NTNU – Trondheim
Norwegian University of
Science and Technology

Implementation and Adaptation of a System for Automatic Classification of Birdsong.

Kristian Selboe

Master of Science in Electronics
Submission date: June 2015
Supervisor: Magne Hallstein Johnsen, IET

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Problem Description

NTNU is in the process of developing a system for bird classification. The system involves several steps. In the first step the incoming acoustic signal is decoded into a sequence of segments. The segments may belong to all legal classes represented by the system. The system finds/decodes the sequence of segments that provides the greatest score (likelihood). In the next step the sequence is analysed with respect to which classes occurs most frequently. This information is stored in a histogram representing the different classes. Based on the histogram one can find the classes which occurs most frequently, i.e. n-best classification. In a final step the histogram is normalized to eliminate the influence of the duration of the recording. The normalized histogram is applied as input to a static post-classifier that provides a probability for each class. A similar n-best ranking of the probabilities has proven to give a better performance than using the histogram directly.

The sequence of segments is generated by a program called Hidden Markov Model Toolkit (HTK) which is free to download. But the remaining steps are implemented in Matlab code. In this thesis, these steps are going to be implemented using free software.

The parameters in models and post-classifier has to be trained by a manually labeled training database. The quality of the identification, and hence the error rate is dependent on the database being "large enough". This is a relative term, but the current amount of available data is clearly too small. One possible strategy is therefore to disclose the current system, i.e. getting users to submit recordings. In addition to providing the user with the name of the class, it is possible to use the recordings with labeling to improve the system. It is then two possible approaches to this problem:

1. Perform the labeling manually.

2. Perform the labeling automatically by using the results from the classifier.

The first method can then be regarded as an upper limit for the second method, especially if it does not occur misclassifications in the manual labeling. However, the second method is fully automated, and is therefore preferable if a reasonable improvement is achieved. In this thesis it will be performed a comparison between these two methods with respect to error rate and the degree of dependency on the

size of the database and user submitted recordings. This comparison will tell if it is possible to implement this adaptation of data automatically in a sufficient way.

Preface

This report treats the work related to my master thesis carried out in my last semester at the Master of Science program in Electronics, at the Department of Electronics and Telecommunications, NTNU. I would like to thank my supervisor Magne H. Johnsen for helpful support during the project.

Trondheim, June 19, 2015
Kristian Selboe

Summary

The background for this master thesis is a collaboration between Able Magic and NTNU. At the request of Able Magic there has been conducted a feasibility study by NTNU, which has led to the development of a bird classification system. This system is based on identification of bird song and are aimed at 21 different Norwegian bird species. Able Magic wants to create a commercial version of this system implemented as an application for the smart phone market.

The system is based on model based segmentation, modeling each bird as a Gaussian Mixture Model (GMM). These models are trained with the Expectation Maximization algorithm on labeled training data. Furthermore, these models are used in the identification of the bird song. Hence, one can say that the system is divided into two parts, namely training and identification.

This thesis is divided into two main tasks. The first task is to convert the basis system created by NTNU into a production model ready for commercial use. The initial system is designed using three different development tools, respectively, Hidden Markov Model Toolkit, Praat and Matlab. Matlab is not free and the scripts developed in Matlab therefore needed to be converted to another language which is free. After converting the different scripts and restructuring them, the production model was created by combining the scripts to a complete system. Now the system consists of three scripts, where one is handling the training process and another one is handling the identification process. In addition there is one script training a linear classifier used in the identification. All three scripts are implemented in Perl.

The second task of the thesis is aimed at improving the system performance by looking at the possibility of doing adaptation of training data. This is an effort of trying to improve the existing models. With this adaptation experiment the possibility of adapting data automatically without any manual labeling of the data is going to be investigated. The adaptation experiment has been performed by training GMMs with a given amount of training data, and then introducing more data used to train new GMMs. The new training data can either be labeled by the system output when doing identification with the initial models (automatically), or using the manual labeling which is known to be correct. By comparing the results obtained by these two methods it is possible to tell if the automatic adaptation provides sufficient results.

From the adaptation experiments it is seen that the number of misclassified birds is increasing when using the new adaptation models generated by automatic labeling of the adaptation data. There is an increase of about 2.5% in the error rate when using these models compared to when using the initial models trained with less data. This shows that there is difficult to perform the adaptation of data automatically with the current system performance and amount of available data.

Sammendrag

Bakgrunnen for dette prosjektet er et sammarbeid mellom Able Magic og NTNU. Etter ønske fra Able Magic har det blitt foretatt en mulighetstudie av NTNU, som har ledet til utviklingen av et system for klassifisering av fuglearter basert på identifikasjon av fuglesang. Systemet er rettet mot 21 forskjellige norske fuglearter.

