency. The third dimension represent the magnitude of a particular frequency for a particular time in the signal. By investigating a spectrogram, it serves as a useful aid when investigating sounds. In fact, spectrograms was used by the experts who labeled the audio files in the data set that is used in this thesis [31].

2.1.2 Mel-Frequency Cepstral Coefficients

A spectrogram is good for analyzing the sounds in an audio file, owever further computations are normally performed on the coefficients when features for speech and sound recognition is desired [11, 19, 23, 25, 26, 27, 29]. For this project the Mel-Frequency Cepstral Coefficients (MFCCs) is used, which are coefficients that together describe the mel-frequency cepstrum [19]. The mel-frequency cepstrum is derived from the spectrum, and, as the spectrum, they carry information about the frequencies in a frame [19, 23]. An overview of the steps for computing the MFCCs are illustrated in Figure 2.7. Each step is described in the following paragraphs.

[2]In practice Fast Fourier transform (FFT) is used for computing the DFT because the algorithm for FFT is computational faster than the basic DFT algorithm [12].

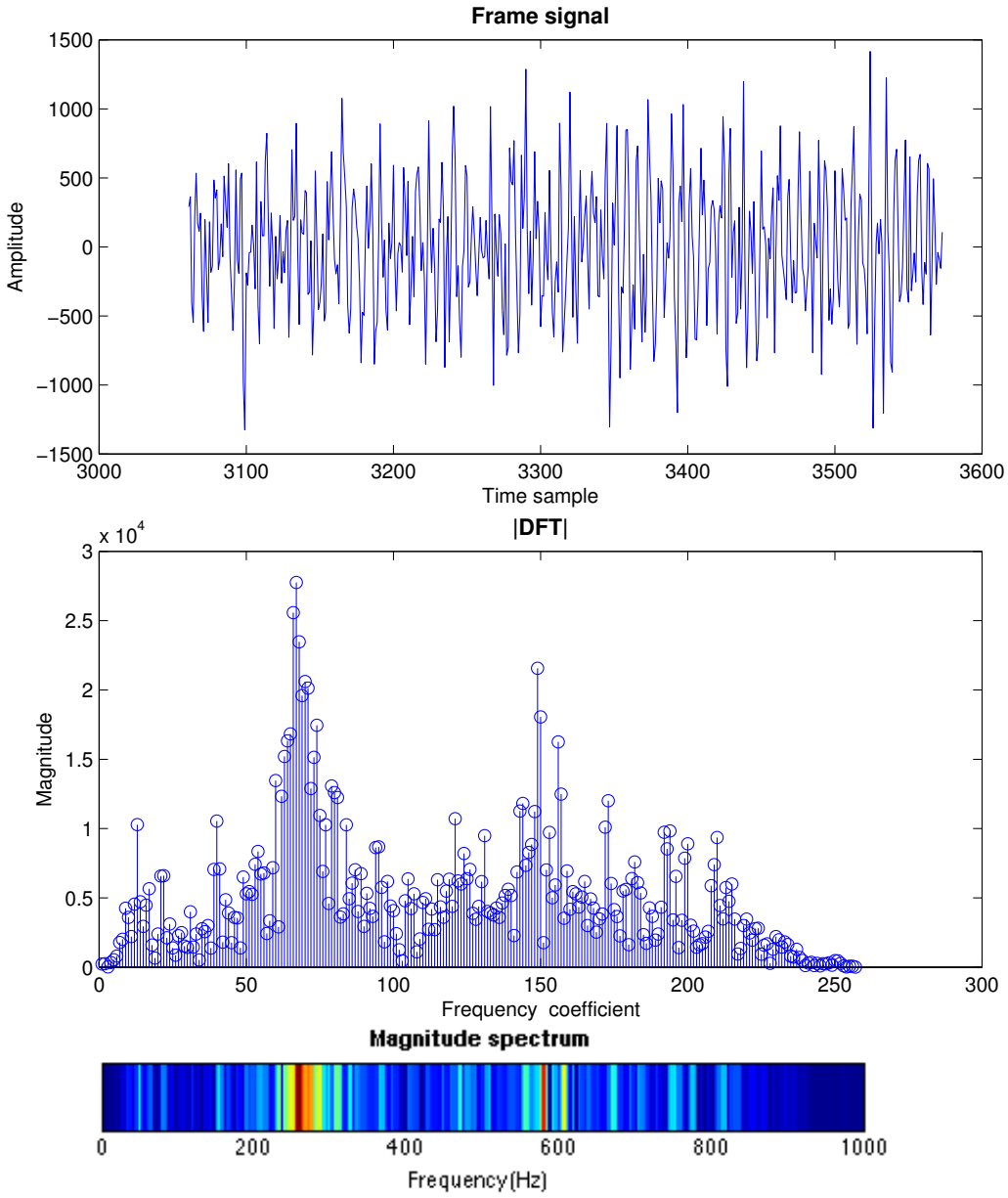


Figure 2.5: Computing the spectrum of a frame.

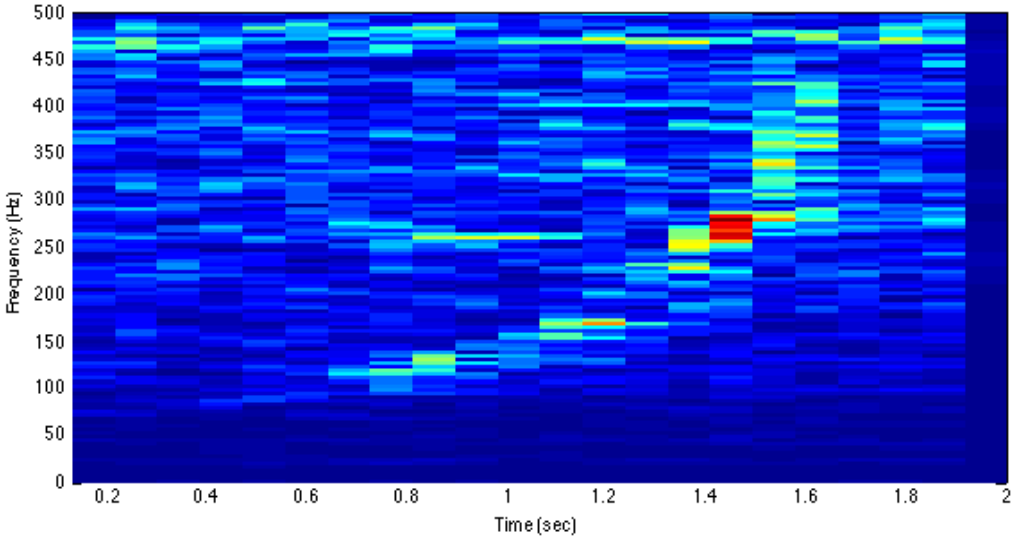


Figure 2.6: Example of a resulting spectrogram.

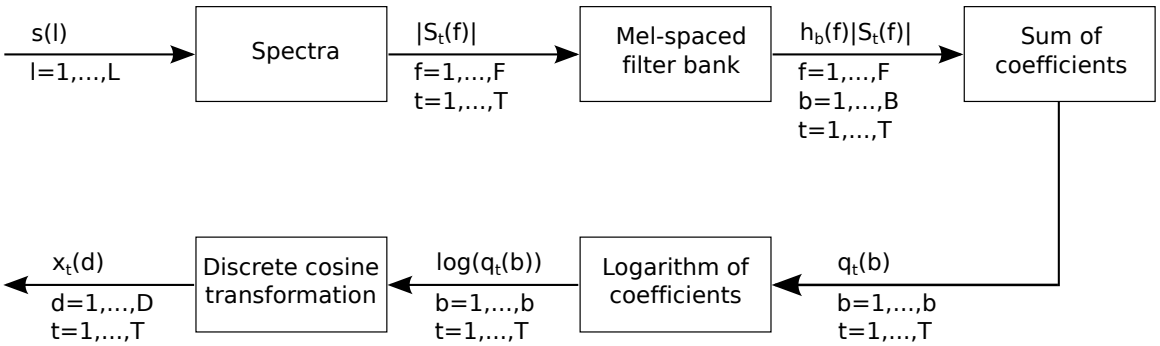


Figure 2.7: Pipeline for computing MFCCs.

Spectrum for each frame The first step is to compute the spectrum for each frame as explained in Section 2.1.1.

Mel-spaced filter bank In order to combine the coefficients of the spectrum for each frame, a filter bank is applied [19]. A filter bank consists of several filters. Each filter covers an area of frequencies. When applied a filter scales each coefficient of the spectrum with a factor. The value of this factor depends on the frequency area associated with the filter. Coefficients outside the area which the filter covers are scaled to 0. The result of applying the filter bank is a filtered spectrum for each filter in the filter bank.

An example of a filter can be seen in Figure 2.8. As it can be seen in the figure, a filter covers an interval for frequencies. When the signal is discrete, as it is in this project, the filters will also be discrete, so a filter is a sequence of numbers $h(f)$ where $f = 1, \ldots, F$. For frequencies f outside the filter $h(f) = 0$. Let B be the number of filters in the filter bank. For each frame t the filters $h_b(f)$, $b = 1, \ldots, B$, are then applied to the spectrum coefficients for the frame. This gives B sequences:

$$\langle h_b(1)|S_t(1)|, h_b(2)|S_t(2)|, \ldots, h_b(F)|S_t(F)|\rangle \text{ for } b = 1, \ldots, B \tag{2.2}$$

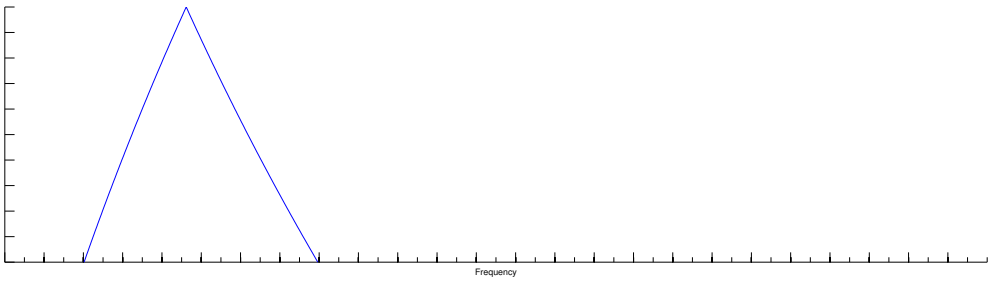


Figure 2.8: Example of one filter.

The size of the frequency area that a filter covers could be the same for all filters in a filter bank. Humans, however, can better distinguish smaller changes in frequencies for a sound if the frequency is small than if the frequency is high [24]. This can be expressed in the filter bank by letting filters which cover a frequency area of low frequencies cover a smaller area than filters which cover an area of high frequencies [19, 24]. Such a filter bank are then said to be mel-spaced. An example of a mel-spaced filter bank with five filters is illustrated in Figure 2.9. There is however no reference in found litterateur for what filter bank to use for right whale calls. Brown and Smaragdis [11], who had a similar problem and uses a similar solution method as us, got good results when using MFCC which is the cepstral coefficients where a mel-spaced filter bank have been used in the derivation. At the same time the MatLab

toolbox VOICEBOX [7] provides a function called *melcepst* which finds the mel-spaced cepstral coefficients, so we have chosen to use a mel-spaced filter bank.

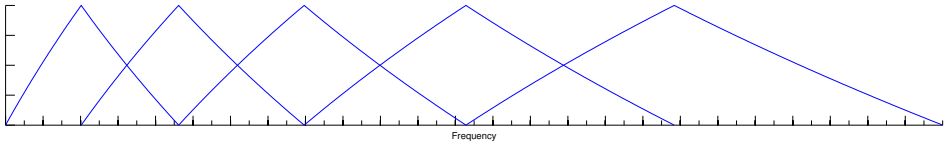


Figure 2.9: Example of a mel-spaced filter bank with 5 filters.

Sum of coefficients for each filter In order to reduce the number coefficients we sum all of the elements in the sequence for each filter, which corresponds to an area in the frequencies. Thus the number of coefficients are reduced from F to B for each frame t:

$$q_t(b) = \sum_{f=1}^{F} h_b(f)|S_t(f)| \text{ for } b = 1, \ldots, B \tag{2.3}$$

Logarithm of coefficients for each filter We compute the logarithm of each of the B coefficients from the previous step. This have the effect on the values that numbers close to zero become large negative values and the large positive values are reduced to smaller positive values. The original motivation behind applying the the logarithm, however is that we potentially could detect and remove echo effects in resulting coefficients [23]. The result is a sequence of B numbers for each frame t:

$$\langle \log q_t(1), \log q_t(2), \ldots \log q_t(B) \rangle \tag{2.4}$$

Discrete cosine transform The fifth step is to convert the coefficients back to the time domain [19]. This is done by making the Discrete Cosine transform (DCT) on the sequence:

$$x_t(d) = \sum_{b=1}^{B} \log(q_t(b)) \cos(\frac{\pi}{B} \cdot d \cdot (b - \frac{1}{2})) \text{ for } d = 1, \ldots, D \tag{2.5}$$

which returns a sequence of B elements which is the MFCCs. In order to construct our feature vector, the desired number of MFCCs is kept while the rest is discarded [19]. Usually it are the coefficients for the high frequencies which are discarded. Let D be the desired number, the result is therefore a sequence of D MFCCs for each frame which we will refer to as the feature vectors or the data of an audio file. How D is chosen is presented in Section 2.2.

We will denote an observed feature vector as $\mathbf{x}$. When we want to specify that it is the feature vector for frame t, $\mathbf{x}_t$ is used, so $\mathbf{x}_t = (x_t(1), \ldots, x_t(D))$. When denoting all observed feature vectors for frames t to t', the notation $\mathbf{x}_{t:t'}$ is used. When the feature vector of a frame t is not observed $\mathbf{X}_t$ is used, and when we want to specify that the feature vector for frame t comes from audio file r we denote it as $\mathbf{x}_t^{(r)}$. Let R be the number of audio files. Then the set of all the observed feature vectors of all R audio files is denoted as $\mathcal{X} = (\mathbf{x}_{1:T}^{(1)}, \ldots, \mathbf{x}_{1:T}^{(R)})$.

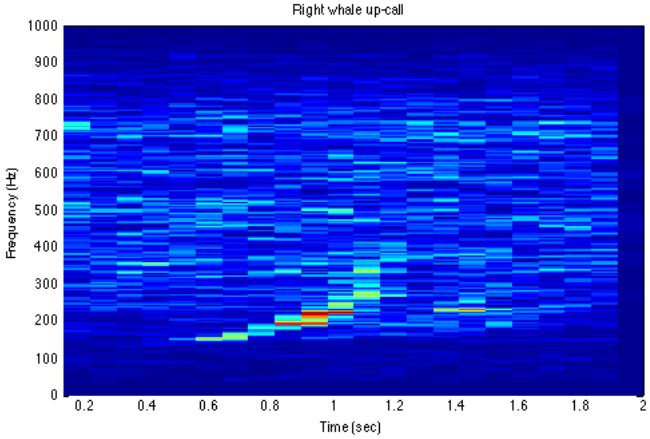
2.2 Data analysis

To extract the digital signal from the audio files and compute the MFCCs, we use the MatLab toolbox called VOICEBOX.

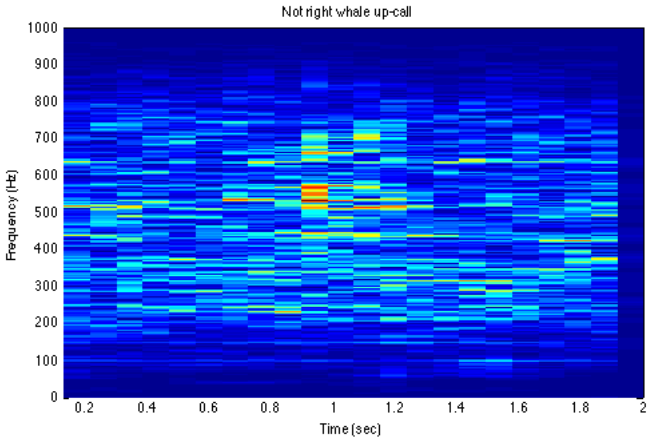
In order to make a spectrogram the length of the frames, and how much the frames overlap must be decided. For this project we used two different frame lengths: A frame length of 4000 samples, and a frame length of 512 samples. The frame length of 4000 samples corresponds to 2 seconds i.e. the entire audio recording. So the audio records was therefore only divided into one frame. Thus is $T = 1$, and there is no overlap. This was used for measuring the effect of going from a simple approach with only one frame to a more complex approach with several frames. The frame length of 512 samples corresponds to a frame length of 256 ms. A overlap of $\frac{2}{3}$ of the frame length was chosen. The frame length of 512 was chosen in order to describe the sound files as nuanced as possible but still limit the number of frames because this has a great effect on the required time for learning the models. The choice was therefore based on keeping the number of frames down such that the models can be learned in the time frame of this project. The overlap of $\frac{2}{3}$ was based on [26]. When dividing the signal into 512 frames with an overlap of $\frac{2}{3}$ there will be 22 frames, so $T = 22$ in this case.

The recordings in the data set contain a mixture of non-biological noise, up-calls, and other whale calls [3]. A spectrogram for three audio files are shown in Figure 2.10. The spectrogram in Figure 2.10(a) contains an up-call, and the spectrogram in Figure 2.10(b) contains noise but no up-call. The spectrogram in Figure 2.10(c) contains an up-call like sound. It is, however, not an up-call, so the audio file for the spectrogram is negative. The positive recordings can contain sounds that also appear in the negative recordings and vice verse, except for the up-calls which only appear in the positive records. Figure 2.10(a) and Figure 2.10(c) illustrates however that a negative labeled recording can contain signal that is very similar to an up-call. The average spectrogram for each class are shown in Figure 2.11 where Figure 2.11(a) shows the average spectrogram for all the positive labeled audio recordings, and Figure 2.11(b) shows the average spectrogram for all the negative labeled audio recordings.

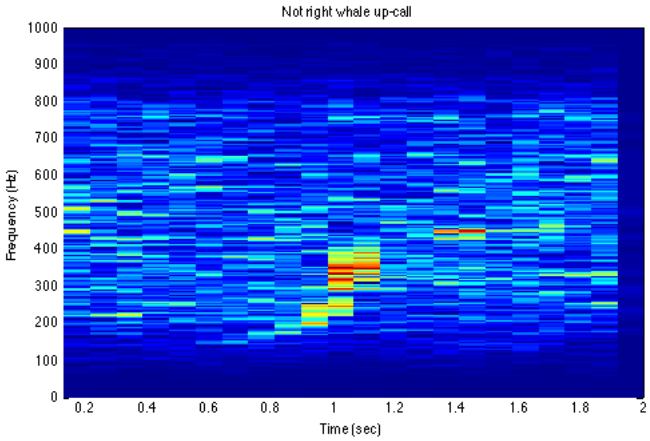
From the spectrogram the MFCCs can be found, but in order to do this we must decide the number of filters in the filter bank. This determine the frequency area of the filters because the



(a) A spectrogram of an audio that are labeled positive.

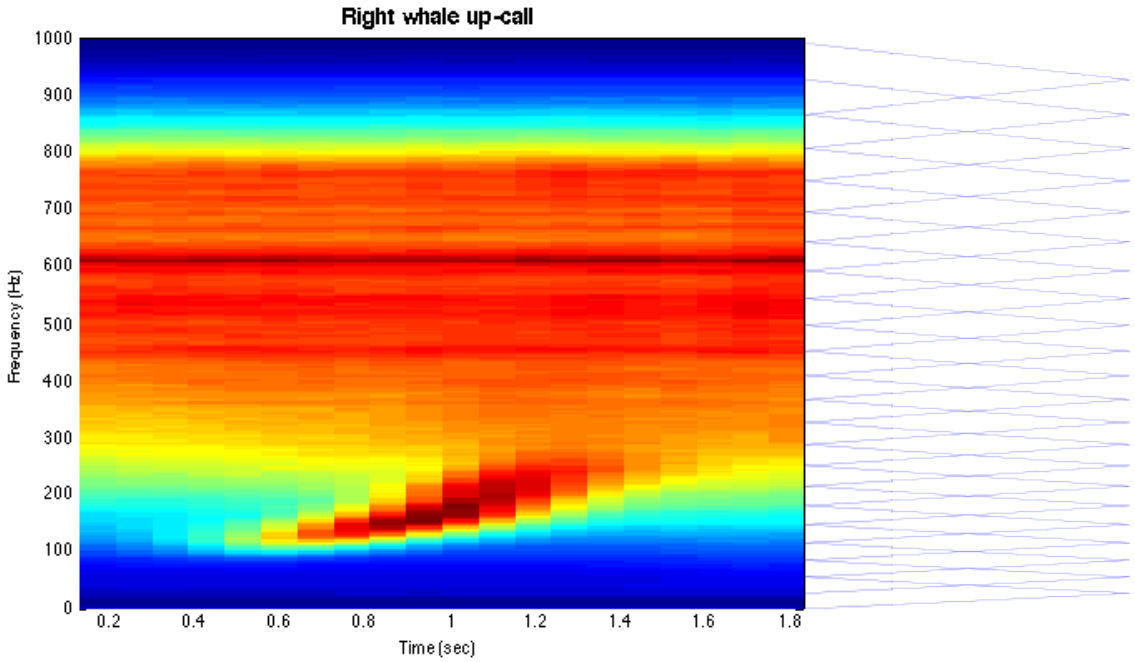


(b) A spectrogram of an audio that are labeled negative containing severe amount of noise.

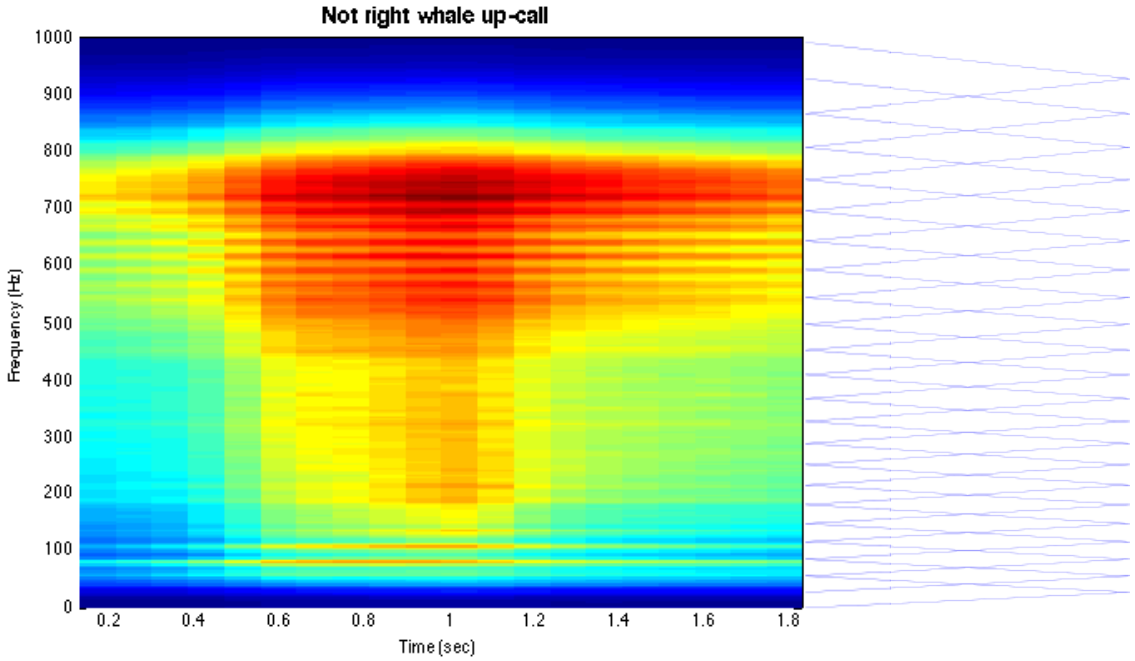


(c) A spectrogram of an audio that are labeled negative containing an up-call like signal.

Figure 2.10: Three spectrograms where frame length is 512 samples, and overlap is $\frac{2}{3}$ of frame length.



(a) The average spectrogram of all audio recordings that are labeled positive.



(b) The average spectrogram of all audio recordings that are labeled negative.

Figure 2.11: Average spectrograms with the used filter bank append to the vertical axis.

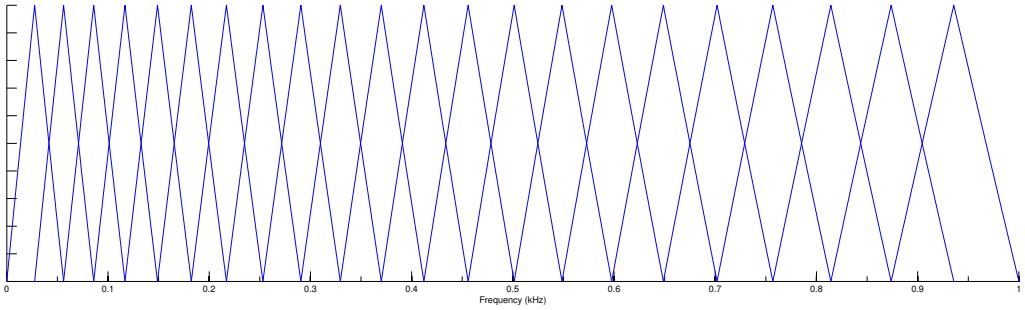


Figure 2.12: The filter bank used for extracting features.

filter bank covers the entire frequency spectrum; here from 0 to 1000 Hz. The filter bank that was used for the work of this thesis had 22 filters and is illustrated in Figure 2.12. Further we must decide the number of MFCCs to keep and use as feature data. For making these decisions it can be useful to look at the average spectrogram for all the audio files in both the positive class and negative class. These are shown in Figure 2.11 with our filter bank illustrated on the vertical axis. From this it can be seen that the up-calls are normally located below 400 Hz, and looking at Figure 2.12 it can be seen that by choosing the first 12 filter banks this area is covered. We therefore chose the number of MFCCs for each frame to be $D = 12$.

2.3 Data cleaning

Some cleaning of the available data set was necessary. We had two remove both outliers and duplicates before the available data was partitioned into subset. A few audio files were found to be outliers. The relative likelihood of observing some feature vectors from these audio files was so small that it came too close to zero, and we could not use them in our models. The content of the audio files for these feature vectors was examined by listening to it, and viewing them as a spectrogram plot. They all seemed to contain some strong interferes. In total 7 audio files were found to be ouliers where 3 of these were labeled as containing an up-call. An example of an positive audio file that was found to be an outlier is shown in figure Figure 2.13.

Doing the Kaggle competition [3] the host announced on the official forum [4] that the public training data set did contain some duplicates and they therefore released a python script that the participants could use to remove the duplicates.

In total was 646 audio files removed from the available data either because they was outliers or duplicates. The data set was therefore reduced to 29354 audio files.

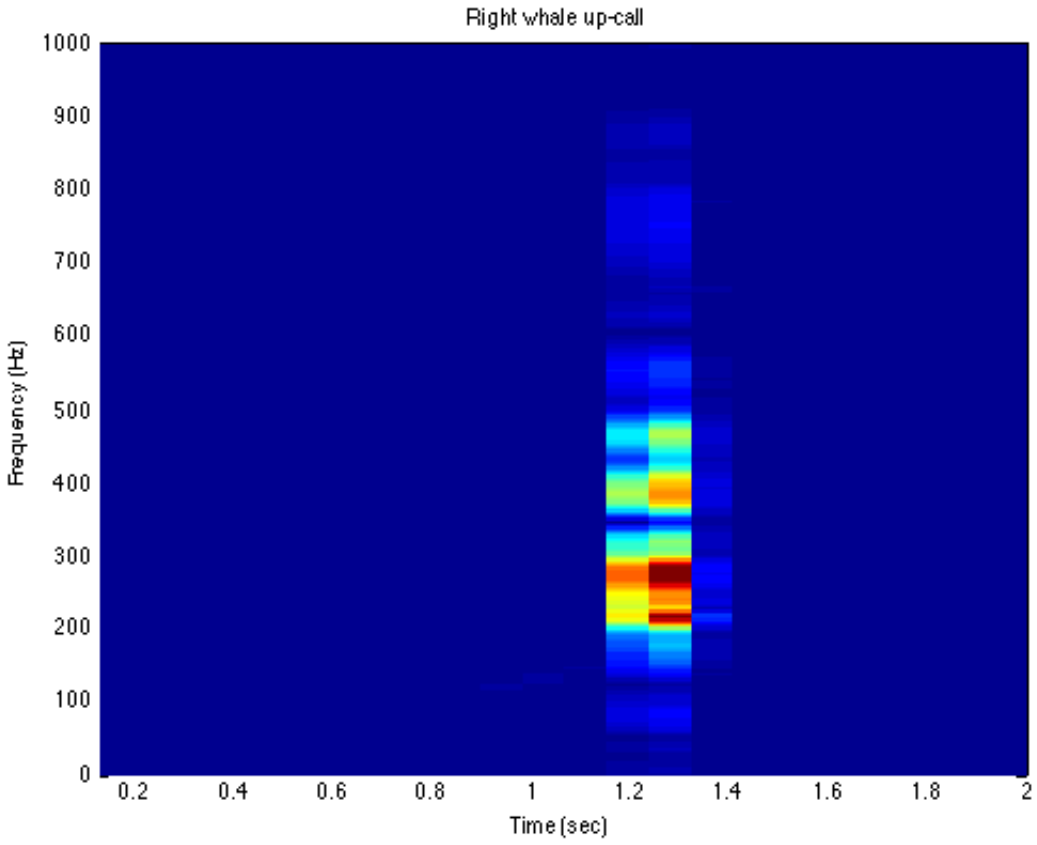


Figure 2.13: Spectrogram for an outlier audio file.

3 Models

The feature data for an audio file is a feature vector for each frame where the vectors consists of the MFCCs for the particular frame. From this the audio file must be classified which means that it must be determined whether a given audio file is positive or negative, i.e. whether it contains an up-call or does not contain an up-call, respectively. In order to do this, a classifier must be made [15]. In this project, this is addressed by using two model types; GMMs and HMMs. A model is then used to represent the positive labeled data while another model is used to represent the negative labeled data. This chapter gives a description of the two model types, and how they can be used to represent the feature vectors in this problem. The GMM is explained in Section 3.1, and the HMM is explained in Section 3.2.

3.1 Gaussian Mixture Model

We can consider the feature data of the audio files in each class as being generated by a probability distribution. An audio file can then be classified by considering the probability that the distribution had generated the feature data of the audio file. Thus by using a probability distribution as a model to represent each class, we can infer about the class of a particular audio file [30]. Let D be the number of MFCCs for each frame, i.e. D is the size of the feature vectors. It is assumed that when plotting the feature vectors in the space of $\mathbb{R}^D$, there are one or several areas where most of the feature vectors are placed. These areas are called clusters, and if there

are more than one of these areas it may be assumed that the feature vectors have been generated by a *mixture distribution* [30]. A mixture distribution consists of several *components*, one for each cluster, each of which is a distribution for the members of the particular cluster. For continuous feature vectors with more than one dimension, the multivariate Gaussian distribution is normally used as distribution for the components. The mixture distribution is then called a Gaussian Mixture Model (GMM). The multivariate Gaussian distribution is explained in Section 3.1.1 and the GMM is explained further in Section 3.1.2. How GMMs is used for representing the classes in this project is explained in Section 3.1.3.

3.1.1 Multivariate Gaussian Distribution

A multivariate Gaussian distribution is an extension of the normal distribution to a space with dimensions $D \geq 1$. It is therefore a distribution over the continues space $\mathbb{R}^D$ and it is defined by [28, 30]:

Definition 3.1.1 (Probability density function of multivariate Gaussian distribution)**.**

$$P(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^D |\boldsymbol{\Sigma}|}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}$$

where $\boldsymbol{\mu}$ is the mean vector, and $\boldsymbol{\Sigma}$ is the covariance matrix.

All vectors in this project are considered as being column vectors, so $\mathbf{x} - \boldsymbol{\mu}$ is a column vector and $(\mathbf{x}-\boldsymbol{\mu})^T$ is a row vector. A multivariate Gaussian distribution is described by its mean vector and covariance matrix, so by setting these parameters a multivariate Gaussian distribution is defined. A multivariate Gaussian distribution is denoted as $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, and the probability density function is thus denoted as $P(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

The mean vector $\boldsymbol{\mu}$ can be considered as the center of a mass where $P(\mathbf{x})$ is the amount of mass concentrated at $\mathbf{x}$ [15]:

Definition 3.1.2 (Mean vector)**.**

$$\boldsymbol{\mu} = E[\mathbf{x}] = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_D \end{bmatrix} = \begin{bmatrix} E[x_1] \\ E[x_2] \\ \vdots \\ E[x_D] \end{bmatrix} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \mathbf{x} p(\mathbf{x}) \mathrm{d}\mathbf{x}$$

The covariance matrix is defined from the covariances σ_{ij} between each dimension i and j. When $i = j$, this is the variances σ_i^2 [15]. The variance for dimension i is defined by:

Definition 3.1.3 (Variances)**.**

$$\sigma_i^2 = E[(x_i - \mu_i)^2] = \int_{-\infty}^{\infty} (x_i - \mu_i)^2 p(x_i) \mathrm{d}x_i \quad i = 1, 2, \ldots, D$$

The variance σ_i^2 is the square of the standard deviation σ_i which is a measure of how far the observations x_i is likely to derive from the mean μ_i. The covariance between the dimension i and j is defined by:

Definition 3.1.4 (Covariances)**.**

$$\sigma_{ij} = E[(x_i - \mu_i)(x_j - \mu_j)] = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (x_i - \mu_i)(x_j - \mu_j) p(x_i, x_j) \mathrm{d}x_i x_j \quad i, j = 1, 2, \ldots, D$$

Note that from Definition 3.1.3 and Definition 3.1.4 it follows that $\sigma_{ij} = \sigma_{ji}$ and $\sigma_{ii} = \sigma_i^2$. The covariance matrix $\mathbf{\Sigma}$ is then defined by:

Definition 3.1.5 (Covariance matrix)**.**

$$\begin{aligned}
\mathbf{\Sigma} &= E[(\mathbf{x} - \boldsymbol{\mu})(\mathbf{x} - \boldsymbol{\mu})^T] \\
&= \begin{bmatrix}
E[(x_1 - \mu_1)(x_1 - \mu_1)] & E[(x_1 - \mu_1)(x_2 - \mu_2)] & \cdots & E[(x_1 - \mu_1)(x_D - \mu_D)] \\
E[(x_2 - \mu_2)(x_1 - \mu_1)] & E[(x_2 - \mu_2)(x_2 - \mu_2)] & \cdots & E[(x_2 - \mu_2)(x_D - \mu_D)] \\
\vdots & \vdots & \ddots & \vdots \\
E[(x_D - \mu_D)(x_1 - \mu_1)] & E[(x_D - \mu_D)(x_2 - \mu_2)] & \cdots & E[(x_D - \mu_D)(x_D - \mu_D)]
\end{bmatrix} \\
&= \begin{bmatrix}
\sigma_1^2 & \sigma_{12} & \cdots & \sigma_{1D} \\
\sigma_{21} & \sigma_{22}^2 & \cdots & \sigma_{2D} \\
\vdots & \vdots & \ddots & \vdots \\
\sigma_{D1} & \sigma_{D2} & \cdots & \sigma_D^2
\end{bmatrix}
\end{aligned}$$

The values of the entries in the covariance matrix determine the orientation and slope of the

multivariate Gaussian distribution [15]. The effect is illustrated in Figure 3.1 where the contours of three multivariate Gaussian distributions for $D = 2$ are shown. If we for two dimensions i and j, $i \neq j$, have $\sigma_{ij} > 0$ then the multivariate Gaussian distribution will increase in dimension j when it increases in dimension i. This effect is illustrated by the multivariate Gaussian distribution to the left in Figure 3.1. If $\sigma_{ij} = 0$, the dimensions are said to be uncorrelated, and the multivariate Gaussian distribution will stay the same in dimension j when it increases in dimension i. This effect is illustrated by the multivariate Gaussian distribution in the middle in Figure 3.1. If $\sigma_{ij} < 0$ the multivariate Gaussian distribution will decrease in dimension j when it increases in dimension i. This effect is illustrated by the multivariate Gaussian distribution to the right in Figure 3.1. The degree of the slope is determined by the variance i.e. the diagonal of the covariance matrix.

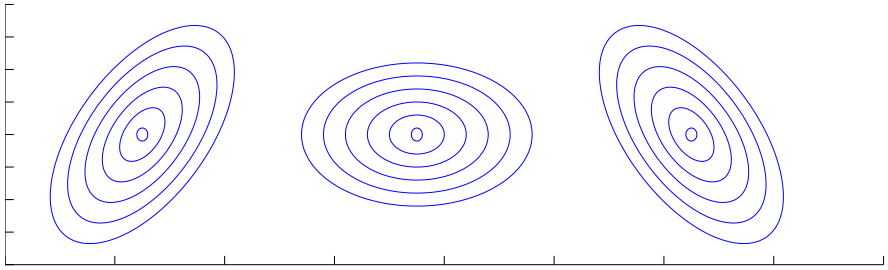


Figure 3.1: The effect of the covariance matrix.

Since $\sigma_{ij} = \sigma_{ji}$ the covariance matrix must be symmetric when a multivariate Gaussian distribution is defined. Further it it also required that $\boldsymbol{\Sigma}$ is positive semi-definite which means that for every vector $\mathbf{w} \in \mathbb{R}^D$ then $\mathbf{w}^T\boldsymbol{\Sigma}\mathbf{w} \geq 0$. This can be seen from the following [21]:

$$\begin{aligned}
\mathbf{w}^T\boldsymbol{\Sigma}\mathbf{w} &= \mathbf{w}^T \cdot \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty}(\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu})^T p(\mathbf{x})\mathrm{d}\mathbf{x}\cdot\mathbf{w} \\
&= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty}\mathbf{w}^T(\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu})^T p(\mathbf{x})\mathbf{w}\mathrm{d}\mathbf{x} \\
&= E[\mathbf{w}^T(\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu})^T\mathbf{w}]
\end{aligned}$$

For the last expression we have that $\mathbf{w}^T(\mathbf{x}-\boldsymbol{\mu}) = (\mathbf{x}-\boldsymbol{\mu})^T\mathbf{w} = (\mathbf{x}-\boldsymbol{\mu})\cdot\mathbf{w}$,where $\cdot$ is the inner product, so the expression becomes $E[((\mathbf{x}-\boldsymbol{\mu})\cdot\mathbf{w})^2]$. Since the result of the inner product is a scalar, the expression is the expected value of a squared scalar, and this is always greater than or equal to zero.

For a multivariate Gaussian distribution it is further required that $\boldsymbol{\Sigma}$ is non-singular because else the variance will be zero. If the $\boldsymbol{\Sigma}$ is non-singular it also follows that the matrix is invertible and the determinant of $\boldsymbol{\Sigma}$ is different from 0 [20] which is required by Definition 3.1.1.

3.1.2 Mixture of Gaussians

A mixture distribution is a distribution with K components each of which is a distribution [28, 30]. Each component k is given a weight $0 \leq w_k \leq 1$ such that $\sum_{k=1}^{K} w_k = 1$. Let c_k be component k. The mixture distribution is then defined by:

Definition 3.1.6 (Probability density function for a mixture distribution)**.**

$$P(\mathbf{x}) = \sum_{k=1}^{K} w_k P(\mathbf{x}|c_k)$$

The weights w_k ensures that $\int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} P(\mathbf{x}) \mathrm{d}\mathbf{x} = 1$.

For continuous data with more than one dimension the multivariate Gaussian distribution is normally used as components. In this case the mixture distribution is denoted as a Gaussian Mixture Model (GMM), which can be described with the parameters $\lambda = ((w_1, \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1), (w_2, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2), \ldots, (w_K, \boldsymbol{\mu}_K, \boldsymbol{\Sigma}_K))$. The probability density function for the GMM is then:

$$P(\mathbf{x}) = \sum_{k=1}^{K} w_k P(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{3.1}$$

where $P(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ is given by Definition 3.1.1.

3.1.3 Representing our feature data using Gaussian Mixture Models

In this thesis we present two approaches where GMMs are used for representing the feature data for each audio file.

3.1.3.1 Representation using a single frame

Each audio file can be interpreted as a single frame with the length of the audio file such that the feature data for an audio file is a single feature vector with 12 MFCCs. This data can then be considered as drawn from the distribution of a GMM. Thus a GMM can be used to represent both classes.

Before the GMM can be learned from data, the number of components K must be chosen. We however do not know how many clusters the feature vectors form, so in order to decide on the number of components to use, we learn several GMMs with different number of components for both the positive and the negative class. For the classification system in Figure 1.1 the best

combination of a model for the positive class and a model for the negative class can then be determined by classifying the files in a validation data set.

3.1.3.2 Representation using several frames

An audio file can be divided into several frames, as explained in Chapter 2, and thereby provide feature data that describe the audio files more nuanced. Let each audio file be divided into T frames then the data $\mathbf{x}_{1:T}$ for an audio file consists of T feature vectors, $\mathbf{x}_{1:T} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T)$. It can then be assumed that the feature vector for frame t for all audio files in a class is generated by the same GMM. Let λ_t be the parameters for the GMM of frame t then such a system could be considered as a model with the parameters $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_T)$. By assuming that the frames are independent, the probability for the feature data of an audio file is then:

$$P(\mathbf{x}_{1:T}) = \prod_{t=1}^{T} P(\mathbf{x}_t) = \prod_{t=1}^{T} \sum_{k=1}^{K} w_k P(\mathbf{x}_t | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{3.2}$$

Again the number of components K must be chosen, and the parameters of the GMMs can be learned from data. We let all the GMMs for each frame have the same number of components. Then, as in Section 3.1.3.1, the GMMs for both the positive and the negative class are learned where the number of components are varied. For the classification system in Figure 1.1 the best combination of a model for the positive class and a model for the negative class can then be determined by classifying the files in a validation data set.

3.2 Hidden Markov Model

As it can be seen in Figure 2.11(a) the digital signal for an up-call has a certain pattern. The digital signal of an audio file can be considered as being generated by an underlying process which is in a certain state for each frame. At the beginning of each frame, the underlying process can change state or possibly stay in the same state. Because an up-call has a certain pattern, it is reasonable to assume that the state of the underlying process for a certain frame depends on the states for the previous frames. We still assume that the feature vector of a frame is generated by a GMM. The component which generated the feature vector is then dependent on the state of underlying process. It is however not possible to observe the state of the process nor the component that generated a particular feature vector, but the feature vector of a frame is observable. Such a system can be represented by a Hidden Markov Model (HMM) with a GMM for each state.

In this chapter HMMs are introduced, and it is presented how HMMs are used in this thesis. An HMM is a special kind of Bayesian Network. We therefore give a description of Bayesian

Networks in Section 3.2.1. In Section 3.2.2 a general formalized description of HMMs is given, and the presentation of how HMMs is used in this thesis is given in Section 3.2.3. Section 3.2.1 and Section 3.2.2 are rewritten from the description of HMMs given in [17].

3.2.1 Bayesian Network

A Bayesian Network is a data structure that represents dependencies among variables. The variables can both be discrete and continuous. It is defined as [30]:

Definition 3.2.1 (Bayesian Network). *A* Bayesian Network *consists of two parts* (G, θ)*:*

- *G is a Directed Acyclic Graph (DAG) where nodes represent variables and edges describe the dependencies between the variables.*
- *θ is a set of conditional probability distributions, one for each node, describing the conditional probabilities for each variable.*

The following applies for the variables and the conditional probabilities:

- *If a variable Y is the parent of variable node X, i.e. there is an edge from node Y to node X, then variable X depends on variable Y.*
- *The conditional probability distribution associated to X is $P(X|pa(X))$ where $pa(X)$ is the set of parent nodes of X. The distribution $P(X|pa(X))$ is then quantifying the effect of the parents on X.*
- *If X is discrete it is required that $\sum_X p(X|pa(X)) = 1$, and if X is continues it is required that $\int_X p(X|pa(X))\mathrm{d}X = 1$.*

The way that information is transmitted through the variables of a Bayesian network can be used for finding conditional independence among the variables. We exploit this in Section 4.3. Transmission of information in a Bayesian network is covered by the following three case [18]:.

Serial connection An example of a serial connection is shown in Figure 3.2. Here C is influenced directly by B and influenced by A through B. However if B is instantiated, C becomes independent of A because no evidence on A will change our belief of the state of C when B is known i.e. A and C becomes d-separated.

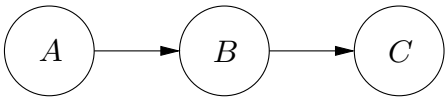


Figure 3.2: Example of a serial connection. The figure is from [18].

Diverging connection An example of a diverging connection is shown in Figure 3.3. If the variable A is instantiated then the children of A become independent i.e. they are d-separated. Otherwise information can pass through A and the children influence each other.

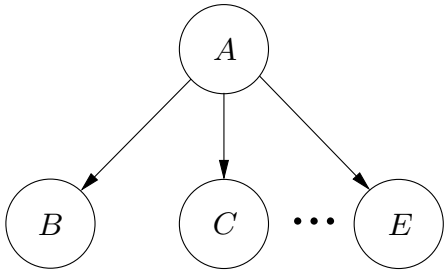


Figure 3.3: Example of a diverging connection. The figure is from [18].

Converging connection An example of a converging connection is shown on Figure 3.4. Here the parents of A are independent unless A changes certainty i.e. information is given that change our belief of the state of variable A. This information can be given if A or any of A's descendants is instantiated.

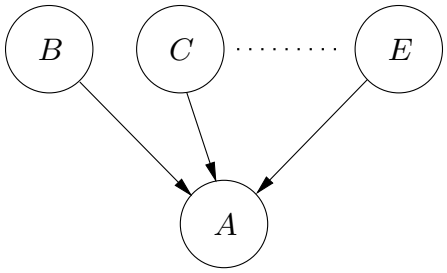


Figure 3.4: Example of a converging connection. The figure is from [18].

Following the rules of d-separation, it is possible to decide if any pair of variables in a Bayesian Network are independent given the evidence entered into the network. The d-separation rules express conditional independence between the variables of the network; if two variables A and B are d-separated because of some other variable C then they are conditionally independent given C because when C is known no knowledge of B will alter the probability of A.

3.2.2 The structure of an Hidden Markov Model

An HMM can be seen as a special type of Bayesian Network that models a system which evolves over time where time is divided into intervals denoted by a discrete time stamp t each refereed to as a time slice [18, 30]. For each time slice there is a discrete state variable S_t. This variable can be hidden which means that it cannot be directly observed i.e. we have no evidence for the variables S_t. This is the case in this thesis. Further, for each time slice there is one or more observation variables which can be either discrete or continuous and do not need to

have the same state space. These variables are observable and can therefore be instantiated when doing inference in the HMM. The HMMs used in this thesis have exactly one continuous multidimensional observation variable $\mathbf{X}_t$ for each time slice. Thus, HMMs with more than one observation variable in each time slice is not discussed.

There are three assumptions underlying an HMM [30]:

1. The observations $\mathbf{X}_t$ at time t are generated by some process that is always in some state, we will refer to this as the Markov process. This state is the value of the hidden state variable S_t which means that the observation variable for each time slice t depends on the hidden state variable S_t.

2. The Markov assumption which states that the present is only dependent on a finite history of previous states. The most simple form of this is the *first-order Markov process* where the given state only depends on the previous state. I.e., given the value of S_{t-1}, the current state S_t is independent of states prior to S_{t-1}. The HMM in Figure 3.5 is a first-order Markov process.

3. The model is stationary: The probability of switching to a particular state, and the probability of a particular observation when in a given state, does not change over time, so the conditional probability distributions is the same for all time slices t.

An example of a DAG for an HMM with one observation variable for each time slice is shown in Figure 3.5.

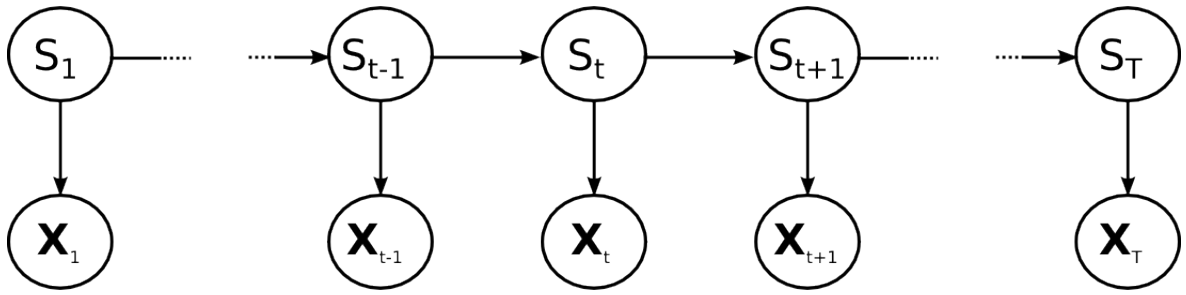


Figure 3.5: A Hidden Markov Model with one observation variable.

The HMM in Figure 3.5 has the following conditional probability distributions:

1. The *transition* conditional probability distribution $P(S_t|S_{t-1})$. This are the probabilities of for the transition between the states of the HMM. As expressed by the graph in Figure 3.5 the current state is only dependent on the state of the previous time slice. Because an HMM is stationary this distribution is the same for all t.

2. The *emission* conditional probability distribution $P(\mathbf{X}_t|S_t)$. As expressed by the graph in Figure 3.5, the observable variable $\mathbf{X}_t$ for each time slice t depend only on the state S_t. Because an HMM is stationary this distribution is the same for all t.

3. The prior distribution $P(S_1)$, this will be refereed to as the *start* distribution in the contents of HMM.

3.2.3 Representing our feature data using Hidden Markov Models

For the problem addressed by this thesis each time slice of the HMM represents a frame of the discrete signal in an audio file, so further on both the phrase *time slice* and *frame* is used to denote a frame of the audio file. There is a single observable multidimensional variable in each time slice which is the feature vector for the associated frame. The feature vectors are considered to be generated by GMMs, so there is a GMM of each state. Let C_t be the component of the GMM which generated the feature vector observed at time t, a time slice for the HMM is then given by Figure 3.6.

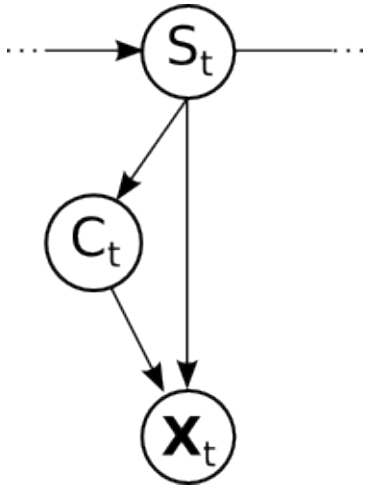


Figure 3.6: Time slice t of the HMM for the problem in this thesis.

Such an HMM has the conditional probability distributions $P(S_1)$, $P(S_t|S_{t-1})$, $P(C_t|S_t)$ and $P(\mathbf{X}_t|S_t, C_t)$. $P(S_1)$ and $P(S_t|S_{t-1})$ are, respectively, the start and transition conditional probability distribution introduced earlier. $P(C_t|S_t)$ is the weight of component C_t of the GMM for state S_t of the HMM, and $P(\mathbf{X}_t|S_t, C_t)$ is the probability of an observation given the component and state to time t which is determined by Definition 3.1.1. We thus have

$$P(C_t = k|S_t = n) = w_{nk} \tag{3.3}$$

$$P(\mathbf{X}_t = \mathbf{x}_t|S_t = n, C_t = k) = P(\mathbf{x}_t|\boldsymbol{\mu}_{nk}, \boldsymbol{\Sigma}_{nk}) \tag{3.4}$$

The emission conditional probability distribution can then be found the following way:

$$\begin{aligned} P(\mathbf{x}_t|S_t = n) &= \sum_{k=1}^{K} P(C_t = k, \mathbf{x}_t|S_t = n) \quad \text{Marginalize} \\ &= \sum_{k=1}^{K} P(\mathbf{x}_t|S_t = n, C_t = k)P(C_t = k|S_t = n) \quad \text{Fund. rule} \end{aligned}$$

From Equation 3.3 and Equation 3.4 we then get that:

$$P(\mathbf{x}_t|S_t = n) = \sum_{k=1}^{K} w_{nk} P(\mathbf{x}_t|\boldsymbol{\mu}_{nk}, \boldsymbol{\Sigma}_{nk}) \tag{3.5}$$

By comparing this with Equation 3.1, it is seen that this probability therefore is the probability density function for the GMM to a given state. Let N be the number of states of the Markov process. The parameters of this HMM are therefore $\lambda = (P(S_1), P(S_t|S_{t-1}), (w_k, \boldsymbol{\mu}_{nk}, \boldsymbol{\Sigma}_{nk})_{n=1,k=1}^{N,K})$.

4

Learning

For a model with parameters λ, we would like determine the values of λ such that the model represent the observed feature data as good as possible. Thus we would like to learn the parameters from the observed features. A general method for learning the parameters of a model is introduced in Section 4.1. Section 4.2 and Section 4.3 explain how this method is applied to GMMs and HMMs, respectably.

4.1 General introduction to learning

When it is possible to observe all variables in a model, the model parameters which best fit the observed features can be found by determining the maximum likelihood of the parameters given the observations. This is explained in Section 4.1.1. However, the method presented in Section 4.1.1 can not always be used e.g. if not all data can be observed. Instead can the EM-algorithm be used to learn the model parameters. This is introduced in Section 4.1.2.

4.1.1 Maximum likelihood [15]

Let R be the number of audio files, and let $\mathbf{x}^{(r)}$ be the feature vectors of the frames for audio file r. Then the observable data are $\mathcal{X} = (\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(R)})$. It is assumed that each $\mathbf{x}^{(r)}$ is generated by the same underlying model with parameters λ, thus they are identical distributed. It is also assumed that the data samples are independent from each other. The data samples

$\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(R)}$ are then said to be independently and identically distributed (i.i.d), and the likelihood of the model parameters λ given the data $\mathcal{X}$ can then be found by Equation 4.1.

$$P(\mathcal{X}|\lambda) = \prod_{r=1}^{R} P(\mathbf{x}^{(r)}|\lambda) \tag{4.1}$$

Normally the log-likelihood is used instead of the likelihood such that the product is replaced with a sum which is computationally safer. This gives us Equation 4.2.

$$\log P(\mathcal{X}|\lambda) = \log \prod_{r=1}^{R} P(\mathbf{x}^{(r)}|\lambda) = \sum_{r=1}^{R} \log P(\mathbf{x}^{(r)}|\lambda) \tag{4.2}$$

For example if our model is a multivariate Gaussian distribution, the parameters of the model would be $\lambda = (\boldsymbol{\mu}, \boldsymbol{\Sigma})$. The likelihood $P(\mathbf{x}^{(r)}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ can then be found using Definition 3.1.1, and $\log P(\mathcal{X}|\lambda)$ can be determined by:

$$\log P(\mathcal{X}|\lambda) = \sum_{r=1}^{R} \log \left(\frac{1}{\sqrt{(2\pi)^D |\boldsymbol{\Sigma}|}} e^{-\frac{1}{2}(\mathbf{x}^{(r)}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x}^{(r)}-\boldsymbol{\mu})} \right)$$

where $\mathbf{x}^{(r)}$ and $\boldsymbol{\mu}$ are column vectors. The values of λ are however not known, but we would like to find some values for λ which maximizes $\log P(\mathcal{X}|\lambda)$ i.e. we want to determine λ^* in Equation 4.3.

$$\lambda^* = \arg\max_{\lambda} \log P(\mathcal{X}|\lambda) \tag{4.3}$$

The values of λ^* can be found by optimizing the expression of $\log P(\mathcal{X}|\lambda)$. For the multivariate Gaussian distribution this is done by finding the partial derivatives of $\log P(\mathcal{X}|\lambda)$ with respect to the parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, setting the expressions equal to 0, and solving the equations for $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ [15]:

$$\frac{\partial}{\partial \boldsymbol{\mu}} \log P(\mathcal{X}|\lambda) = 0 \Rightarrow \widehat{\boldsymbol{\mu}} = \frac{1}{R} \sum_{r=1}^{R} \mathbf{x}^{(r)}$$

$$\frac{\partial}{\partial \boldsymbol{\Sigma}} \log P(\mathcal{X}|\lambda) = 0 \Rightarrow \widehat{\boldsymbol{\Sigma}} = \frac{1}{R} \sum_{r=1}^{R} (\mathbf{x}^{(r)} - \widehat{\boldsymbol{\mu}})(\mathbf{x}^{(r)} - \widehat{\boldsymbol{\mu}})^T$$

There are however cases where this method can not be used e.g. if the model has some hidden variables.

4.1.2 The Expectation-Maximization algorithm

It may not always be possible to find the maximum likelihood parameters for example when a model has hidden variables, or if not all data have been observed. Instead the EM-algorithm can be used to estimate the parameters of the model [10, 15, 30]; a method which was originally proposed in [14]. Let $\mathcal{X}$ be the observable variables and let $\mathcal{Y}$ be the hidden variables such that $\mathcal{X}$ and $\mathcal{Y}$ together includes all the variables of the model. $P(\mathcal{X}|\lambda)$ is then called the *incomplete-data likelihood*, and $P(\mathcal{X},\mathcal{Y}|\lambda)$ is called the *complete-data likelihood*. The EM-algorithm consists of two steps [10, 15]. In the first step it is assumed that the values of the parameters are known by assigning values to the parameters, and then the expected value of the complete-data log-likelihood given the initialized parameters and the observed data is found. This step is called the *expectation step*. In the second step the values of the parameters which maximizes the expected value of the complete-data log-likelihood is found. This step is called the *maximization step*. These values are then used for the parameters in the next expectation step. The two steps are iterated until convergence.

The expectation step First it is assumed that λ is known by assigning values to the parameters. This is denoted as $\lambda^{(i-1)}$. Then the expected value of $\log P(\mathcal{X},\mathcal{Y}|\lambda)$ given $\mathcal{X}$ and $\lambda^{(i-1)}$, denoted $Q(\lambda,\lambda^{(i-1)})$, is found:

$$\begin{aligned} Q(\lambda,\lambda^{(i-1)}) &= E[\log P(\mathcal{X},\mathcal{Y}|\lambda)|\mathcal{X},\lambda^{(i-1)}] \\ &= \sum_{\mathcal{Y}} \log(P(\mathcal{X},\mathcal{Y}|\lambda))P(\mathcal{Y}|\mathcal{X},\lambda^{(i-1)}) \end{aligned} \tag{4.4}$$

For the models used in this thesis all the hidden variables are discrete, and the expected value is therefore calculated as a sum in the last expression of Equation 4.4. This will give an expression dependent on the parameters λ.

The maximization step Then we would like to find the parameters which maximizes the expected value of $\log P(\mathcal{X},\mathcal{Y}|\lambda)$:

$$\lambda^{(i)} = \arg\max_{\lambda} Q(\lambda,\lambda^{(i-1)}) \tag{4.5}$$

This can be done by optimizing $Q(\lambda,\lambda^{(i-1)})$ according to the parameters. $\lambda^{(i)}$ is then used to find the expected value of the log-likelihood in the next iteration.

Termination For the EM-algorithm the incomplete-data likelihood $P(\mathcal{X}|\lambda^{(i)})$ will converge toward a local maximum or saddle point which means that $P(\mathcal{X}|\lambda^{(i)}) \geq P(\mathcal{X}|\lambda^{(i-1)})$ [35]. This fact can be used to determine when to terminate. For example if the absolute difference

between $P(\mathcal{X}|\lambda^{(i)})$ and $P(\mathcal{X}|\lambda^{(i-1)})$ becomes below a threshold ϵ:

$$P(\mathcal{X}|\lambda^{(i)}) - P(\mathcal{X}|\lambda^{(i-1)}) \leq \epsilon \tag{4.6}$$

Alternatively the relative difference can be used. Hence, the algorithm will terminate when:

$$\frac{P(\mathcal{X}|\lambda^{(i)}) - P(\mathcal{X}|\lambda^{(i-1)})}{P(\mathcal{X}|\lambda^{(i-1)})} \leq \epsilon \tag{4.7}$$

As earlier mention log-likelihood is often used instead of likelihood, and then the relative difference termination criterion becomes:

$$\frac{\log P(\mathcal{X}|\lambda^{(i)}) - \log P(\mathcal{X}|\lambda^{(i-1)})}{|\log P(\mathcal{X}|\lambda^{(i-1)})|} \leq \epsilon \tag{4.8}$$

Either way we must determine the likelihood or the log-likelihood of $\lambda^{(i)}$ and $\lambda^{(i-1)}$ given $\mathcal{X}$.

4.2 Gaussian Mixture Model

As mentioned in Section 3.1 a GMM has the parameters $\lambda = ((w_1, \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1), (w_2, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2), \ldots, (w_K, \boldsymbol{\mu}_K, \boldsymbol{\Sigma}_K))$, where K is the number of components. Assuming that the feature vector $\mathbf{x}$ has been generated by a GMM with parameters λ then the likelihood of λ given $\mathbf{x}$ can be found using Equation 3.1:

$$P(\mathbf{x}|\lambda) = \sum_{k=1}^{K} w_k P(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{4.9}$$

So for all the feature vectors $\mathcal{X} = (\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(R)})$ generated by the same GMM with the parameters λ, the log-likelihood of λ given $\mathcal{X}$ is [10]:

$$\log P(\mathcal{X}|\lambda) = \sum_{r=1}^{R} \log \sum_{k=1}^{K} w_k P(\mathbf{x}^{(r)}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{4.10}$$

This is hard to optimize because it contains a logarithm of a sum, so the parameters is found using the EM-algorithm [10]. This is presented in Section 4.2.1. For the first iteration, we need some initial values for λ. How these are chosen is explained in Section 4.2.2. In Section 4.2.3 an illustrative example of the effect of the EM-algorithm for GMM is presented.

When representing the audio files as a single single frame there is only one feature vector per audio file and therefore only one GMM to represent the class. The log-likelihood can therefore be found using Equation 4.10 and the parameters can be learned as explained in Sec-

tion 4.2.1. When representing the audio files as several frames there is a GMM for each frame. Let λ_t be the parameters for the GMM for frame t such that the system has the parameters $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_T)$. Each λ_t is learned separately as it is presented in Section 4.2.1. The log-likelihood of λ can then be found using Equation 3.2:

$$\begin{aligned}\log P(\mathcal{X}|\lambda) &= \sum_{r=1}^{R} \log \prod_{t=1}^{T} P(\mathbf{x}_t^{(r)}) \\ &= \sum_{r=1}^{R} \sum_{t=1}^{T} \log P(\mathbf{x}_t^{(r)}) \\ &= \sum_{r=1}^{R} \sum_{t=1}^{T} \log \sum_{k=1}^{K} w_k P(\mathbf{x}_t^{(r)}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\end{aligned} \tag{4.11}$$

4.2.1 Estimation of parameters

The feature vector $\mathbf{x}$ can be observed, so the observable data are $\mathcal{X} = (\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(R)})$. It is however not possible to observe which component that generated the feature vector $\mathbf{x}^{(r)}$. Let $C^{(r)} \in \{1, \ldots, K\}$ be a hidden variable where $C^{(r)} = k$ if $\mathbf{x}^{(r)}$ is generated by component k. We then have the hidden variables $C = \{C^{(1)}, \ldots, C^{(R)}\}$. The steps of the EM-algorithm for finding the parameters of the GMM can then be determined [10, 30].

The expectation step First the expected value of $\log P(\mathcal{X}, C|\lambda)$ given $\mathcal{X}$ and $\lambda^{(i-1)}$ is found using Equation 4.12. How the values of $\lambda^{(i-1)}$ for the first iteration are found is explained in Section 4.2.2.

$$\begin{aligned}Q(\lambda, \lambda^{(i-1)}) &= E[\log P(\mathcal{X}, C|\lambda)|\mathcal{X}, \lambda^{(i-1)}] \\ &= \sum_{r=1}^{R} \sum_{k=1}^{K} \log(P(\mathbf{x}^{(r)}, C^{(r)} = k|\lambda)) P(C^{(r)} = k|\mathbf{x}^{(r)}, \lambda^{(i-1)})\end{aligned} \tag{4.12}$$

We will handle the two terms $P(\mathbf{x}^{(r)}, C^{(r)} = k|\lambda)$ and $P(C^{(r)} = k|\mathbf{x}^{(r)}, \lambda^{(i-1)})$ handle separately.

The term $P(\mathbf{x}^{(r)}, C^{(r)} = k|\lambda)$ can be rewritten using the fundamental rule:

$$P(\mathbf{x}^{(r)}, C^{(r)} = k|\lambda) = P(\mathbf{x}^{(r)}|C^{(r)} = k, \lambda) P(C^{(r)} = k|\lambda)$$

Here $P(\mathbf{x}^{(r)}|C^{(r)} = k, \lambda)$ is the probability that $\mathbf{x}^{(r)}$ was generated by component k i.e. $P(\mathbf{x}^{(r)}|C^{(r)} = k, \lambda) = P(\mathbf{x}^{(r)}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$, and $P(C^{(r)} = k|\lambda)$ is the weight of component k, i.e. $P(C^{(r)} = k|\lambda) = w_k$. The term $P(\mathbf{x}^{(r)}, C^{(r)} = k|\lambda)$ therefore becomes:

$$P(\mathbf{x}^{(r)}, C^{(r)} = k|\lambda) = w_k P(\mathbf{x}^{(r)}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

The term $P(C^{(r)} = k|\mathbf{x}^{(r)}, \lambda^{(i-1)})$ can be rewritten using Bayes' rule:

$$\begin{aligned} P(C^{(r)} = k|\mathbf{x}^{(r)}, \lambda^{(i-1)}) &= \frac{P(\mathbf{x}^{(r)}|C^{(r)} = k, \lambda^{(i-1)})P(C^{(r)} = k|\lambda^{(i-1)})}{P(\mathbf{x}^{(r)}|\lambda^{(i-1)})} \\ &= \varphi P(\mathbf{x}^{(r)}|C^{(r)} = k, \lambda^{(i-1)})P(C^{(r)} = k|\lambda^{(i-1)}) \end{aligned}$$

where φ is the normalization factor. Again $P(\mathbf{x}^{(r)}|C^{(r)} = k, \lambda^{(i-1)})$ is the probability that $\mathbf{x}^{(r)}$ was generated by component k, and $P(C^{(r)} = k|\lambda^{(i-1)})$ is the weight of component k. Let $w_k^{(i-1)}$, $\boldsymbol{\mu}_k^{(i-1)}$, and $\boldsymbol{\Sigma}_k^{(i-1)}$ imply that it is the values of w_k, $\boldsymbol{\mu}_k$, and $\boldsymbol{\Sigma}_k$ from $\lambda^{(i-1)}$. The term $P(C^{(r)} = k|\mathbf{x}^{(r)}, \lambda^{(i-1)})$ then becomes:

$$P(C^{(r)} = k|\mathbf{x}^{(r)}, \lambda^{(i-1)}) = \frac{w_k^{(i-1)} P(\mathbf{x}^{(r)}|\boldsymbol{\mu}_k^{(i-1)}, \boldsymbol{\Sigma}_k^{(i-1)})}{\sum_{k=1}^{K} w_k^{(i-1)} P(\mathbf{x}^{(r)}|\boldsymbol{\mu}_k^{(i-1)}, \boldsymbol{\Sigma}_k^{(i-1)})} = \varphi \cdot w_k^{(i-1)} P(\mathbf{x}^{(r)}|\boldsymbol{\mu}_k^{(i-1)}, \boldsymbol{\Sigma}_k^{(i-1)}) \tag{4.13}$$

The expected value of $\log P(\mathcal{X}, C|\lambda)$ given $\mathcal{X}$ and $\lambda^{(i-1)}$ becomes:

$$\begin{aligned} Q(\lambda, \lambda^{(i-1)}) &= \sum_{r=1}^{R}\sum_{k=1}^{K} \log(w_k P(\mathbf{x}^{(r)}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)) \cdot \varphi w_k^{(i-1)} P(\mathbf{x}^{(r)}|\boldsymbol{\mu}_k^{(i-1)}, \boldsymbol{\Sigma}_k^{(i-1)}) \\ &= \sum_{r=1}^{R}\sum_{k=1}^{K} \log(w_k) \cdot \varphi w_k^{(i-1)} P(\mathbf{x}^{(r)}|\boldsymbol{\mu}_k^{(i-1)}, \boldsymbol{\Sigma}_k^{(i-1)}) \\ &\quad + \sum_{r=1}^{R}\sum_{k=1}^{K} \log(P(\mathbf{x}^{(r)}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)) \cdot \varphi w_k^{(i-1)} P(\mathbf{x}^{(r)}|\boldsymbol{\mu}_k^{(i-1)}, \boldsymbol{\Sigma}_k^{(i-1)}) \end{aligned} \tag{4.14}$$

The maximization step $Q(\lambda, \lambda^{(i-1)})$ must then be maximized by finding the partial derivatives of w_k, $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$ for Equation 4.14, setting these equal to zero, and solve for the parameters. When maximizing w_k Lagrange multipliers are used to ensure that the weights sum to one [10]. This gives the following expressions:

$$\widehat{w_k} = \frac{1}{K}\sum_{r=1}^{R} P(C^{(r)} = k|\mathbf{x}^{(r)}, \lambda^{(i-1)}) \tag{4.15}$$

$$\widehat{\boldsymbol{\mu}_k} = \frac{\sum_{r=1}^{R} P(C^{(r)} = k|\mathbf{x}^{(r)}, \lambda^{(i-1)})\mathbf{x}^{(r)}}{\sum_{r=1}^{R} P(C^{(r)} = k|\mathbf{x}^{(r)}, \lambda^{(i-1)})} \tag{4.16}$$

$$\widehat{\boldsymbol{\Sigma}_k} = \frac{\sum_{r=1}^{R} P(C^{(r)} = k|\mathbf{x}^{(r)}, \lambda^{(i-1)})(\mathbf{x}^{(r)} - \widehat{\boldsymbol{\mu}_k})(\mathbf{x}^{(r)} - \widehat{\boldsymbol{\mu}_k})^T}{\sum_{r=1}^{R} P(C^{(r)} = k|\mathbf{x}^{(r)}, \lambda^{(i-1)})} \tag{4.17}$$

where $P(C^{(r)} = k|\mathbf{x}^{(r)}, \lambda^{(i-1)})$ is given by Equation 4.13. These must be calculated for $k = 1, \ldots, K$ for finding $\lambda^{(i)} = ((\widehat{w_1}, \widehat{\boldsymbol{\mu}_1}, \widehat{\boldsymbol{\Sigma}_1}), \ldots, (\widehat{w_K}, \widehat{\boldsymbol{\mu}_K}, \widehat{\boldsymbol{\Sigma}_K}))$. These values are then used for the parameters in the next iteration.

Termination The $\log P(\mathcal{X}|\lambda^{(i)})$ and $\log P(\mathcal{X}|\lambda^{(i-1)})$ can be found using Equation 4.10.

4.2.2 Initial values of the parameters

Initial values of the parameters λ for a GMM with K components can be chosen by first clustering in K clusters which can be done with the k-means algorithm. The weight, mean vector and covariance matrix for each component k can then be estimated from the data in the corresponding cluster [26]. The k-means algorithm used is presented in Algorithm 1.

Algorithm 1: k-means [15]

Input: Data samples $\mathcal{X} = (\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(R)})$, number of clusters K

Create K lists: $l_1, l_2, \ldots l_K$
Create K centroids: $\boldsymbol{\rho}_1, \boldsymbol{\rho}_2, \ldots, \boldsymbol{\rho}_K$
$\mathcal{X}' = \mathcal{X}$
`// Initialize centroids`
for $k = 1, \ldots, K$ **do**
    Chose random data sample $\mathbf{x}$ from $\mathcal{X}'$
    $\boldsymbol{\rho}_k = \mathbf{x}$
    $\mathcal{X}' = \mathcal{X}' \backslash \mathbf{x}$
end
repeat
    `// Put data in clusters`
    Empty the lists $l_1, l_2, \ldots l_K$
    for $r = 1, \ldots, R$ **do**
        $k' = \arg\min_{k=1,\ldots,K} \sqrt{(\mathbf{x}^{(r)} - \boldsymbol{\rho}_k)^T (\mathbf{x}^{(r)} - \boldsymbol{\rho}_k)}$
        Put $\mathbf{x}^{(r)}$ in list $l_{k'}$
    end
    `// Update centroids`
    $\mathcal{X}' = \mathcal{X}$
    for $k = 1, \ldots, K$ **do**
        if $|l_k| \leq$ *dimension of state space* **then**
            Chose random data sample $\mathbf{x}$ from $\mathcal{X}'$
            $\boldsymbol{\rho}_k = \mathbf{x}$
            $\mathcal{X}' = \mathcal{X}' \backslash \mathbf{x}$
        else
            Set $\boldsymbol{\rho}_k$ to the mean of the data in l_k
    end
until *No change of* $\boldsymbol{\mu}_k$ *for* $k = 1, \ldots, K$;

The term $\sqrt{(\mathbf{x}^{(r)} - \boldsymbol{\mu}_k)^T (\mathbf{x}^{(r)} - \boldsymbol{\mu}_k)}$ is the Euclidean distance from $\mathbf{x}^{(r)}$ to $\boldsymbol{\mu}_k$. K-means makes a list l_k and a centroid $\boldsymbol{\rho}_k$ for each cluster k. First each centroid is set to be a data point $\mathbf{x}$. Then for each data point it puts the data point in the cluster whose centroid is closest to the data point. When all data points have been put in a cluster, the centroid for each cluster is updated to the center of the points in the particular cluster. This is repeated until the centroids does not

change for an iteration.

After the k-means have been applied let $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(r_k)}$ be the data in cluster k. Then the weight, mean vector and covariance matrix of component k can be estimated from the data in a cluster by the following formulas [15, 26]:

$$\widehat{w_k} = \frac{r_k}{R} \tag{4.18}$$

$$\widehat{\boldsymbol{\mu}_k} = \frac{1}{r_k} \sum_{i=1}^{r_k} \mathbf{x}^{(i)} \tag{4.19}$$

$$\widehat{\boldsymbol{\Sigma}_k} = \frac{1}{r_k} \sum_{i=1}^{r_k} (\mathbf{x}^{(i)} - \widehat{\boldsymbol{\mu}_k})(\mathbf{x}^{(i)} - \widehat{\boldsymbol{\mu}_k})^T \tag{4.20}$$

$\widehat{w_k}$ will sum to one for all K components as required, and $\widehat{\boldsymbol{\Sigma}_k}$ is symmetric as required because the matrix $(\mathbf{x}^{(i)} - \widehat{\boldsymbol{\mu}_k})(\mathbf{x}^{(i)} - \widehat{\boldsymbol{\mu}_k})^T$ is symmetric. Further, $\widehat{\boldsymbol{\Sigma}_k}$ is semi-definite as required. The proof for this follow the same pattern as the proof that $\boldsymbol{\Sigma}$ is semi-definite. It can be seen from the following [9].

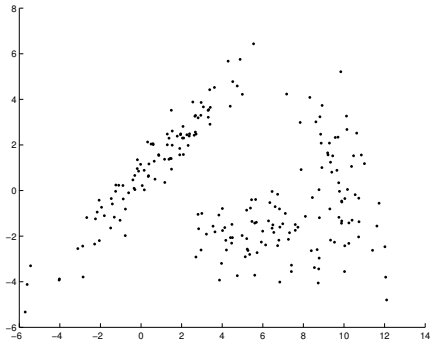
$$\begin{aligned}
\mathbf{w}^T \widehat{\boldsymbol{\Sigma}_k} \mathbf{w} &= \mathbf{w}^T \frac{1}{r_k} \sum_{i=1}^{r_k} ((\mathbf{x}^{(i)} - \widehat{\boldsymbol{\mu}_k})(\mathbf{x}^{(i)} - \widehat{\boldsymbol{\mu}_k})^T) \mathbf{w} \\
&= \frac{1}{r_k} \sum_{i=1}^{r_k} (\mathbf{w}^T (\mathbf{x}^{(i)} - \widehat{\boldsymbol{\mu}_k})(\mathbf{x}^{(i)} - \widehat{\boldsymbol{\mu}_k})^T \mathbf{w})
\end{aligned}$$

For the last expression we have $\mathbf{w}^T(\mathbf{x}^{(i)} - \widehat{\boldsymbol{\mu}_k}) = (\mathbf{x}^{(i)} - \widehat{\boldsymbol{\mu}_k})^T \mathbf{w} = (\mathbf{x} - \widehat{\boldsymbol{\mu}_k})) \cdot \mathbf{w}$, where $\cdot$ is the inner product. The expression therefore becomes $\frac{1}{r_k} \sum_{i=1}^{r_k} ((\mathbf{x}^{(i)} - \widehat{\boldsymbol{\mu}_k})) \cdot \mathbf{w})^2$. Since the result of the inner product is a scalar the expression is the sum of squared scalars, and this is always greater than or equal to zero. It is therefore ensured that $\mathbf{w}^T \widehat{\boldsymbol{\Sigma}_k} \mathbf{w} \geq 0$. As mentioned in Section 3.1.1 it is also required that $\widehat{\boldsymbol{\Sigma}}$ is non-singular else the variance is zero. This can be checked by calculating the determinant of the matrix. If this is 0 then the found estimates can not be used, and k-means must be applied again with another initialization of the centroids.

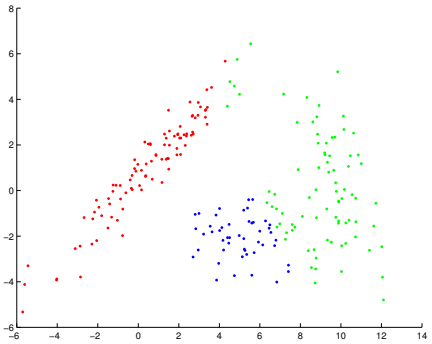
4.2.3 Illustrative example of learning a Gaussian Mixture Model

Figure 4.1 illustrates an example of the results for the steps that we perform in order to learn a GMM. In the example the size of the feature vectors is two, so $D = 2$. In Figure 4.1(a) the feature vectors are plotted into a graph. We will then learn a GMM with three components that fit this data. First the k-means algorithm is run which partitions the points into three clusters. The result of k-means is illustrated in Figure 4.1(b) where each color represents a cluster. Then the parameters of the GMM is initialized by calculating Equation 4.18-4.20 for each cluster. The contour of the initialized GMM is illustrated in Figure 4.1(c). The EM-algorithm is then run until convergence. The contour of the GMM after the EM-algorithm has terminated is illustrated in

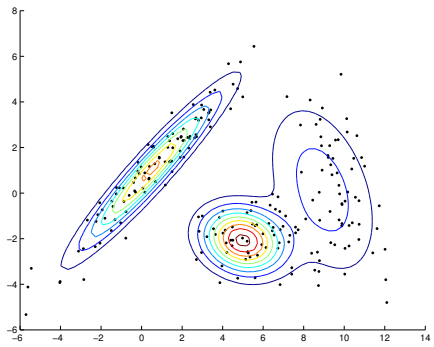
Figure 4.1(d). The GMM in Figure 4.1(c) and Figure 4.1(d) looks similar. However, the GMM in Figure 4.1(d) is better fitted to the points because the likelihood of the parameters of the model given the points increases in every iteration of the EM-algorithm.



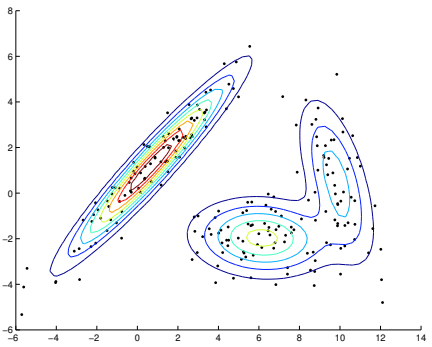
(a) Points of feature vectors.



(b) Result of k-means.



(c) GMM with initialized parameters.



(d) The GMM after applying the EM-algorithm.

Figure 4.1: Illustrative example of learning a GMM.

4.3 Hidden Markov Model

The state of an HMM is hidden, and, thus, the state variable for each time slice S_t can not be observed. The EM-algorithm is therefore used to find some parameters of the HMM which fit the observed data. The EM-algorithm for an HMM requires several tasks; *forward*, *backward* and *smoothing*. These are therefore introduced in Section 4.3.1-4.3.3 before the steps of the EM-algorithm are explained in Section 4.3.4. In Section 4.3.4 it is however only described how to estimate the parameters when a single audio file is given. In Section 4.3.5 it is explained how observations from multiple audio files is used for estimating the model parameters. In Section 4.3.6 it is explained how we initialize the model parameters for the EM-algorithm.

In [17] it is described how HMMs is learned when the observable data is discrete. Section 4.3.1, Section 4.3.2, and Section 4.3.3 are rewritten from the description given in [17], and the de-

scription of reestimating $P(S_1)$ and $P(S_t|S_{t-1})$ in Section 4.3.4 is inspired from the description given in [17].

4.3.1 Forward message

The forward message is defined as [26]:

Definition 4.3.1 (Forward message). *The* forward message α_t *is the probability of observing the observations from time* 1 *to* t*, and being in a certain state of the Markov process at time* t *:*

$$\alpha_t = P(\mathbf{x}_{1:t}, S_t)$$

We have that $P(\mathbf{x}_{1:t}, S_t) \to 0$ for $t \to \infty$. This poses an underflow problem when implementing the forward message. It is explained in Section 5.3.1.1 how this is handled.

The value of the forward message can be found the following way [30]:

$$\begin{aligned}
\alpha_t &= P(\mathbf{x}_{1:t}, S_t) \\
&= P(\mathbf{x}_t|S_t, \mathbf{x}_{1:t-1}) \cdot P(S_t, \mathbf{x}_{1:t-1}) \quad \text{Fund. rule} \\
&= P(\mathbf{x}_t|S_t) \cdot P(S_t, \mathbf{x}_{1:t-1}) \quad \text{Cond. indep.} \\
&= P(\mathbf{x}_t|S_t) \sum_{S_{t-1}} P(S_{t-1}, S_t, \mathbf{x}_{1:t-1}) \quad \text{Marginalize} \\
&= P(\mathbf{x}_t|S_t) \sum_{S_{t-1}} P(S_t|S_{t-1}, \mathbf{x}_{1:t-1}) \cdot P(\mathbf{x}_{1:t-1}, S_{t-1}) \quad \text{Fund. rule} \\
&= P(\mathbf{x}_t|S_t) \sum_{S_{t-1}} P(S_t|S_{t-1}) \cdot P(\mathbf{x}_{1:t-1}, S_{t-1}) \quad \text{Cond. indep.}
\end{aligned}$$

By noticing that $\alpha_{t-1} = P(\mathbf{x}_{1:t-1}, S_{t-1})$ the expression of the forward message becomes:

$$\alpha_t = P(\mathbf{x}_t|S_t) \sum_{S_{t-1}} P(S_t|S_{t-1}) \cdot \alpha_{t-1} \tag{4.21}$$

So the forward message can be calculated recursively. The term $P(\mathbf{x}_t|S_t)$ is given by Equation 3.5 and the term $P(S_t|S_{t-1})$ is the transition distribution. For the first time step $t = 1$ the forward variable is

$$\alpha_1 = P(S_1) \cdot P(\mathbf{x}_1|S_1) \tag{4.22}$$

We will denote the value of the forward message for a particular state n as [26, 30]:

$$\alpha_t(n) = P(\mathbf{x}_{1:t}, S_t = n)$$

4.3.2 Backward message

The backward message is defined as [26, 30]:

Definition 4.3.2 (Backward message). *The* backward message β_t *is the likelihood of the state of the Markov process at time* t *given the observations from time* $t+1$ *to* T*:*

$$\beta_t = P(\mathbf{x}_{t+1:T}|S_t)$$

As with the forward message, the backward message approaches 0 the longer the sequence $\mathbf{x}_{t+1:T}$ gets, so this also poses an underflow problem when implementing the backward message. In Section 5.3.1.1 it is explained how the underflow problem is handled for the forward and backward message.

The value of the backward message can be found the following way [30]:

$$\begin{aligned}
\beta_t &= P(\mathbf{x}_{t+1:T}|S_t) \\
&= \sum_{S_{t+1}} P(\mathbf{x}_{t+1:T}, S_{t+1}|S_t) \quad \text{Marginalize} \\
&= \sum_{S_{t+1}} P(\mathbf{x}_{t+1:T}|S_t, S_{t+1}) \cdot P(S_{t+1}|S_t) \quad \text{Fund. rule} \\
&= \sum_{S_{t+1}} P(\mathbf{x}_{t+1:T}|S_{t+1}) \cdot P(S_{t+1}|S_t) \quad \text{Cond. indep.} \\
&= \sum_{S_{t+1}} P(\mathbf{x}_{t+1}|\mathbf{x}_{t+2:T}, S_{t+1}) \cdot P(\mathbf{x}_{t+2:T}|S_{t+1}) \cdot P(S_{t+1}|S_t) \quad \text{Fund. rule} \\
&= \sum_{S_{t+1}} P(\mathbf{x}_{t+1}|S_{t+1}) \cdot P(\mathbf{x}_{t+2:T}|S_{t+1}) \cdot P(S_{t+1}|S_t) \quad \text{Cond. indep.}
\end{aligned}$$

By noticing that $\beta_{t+1} = P(\mathbf{x}_{t+2:T}|S_{t+1})$ the expression of the backward variable becomes:

$$\beta_t = \sum_{S_{t+1}} P(\mathbf{x}_{t+1}|S_{t+1}) \cdot \beta_{t+1} \cdot P(S_{t+1}|S_t) \tag{4.23}$$

The backward message can therefore, as the forward message, also be calculated recursively. The term $P(\mathbf{x}_{t+1}|S_{t+1})$ is given by Equation 3.5, and the term $P(S_{t+1}|S_t)$ is the transition distribution. For the first backward variable, that is for the last time step $t = T$, there is no emission,

so for all states of S_T the likelihood are set to 1 in order to rate them equally [26, 30]:

$$\beta_T = (1, 1, \ldots, 1) \tag{4.24}$$

We will denote the value of the backward message for a particular state n as:

$$\beta_t(n) = P(\mathbf{x}_{t+1:T}|S_t = n)$$

4.3.3 Smoothing

Smoothing is defined as [26, 30]:

Definition 4.3.3. *The* smoothing variable γ_t *is the probability of the states of the Markov process at time* t *given the observations from time* 1 *to time* T*:*

$$\gamma_t = P(S_t|\mathbf{x}_{1:T})$$

The value of the smoothing variable can be found the following way [30]:

$$\begin{aligned}
\gamma_t &= P(S_t|\mathbf{x}_{1:T}) \\
&= \frac{P(\mathbf{x}_{t+1:T}|S_t, \mathbf{x}_{1:t}) \cdot P(S_t|\mathbf{x}_{1:t})}{P(\mathbf{x}_{t+1:T}|\mathbf{x}_{1:t})} && \text{Bayes' rule} \\
&= \frac{P(\mathbf{x}_{t+1:T}|S_t) \cdot P(S_t|\mathbf{x}_{1:t})}{P(\mathbf{x}_{t+1:T}|\mathbf{x}_{1:t})} && \text{Cond. indep.} \\
&= \frac{P(\mathbf{x}_{t+1:T}|S_t) \cdot P(S_t, \mathbf{x}_{1:t})}{P(\mathbf{x}_{t+1:T}|\mathbf{x}_{1:t})P(\mathbf{x}_{1:t})} && \text{Fund. rule} \\
&= \varphi P(\mathbf{x}_{t+1:T}|S_t)P(S_t, \mathbf{x}_{1:t}) && \varphi \text{ is normalization factor}
\end{aligned}$$

The term $P(S_t, \mathbf{x}_{1:t})$ is the forward message and the term $P(\mathbf{x}_{t+1:T}|S_t)$ is the backward message, so smoothing becomes:

$$\gamma_t = \frac{\alpha_t \cdot \beta_t}{\sum_{S_t} \alpha_t \cdot \beta_t} \tag{4.25}$$

It is worth noting that for γ_t the forward message α_t takes observations from time step 1 to t into consideration while β_t takes observations from time step $t+1$ to T into consideration, so all observations are included exactly once in the expression.

We will denote the smoothing value for a particular state n as $\gamma_t(n) = P(S_t = n|\mathbf{x}_{1:T})$, and this is therefore given as:

$$\gamma_t(n) = \frac{\alpha_t(n) \cdot \beta_t(n)}{\sum_{n=1}^{N} \alpha_t(n) \cdot \beta_t(n)} \tag{4.26}$$

4.3.4 Estimation of parameters

The considerations and calculations in this section has its base in [10]. The observable data are the feature vectors of the audio files, $\mathcal{X} = (\mathbf{x}_{1:T}^{(1)}, \ldots, \mathbf{x}_{1:T}^{(R)})$. However, we first treat the case with feature vectors from one audio file $\mathbf{x}_{1:T}$ i.e. $\mathcal{X} = \mathbf{x}_{1:T}$. The case with multiplied audio files is handled in the next section. The hidden variables $S_1, S_2, \ldots, S_T$, which we denote as $S_{1:T}$, are the state of the Markov process at each time t, and the hidden variables $C_1, C_2, \ldots, C_T$, which we denote as $C_{1:T}$, are the component used at each time t.

The expectation step Let $\sum_{S_{1:T}}$ denote $\sum_{S_1}\sum_{S_2}\cdots\sum_{S_T}$ and $\sum_{C_{1:T}}$ denote $\sum_{C_1}\sum_{C_2}\cdots\sum_{C_T}$. The expected value of the log-likelihood of the parameters, $\log P(\mathbf{x}_{1:T}, S_{1:T}, C_{1:T}|\lambda)$, given $\mathbf{x}_{1:T}$ and the parameters $\lambda^{(i-1)}$ is given by:

$$Q(\lambda, \lambda^{(i-1)}) = \sum_{S_{1:T}}\sum_{C_{1:T}} \log(P(\mathbf{x}_{1:T}, S_{1:T}, C_{1:T}|\lambda))P(S_{1:T}, C_{1:T}|\mathbf{x}_{1:T}, \lambda^{(i-1)}) \tag{4.27}$$

By looking at the Bayesian network of the HMM in Figure 3.6 the term $P(\mathbf{x}_{1:T}, S_{1:T}, C_{1:T}|\lambda)$ can be rewritten using the chain rule [18]:

$$P(\mathbf{x}_{1:T}, S_{1:T}, C_{1:T}|\lambda) = P(S_1|\lambda)P(C_1|S_1, \lambda)P(\mathbf{x}_1|C_1, S_1, \lambda)\prod_{t=2}^{T} P(S_t|S_{t-1})P(C_t|S_t, \lambda)P(\mathbf{x}_t|C_t, S_t, \lambda)$$

This can be used to rewrite Equation 4.27:

$$\begin{aligned}
Q(\lambda, \lambda^{(i-1)}) &= \sum_{S_{1:T}}\sum_{C_{1:T}} \log(P(S_1\lambda)P(C_1|S_1, \lambda)P(\mathbf{x}_1|C_1, S_1, \lambda)\prod_{t=2}^{T} P(S_t|S_{t-1}, \lambda)P(C_t|S_t, \lambda)P(\mathbf{x}_t|C_t, S_t, \lambda)) \\
&\quad \cdot P(S_{1:T}, C_{1:T}|\mathbf{x}_{1:T}, \lambda^{(i-1)}) \\
&= \sum_{S_{1:T}}\sum_{C_{1:T}} \log(P(S_1|\lambda))P(S_{1:T}, C_{1:T}|\mathbf{x}_{1:T}, \lambda^{(i-1)}) \qquad \text{(i)} \\
&\quad + \sum_{S_{1:T}}\sum_{C_{1:T}}\sum_{t=2}^{T} \log(P(S_t|S_{t-1}, \lambda))P(S_{1:T}, C_{1:T}|\mathbf{x}_{1:T}, \lambda^{(i-1)}) \qquad \text{(ii)} \\
&\quad + \sum_{S_{1:T}}\sum_{C_{1:T}}\sum_{t=1}^{T} \log(P(C_t|S_t, \lambda)P(\mathbf{x}_t|C_t, S_t, \lambda))P(S_{1:T}, C_{1:T}|\mathbf{x}_{1:T}, \lambda^{(i-1)}) \qquad \text{(iii)}
\end{aligned}$$

We will rewrite the three terms i, ii, and iii separately. The term i becomes:

$$\begin{aligned}\sum_{S_{1:T}}\sum_{C_{1:T}}\log(P(S_1|\lambda))P(S_{1:T},C_{1:T}|\mathbf{x}_{1:T},\lambda^{(i-1)}) &= \sum_{S_{1:T}}\log(P(S_1|\lambda))\sum_{C_{1:T}}P(S_{1:T},C_{1:T}|\mathbf{x}_{1:T},\lambda^{(i-1)})\\ &= \sum_{S_{1:T}}\log(P(S_1|\lambda))P(S_{1:T}|\mathbf{x}_{1:T},\lambda^{(i-1)})\\ &= \sum_{S_1}(\log(P(S_1|\lambda))\sum_{S_{2:T}}P(S_{1:T}|\mathbf{x}_{1:T},\lambda^{(i-1)}))\\ &= \sum_{S_1}\log(P(S_1|\lambda))P(S_1|\mathbf{x}_{1:T},\lambda^{(i-1)})\end{aligned}$$

The term ii becomes:

$$\begin{aligned}&\sum_{S_{1:T}}\sum_{C_{1:T}}\sum_{t=2}^{T}\log(P(S_t|S_{t-1},\lambda))P(S_{1:T},C_{1:T}|\mathbf{x}_{1:T},\lambda^{(i-1)})\\ &= \sum_{S_{1:T}}\sum_{t=2}^{T}\log(P(S_t|S_{t-1},\lambda))\sum_{C_{1:T}}P(S_{1:T},C_{1:T}|\mathbf{x}_{1:T},\lambda^{(i-1)})\\ &= \sum_{S_{1:T}}\sum_{t=2}^{T}\log(P(S_t|S_{t-1},\lambda))P(S_{1:T}|\mathbf{x}_{1:T},\lambda^{(i-1)})\\ &= \sum_{t=2}^{T}\sum_{S_{t-1}}\sum_{S_t}(\log(P(S_t|S_{t-1},\lambda))\sum_{S_{1:t-2}}\sum_{S_{t+1:T}}P(S_{1:T}|\mathbf{x}_{1:T},\lambda^{(i-1)}))\\ &= \sum_{t=2}^{T}\sum_{S_{t-1}}\sum_{S_t}\log(P(S_t|S_{t-1},\lambda))P(S_{t-1},S_t|\mathbf{x}_{1:T},\lambda^{(i-1)})\end{aligned}$$

The factor $P(S_{t-1},S_t|\mathbf{x}_{1:T})$ is given by the following where $\lambda^{(i-1)}$ is omitted to make the derivation more readable:

$$\begin{aligned}P(S_{t-1},S_t|\mathbf{x}_{1:T}) &= P(S_{t-1},S_t|\mathbf{x}_{1:t-1},\mathbf{x}_t,\mathbf{x}_{t+1:T})\\ &= \frac{P(\mathbf{x}_{t+1:T}|S_{t-1},S_t,\mathbf{x}_{1:t-1},\mathbf{x}_t)\cdot P(S_{t-1},S_t|\mathbf{x}_{1:t-1},\mathbf{x}_t)}{P(\mathbf{x}_{t+1:T}|\mathbf{x}_{1:t})} \quad \text{Bayes' rule}\\ &= \frac{P(\mathbf{x}_{t+1:T}|S_t)\cdot P(S_{t-1},S_t|\mathbf{x}_{1:t-1},\mathbf{x}_t)}{P(\mathbf{x}_{t+1:T}|\mathbf{x}_{1:t})} \quad \text{Cond. indep.}\\ &= \frac{P(\mathbf{x}_{t+1:T}|S_t)\cdot P(\mathbf{x}_t|S_{t-1},S_t,\mathbf{x}_{1:t-1})\cdot P(S_{t-1},S_t|\mathbf{x}_{1:t-1})}{P(\mathbf{x}_{t+1:T}|\mathbf{x}_{1:t})\cdot P(\mathbf{x}_t|\mathbf{x}_{1:t-1})} \quad \text{Bayes' rule}\\ &= \frac{P(\mathbf{x}_{t+1:T}|S_t)\cdot P(\mathbf{x}_t|S_t)\cdot P(S_{t-1},S_t|\mathbf{x}_{1:t-1})}{P(\mathbf{x}_{t+1:T}|\mathbf{x}_{1:t})\cdot P(\mathbf{x}_t|\mathbf{x}_{1:t-1})} \quad \text{Cond. indep.}\end{aligned}$$

$$
\begin{aligned}
&= \frac{P(\mathbf{x}_{t+1:T}|S_t) \cdot P(\mathbf{x}_t|S_t) \cdot P(S_t|S_{t-1}, \mathbf{x}_{1:t-1}) \cdot P(S_{t-1}|\mathbf{x}_{1:t-1})}{P(\mathbf{x}_{t+1:T}|\mathbf{x}_{1:t}) \cdot P(\mathbf{x}_t|\mathbf{x}_{1:t-1})} \quad \text{Fund. rule} \\
&= \frac{P(\mathbf{x}_{t+1:T}|S_t) \cdot P(\mathbf{x}_t|S_t) \cdot P(S_t|S_{t-1}) \cdot P(S_{t-1}|\mathbf{x}_{1:t-1})}{P(\mathbf{x}_{t+1:T}|\mathbf{x}_{1:t}) \cdot P(\mathbf{x}_t|\mathbf{x}_{1:t-1})} \quad \text{Cond. indep.} \\
&= \frac{P(\mathbf{x}_{t+1:T}|S_t) \cdot P(\mathbf{x}_t|S_t) \cdot P(S_t|S_{t-1}) \cdot P(S_{t-1}, \mathbf{x}_{1:t-1})}{P(\mathbf{x}_{t+1:T}|\mathbf{x}_{1:t}) \cdot P(\mathbf{x}_t|\mathbf{x}_{1:t-1}) \cdot P(\mathbf{x}_{1:t-1})} \quad \text{Fund. rule} \\
&= \varphi P(\mathbf{x}_{t+1:T}|S_t) P(\mathbf{x}_t|S_t) P(S_t|S_{t-1}) P(S_{t-1}, \mathbf{x}_{1:t-1}) \quad \varphi \text{ is normalization factor}
\end{aligned}
$$

Here $P(\mathbf{x}_{t+1:T}|S_t)$ is the backward message to time t, $P(\mathbf{x}_t|S_t)$ is given by Equation 3.5, $P(S_t|S_{t-1})$ is the transition distribution, and $P(S_{t-1}, \mathbf{x}_{1:t-1})$ is the forward message to time t. $P(S_{t-1}, S_t|\mathbf{x}_{1:T})$ then becomes:

$$
P(S_{t-1}, S_t|\mathbf{x}_{1:T}) = \frac{\alpha_{t-1} \cdot P(S_t|S_{t-1}) \cdot P(\mathbf{x}_t|S_t) \cdot \beta_t}{\sum_{S_{t-1}} \sum_{S_t} \alpha_{t-1} \cdot P(S_t|S_{t-1}) P(S_{t-1} \cdot P(\mathbf{x}_t|S_t) \cdot \beta_t} \tag{4.28}
$$

The term iii becomes:

$$
\begin{aligned}
&\sum_{S_{1:T}} \sum_{C_{1:T}} \sum_{t=1}^{T} \log(P(C_t|S_t, \lambda) P(\mathbf{x}_t|C_t, S_t, \lambda)) P(S_{1:T}, C_{1:T}|\mathbf{x}_{1:T}, \lambda^{(i-1)}) \\
&= \sum_{t=1}^{T} \sum_{S_t} \sum_{C_t} (\log(P(C_t|S_t, \lambda) P(\mathbf{x}_t|C_t, S_t, \lambda)) \sum_{S_{1:t-1}} \sum_{S_{t+1:T}} \sum_{C_{1:t-1}} \sum_{C_{t+1:T}} P(S_{1:T}, C_{1:T}|\mathbf{x}_{1:T}, \lambda^{(i-1)})) \\
&= \sum_{t=1}^{T} \sum_{S_t} \sum_{C_t} \log(P(C_t|S_t, \lambda) P(\mathbf{x}_t|C_t, S_t, \lambda)) P(S_t, C_t|\mathbf{x}_{1:T}, \lambda^{(i-1)}) \\
&= \sum_{t=1}^{T} \sum_{n=1}^{N} \sum_{k=1}^{K} \log(w_{nk} P(\mathbf{x}_t|\boldsymbol{\mu}_{nk}, \boldsymbol{\Sigma}_{nk})) P(S_t = n, C_t = k|\mathbf{x}_{1:T}, \lambda^{(i-1)}) \quad \text{Equation 3.3 and Equation 3.4} \\
&= \sum_{t=1}^{T} \sum_{n=1}^{N} \sum_{k=1}^{K} \log(w_{nk}) P(S_t = n, C_t = k|\mathbf{x}_{1:T}, \lambda^{(i-1)}) \\
&\quad + \sum_{t=1}^{T} \sum_{n=1}^{N} \sum_{k=1}^{K} \log(P(\mathbf{x}_t|\boldsymbol{\mu}_{nk}, \boldsymbol{\Sigma}_{nk})) P(S_t = n, C_t = k|\mathbf{x}_{1:T}, \lambda^{(i-1)})
\end{aligned}
$$

The factor $P(S_t = n, C_t = k|\mathbf{x}_{1:T}, \lambda^{(i-1)})$ is given by the following where $\lambda^{(i-1)}$ is omitted to make the derivation more readable:

$$
\begin{aligned}
P(S_t = n, C_t = k|\mathbf{x}_{1:T}) &= P(C_t = k|S_t = n, \mathbf{x}_{1:T}) \cdot P(S_t = n|\mathbf{x}_{1:T}) \quad \text{Fund. rule} \\
&= P(C_t = k|S_t = n, \mathbf{x}_t) \cdot P(S_t = n|\mathbf{x}_{1:T}) \quad \text{Cond. indep.} \\
&= \frac{P(\mathbf{x}_t|C_t = k, S_t = n) \cdot P(C_t = k|S_t = n)}{P(\mathbf{x}_t|S_t = n)} \cdot P(S_t = n|\mathbf{x}_{1:T}) \quad \text{Bayes' rule} \\
&= \varphi P(\mathbf{x}_t|C_t = k, S_t = n) \cdot P(C_t = k|S_t = n) \cdot P(S_t = n|\mathbf{x}_{1:T}) \\
&\qquad \varphi \text{ is normalization factor}
\end{aligned}
$$

Here $P(\mathbf{x}_t|C_t = k, S_t = n)$ is given by Equation 3.4, $P(C_t = k|S_t = n)$ is given by Equation 3.3, and $P(S_t = n|\mathbf{x}_{1:T})$ is the smoothing variable for state n. $P(S_t = n, C_t = k|\mathbf{x}_{1:T})$ then becomes:

$$P(S_t = n, C_t = k|\mathbf{x}_{1:T}) = \frac{\alpha_t(n) \cdot \beta_t(n)}{\sum_{n=1}^{N} \alpha_t(n) \cdot \beta_t(n)} \cdot \frac{w_{nk} P(\mathbf{x}_t|\boldsymbol{\mu}_{nk}, \boldsymbol{\Sigma}_{nk})}{\sum_{k=1}^{K} w_{nk} P(\mathbf{x}_t|\boldsymbol{\mu}_{nk}, \boldsymbol{\Sigma}_{nk})} \tag{4.29}$$

$Q(\lambda, \lambda^{(i-1)})$ is therefore the sum of four terms:

$$\begin{aligned} Q(\lambda, \lambda^{(i-1)}) = &\sum_{S_1} \log(P(S_1|\lambda)) P(S_1|\mathbf{x}_{1:T}, \lambda^{(i-1)}) && \text{(I)} \\ &+ \sum_{t=2}^{T} \sum_{S_{t-1}} \sum_{S_t} \log(P(S_t|S_{t-1}, \lambda)) P(S_{t-1}, S_t|\mathbf{x}_{1:T}, \lambda^{(i-1)}) && \text{(II)} \\ &+ \sum_{t=1}^{T} \sum_{n=1}^{N} \sum_{k=1}^{K} \log(w_{nk}) P(S_t = n, C_t = k|\mathbf{x}_{1:T}, \lambda^{(i-1)}) && \text{(III)} \\ &+ \sum_{t=1}^{T} \sum_{n=1}^{N} \sum_{k=1}^{K} \log(P(\mathbf{x}_t|\boldsymbol{\mu}_{nk}, \boldsymbol{\Sigma}_{nk})) P(S_t = n, C_t = k|\mathbf{x}_{1:T}, \lambda^{(i-1)}) && \text{(IV)} \end{aligned}$$

The maximization step The parameters which must be maximized are the start distribution $P(S_1)$, the transition distribution $P(S_t|S_{t-1})$, and for each component k of the GMMs for each state n of the Markov process the weight w_{nk}, the mean vector $\boldsymbol{\mu}_{nk}$, and the covariance matrix $\boldsymbol{\Sigma}_{nk}$. As described in Section 4.1, this includes taking the partial derivatives of $Q(\lambda, \lambda^{(i-1)})$.

When optimizing $Q(\lambda, \lambda^{(i-1)})$ according to $P(S_1)$ the terms II, III, and IV cancels out, so it is only required to differentiate the term I. Since it is required that $\sum_{n=1}^{N} \widehat{P}(S_1 = n) = 1$ Lagrange multipliers are used. The equation for $\widehat{P}(S_1)$ becomes [26]:

$$\widehat{P}(S_1) = P(S_1|\mathbf{x}_{1:T}, \lambda^{(i-1)}) = \gamma_{1,\lambda^{(i-1)}} \tag{4.30}$$

where $\gamma_{1,\lambda^{(i-1)}}$ implies that γ_1 has been found using the values of $\lambda^{(i-1)}$. $\widehat{P}(S_1)$ can be interpreted as the expected number of transitions from S_1.

When optimizing $Q(\lambda, \lambda^{(i-1)})$ according to $P(S_t|S_{t-1})$ the terms I, III, and IV cancels out, so it is only required to differentiate the term II. For the result $\widehat{P}(S_t|S_{t-1})$ it is required that $\sum_{n'=1}^{N} \widehat{P}(S_t = n'|S_{t-1} = n'') = 1$ for $n'' = 1, \ldots, N$, so Lagrange multipliers are used. The equation for $\widehat{P}(S_t|S_{t-1})$ becomes [26]:

$$\widehat{P}(S_t|S_{t-1}) = \frac{\sum_{t=2}^{T} P(S_{t-1}, S_t|\mathbf{x}_{1:T}, \lambda^{(i-1)})}{\sum_{t=2}^{T} P(S_{t-1}|\mathbf{x}_{1:T}, \lambda^{(i-1)})} = \frac{\sum_{t=2}^{T} P(S_{t-1}, S_t|\mathbf{x}_{1:T}, \lambda^{(i-1)})}{\sum_{t=2}^{T} \gamma_{t-1,\lambda^{(i-1)}}} \tag{4.31}$$

where $P(S_{t-1}, S_t|\mathbf{x}_{1:T}, \lambda^{(i-1)})$ can be found with Equation 4.28, and $\gamma_{t-1,\lambda^{(i-1)}}$ implies that γ_{t-1} has been found using the values of $\lambda^{(i-1)}$. The denominator can be found with smoothing, but it can also be found by marginalizing S_t out of the nominator. The reestimation of $P(S_t|S_{t-1})$ can be interpreted as [26]:

$$\widehat{P}(S_t|S_{t-1}) = \frac{\text{expected number of transistions from } S_{t-1} \text{ to } S_t}{\text{expected number of times in state } S_{t-1}}$$

When optimizing $Q(\lambda, \lambda^{(i-1)})$ according to w_{nk} the terms I, II, and IV cancels out, so it is only required to differentiate the term III. It is required that $\sum_{k=1}^{K} \widehat{w_{nk}} = 1$ for $n = 1, \ldots, N$ so Lagrange multipliers are used again. The equation for $\widehat{w_{nk}}$ becomes [26]:

$$\widehat{w_{nk}} = \frac{\sum_{t=1}^{T} P(S_t = n, C_t = k|\mathbf{x}_{1:T}, \lambda^{(i-1)})}{\sum_{t=1}^{T}\sum_{k=1}^{K} P(S_t = n, C_t = k|\mathbf{x}_{1:T}, \lambda^{(i-1)})} \tag{4.32}$$

where $P(S_t = n, C_t = k|\mathbf{x}_{1:T}, \lambda^{(i-1)})$ is found using Equation 4.29.

When optimizing $Q(\lambda, \lambda^{(i-1)})$ according to $\boldsymbol{\mu}_{nk}$ and $\boldsymbol{\Sigma}_{nk}$ the terms I, II, and III cancels out, so it is only required to differentiate the term IV. The equations for $\widehat{\boldsymbol{\mu}_{nk}}$ and $\widehat{\boldsymbol{\Sigma}_{nk}}$ becomes [26]:

$$\widehat{\boldsymbol{\mu}_{nk}} = \frac{\sum_{t=1}^{T} P(S_t = n, C_t = k|\mathbf{x}_{1:T}, \lambda^{(i-1)}) \cdot \mathbf{x}_t}{\sum_{t=1}^{T} P(S_t = n, C_t = k|\mathbf{x}_{1:T}, \lambda^{(i-1)})} \tag{4.33}$$

$$\widehat{\boldsymbol{\Sigma}_{nk}} = \frac{\sum_{t=1}^{T} P(S_t = n, C_t = k|\mathbf{x}_{1:T}, \lambda^{(i-1)})(\mathbf{x}_t - \widehat{\boldsymbol{\mu}_{nk}})(\mathbf{x}_t - \widehat{\boldsymbol{\mu}_{nk}})^T}{\sum_{t=1}^{T} P(S_t = n, C_t = k|\mathbf{x}_{1:T}, \lambda^{(i-1)})} \tag{4.34}$$

where $P(S_t = n, C_t = k|\mathbf{x}_{1:T}, \lambda^{(i-1)})$ is found using Equation 4.29.

Equation 4.32-4.34 must be calculated for $n = 1, \ldots, N$ and $k = 1, \ldots, K$. The parameters for the next iteration then becomes $\lambda^i = (\widehat{P}(S_1), \widehat{P}(S_t|S_{t-1}), (\widehat{w_{nk}}, \widehat{\boldsymbol{\mu}_{nk}}, \widehat{\boldsymbol{\Sigma}_{nk}})_{n=1,k=1}^{N,K})$.

Termination The likelihood of the parameters given the observations can be found from the forward message to the last time step T by marginalizing the states of the Markov process out [26]:

$$\begin{aligned} P(\mathbf{x}_{1:T}|\lambda) &= \sum_{S_T} P(\mathbf{x}_{1:T}, S_T|\lambda) \\ &= \sum_{S_T} \alpha_T \end{aligned} \tag{4.35}$$

4.3.5 Multiple audio files

When we have multiple audio files $\mathcal{X} = (\mathbf{x}_{1:T}^1, \ldots, \mathbf{x}_{1:T}^R)$ all of these can be used for learning the model parameters. The reestimation formulas given in Equation 4.30-4.34 then be-

comes [26]:

$$\widehat{P}(S_1) = \frac{\sum_{r=1}^{R} \gamma_{1,\lambda^{(i-1)}}^{(r)}}{R} \tag{4.36}$$

$$\widehat{P}(S_t|S_{t-1}) = \frac{\sum_{r=1}^{R}\sum_{t=2}^{T} P(S_{t-1}, S_t|\mathbf{x}_{1:T}^{(r)}, \lambda^{(i-1)})}{\sum_{r=1}^{R}\sum_{t=2}^{T} \gamma_{t-1,\lambda^{(i-1)}}^{(r)}} \tag{4.37}$$

$$\widehat{w_{nk}} = \frac{\sum_{r=1}^{R}\sum_{t=1}^{T} P(S_t = n, C_t = k|\mathbf{x}_{1:T}^{(r)}, \lambda^{(i-1)})}{\sum_{r=1}^{R}\sum_{t=1}^{T}\sum_{k=1}^{K} P(S_t = n, C_t = k|\mathbf{x}_{1:T}^{(r)}, \lambda^{(i-1)})} \tag{4.38}$$

$$\widehat{\boldsymbol{\mu}_{nk}} = \frac{\sum_{r=1}^{R}\sum_{t=1}^{T} P(S_t = n, C_t = k|\mathbf{x}_{1:T}^{(r)}, \lambda^{(i-1)}) \cdot \mathbf{x}_t^{(r)}}{\sum_{r=1}^{R}\sum_{t=1}^{T} P(S_t = n, C_t = k|\mathbf{x}_{1:T}^{(r)}, \lambda^{(i-1)})} \tag{4.39}$$

$$\widehat{\boldsymbol{\Sigma}_{nk}} = \frac{\sum_{r=1}^{R}\sum_{t=1}^{T} P(S_t = n, C_t = k|\mathbf{x}_{1:T}^{(r)}, \lambda^{(i-1)})(\mathbf{x}_t^{(r)} - \widehat{\boldsymbol{\mu}_{nk}})(\mathbf{x}_t^{(r)} - \widehat{\boldsymbol{\mu}_{nk}})^T}{\sum_{r=1}^{R}\sum_{t=1}^{T} P(S_t = n, C_t = k|\mathbf{x}_{1:T}^{(r)}, \lambda^{(i-1)})} \tag{4.40}$$

In order to determine when to terminate, $P(\mathcal{X}|\lambda)$ must be found. Using Equation 4.1 and Equation 4.35 this becomes [26]:

$$P(\mathcal{X}|\lambda) = \prod_{r=1}^{R}\sum_{S_T} \alpha_T^{(r)} \tag{4.41}$$

where $\alpha_T^{(r)} = P(\mathbf{x}_{1:T}^{(r)}, S_T|\lambda)$. As mentioned earlier the log-likelihood, $\log P(\mathcal{X}|\lambda)$, is used in order to get a sum instead of a multiplication because it is computational better. This is then given by:

$$\log P(\mathcal{X}|\lambda) = \sum_{r=1}^{R} \log(\sum_{S_T} \alpha_T^r) \tag{4.42}$$

4.3.6 Initial values of the parameters

The EM-algorithm is not ensured to find the global optimum, so the EM-algorithm must be run several times with different initial values in order to increase the probability for finding a good local optimum. According to Rabiner [26] there is no known good selection for initial parameters only knowledge about the particular domain can be used as guidance. Normally the parameters are initialized randomly. We choose to initialize the start and transition probability distribution uniformly in order to limit the parameter space to search, and, thus, only use a randomness when initializing the weights, mean vectors, and covariance matrices. In order to find good values for these parameters, the feature vectors of all frames of all annotated audio files are randomly distributed to each state. Then k-means is applied on the feature vectors in each state, and the parameters for the GMM for each state is estimated as explained in Section 4.2.2.

5 Implementation

In order to learn and test the models several implementation tasks were required. Extractions of the feature vectors and validation were performed using methods from MatLab while the the models, learning the models, and classification of the audio files were implemented in C++. In Section 5.1 the pipeline of the work process is presented which gives an overview of the implementation tasks required. In Section 5.2 implementation of GMM is described in more details, in Section 5.3 implementation of HMM is described in more details, and in Section 5.4 implementation of classification is described.

5.1 Pipeline

The pipeline of handling data is shown in Figure 5.1.

First the data set is devided into three disjoint subsets:

1. A subset for learning
2. A subset for validation
3. A subset for testing[1]

[1]For the Kaggle competition this subset is given by Kaggle. These audio files are not annotated, so it can not be seen whether a audio file is positive or negative, but the result of the classification of these audio files can be loaded up to the Kaggle web page [3], and a result is returned based on AUC of a ROC-curve. AUC and ROC-curves are explained in more details in Section 6.1.1.

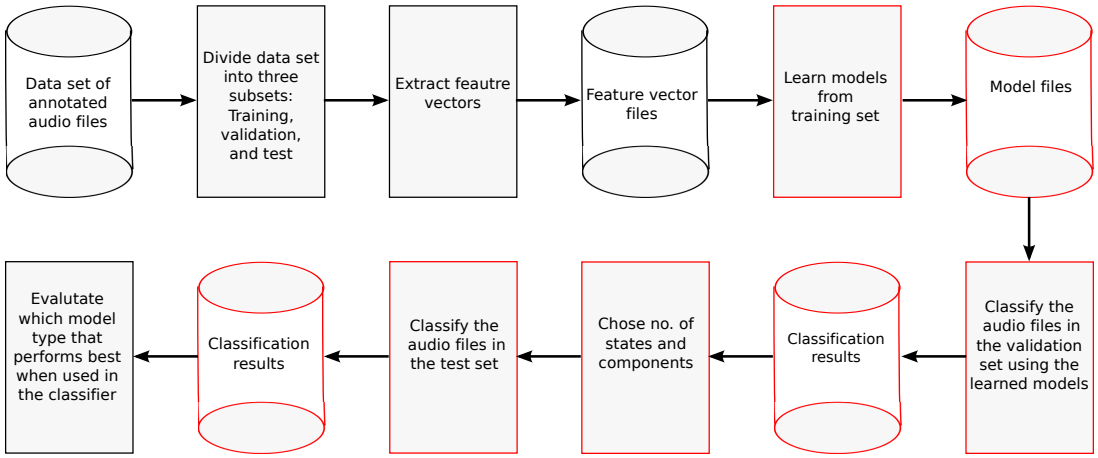


Figure 5.1: The pipeline of handling data. The part of the process which are red is performed for each model type.

After this the feature vectors are extracted from all the audio files using the MatLab toolbox VOICEBOX. This returns four data files each of which consists of the feature vectors for the audio files in the particular set:

1. A data file for positive audio files used for learning the model parameters
2. A data file for negative audio files used for learning the model parameters
3. A data file for audio files used for validation
4. A data file for audio files used for testing

To learn the models for the positive and negative class, the data file containing the feature data for the training set is loaded. From this different models are learned where the number of components is varied for the models using GMMs, and number of states and components are varied for the models using HMMs. Afterwards the learned models are stored as files such that they can be used as candidates for the final classifier.

For both positive and negative models, the learned models are loaded together with the feature vectors for the audio files in the validation set. For each learned model each of these audio files are classified, and the results are saved to a file. For each model type all the positive models are paired against all negative models, and the validation set is used to decide which pair of positive and negative model that performs best when used in the classifier in Figure 1.1. It is then tested which model type that performs best when used in the classifier by loading the selected positive and negative model, and the features vector for the audio files in the test set. The audio files in the test set are then classified, and it is evaluated which model type is the best performs best when used in the classifer in Figure 1.1. Validation and testing are explained further in Chapter 6.

5.2 Gaussian Mixture Model

An overview of the implementation of the GMM can be seen in Appendix A where the header file of the implementation is presented. It consists of three classes: *MultivariateGaussianDistribution*, *Component* and *GMM*. A class-diagram of the implementation of GMM can be seen in Figure 5.2 where the most relevant attributes and methods are shown.

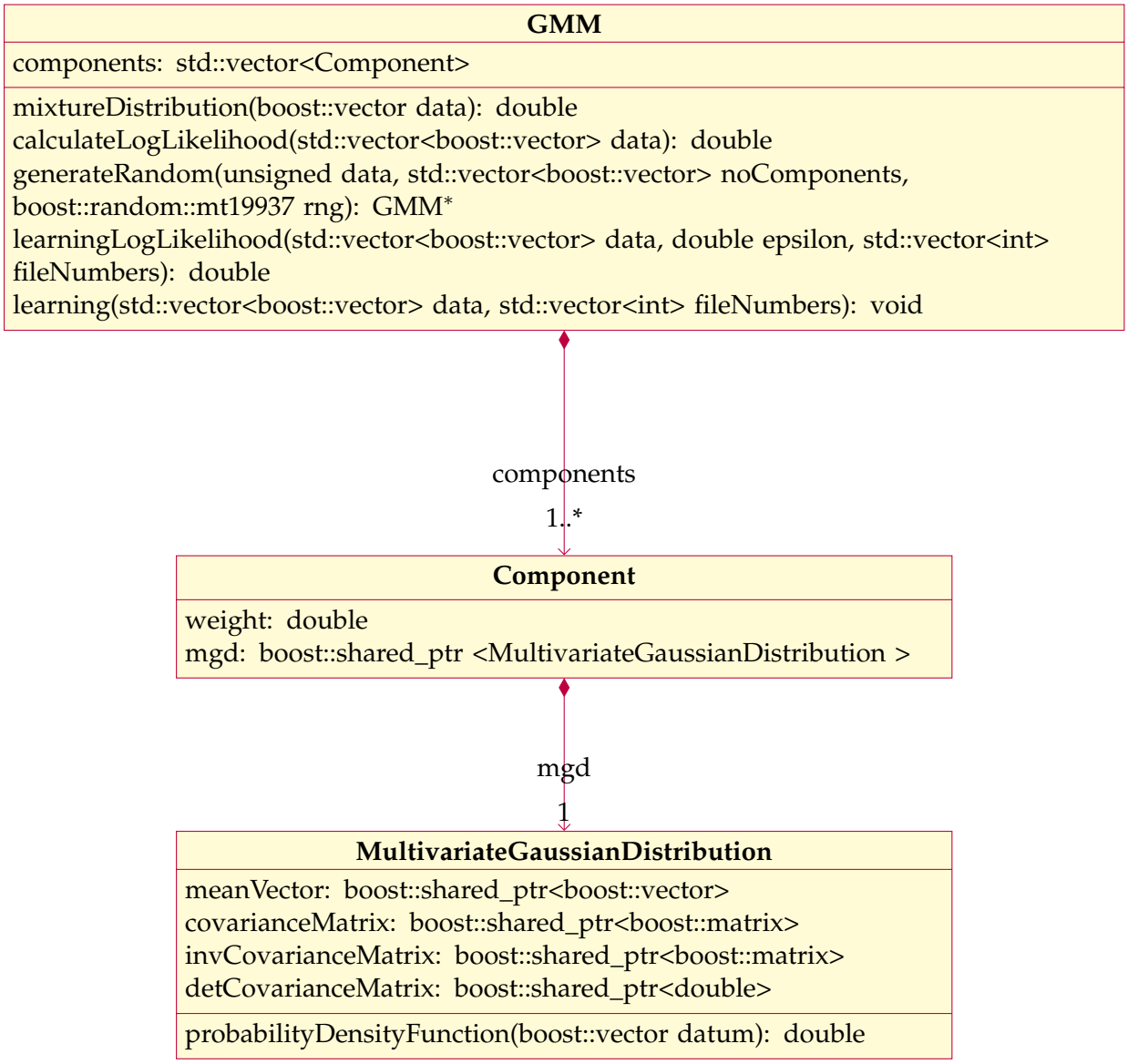


Figure 5.2: Class-diagram for Gaussian Mixture Model.

The class *MultivariateGaussianDistribution* contain a mean vector and a covariance matrix. Further it has a method for calculating the probability density function in Definition 3.1.1. This can be seen in Listing 5.1. When calculating the probability density function the determinant and inverse of the covariance matrix must be found. The method are called several times e.g. during learning where the determinant and inverse of the covariance matrix are the same. In order

to save calculations, the determinant and inverse covariance matrix are therefore cached. This is done by having an attribute for the determinant and the inverse covariance matrix which is set to null when the value of the covariance matrix is updated e.g. after one iteration of learning. The first time the determinant and the inverse of the covariance matrix are used after the covariance matrix has been updated the determinant and the inverse covaraince matrix are calculated and cached, and then these values are used later on instead of making the same calculations again. This can be seen in Line 6 and Line 14 of Listing 5.1.

```
1  double MultivariateGaussianDistribution::probabilityDensityFunction(const vector<
       double> &x)
2  {
3      if(detCovarianceMatrix == NULL)
4      {
5          double det = determinantLU(*covarianceMatrix);
6          detCovarianceMatrix.reset(new double(det));
7      }
8      if(*detCovarianceMatrix<1e-300)
9      {
10         throw SingularCovarianceMatrixException(*detCovarianceMatrix);
11     }
12     if(invCovarianceMatrix == NULL)
13     {
14         invCovarianceMatrix.reset(invertGaussJordan(*covarianceMatrix));
15     }
16     //exponent
17     vector<double> diff = x-*meanVector;
18     vector<double> prod1 = prod(*invCovarianceMatrix, diff);
19     double prod2 = inner_prod(diff, prod1);
20     double  exponent = -0.5*prod2;
21
22     //normalization factor
23     double tmp = pow(2*pi, dim);
24     tmp = tmp * (*detCovarianceMatrix);
25     tmp = sqrt(tmp);
26     double normalizeFactor = 1.0/tmp;
27
28     double result = normalizeFactor*exp(exponent);
29     if(result == std::numeric_limits<double>::infinity())
30     {
31             throw InfLikelihoodException(result);
32     }
33     return result;
34 }
```

Listing 5.1: Probability density function for Multivariate Gaussian Distribution.

The class *Component* contains a *MultivariateGaussianDistritbution* and a double for the weight of the component, and the class *GMM* contains several components. The mixture distribution given in Equation 3.1 was implemented directly from the formula in the method `mixtureDistribution`. The learning and initialization of a GMM were also implemented directly from the explanations

given in Section 4.2.1 and Section 4.2.2, respectively. The flow chart for learning is shown in Figure 5.3. First the parameters of the GMM must be initialized. This is done with the method `generateRandom`. For initialization the k-means algorithm was also implemented. The method `learning` runs one iteration of the EM-algorithm. Different methods which terminates the EM-algorithm on different criteria can then be made. These can run several iterations of the EM-algorithm, and call the `learning` method for performing an iteration. The `learningLogLikelihood` method is such a method which terminates when the increase in log-likelihood is below a threshold. When one iteration of learning has been run, a new *GMM* object is not created instead the values of this *GMM* object are updated. Computation of the log-likelihood for the model parameters given the data was implemented directly from Equation 4.10 using method *MultivariateGaussianDistritbution*.

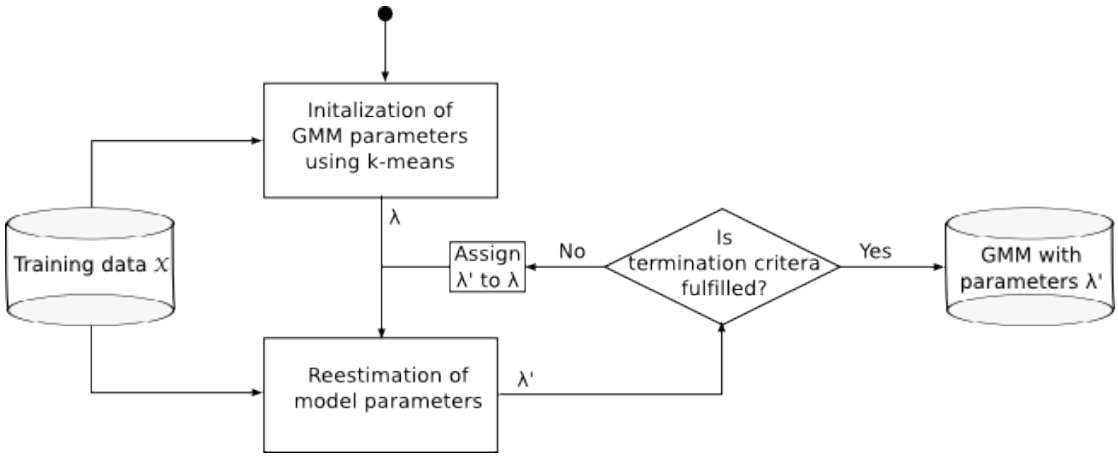


Figure 5.3: Flow chart for learning a GMM.

According to Rabiner [26] most problems of interest have a very complex optimization surfaces, and similarly many local maximums for optimization. When learning a model, several different initial values of the parameters λ are therefore used in order to increase the probability of finding a local maximum closer to the global maximum. The model with the initial values which led to the highest log-likelihood is then used further on for validation. As mentioned in Section 4.2.2 we used the k-means algorithm to make the initial clustering of the feature vectors. In the k-means algorithm the first choice of centroids is made by randomly picking feature vectors. The initial values of λ can therefore be changed by picking different feature vectors to use as initial centroids.

5.3 Hidden Markov Model

This section has been rewritten from the description of scaling and implementation given in [17]. The header file for the implementation of HMM can be seen in Appendix B. The class of HMM has four attributes:

- `noStates`: An unsigned integer to represent the number of states the HMM.
- `start`: A vector of doubles to represent the prior probability $P(S_1)$, so entry i of the vector is the prior probability for state i.
- `transition`: A matrix of doubles to represent the transition conditional probability distribution $P(S_t|S_{t-1})$. The value at entry (i, j) of the matrix is the probability for making a transition from state i to state j.
- `emission`: A list of objects of type *GMM*, one for each state.

The implementation was made almost directly from the description given in Section 3.2 and Section 4.3 where matrices and vectors were used as data structures for the probability distributions. There is however some technical specification when implementing the forward and backward message. These are introduced in Section 5.3.1. Specifications of learning are discussed in Section 5.3.2.

5.3.1 Forward and backward message

When calculating the forward and backward message from Equation 4.21 and Equation 4.23 it includes many multiplications with numbers below one, so for large values of t there is a great risk that α_t and β_t will come so close to zero that it will lead to underflow. How this is handled is explained in Section 5.3.1.1. Matrix operations can be exploited when implementing the forward and backward message. This is explained in Section 5.3.1.2.

5.3.1.1 Scaling

Underflow of forward and backward message can be avoided by scaling α_t and β_t in each time step [26]. For the forward message this is done by multiplying with a normalization factor in each time step, so when computing α_t it will be normalized. We refer to the result of this as f_t, and this is then used in the next iteration when computing α_{t+1}. Let the normalization factor to time step t be c_t, the initialization step and induction step for the forward message then becomes:

$$f_1 = \frac{P(S_1) \cdot P(\mathbf{x}_1|S_1)}{\sum_{S_t} P(S_1) \cdot P(\mathbf{x}_1|S_1)} = c_1 P(S_1) P(\mathbf{x}_1|S_1) \tag{5.1}$$

$$f_t = \frac{P(\mathbf{x}_t|S_t) \sum_{S_{t-1}} P(S_t|S_{t-1}) \cdot f_{t-1}}{\sum_{S_t} P(\mathbf{x}_t|S_t) \sum_{S_{t-1}} P(S_t|S_{t-1}) f_{t-1}} = c_t P(\mathbf{x}_t|S_t) \sum_{S_{t-1}} P(S_t|S_{t-1}) f_{t-1} \tag{5.2}$$

For the backward message the same scaling factors are used i.e. c_t is also used for scaling β_t. We denote the result as b_t. As with the forward message b_t is used in the next iteration when computing β_{t-1}. The initialization and induction step of the backward message then

becomes:

$$b_T = c_T \cdot (1, 1, \ldots, 1) \tag{5.3}$$

$$b_t = c_t \sum_{S_{t+1}} P(\mathbf{x}_{t+1}|S_{t+1}) \cdot b_{t+1} \cdot P(S_{t+1}|S_t) \tag{5.4}$$

Using the scaled forward and backward message does not have an effect on the EM-algorithm because the scaling factors cancels out from the nominator and denominator in Equation 4.36-4.40 [26]. We must however still be able to calculate the log-likelihood of λ given the data $\mathcal{X}$. Recall from Equation 4.35 that the likelihood for λ given an audio file r is:

$$P(\mathbf{x}_{1:T}^{(r)}|\lambda) = \sum_{S_T} P(\mathbf{x}_{1:T}^{(r)}, S_T|\lambda) = \sum_{S_T} \alpha_T^{(r)}$$

where $\alpha_T^{(r)}$ is the forward message to the last time step T for audio file r. Further recall from Equation 4.42 that the log-likelihood of λ for the data $\mathcal{X}$ then is:

$$\log P(\mathcal{X}|\lambda) = \sum_{r=1}^{R} \log P(\mathbf{x}_{1:T}^{(r)}|\lambda) = \sum_{r=1}^{R} \log(\sum_{S_T} \alpha_T^{(r)})$$

Let $f_t^{(r)}$ be the scaled forward message to time step t for audio file r, and let $c_t^{(r)}$ be the scaling factor to time step t for audio file r. By using induction, it can be shown from Equation 5.2 that [26]:

$$f_t^{(r)} = \prod_{i=1}^{t} c_i^{(r)} \alpha_t^{(r)} \tag{5.5}$$

Because $f_T^{(r)}$ is normalized, we have:

$$\sum_{S_T} f_T^{(r)} = 1$$
$$\Updownarrow \quad \text{Using Equation 5.5}$$
$$\sum_{S_T} \prod_{t=1}^{T} c_t^{(r)} \alpha_T^{(r)} = 1$$
$$\Updownarrow \quad c_t^{(r)} \text{ is not dependend of } S_T$$
$$\prod_{t=1}^{T} c_t^{(r)} \sum_{S_T} \alpha_T^{(r)} = 1$$
$$\Updownarrow \quad \text{Using Equation 4.35}$$
$$\prod_{t=1}^{T} c_t^{(r)} P(\mathbf{x}_{1:T}^{(r)}|\lambda) = 1$$
$$\Updownarrow$$
$$P(\mathbf{x}_{1:T}^{(r)}|\lambda) = \frac{1}{\prod_{t=1}^{T} c_t^{(r)}}$$

So $P(\mathbf{x}_{1:T}^{(r)}|\lambda)$ can be found from the scaling factors, and the log-likelihood of λ given all the data $\mathcal{X}$ then becomes:

$$\log P(\mathcal{X}|\lambda) = \sum_{r=1}^{R} \log \frac{1}{\prod_{t=1}^{T} c_t^{(r)}} = -\sum_{r=1}^{R} \sum_{t=1}^{T} \log c_t^{(r)} \tag{5.6}$$

5.3.1.2 Using matrix operations

Let the integer `noStates` be denoted N, the vector `start` be denoted by π, and let the matrix `transition` be denoted by A. For an observation at time t, $\mathbf{x}_t$, the emission $P(\mathbf{x}_t|S_t)$ can be calculated for each state using Equation 3.5 from Section 3.2.3. The result can then be represented by a vector E_t where entry n is the result of the mixture distribution of the GMM to state n i.e. the result of $P(\mathbf{x}_x|S_t = n)$. Equation 5.2 for forward message can then be written as [30]:

$$f_{1:t} = c_t \cdot E_t \cdot A^T \cdot f_{1:t-1} \tag{5.7}$$

Russell and Norvig [30] suggest that E_t can be represented as a diagonal matrix, and the calculations can then be performed by using matrix-vector operations. However, this will result in several unnecessary calculations, and in order to avoid this component wise multiplication with E_t can be used instead. This is done by calculating $A^T \cdot f_{1:t-1}$ first with matrix-vector

multiplication. The result is a vector which must be multiplied component wise with E_t and scaled. The implementation of this can be seen in Listing 5.2. In the listing `lastForward` is the scaled forward from last iteration, and $sensor$ is the result of $P(\mathbf{x}_x|S_t = n)$. Scaling is done after the call to this method.

```
1   vector<double> HMM::forward(const vector<double> &lastForward, const vector<double> &
        sensor) const
2   {
3       vector<double> step1(noStates);
4       vector<double> step2(noStates);
5       step1 = prod(trans(*transition),lastForward);
6       step2 = componentwiseProd(sensor, step1);
7       double sum = 0;
8       for(unsigned i=0; i<this->noStates; ++i)
9       {
10          sum += step2[i];
11      }
12      if(!boost::math::isnormal(sum))
13      {
14          throw OutlierException(sum);
15      }
16      return step2;
17  }
```

Listing 5.2: Forward message.

Equation 5.4 for backward message can be written as [30]:

$$b_{t:T} = A \cdot E_{t+1} \cdot b_{t+1:T} \tag{5.8}$$

Similar to the forward message, it is suggested by Russell and Norvig [30] to represent E_t as a diagonal matrix. But like with the forward message, we used component wise multiplication of vectors instead. The implementation of this can be seen in Listing 5.3. In the listing `lastBackward` is the scaled backward message from last iteration, and $sensor$ is the result of $P(\mathbf{x}_x|S_t = n)$. Scaling is done after the call to this method.

```
1   vector<double> HMM::backward(const vector<double> &lastBackward, const vector<double>
        &sensor)
2   {
3       vector<double> step1(noStates);
4       vector<double> step2(noStates);
5       step1 = componentwiseProd(sensor, lastBackward);
6       step2 = prod(*transition,step1);
7       double sum = 0;
8       for(unsigned i=0; i<this->noStates; ++i)
9       {
10          sum += step2[i];
11      }
12      if(!boost::math::isnormal(sum))
13      {
14          throw OutlierException(sum);
15      }
16      return step2;
17  }
```

Listing 5.3: Backward message.

When calculating f_t, the value of the last forward step f_{t-1} is needed, so all the forward steps for $t = 1, \ldots, T$ are calculated first in learning and cached such that it is not necessary to calculated $f_{1:t-1}$ every time f_t is used. The same is done for backward message because b_{t+1} is required in order to calculate b_t.

5.3.2 Learning

Learning was implemented directly from the explanation given in Section 4.3. The values of forward messages, backward messages, smoothing variables, and $P(S_t, C_t|\mathbf{x}_{1:T})$ are cached because these values are used several times. The flow chart of learning can be seen in Figure 5.4. As with GMM several different initial values of the parameters λ are used in order to increase the probability of finding a local maximum close to the global maximum. The initial values of the start and transition distribution are the same for each initial value of λ, but the parameters of the GMMs are changed. This is done by randomly distribute the feature vectors to the states differently for each start point and use k-means to cluster the feature vectors.

5.4 Classification

When a positive and a negative model have been learned, they can be used to classify new audio files, as it is shown in our classification system in Figure 1.1. Let $\lambda_{positive}$ be the parameters of the positive model, let $\lambda_{negative}$ be the parameters for the negative model, and let $\mathbf{x}_{1:T}$ be the feature vectors of an audio file that must be classified. We then find $\frac{P(\mathbf{x}_{1:T}|\lambda_{positive})}{P(\mathbf{x}_{1:T}|\lambda_{negative})}$ which is compared with a threshold. However as explained earlier, we have implemented

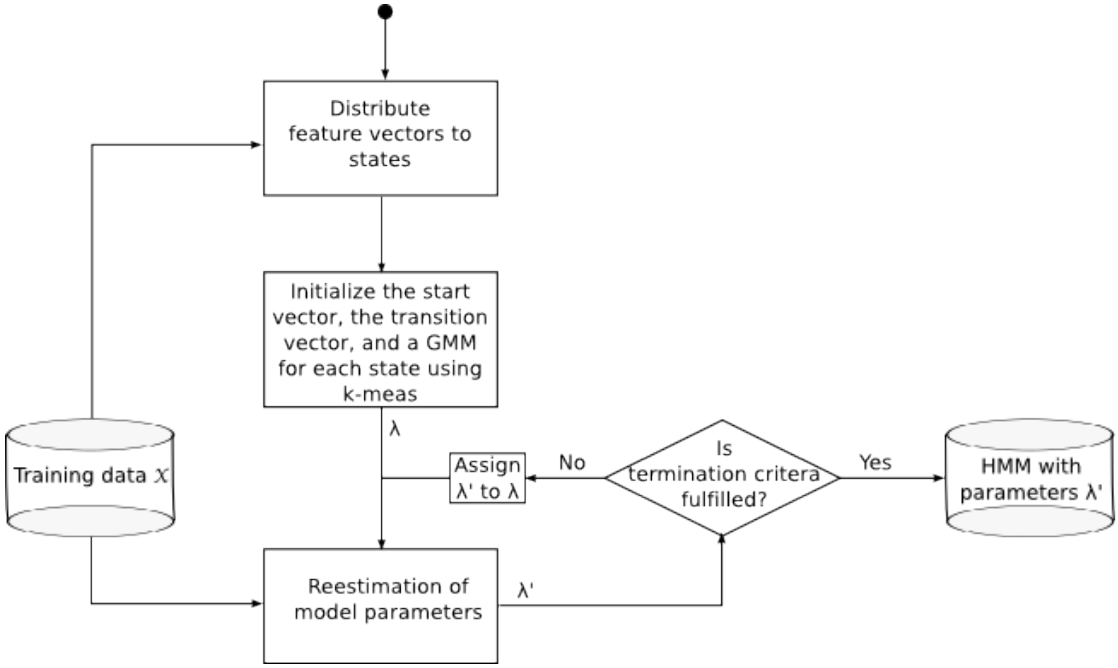


Figure 5.4: Flow chart for learning a HMM.

the log-likelihood instead of the likelihood, so $\frac{P(\mathbf{x}_{1:T}|\lambda_{positive})}{P(\mathbf{x}_{1:T}|\lambda_{negative})}$ must be rewritten such that the log-likelihood can be used for calculating the ratio:

$$\begin{aligned}\frac{P(\mathbf{x}_{1:T}|\lambda_{positive})}{P(\mathbf{x}_{1:T}|\lambda_{negative})} &= e^{\log\left(\frac{P(\mathbf{x}_{1:T}|\lambda_{positive})}{P(\mathbf{x}_{1:T}|\lambda_{negative})}\right)} \\ &= e^{\log(P(\mathbf{x}_{1:T}|\lambda_{positive}))-\log(P(\mathbf{x}_{1:T}|\lambda_{positive}))}\end{aligned}$$

Here $\log(P(\mathbf{x}_{1:T}|\lambda_{positive}))$ is the log-likelihood of the positive model, and $\log(P(\mathbf{x}_{1:T}|\lambda_{negative}))$ is the log-likelihood for the negative model. The implementation of classification therefore finds $\log(P(\mathbf{x}_{1:T}|\lambda_{positive}))$ and $\log(P(\mathbf{x}_{1:T}|\lambda_{negative}))$, and returns $e^{\log(P(\mathbf{x}_{1:T}|\lambda_{positive}))-\log(P(\mathbf{x}_{1:T}|\lambda_{negative}))}$.

6

Experiments

The classification approach presented in this thesis was tested by classifying the audio files in a data set that was reserved for testing, and from this compute various metrics for the performance of the classification, i.e. how well the classifier detects the files containing up-calls, and rejects the files not containing up-calls. The metrics used for measuring the performance of the classifier are introduced in Section 6.1.

As shown in Figure 1.1 two models are used when classifying an audio file with feature data $\mathbf{x}$: A model learned from the positive labeled files, and another model learned from the negative labeled files. Let $P(\mathbf{x}|\lambda_{positive})$ be the likelihood of the model for the positive model, and let $P(\mathbf{x}|\lambda_{negative})$ be the likelihood for the negative model. Then the result of the classification of the audio file is the ratio $\frac{P(\mathbf{x}|\lambda_{positive})}{P(\mathbf{x}|\lambda_{negative})}$. Thus the result is high, if the likelihood of the model for the positive class is high, and the result is low, if the likelihood of the model for the negative class is high. A threshold must be chosen in order to decide the class of the audio file from the found ratio. If the ratio of the classification of an audio file is higher than the threshold, the audio file is classified as being positive, otherwise it is classified as being negative.

We have investigated the performance of three different types of models when used in the classification procedure shown in Figure 1.1. The first model type considered each audio file as a single frame, when extracting the features, and modeled these using a single GMM. For the second model type, each audio signal was divided up into overlapping frames and the features

were then extracted from each of them. The features for each frame were then modeled using a GMM for each frame. These GMMs had the same number of components but were learned individually from different frames. For the third model type an HMM was used in order to capture the development in the process that we assumed generated the audio signal. For the HMM, the audio files were also divided into overlapping frames. Models of different types can be combined in the classification process, e.g. having an HMM for the positive class and a GMM for the negative class. However we limited ourselves to not combining the models of different type in order to keep the testing less complicated.

In order to find the two models for the classification procedure there are many parameters which must be determined. For the feature extraction process, see Section 2.1, the frame length and the size of the overlap of the frames must be decided in order to compute the feature data. Furthermore a particular model contains various parameters such as mean vectors, covariance matrices, and probability tables. These parameters are learned from data using the EM-algorithm, and thus we refer to this process as *parameter learning*. Finally the number of components and states must be decided, these parameters denote the cardinality of variables in the model, and can not be learned using the EM-algorithm since they determine the number of model parameters. These parameters are decided by learning several different models where the number of components and states are varied, and then the learned models are validated. Thus we have three groups of parameters. The first group is parameters for the feature extraction, the second group is the model parameters, which we learn with the EM- algorithm, and the third group is the parameters that describe the cardinality of the model parameters which we determined by validation.

We therefore require three data sets. A data set for parameter learning, we refer to this as the *training set*, a data set for selecting among the learned models, we refer to this as the *validation set*, and a data set for testing the final model by measuring it performance, we refer to this as the *test set*.

Our test setup, and how the parameter space was explored for validation are described in Section 6.2. The classification procedure presented in this thesis was tested in two stages. The purpose of the first stage was to demonstrate how well our method performs in relation to the Kaggle competition. This is presented in Section 6.3. The purpose of the second stage was to present a further analysis of the results, and investigate what an HMM captures from the training data. This is presented in Section 6.4.

6.1 Performance metrics

The performance metrics, which we used, can be computed from the entries of the so called *confusion matrix* which is illustrated in Figure 6.1 [32]. The binary result of the classifier is re-

ferred to as positive if it was predicted to contain a right whale up-call and negative otherwise. Whether the classification was correct or incorrect is refereed to as true or false, respectively. The entries of the confusion matrix are:

- True Positive (TP): Number of audio files which were classified to be positive and contained an up-call
- True Negative (TN): Number of audio files which were classified to be negative and did not contain an up-call
- False Positive (FP): Number of audio files which were classified to be positive but did not contain an up-call
- False Negative (FN): Number of audio files which were classified to be negative but did contain an up-call

	Predicted positive	Predicted negative
Actual positive	TP	FN
Actual negative	FP	TN

Figure 6.1: Confusion Matrix.

6.1.1 ROC curves

This section is rewritten from a section in [17] which describes ROC curves. The evaluation metric used for selecting the number of components and states are the Area Under Curve of the Receiver Operating Characteristic (ROC) curves. AROC curve illustrates the relation between the two rates [32]:

- True Positive Rate (TPR): The fraction of positive labeled audio files that was classified correctly
- False Positive Rate (FPR): The fraction of negative labeled audio files that was classified as being positive

Then TPR and FPR can be calculated using Equation 6.1 and Equation 6.2, respectively.

$$TPR = \frac{TP}{TP + FN} \tag{6.1}$$

$$FPR = \frac{FP}{FP + TN} \tag{6.2}$$

The classification procedure, shown in Figure 1.1, must be performed for each audio file in the test data set. The computed class for each audio file must then be compared to the its actual

label in order to determine in which of the four entries of the confusion matrix the audio file belong. When every file in the test data set has been assigned to an entry of the confusion matrix, the TPR and FPR can be computed. For a given threshold the TPR and the FPR constitute a point on the ROC curve. By varying the classification threshold over an appropriate interval several points on the ROC curve are obtained. The ROC curve can be plotted in a coordinate system where the horizontal axis represents the FPR and the vertical axis represents the TPR [32]. Each point on the curve represents the performance of the classifier for a particular threshold. If TPR = FPR it would correspond to random guessing, and if TPR > FPR the prediction is better than random guessing. The larger TPR is, compared to FPR, the better. A good classification should be close to the point $(0, 1)$. To compare classifiers it is convenient to reduce the ROC curve to a single scalar representing the average performance of the classifier. A common method is to use the Area Under Curve (AUC) of the ROC curve [32]. If the classifier is perfect the AUC is 1, while the AUC for random guessing is 0.5. A classifier that is better than another classifier would have a larger AUC. The ROC curves and AUC used in the work of this thesis was computed using the Matlab function *perfcurve* [8].

6.1.2 Precision, recall and accuracy

The entries of the confusion matrix can be used for calculate other performance metrics. In this thesis we used *accuracy*, *precision*, and *recall* which can found by Equation 6.3, Equation 6.4, and Equation 6.5, respectively [16, 32].

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{6.3}$$

$$Precision = \frac{TP}{TP + FP} \tag{6.4}$$

$$Recall = \frac{TP}{TP + FN} \tag{6.5}$$

The accuracy is the percentage of instances that the classifier predicted correctly, and thus it does not make any distinctions between the positive and negative class. For the problem addressed by this thesis, the audio files in the positive class are more significant than the negative, since it is the positive that we are trying to recognize. When this is the case precision and recall are often used [32]. They are computed with respect to one of the class of interest. For the problem addressed in this thesis we are interested in the audio files containing a right whale up call. Precision is the fraction of audio files that actually contain an up-call in the group of audio files that the classifier predicted to be positive. Precision is high when the classifier makes few false positive errors. Recall is the fraction of audio files containing an up-call that the classifier

recognized correctly. Recall is high when the classifier made few false negative errors. We can increase precision of our classifier by increasing the threshold in Figure 1.1. This will decrease the number of files that are predicted positive. However this will reduce the recall since it also increases the risk of predicting a file to not contain an up-call although it actually did. Precision and recall expresses the tradeoff between the error of FP and FN and are sometimes visualized as a prediction-recall curve. For making this curve, the precision and recall are computed for various thresholds and plotted into a graph in a coordinate system. All the precision-recall curves used for this thesis were computed using the Matlab function *perfcurve* [8].

Precision and recall can be combined into a single scalar, called the F_β-*measure*, which is calculated by Equation 6.6 where the value of β express the tread-off between prediction and recall [32]. Since we do not know the utility function for the right whale detection problem, we used the F_1-measure which is the harmonic mean between the recall and precision. The F_1-measure is shown in Equation 6.7.

$$F_\beta = \frac{(1+\beta^2)recall \cdot precision}{recall + \beta^2 precision} \tag{6.6}$$

$$F_1 = \frac{(1+1^2)recall \cdot precision}{recall + 1^2 precision} = \frac{2 \cdot recall \cdot precision}{recall + precision} \tag{6.7}$$

6.2 Test setup

We compare the models types by computing the AUC of the ROC curves for the learned models on the test set. This is done in Section 6.3. In order to perform further analysis of the models we also computed the accuracy, precision and recall in Section 6.4. In this section it also presented that we found the most likely path through the state space for the Markov process for three positive audio files using a learned positive HMM, and compared it with spectrogram plots for the audio files in order to investigate if we could identify states that are part of the up-call and states that are noise. Further we learned additional linear models to investigate whether it is possible to detect the position of an up-call in an audio file from the most likely state sequence for the audio file.

As mentioned in the introduction of this chapter the EM-algorithm can be used to learn the model parameters such as the probability tables, but not the number of states and components using the EM-algorithm. Instead we learned models with different number of states and components for both the positive and negative model. Then we used the validation set to select the best model pair containing both a positive and a negative model. This was done by pairing each positive model with each negative model and then use each of these pairs in the classification process on the validation set. We then selected the models of the pair that yield the highest

AUC of the ROC curves. In this way the number of states and components were selected for both the positive and the negative model. The model pairs can be any combination of a positive and negative model. However, in order to simplify our experiments chapter, we limited ourselves to only pair models of the same type.

The number of components and states for the models that we learned from the training set, and validated on our validation set are presented in Table 6.1. In order to simplify the test, we varied the number of components and states over the same values for the positive models as we did for the negative models. As shown in the first and second row of Table 6.1, for the models using either a single GMM or several GMMs, we varied the number of components from 1 up to 10. We did not exceed 10 components because we wanted to restrict the complexity of the used GMMs in order to learn the model parameters in a reasonable time. As shown in the third row of Table 6.1, for the models using an HMM we varied the number of states from 5 to 15 in steps of 5, and the number of components from 1 to 4. Here we restricted the number of components to be 4 with the presumption that the space of the HMM compensate for the reduced number of components. We only validated HMMs with 5, 10, and 15 states in order limit the parameter space that was explored. Ideally there should not be any gap between the parameters for the models that were investigated, however it was necessary here because of time constraints and limited computer power. As mentioned in Section 2.2, we used a frame length of both 4000 samples and 512 samples. The frame length of 4000 corresponds to two seconds, i.e. the duration of each audio file, so the audio file was therefore represented as one frame. This was the frame length used for the model type which used a singe GMM. The frame length of 512 samples was used for the model type which used several GMMs and the model type which used a HMM. The overlap of the frames was set to $\frac{2}{3}$. This gave 22 frames, so $T = 22$.

As termination criterion we used change in absolute value of log-likelihood and change in relative value of log-likelihood. For the experiments in Section 6.3 we terminated the EM algorithm when the absolute value went below a threshold of 10^{-6}, while in Section 6.4 we terminated the EM-algorithm when the relative value went below a threshold of 10^{-5}. The latter leads to earlier termination, and we chose to use this rather than the absolute value for all experiments subsequent to the experiments in order to learn the required models in the time window of this project.

6.3 Kaggle results

Kaggle provided two disjoint data sets; a public training set containing 29354[1] labeled audio files, and a private test set containing 54503 audio files where the label was not published and

[1]The actually size of the public training set was 30000, but, as explained in Section 2.3, 646 outliers and duplicates was removed.

GMM	No. of components	
	1, 2, 3, 4, 5, 6, 7, 8, 9 and 10	
GMMs	No. of components	
	1, 2, 3, 4, 5, 6, 7, 8, 9 and 10	
HMM	No. of states	No. of components
	5	1
	5	2
	5	3
	5	4
	10	1
	10	2
	10	3
	10	4
	15	1
	15	2
	15	3
	15	4

Table 6.1: The different number of components and states used for the learned models.

therefore unknown for the Kaggle competition participants. The public data set was split up such that $2/3$ of the files for each label were used for learning the model parameters λ with the EM algorithm. The remaining audio files in the public data set were used for selecting the number of components and states. The private Kaggle data set was used for testing the performance of our models when used for classification. Figure 6.2 illustrates the partitioning of the available audio files. The models for the Kaggle results were trained with five initial points for the EM algorithm, and the absolute difference of log-likelihood with a threshold of 10^{-6} was used as termination criterion.

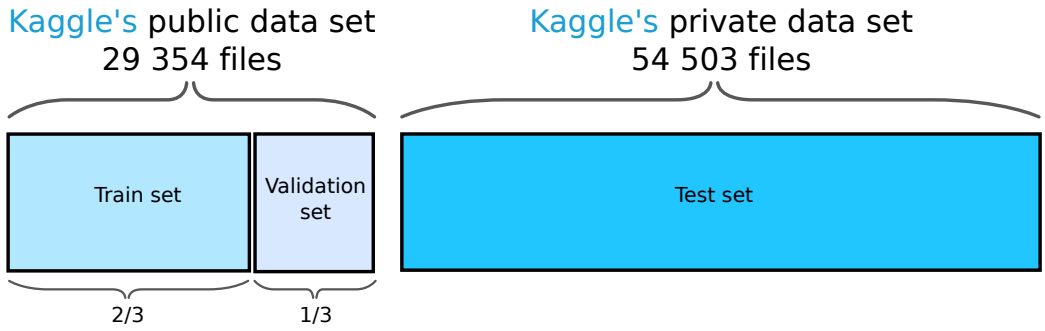


Figure 6.2: Partitioning of the available data.

6.3.1 Model selection

The model parameters such as weights, mean vectors, and covariance matrices for a GMM, or start and transition conditional probabilities for an HMM were learned using the EM algorithm. However, in order to determine the number of components and number of states, several models were learned and compared using the validation set. This data set contained 9784 of the files from the public data set. The values that the number of components and states were varied over can be seen in Table 6.1. For each model type each positive model was paired with each negative model in the classifier for classifying the audio files in the validation set, and we selected the pair which gave the highest AUC of the ROC curve.

The five best validation results for a classifier using a single GMM for each class are shown in Table 6.2. The best result was for a positive model with 5 components and a negative model with 10 components. These were then used for testing in Section 6.3.2.

The five best validation results for a classifier using 22 GMMs for each class are shown in Table 6.3. The best result was for a positive model with 10 components and a negative model with 10 components. These were then used for testing in Section 6.3.2.

The five best validation results for a classifier using an HMM for each class are shown in Table 6.4. The best result was for a positive model with 10 states and 2 components and a negative model with 10 states and 4 components. These were then used for testing in Section 6.3.2.

Positive model no. components	Negative model no. components	AUC of ROC curve
5	10	0.92371
6	10	0.92269
7	10	0.92264
5	9	0.92244
5	7	0.92186

Table 6.2: The 5 best results for validation when modeling the signal as a single frame using a GMM.

Positive model no. components	Negative model no. components	AUC of ROC curve
10	10	0.91675
10	9	0.91643
8	10	0.9164
9	10	0.91614
8	9	0.91612

Table 6.3: The 5 best results for validation when modeling the signal as 22 frames using a GMM for each frame.

Positive model		Negative model		
no. states	no. components	no. states	no. components	AUC of ROC curve
10	2	10	4	0.89621
15	3	15	3	0.89607
15	3	15	4	0.8959
15	2	15	3	0.89565
15	4	15	3	0.89494

Table 6.4: The 5 best results for validation when modeling the signal as 22 frames using an HMM.

6.3.2 Test

Using the pair of models which gave the best results in Table 6.2, Table 6.3, and Table 6.4, respectively, $\frac{P(\mathbf{x}|\lambda_{positive})}{P(\mathbf{x}|\lambda_{negative})}$ was calculated for the feature data $\mathbf{x}$ for each of the 54503 audio files in the private data set from Kaggle. The results were load up to the Kaggle web page, and AUC of the ROC curve was returned. The results are listed in Table 6.5.

Model type	**Positive model**	**Negative model**	AUC of ROC curve
GMM	5 components	10 components	0.92106
GMMs	10 components	10 components	0.91260
HMM	10 states, 2 components	10 states 4 components	0.88902

Table 6.5: Test results for the Kaggle setup tests.

The AUC for the classifier using HMMs is smaller than for the two classifiers using respectively one and several GMMs. In fact the simplest approach using only a single GMM performs best. This do, indeed, not support our expectations; by using an HMM the model should capture the development in the process that generated the signals in the audio files and, thus, perform better when used for classifying the audio files in the test set. If we look at the last column in Table 6.2, Table 6.3, and Table 6.4, the difference in AUC is in the third decimal point. The AUC therefore seems robust for change in the number of components and states. In Section 6.4 a closer analysis of the models, and an investigation of what the classifier using HMMs have problem capturing are presented.

6.4 Further analysis

In this section we make a further analysis of the models, in order to investigate what they have problems capturing. In order to analyze the models we looked at additional metrics for measuring the models performance. Further we investigated whether a group of states of the Markov process corresponds to an up-call.

6.4.1 Our setup

In Section 6.3 we only looked at the AUC of the ROC curves. In this section it is presented how we analyzed the classifiers more thoroughly by looking at accuracy, precision, and recall. Further we drew the ROC curve, precision-recall curve, and computed the confusion matrices for a single threshold value. This required that we had access to the actual labels of each audio file in the data set, and we could therefore not use the private Kaggle test data set. The models in this section were therefore learned independently from the models in Section 6.3. The available data were partitioned into new disjoint data sets, but here we only used the audio files from the public Kaggle data set. $1/3$ of the files in the public data set were reserved for tests, $4/9$ were used for learning the weights, mean vectors, covariance matrices, and the start and transition probability distribution tables using the EM algorithm, and the remaining $2/9$ were used for selecting the number of components and states. The partition of the data can be seen in Figure 6.3.

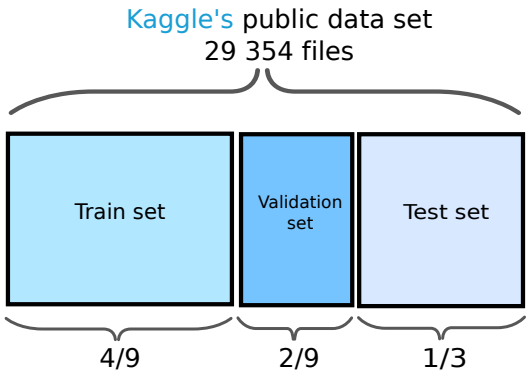


Figure 6.3: Partitioning the data for the further analysis.

6.4.2 Model selection

As in Section 6.3.1, we selected the cardinality of the hidden variables by learning the models presented in Table 6.1. However, this time the training set presented in Section 6.4.1 was used. Again, for each model type each positive model was paired with each negative model in the classifier for classifying the audio files in the validation set, and we selected the pair which gave the highest AUC of the ROC curve. The model was trained with five randomly selected initial points for the EM algorithm, and the relative distance of log-likelihood with a threshold of 10^{-4} was used as termination criterion. We varied the number of components and states over the same values as in Section 6.3.1, which can be seen in Table 6.1

The five best validation results for a classifier using a single GMM for each class is shown in Table 6.6. The best result was for a positive model with 3 components and a negative model with 5 components. These were then used for testing in Section 6.4.3.

The five best model selection results for a classifier using several GMMs for each class is shown

in Table 6.7. The best result was for a positive model with 10 components and a negative model with 9 components. These were then used for testing in Section 6.4.3.

The five best model selection results for a classifier using a HMM for each class is shown in Table 6.8. The best result was for a positive model with 15 states and 2 components, and a negative model with 15 states and 3 components. These were then used for testing in Section 6.4.3.

Positive model no. components	Negative model no. components	AUC of ROC curve
3	5	0.9176
4	8	0.91738
4	5	0.91727
7	10	0.9172
3	6	0.91639

Table 6.6: The 5 best results for validation when modeling the signal as a single frame using GMM.

Positive model no. components	Negative model no. components	AUC of ROC curve
10	9	0.90315
7	10	0.90311
6	10	0.90288
7	9	0.90287
10	10	0.90286

Table 6.7: The 5 best results for validation when modeling the signal as 22 frames using a GMM for each frame.

Positive model		Negative model		
no. states	no. components	no states	no. components	AUC of ROC curve
15	2	15	3	0.88954
15	4	15	3	0.88848
10	3	15	3	0.88671
10	4	15	3	0.8867
15	1	15	3	0.88662

Table 6.8: The 5 best results for validation when modeling the signal as 22 frames using HMM.

6.4.3 Tests

Using the files from the data set that was reserved for testing, see Figure 6.3, the performance measures can be computed on the best model pairs in Table 6.6, Table 6.7 and Table 6.8.

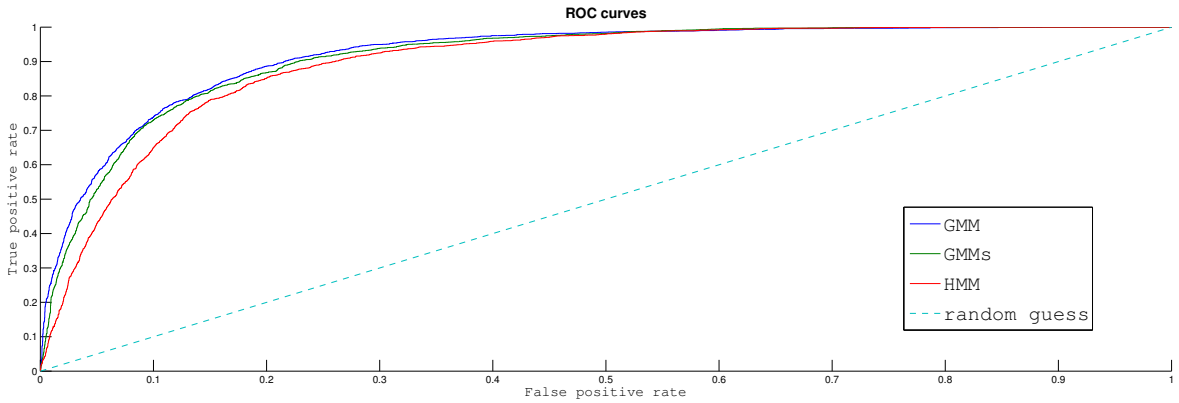


Figure 6.4: ROC curves of the final models.

6.4.3.1 Receiver operating characteristic curves

Figure 6.4 shows the ROC curve for the classifier using, respectively, a GMM, several GMMs, and an HMM for representing the features of an audio file. The ROC curves also indicate that the approaches using either a single GMM or several GMMs perform better than the approach using a HMM, since the curve for the HMM based classifier is dominated by the classifiers using a single and several GMMs respectively. Actually, as their AUC, the ROC curves indicate that the simplest approach with a single GMM for each class of audio files performs best.

6.4.3.2 Precision-recall curves

Figure 6.5 shows the precision-recall curve for the classifier using respectively a GMM, several GMMs, and an HMM for representing the features of an audio file. The precision-recall curves unfortunately also shows that the approaches using GMMs overall performs better than the approach using HMMs, since their curves dominate the curve for the HMM based classifier.

6.4.3.3 Accuracy and F_1-value

For each threshold there is an accuracy and a F_1-measure. In Table 6.9 the largest accuracy and F_1-measure of all thresholds are shown. Furthermore AUC of the ROC curves are shown for the classifier using, respectively, a GMM, several GMMs, and a HMM for representing the features of an audio file. Both accuracy and the F_1-value indicate that GMM is better than HMM at classifying the audio files.

This contradicts with results to those found by Brown and Smaragdis [11] who investigate the use of GMMs and HMMs to classify killer whale calls into seven classes and report accuracy as their result. The best accuracy found by Brown and Smaragdis was over 95%, which was obtained for the HMMs with 13 to 17 states, one GMM per state, and 24 to 30 MFCCs. Their best

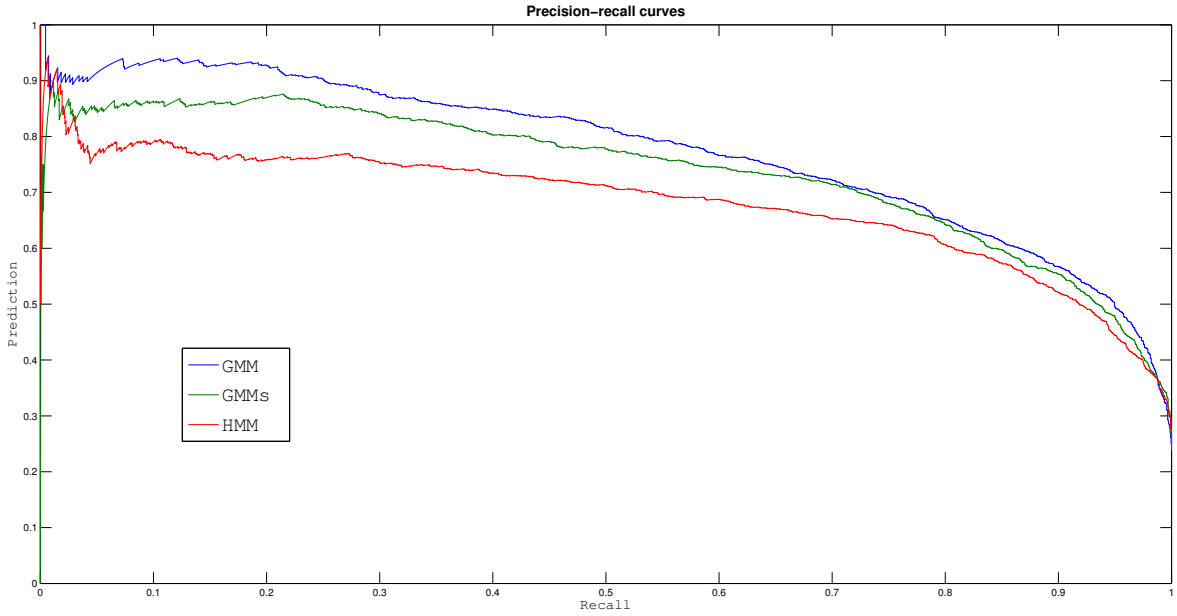


Figure 6.5: Precision-recall for final Models.

result using a GMM was 92%, which was obtained by using GMMs with two components and 30 MFCCs. It is not easy to compare our result to theirs. Besides working on a different domain and problem, they do not report the width of the filters in the their filter bank or the number of filters in the bank, thus do we not know the frequency area that their features cover. However, for 12 MFCCs, which is the number coefficients we used, they obtained an accuracy from 78% to 82% for GMMs, and an accuracy from 85% to 89% using HMMs. Generally their HMMs performed better than their GMMs.

Model type	Best Accuracy	best F_1-measure	AUC
GMM	0.86427	0.72399	0.91931
GMMs	0.86202	0.71615	0.91193
HMM	0.84076	0.69606	0.89514

Table 6.9: Performance metrics for final models.

6.4.3.4 Confusion matrix

In Figure 6.6, Figure 6.7, and Figure 6.8 the confusion matrix for the classifier that uses a GMM, several GMMs, and an HMM are shown. The threshold used is the threshold for the best result of the ROC curves in Figure 6.4. The HMM has lowest numbers of TP and TN predictions, and highest numbers of FP and FN predictions. The HMM has especially a high number of FP predictions. This indicates that it recognizes noisy audio files as containing an up-call.

	Predicted positive	Predicted negative
Actual positive	2030	309
Actual negative	1357	6088

Figure 6.6: Confusion Matrix for GMM threshold found from ROC curve.

	Predicted positive	Predicted negative
Actual positive	1990	349
Actual negative	1342	6103

Figure 6.7: Confusion Matrix for GMMs threshold found from ROC curve.

6.4.4 Model investigation

In order to investigate whether there are a connection between the state of the Markov process and the frame in the audio file containing an up-call we computed the most likely path through the state space of a positive HMM, and compared this to a spectrogram of the audio file. For a particular HMM we can use the Viterbi algorithm [26] to compute $\arg\max_{s_{1:T}} P(s_{1:T}|\mathbf{x}_{1:T})$ which is a sequence of states $s_{1:T}$ that is most likely to have generated a particular observation sequence $\mathbf{x}_{1:T}$. This is sometimes referred to as the Viterbi path or the most likely explanation. The Viterbi algorithm is presented in Appendix C. By computing the Viterbi path for an audio file in our test set we can analyze the meaning of the states and try identifying if there are some states which correspond to an up-call. We have chosen to do this for three audio files. We did this for the HMM with the highest AUC score seen in Table 6.8 and for four linear left-right HMMs, each having 3 states but different number of components.

6.4.4.1 Hidden Markov Model from Section 6.4.3

We use the positive HMM with the largest AUC in Table 6.8 i.e. the positive HMM with 15 states and 2 components. The result is shown in Figure 6.9 where the spectrogram of the three audio files and the Viterbi path for each audio file is shown. In order to understand the figure one must recall that the frames overlap with $\frac{2}{3}$ which means that they are displaced with $\frac{1}{3}$ of a frame from the right. A frame starts at a certain line, and ends when the next line of the same type appears where a new frames also starts. The Viterbi path is written such that the state of a frame is written right after line which starts the frame, i.e. under the first $\frac{1}{3}$ part of the

	Predicted positive	Predicted negative
Actual positive	1969	370
Actual negative	1413	6032

Figure 6.8: Confusion Matrix for HMM threshold found from ROC curve.

frame.

No ordering was assumed prior to executing the EM algorithm for the models in Table 6.8, and thus the state index do not imply any order. The result indicates that state $4, 5, 8, 9$ and 10 are used for modeling noise, while state $1, 6, 7, 9, 10, 12$ and 15 are used for modeling up-calls. There are some similarities for the up-calls in the first and second audio file while the up-call in the third audio file seems to differ. The Viterbi path for the up-call in the first, second, and third audio file visits the states $1 - 7 - 10$, $1 - 15 - 7 - 10$, and $9 - 6 - 12 - 10$, respectively. When not looking at how many frames the model stay in each state the only different between the two paths in the first and second audio file are that the up-call in the second audio file goes through state 15 which the up-call in the first audio file does not.

The state 9 and 10 are used both for modeling noise and up-calls which is unfortunate. State 10 seems to represent the time in the end and after the up-call. These results however indicates that the HMM detects up-call in some degree.

6.4.4.2 Linear left-right models

That the GMM classifier performs better than a classifier using HMMs do, indeed, not correspond with our expectations. We assumed that an HMM would capture the development in the signals and model this as a noise part, followed by the up-call, and then followed by a noise part. The learned HMM in Section 6.4.3 was, however, ergodic which means that all states can be reached from every states [26]. A more directly approach could be to learn a simple linear left-right model with only three states which should model the noise - up-call - noise development. A linear left-right HMM is an HMM with transitions such that the model only can stay in the same state or transit to a state with a number one higher than the current state number. The initializing of the start and transition distribution before learning then is:

$$P(S_1) = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

$$P(S_t|S_{t-1}) = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{pmatrix}$$

The entries set to 0 will stay as 0 doing learning. In order to initialize the GMMs associated to the states, the first a frames of an audio file are distributed to state 1, the next b frames are distributed to state 2, and the last c states are distributed to state 3. For each audio file a, b and c is drawn randomly such that $a + b + c = 22$. The GMMs are then initialized using k-means as

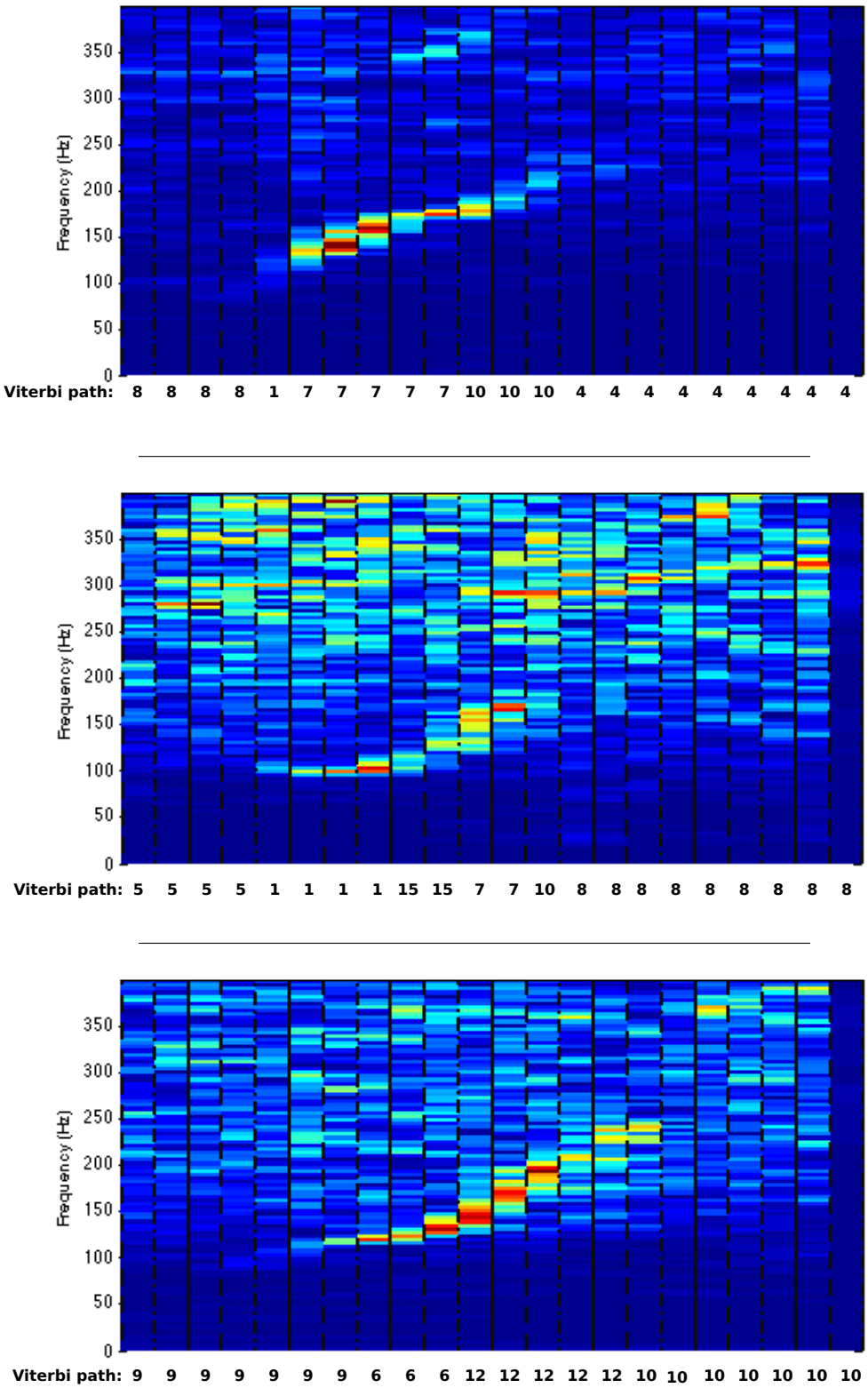


Figure 6.9: Viterbi paths for the positive HMM from Table 6.8, and three different audio files containing an up-call.

explained in Section 4.2.2.

The point then is that state 1 takes care of the frames before the up-call, state 2 takes care of the frames during the up-call, and state 3 takes care of the frames after the up-call. Such a model would not capture the development of the up-call because there is only one state for the up-call frames, and, thus, the up-call part must be captured using the GMM associated with the up-call state.

Again we would like to see whether the model can capture the up-call by looking at the Viterbi path. We learn four positive linear left-right HMMs with 3 states, and, respectably, $1, 2, 3$ and 4 components. We then compute the Viterbi path for each HMM on the audio files shown in Figure 6.9. The result is shown in Figure 6.10. The first line of states under the spectrograms is the Viterbi path for the HMM with 1 component, the second line is the Viterbi path for the HMM with two components, and so on. The HMM with 1 component is not able to capture the up-call in any of the audio files. The Viterbi paths for the HMMs with $2, 3$ and 4 components gives almost the same result. For the first audio file, the Viterbi paths changes to state 2 when the up-call starts and to state 3 when the up-call ends. For the second audio file, the Viterbi paths changes to state 2 too late, i.e. after the up-call has started, but changes to state 3 at the end of the up-call. However, the HMM with 4 components perhaps changes a bit too early to state 3 it can, however, be hard to see. In the third audio file it looks like all three paths changes too early to state 2, but changes to state 3 at the end of the up-call. It can, however, be the case that the up-call starts this early, but the spectrogram does not capture it. It looks like the three linear left-right HMMs with $2, 3$ and 4 components are able to detect the position of the up-call to a fair degree. Actually this could be use for removing the noisy part of the audio signal by only considering the part where the HMM is in state 2.

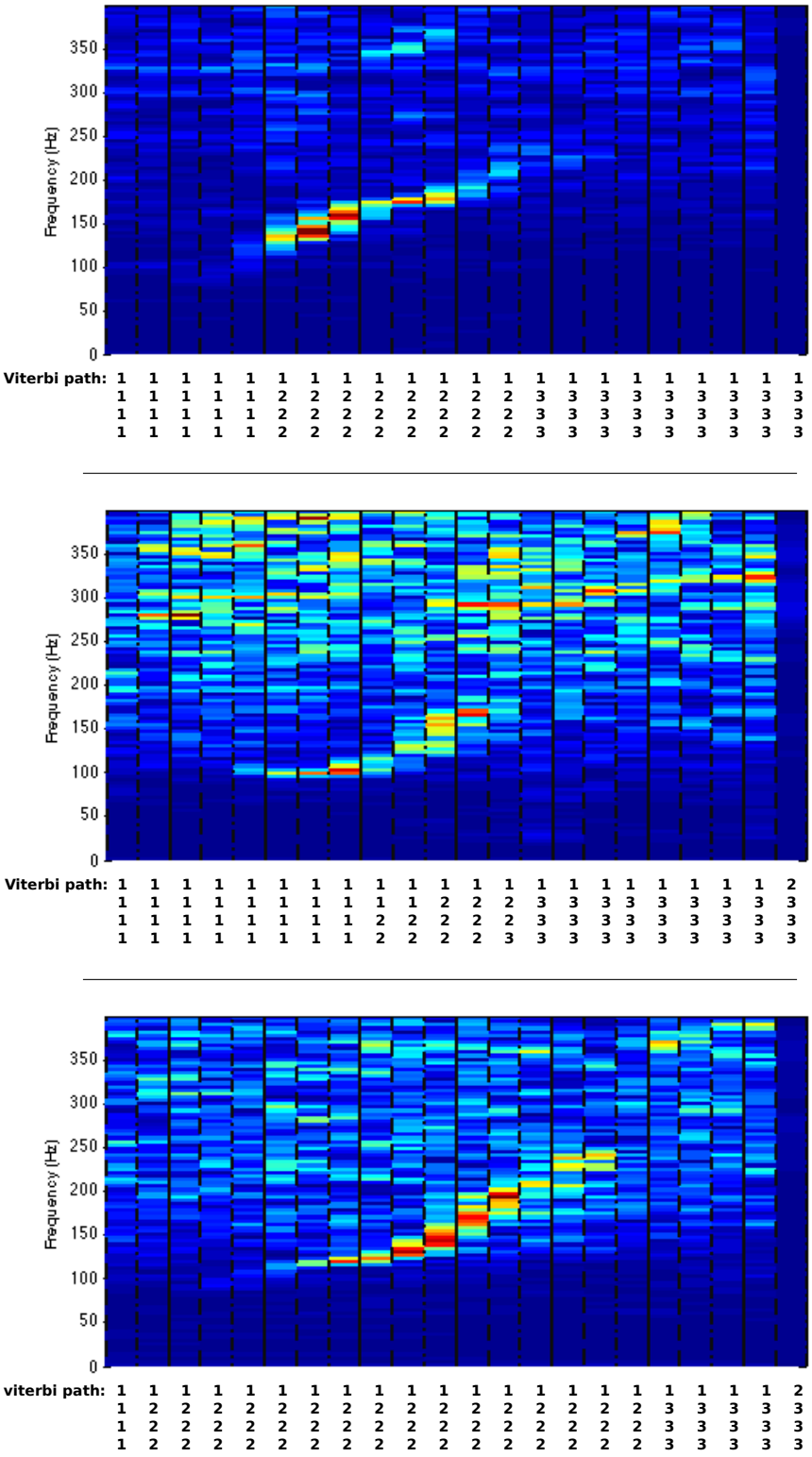


Figure 6.10: Viterbi paths for the three linear left-right HMMs having one to four components and three different audio files containing an up-call.

7

Conclusion

We have designed and implemented a classification system which, to some extend, can classify an audio file as either containing an up-call from a North Atlantic right whale or not. An audio file was represented by a feature vector for each frame. The entries of the the feature vectors were the MFCCs for the particular frame. We experimented with using different model types in the classification system to compare the performance of these for our application. The model types used in the classification system was based on GMMs and HMMs. We tried two different approaches using GMMs. In the first approach each audio file was considered as a single frame and therefore represented with one feature vector. The feature vectors for the audio files were then considered as being generated by a single GMM. In the second approach the audio files were divided up into overlapping frames. The feature vector for each frame was then considered as being generated by a GMM such that we had a GMM for each frame. In addition to the two approaches using GMMs, we also tried an approach using an HMM where there was associated a GMM to each state. Each model type was designed and implemented. The models were learned from annotated audio files using the the EM-algorithm. We therefore investigated the general theory and of the EM-algorithm, and how the two steps of the algorithm were derived for GMMs and HMMs with a GMM for each state.

In order to compare the model types, we used ROC curves, precision-recall curves, and calculated the accuracy, F_1-measure, and the AUC of the ROC curves. For all of these performance metrics, the model type that represented the audio files as a single feature vector using a GMM

gave the best results, the model type using several GMMs gave the second best results, and the HMM gave the worst result. We had expected that the HMM would give the best result because HMMs can capture the structure of an up-call. Brown and Smaragdis [11] found that HMMs were better at classifying than GMMs when classifying calls made by killer whales into seven classes. Further they obtained a higher accuracy than the accuracy we got. They, however, also experimented with different numbers of MFCCs, so it might had shown an effect in our results if we had had time for testing with different width of the filters and varied the number of MFCCs. It is, however, also worth noticing that their audio files had a sampling rate of 44100 samples/s compared to the sampling rate of 2000 samples/s which the audio files, we classified, had.

In order to investigate the models further we constructed the confusion matrices for the threshold which correspond to the point on the ROC curve closest to $(0, 1)$. For all three model types the classification system had a high number of false positive classification, but it was much higher for the HMM than for the two GMMs approaches. Since we had expected that the HMM would perform best we investigated this model type further by computing the Viterbi path for three different audio files containing an up-call in order to see which states an positive HMM was in before, during, and after an up-call. This was done for both an ergodic HMM and four linear left-right HMMs. To a certain degree the states of the ergodic HMM seemed to correspond either to the up-call part of an audio file or the noise part of the audio file. However, two states were visited both for the frames during an up-call and for frames before and after the up-call. The linear left-right HMMs were learned with three states and one to four components. The HMM with one component was not able to detect the up-calls, but the HMMs which had two, three and four components were able to detect the up-call to some degree.

7.1 Future work

Further analysis is required in order to find what the models have trouble capturing. The accuracy in our results is relatively low. Especially is precision an issue since our classifiers have a tendency to make many false positive errors. It is reasonable to believe that it is possible to improve on our results. A way to approach this would be to find and visualize the audio files in the false positive and false negative entries of the confusion matrix in order to identify which type of audio files that are hard to classify. This could for example be audio files with a noisy spectrogram and no up-call like structure but many different frequencies in each frame, or an audio file with a call that is very similar to the right whale up-calls but were made by other whale species.

We obtained better results for the classifiers using a GMM rather than an HMM for representing each class of audio files. This is conflicting with our expectations and the results reported by Brown and Smaragdis [11]. A way to investigate what the GMMs captures, that the HMMs do

not, could be to learn HMMs with only one state, and use those with our classifier. Such models would be very close to the models using a GMM because an HMM with one state, and a GMM for each state would only have the parameters of the GMM to estimate. By increasing the number of state it may be possible to analyze when the performance decreases. An approach with HMMs with only one state would not be identical to the approach using a single GMM because the latter represents the audio signal as a single frame. It could also be interesting to investigate the performance of linear left-right HMMs with the same number of states as frames, and with zero probability of transiting to the same state. Such a model would be close to the model using several independent GMMs where the audio files are represented as several frames.

Several factors can be investigated in the future to improve the classification system. Future work should include investigating the effect of using different features types. We restrained from this in order to focus on the machine learning part of the problem. The feature extraction process explained in Section 2.1 includes many parameters e.g. frame length and overlap of the frames, the number of coefficients, and the size and shape of the filters in the filter bank. By changing these it might lead to more appropriate features which carry information that could help classify the audio files containing up-calls with higher accuracy. E.g. would smaller filters and more coefficients give a better resolution of the frequency area for the up-calls, or shorter frame lengths could help capturing the development in the audio file with higher granularity.

Other types of features could be investigated e.g. the filters could be spaced linear, or the temporal derivatives of coefficients could be used [26]. Alternatively the spectrum could be used directly as features rather than the cepstrum. However, we did some experiments early in the project process which indicated that spectrum would not perform better than the MFCC.

Preprocessing of the audio files could also be investigated. Noise seemed to have a very negative effect on the performance of the classifier. It could be interesting to investigate the effect of applying different noise reducing filters such as spectral subtraction [33] to the audio files in order to suppress the marine noise. This could possibly make the HMM able to describe the up-call in a better way without being disturbed by intrusive noise.

An HMM could be used to detect that start and end frame of the up-call. The results from Section 6.4.4 indicated that it is possible to learn HMMs that can be used to detect the start and end of an up-call by looking at the Viterbi path. For example could the linear left-right HMMs with two or more components in Section 6.4.4.2 be used to detect the relevant frames. When the Viterbi path for the model transit to state 2 the relevant part of the audio signal begins, and when it transit to state 3 the relevant part of the audio file ends. A way to improve this further could be to make additional annotation of the audio files. The start and the end times for the relevant part of the audio files that contain an up-call could be annotated, and then use this

when initializing the parameters of the model. For the linear left-right HMMs with three states this would mean that the GMM of state 1, 2, and 3 would be initialized by estimating from the samples of the frames that are respectively before, under, and after the the up-call. A drawback of this approach is of cause that it requires further annotation of the audio files which is time consuming.

A

gaussian.h

```
1  /*
2   * gaussian.h
3   * Implementation of a Gaussian mixture model (GMM).
4   * Contains a Mulitvariate Gaussian Distribution class, a Component class, and a GMM
       class.
5   *
6   *  Created on: Feb 19, 2013
7   *  Author: Morten Albeck Nielsen and Stine Back Larsen
8   */
9
10 #ifndef GAUSSIAN_H_
11 #define GAUSSIAN_H_
12
13 #include <assert.h>
14
15 #include <boost/numeric/ublas/vector.hpp>
16 #include <boost/numeric/ublas/matrix.hpp>
17 #include <boost/shared_ptr.hpp>
18 #include <boost/random/uniform_real_distribution.hpp>
19 #include <boost/random/mersenne_twister.hpp>
20 #include <boost/algorithm/string/split.hpp>
21 #include <boost/algorithm/string/classification.hpp>
22 #include <boost/lexical_cast.hpp>
23
24 #include <boost/math/special_functions/fpclassify.hpp>
25
26 #include <cmath>
27 #include <fstream>
```

```
28 #include <iostream>
29 #include <limits>       // std::numeric_limits
30 #include <exception>
31
32 #include "arglist.h"
33
34 using namespace boost::numeric::ublas;
35
36 //---------------------Exception classes---------------------
37 class LoglikelihoodUnderflowException: public std::exception
38 {
39         double logLikelihood;
40 public:
41         LoglikelihoodUnderflowException(double logLikelihood):logLikelihood(
              logLikelihood){};
42
43         virtual const char* what() const throw()
44         {
45
46                 std::stringstream msg;
47                 msg<< "Loglikelihood underflow: loglikelihood="<<logLikelihood;
48
49                 return msg.str().c_str();
50         }
51 };
52
53 class SingularCovarianceMatrixException: public std::exception
54 {
55         double det;
56 public:
57         SingularCovarianceMatrixException(double det):det(det){};
58
59         virtual const char* what() const throw()
60         {
61
62                 std::stringstream msg;
63                 msg<< "Singular covariance matrix: determinant="<<det;
64
65                 return msg.str().c_str();
66         }
67 };
68
69 class InfLikelihoodException: public std::exception
70 {
71         double likelihood;
72 public:
73         InfLikelihoodException(double likelihood):likelihood(likelihood){};
74
75         virtual const char* what() const throw()
76         {
77
78                 std::stringstream msg;
79                 msg<< "likelihood infinity: likelihood="<<likelihood;
80
```

```
81                  return msg.str().c_str();
82          }
83  };
84
85  class ZeroDataSizeException: public std::exception
86  {
87          double size;
88  public:
89          ZeroDataSizeException(double size):size(size){};
90
91          virtual const char* what() const throw()
92          {
93
94                  std::stringstream msg;
95                  msg<< "Data size to small: size="<<size;
96
97                  return msg.str().c_str();
98          }
99  };
100
101 class OutlierException: public std::exception
102 {
103         double sumEx;
104 public:
105         OutlierException(double sum):sumEx(sum){};
106
107         virtual const char* what() const throw()
108         {
109
110                 std::stringstream msg;
111                 msg << "outlier: ";
112                 msg << sumEx;
113                 msg << " ";
114
115                 return msg.str().c_str();
116         }
117 };
118
119 //---------------------Mulitvariate Gaussian Distribution---------------------
120 class MultivariateGaussianDistribution {
121 private:
122         //Attributes:
123         unsigned dim;
124         boost::shared_ptr<vector<double> > meanVector;
125         boost::shared_ptr<matrix<double> > covarianceMatrix;
126
127         //Caches:
128         boost::shared_ptr<matrix<double> > invCovarianceMatrix;
129         boost::shared_ptr<double> detCovarianceMatrix;
130
131
132 public:
133         MultivariateGaussianDistribution(unsigned dim, const vector<double> &meanVector,
                const matrix<double> &covarianceMatrix);
```

```
134         MultivariateGaussianDistribution(const MultivariateGaussianDistribution &copy);
135
136         virtual ~MultivariateGaussianDistribution();
137
138         unsigned getDim() const {return dim;}
139         boost::shared_ptr<vector<double> > getMeanVector() const {return meanVector;}
140         boost::shared_ptr<matrix<double> > getCovarianceMatrix() const {return
              covarianceMatrix;}
141         boost::shared_ptr<double> getDetCovarianceMatrix();
142
143         void setMeanVector(const vector<double> &v);
144         void setCovarianceMatrix(const matrix<double> &m);
145
146         void resetCovarianceMatrixCach();
147
148         double probabilityDensityFunction(const vector<double> &x);
149         double probabilityDensityFunction(const vector<double> &x, const double &
              exponent);
150         double calculateExponent(const vector<double> &x);
151
152
153 };
154
155 //---------------------Component--------------------
156 class Component
157 {
158 private:
159         double weight;
160         boost::shared_ptr<MultivariateGaussianDistribution> mgd;
161
162 public:
163         Component(double weight, unsigned dim,const vector<double> &meanVector, const
              matrix<double> &covarianceMatrix);
164         Component(double weight, const MultivariateGaussianDistribution &mgd);
165         Component(const Component& copy);// copy constructor
166
167         double getWeight() const {
168                 return weight;
169         }
170         boost::shared_ptr<MultivariateGaussianDistribution>
              getMultivariateGaussianDistribution() const {return mgd;}
171         unsigned getDim() const {return mgd->getDim();}
172         boost::shared_ptr<vector<double> > getMeanVector() const {return mgd->
              getMeanVector();}
173         boost::shared_ptr<matrix<double> > getCovarianceMatrix() const {return mgd->
              getCovarianceMatrix();}
174
175
176         void setWeight(double weight);
177
178         void setMultivariateGaussianDistribution(MultivariateGaussianDistribution mgd);
179         void setMeanVector(const vector<double> &v);
180         void setCovarianceMatrix(const matrix<double> &m);
181
```

```
182         std::string printComponent(bool print=true) const;
183
184         virtual ~Component();
185 };
186
187 //---------------------Gaussian Mixture Model----------------------
188 class GMM {
189 private:
190         unsigned dim;
191
192         static void setMinMaxVecFromData(vector<double> *min, vector<double> *max, const
                std::vector<vector<double> > &data);
193
194
195
196 public:
197         std::vector<Component> components;
198         GMM(unsigned dim, const std::vector<Component> &components);
199
200         GMM(const GMM& copy); //(deep) copy constructor
201         virtual ~GMM();
202
203         unsigned getDim(){return dim;};
204         unsigned getNoComponents() const {return components.size();}
205
206         static vector<double> calculateSampleMean(const std::vector<vector<double> > &
                observations);
207         static matrix<double> calculateSampleCovMatrix(const std::vector<vector<double>
                > &observations, const vector<double> &mean);
208         static GMM *generateRandom(const unsigned noComponents, const std::vector<vector
                <double> > &data, boost::random::mt19937 &rng);
209
210         double mixtureDistribution(const vector<double> &obs);
211
212         double calculateLogLikelihood(const vector<double> &obs);
213         double calculateLogLikelihood(const std::vector<vector<double> > &obsSeqs);
214
215
216         double learningLogLikelihood(const std::vector<vector<double> > &obsSeqs, double
                epsilon, const std::vector<int> &fileNumbers);
217         void learning(const std::vector<vector<double> > observations, const std::vector
                <int> &fileNumbers);
218
219         std::string printGMM (bool print=true) const;
220         void saveGMM();
221         static void saveGMMS(const std::vector<GMM> &gmms);
222
223         bool containSingularCovarianceMatrix(void);
224 };
225
226 #endif /* GAUSSIAN_H_ */
```

Listing A.1: Header file of implementation of Gaussian Mixture Model.

B

hmm.h

```
1  /*
2   * hmm.h
3   * Implementation of a Hidden Markov Model (HMM) with a Gaussian mixture model (GMM) for
        each hidden state.
4   *
5   *  Created on: Feb 11, 2013
6   *  Author: Morten Albeck Nielsen and Stine Back Larsen
7   */
8
9  #ifndef HMM_H_
10 #define HMM_H_
11
12 #include <boost/numeric/ublas/vector.hpp>
13 #include <boost/numeric/ublas/matrix.hpp>
14 #include <boost/shared_ptr.hpp>
15 #include <boost/random/mersenne_twister.hpp>
16 #include <boost/random/uniform_int_distribution.hpp>
17 #include <boost/random/discrete_distribution.hpp>
18
19 #include <boost/math/special_functions/fpclassify.hpp>
20
21 #include <assert.h>
22 #include <limits>
23
24 #include <iostream>
25 #include <fstream>
26
27 #include <cmath>
```

```
28
29 #include <exception>
30 #include <string>
31 #include <sstream>
32
33 #include "utilities.h"
34 #include "gaussian.h"
35
36 using namespace boost::numeric::ublas;
37
38 class ComponentInfLikelihoodException: public std::exception
39 {
40         unsigned state;
41         unsigned component;
42 public:
43         ComponentInfLikelihoodException(unsigned state, unsigned component):state(state)
              , component(component){};
44
45         unsigned getState(){return state;}
46         unsigned getComponent(){return component;}
47
48         virtual const char* what() const throw()
49         {
50
51                 std::stringstream msg;
52                 msg<< "For state "<<state<<" component "<<component<<" likelihood is
                      infinity";
53
54                 return msg.str().c_str();
55         }
56 };
57
58 class HMM
59 {
60 private:
61         unsigned noStates;//a state is indicated by a number
62
63         boost::shared_ptr<vector<double> > start;
64         boost::shared_ptr<matrix<double> > transition;
65         boost::shared_ptr<matrix<double> > logTransition;
66         boost::shared_ptr<std::vector<GMM> > emission; //A GMM for each state
67
68
69
70         static std::vector<GMM> generateEmission(std::vector<std::vector<vector<double>
              > > states, unsigned noComponents, boost::random::mt19937 &rng);
71
72         std::vector< std::vector< matrix<double> > > *calculateGamma(
73                         std::vector<const std::vector<vector<double> > *> &obsSeqs,
74                         const std::vector<std::vector<vector<double> > > &smoothingsSeqs
75                 ); //used for learning
76
77         void resetComponent(const std::vector<std::vector<vector<double> > > &obsSeqs,
              unsigned state, unsigned component,      boost::random::mt19937& rng);
```

```
78
79  public:
80          HMM(const unsigned noStates,const vector<double> &start,const matrix<double> &
               transition, const std::vector<GMM> &emission);
81          ~HMM(void);
82
83
84          unsigned getNoStates() const {return noStates;}
85          boost::shared_ptr<vector<double> > getStart() const {return start;}
86          boost::shared_ptr<matrix<double> > getTransition() const {return transition;}
87          boost::shared_ptr<std::vector<GMM> > getEmission() const {return emission;}
88
89          void setMembers(const unsigned noStates,const vector<double> &start, const
               matrix<double> &transition, const std::vector<GMM> &emission);
90          void setStart(unsigned state, double probability);
91          void setStart(const vector<double> &v);
92          void setTransition(unsigned startState, unsigned endState, double probability);
93          void setTransition(const matrix<double> &m);
94          void setEmission(const std::vector<GMM> &e);
95
96          static HMM *generateRandom(const unsigned noStates, const unsigned noComponents,
                const std::vector<std::vector<vector<double> > > &data, boost::random::
               mt19937 &rng, bool leftRight=false);
97
98          vector<double> forward(const vector<double> &previousForward, const vector<
               double> &sensor) const;
99          vector<double> backward(const vector<double> &previousBackward, const vector<
               double> &sensor);
100         vector<double> maxProp(const vector<double> &previousMaxProp, vector<double>&
               psi);
101
102         std::vector<vector<double> > filtering(const std::vector<vector<double> > &
               obsSeq) const;
103         std::vector<vector<double> > filtering(const std::vector<vector<double> > &
               obsSeq, std::vector<double> &scalingFactors) const;
104
105         std::vector<vector<double> > backwardInduction(const std::vector<vector<double>
               > &obsSeq);
106         std::vector<vector<double> > backwardInduction(const std::vector<vector<double>
               > &obsSeq, const std::vector<double> &scalingFactors);
107
108         vector<double> prediction(const std::vector<vector<double> > &obsSeq, unsigned
               timestep) const;
109
110
111         std::vector<vector<double> > smoothing(const std::vector<vector<double> > &
               obsSeq);
112         std::vector<vector<double> > smoothing(const std::vector<vector<double> > &
               obsSeq, std::vector<vector<double> > &forwards, std::vector<vector<double> >
               &backwards, std::vector<double> &scalingFactors);
113
114
115         std::vector<unsigned > mle(const std::vector<vector<double> > &obsSeq);
```

```
116         std::vector<unsigned > mle(const std::vector<vector<double> > &obsSeq, double &
                logP);
117
118         static double calculateLogLikelihood(const std::vector<double> &scalingFactors);
119         static double calculateLogLikelihood(const std::vector<std::vector<double> > &
                scalingFactorsSeqs);
120
121         double learningKMeans(const std::vector<std::vector<vector<double> > > &obsSeqs,
                 const double epsilon, boost::random::mt19937 &rng, const std::vector<int> &
                fileNumbers);
122         double learningLogLikelihood(const std::vector<std::vector<vector<double> > > &
                obsSeqs, const double epsilon, const std::vector<int> &fileNumbers,
                boost::random::mt19937& rng);
123         bool learning(const std::vector<std::vector<vector<double> > > &obsSeqs, std::
                vector<std::vector<double> > &scalingFactorsSeqs, const std::vector<int> &
                fileNumbers);
124
125         vector<double> calculateEvidence(const vector<double> &obs) const;
126
127         std::vector<vector<double> > generateSequence(const unsigned &lenght, boost::
                random::mt19937 &rng);
128         double distance(const HMM &other, const unsigned &lenght, boost::random::mt19937
                 &rng); //not symmetric
129
130         std::string printHMM(bool print=true);
131         void saveHMM(const std::string prefix="");
132
133         bool containSingularCovarianceMatrix();
134 };
135
136
137
138
139 #endif /* HMM_H_ */
```

Listing B.1: Header file of implementation of Hidden Markov Model.

C

Viterbi algorithm

The description given on the Viterbi algorithm in this appendix is based on the description given by Rabiner [26]. Let $s_{1:T}$ be the sequence of state for an HMM with with highest probability when observation $\mathbf{x}_{1:T}$. We would then like to determine $s_{1:T}$. Let λ be the parameters of the HMM, and let $\delta_t = \max_{S_{1:t-1}} P(S_{1:t}, \mathbf{x}_{1:t}|\lambda)$, i.e. δ_t is the highest probability for a path through $S_{1:t-1}$, observing $\mathbf{x}_{1:t}$ and ending in state S_t. $s_{1:T}$ can then be found using Algorithm 2. Both the path $s_{1:T}$ and the probability $\Delta = \max_{n=1,\dots,N}(\delta_T(n))$ can be of interest.

Algorithm 2: Viterbi algorithm [26]

Input: Observation sequence $\mathbf{x}_{1:T}$

// Initialization

$\delta_1 = P(S_1) \cdot P(\mathbf{x}_1|S_t)$

$\psi_1 = \mathbf{0}$

// Recursion

for $t = 2, \ldots, T$ **do**

  // $P(\mathbf{x}_t|S_t = n)$ is given by Equation 3.5

  $\delta_t(n) = \max_{n'=1,\ldots,N}(\delta_{t-1}(n')P(S_t = n|S_{t-1} = n')) \cdot P(\mathbf{x}_t|S_t = n)$ for $n = 1, \ldots N$

  $\psi_t(n) = \arg\max_{n'=1,\ldots,N}(\delta_{t-1}(n')P(S_t = n|S_{t-1} = n')) \cdot P(\mathbf{x}_t|S_t = n)$ for $n = 1, \ldots N$

end

// Termination

$\Delta = \max_{n=1,\ldots,N}(\delta_T(n))$

$s_T = \arg\max_{n=1,\ldots,N}(\delta_T(n))$

// Path backtracking

for $t = T - 1, \ldots, 1$ **do**

  $s_t = \psi_{t+1}(s_{t-1})$

end

Bibliography

[1] Cornell University's Bioacoustic Research Program. URL http://www.birds.cornell.edu/brp/. Retrieved April 29, 2013.

[2] Kaggle, . URL https://www.kaggle.com/. Retrieved April 22, 2013.

[3] The Marinexplore and Cornell University Whale Detection Challenge, . URL https://www.kaggle.com/c/whale-detection-challenge/. Retrieved April 22, 2013.

[4] The Marinexplore and Cornell University Whale Detection Challenge Forum, . URL http://www.kaggle.com/c/whale-detection-challenge/forums/. Retrieved April 22, 2013.

[5] Right Whale Listening Network. URL http://www.listenforwhales.org/. Retrieved April 29, 2013.

[6] Marinexplore. URL http://marinexplore.org/. Retrieved April 29, 2013.

[7] Voicebox: Speech Processing Toolbox for MatLab. URL http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html. Retrieved April 22, 2013.

[8] Compute Receiver Operating Characteristic (ROC) curve or other performance curve for classifier output. URL http://www.mathworks.se/help/stats/perfcurve.html. Retrieved May 23, 2013.

[9] Is a sample covariance matrix always symmetric and positive definite? URL http://stats.stackexchange.com/questions/52976/is-a-sample-covariance-matrix-always-symmetric-and-positive-definite. Retrieved April 30, 2013.

[10] Jeff A Bilmes et al. A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian Mixture and Hidden Markov Models. *International Computer Science Institute*, 4(510):126, 1998.

[11] Judith C. Brown and Paris Smaragdis. Hidden markov and gaussian mixture models for automatic call classification. *Journal of the Acoustical Society of America*, 125(6):EL221–EL224, 2009.

[12] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, MA, USA, 2nd edition, 2001.

[13] S Datta and C Sturtivant. Dolphin whistle classification for determining group identities. *Signal processing*, 82(2):251–258, 2002.

[14] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38, 1977.

[15] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification*. Wiley-interscience, 2nd edition, 2012.

[16] Xinjian Guo, Yilong Yin, Cailing Dong, Gongping Yang, and Guangtong Zhou. On the class imbalance problem. In *Natural Computation, 2008. ICNC'08. Fourth International Conference on*, volume 4, pages 192–201. IEEE, 2008.

[17] Anders Hesslager-Olesen, Stine Back Larsen, and Morten Albeck Nielsen. Predicting Unit Production in StarCraft using Hidden Markov Models. Department of Computer Science, Aalborg University, Autumn 2012.

[18] Finn V. Jensen and Thomas D. Nielsen. *Bayesian Networks and Decision Graphs*. Springer Publishing Company, Incorporated, 2nd edition, 2007. ISBN 9780387682815.

[19] A. Klapuri and M. Davy. *Signal Processing Methods for Music Transcription*. Springer Science+Business Media LLC, 2006. ISBN 9780387328454. URL http://www.google.dk/books?id=AF30yR41GIAC.

[20] S. J. Leon. *Linear Algebra With Applications*. Maxwell Macmillan international editions. Pearson, 2006. ISBN 0-13-200306-6.

[21] David McAllester. The Covariance Matrix. Course TTIC 103 (CMSC 35420): Statistical Methods for Artificial Intelligence, at Toyota Technological Institute at Chicago, Autumn 2007. URL http://ttic.uchicago.edu/~dmcallester/ttic101-07/lectures/Gaussians/Gaussians.pdf. Retrieved April 14, 2013.

[22] David K Mellinger and Christopher W Clark. Recognizing transient low-frequency whale sounds by spectrogram correlation. *The Journal of the Acoustical Society of America*, 107:3518, 2000.

[23] A.V. Oppenheim and R.W. Schafer. *Discrete-Time Signal Processing*. Prentice Hall signal processing series. Pearson Education, Limited, 2009. ISBN 9780132067096. URL http://books.google.dk/books?id=5vajQAAACAAJ.

[24] F.J. Owens. *Signal processing of speech*. Macmillan new electronics series. McGraw-Hill

Ryerson, Limited, 1993. ISBN 9780070479555. URL http://books.google.dk/books?id=qpAeAQAAIAAJ.

[25] Lawrence Rabiner and Biing-Hwang Juang. Fundamentals of speech recognition. 1993.

[26] Lawrence R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. In *Proceedings of the IEEE*, pages 257–286, 1989.

[27] D. Reby, R. Andre-Obrecht, A. Galinier, J. Farinas, and B. Cargnelutti. Cepstral coefficients and hidden markov models reveal idiosyncratic voice characteristics in red deer (cervus elaphus) stags. *Journal of the Acoustical Society of America*, 120(6):4080–4089, 2006. URL http://sro.sussex.ac.uk/756/.

[28] Douglas Reynolds. Gaussian mixture models. *Encyclopedia of Biometric Recognition*, pages 14–68, 2008.

[29] Marie A Roch, Melissa S Soldevilla, Jessica C Burtenshaw, E Elizabeth Henderson, and John A Hildebrand. Gaussian mixture model classification of odontocetes in the Southern california bight and the gulf of california. *J Acoust Soc Am*, 121(3):1737–48, 2007. URL http://www.biomedsearch.com/nih/Gaussian-mixture-model-classification-odontocetes/17407910.html.

[30] Stuart Jonathan Russell, Peter Norvig, John F Canny, Jitendra M Malik, and Douglas D Edwards. *Artificial Intelligence: A Modern Approach*. Prentice hall Englewood Cliffs, 2nd edition, 1995.

[31] Eric Spaulding, Matt Robbins, Thomas Calupca, Christopher W. Clark, Christopher Tremblay, Amanda Waack, Ann Warde, John Kemp, and Kris Newhall. An autonomous, near-real-time buoy system for automatic detection of north atlantic right whale calls. *Proceedings of Meetings on Acoustics*, 6(1), 2009. doi: 10.1121/1.3340128. URL http://link.aip.org/link/?PMA/6/010001/1.

[32] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005. ISBN 0321321367.

[33] S.V. Vaseghi. *Advanced Digital Signal Processing and Noise Reduction*. Wiley, 2008. ISBN 9780470740163. URL http://books.google.ca/books?id=vVgLv0ed3cgC.

[34] BA Weisburn, SG Mitchell, CW Clark, and TW Parks. Isolating biological acoustic transient signals. In *Acoustics, Speech, and Signal Processing, 1993. ICASSP-93., 1993 IEEE International Conference on*, volume 1, pages 269–272. IEEE, 1993.

[35] CF Wu. On the convergence properties of the em algorithm. *The Annals of Statistics*, 11(1): 95–103, 1983.