Systemet modellerer hver fugleart i tillegg til "pause" som en Gaussian Mixture Model (GMM). Disse modellene er trent med Expectation Maximization (EM) algoritmen. I denne treningen brukes merket treningsdata. Videre blir disse modellene brukt til identifikasjon av fuglesang. Man kan derfor si at systemet er delt opp i to hoveddeler, henholdsvis trening og identifikasjon.

Dette prosjektet er delt inn i to hovedoppgaver. Den første oppgaven går ut på å lage en produksjonsmodell av systemet utviklet av NTNU, som er klar for kommersiell bruk. Det opprinnelige systemet er utviklet ved hjelp av tre ulike utviklingsverktøy, henholdsvis Hidden Markov Model Toolkit (HTK), Praat og Matlab. Matlab er ikke gratis og scriptene skrevet i Matlab er derfor konvertert til Perl som er gratis. Etter at dette er gjort settes scriptene sammen til et komplett system. Systemet består nå av tre script, der et tar seg av treningen av modellene og et annet tar seg av identifikasjonen. I tilegg er det et script som tar seg av treningen av en lineær klassifiserer som brukes under identifikasjonen.

Den andre oppgaven går ut på å forbedre ytelsen til systemet ved å se på muligheten for å adaptere ytterligere treningsdata. Dette er et forsøk på å forbedre eksisterende modeller ved å trene nye adapsjonsmodeller med et økt treningsgrunnlag. Det skal undersøkes om denne adapsjonen kan gjøres automatisk. Denne adapsjonen kan enten gjøres ved å merke filene som skal adapteres ved hjelp av resultatene fra klassifiseringen (automatisk), eller man kan bruke merking som er foretatt manuelt og som man vet er korrekt. Ved å foreta en sammenligning av disse to metodene kan man si om det er mulig å foreta denne adapsjonen automatisk med tilfredsstillende resultater.

Det viser seg at ytelsen til systemet er dårligere med de nye adapsjonsmodellene generert ved automatisk adapsjon, sammenlignet med de initielle modelle. Feilraten har økt med 2.5%. Dette viser at systemets ytelse i kombinasjon med begrenset tilgang på data er for dårlig til å gjennomføre adapsjonen automatisk.

Table of Contents

List of Tables

List of Figures

Abbreviations

HTK	=	Hidden Markov Model Toolkit
GMM	=	Gaussian Mixture Model
HMM	=	Hidden Markov Model
MFC	=	Mel Frequency Cepstrum
MFCC	=	Mel Frequency Cepstrum Coefficient
DCT	=	Discrete Cosine Transform
MAP	=	Maximum a Posteriori
ML	=	Maximum Likelihood
EM	=	Expectation Maximization
MLF	=	Master Label File

Chapter 1

Introduction

This chapter is an introduction to this report. It will be given a brief overview of the motivation and background for the thesis in section 1.1. The work and the chosen approach will be presented in section 1.2. Finally there is given an overview of the structure in this report in section 1.3.

1.1 Motivation and Background

The background for this master thesis is a collaboration between Able Magic and NTNU. At the request of Able Magic there has been conducted a feasibility analysis by NTNU, which has led to the development of a system for bird classification based on identification of bird song. Able Magic wants to create a commercial system in the form of an application for the smart phone market. Therefore, it is desired that the system is developed using free software. The system developed by NTNU is designed using three different development tools, respectively Praat, Hidden Markov Model Toolkit (HTK) and Matlab. Matlab is not free, and the programs developed in Matlab has therefore been converted to Perl code. The system is initially aimed at 21 different birds.

The desired behaviour of the system is to get an audio recording as input and then be able to identify which bird the recording contains and return a label representing this bird as output. If one should recognize speech, distinction is often made between classification and recognition. Classification is about recognizing individual words, while recognition is about determining the number of words, the word sequence and sometimes the time boundaries between words. When it comes to bird song this can generally be placed between classification and recognition.

Since the goal of this particular system is to recognize a file as one bird, and not look at different stanzas this can be looked at as a classification problem. In order to implement this, one is dependent of representing the bird song mathematically.

There is no mathematical formula for bird song, therefore it is necessary to find a mathematical approximation often called models, good enough to represent the signal in a sufficient way. Speech and most time signals have information in temporary form. This means that the order of the signals content is crucial to understand the meaning. Turning to bird song it is fairly certain that each stanza has temporary information, while repeating stanzas probably do not provide additional information.

This thesis is based on the assumption that the time signal is stochastic. With regards to stochastic signals a distinction is made between stationary and non-stationary signals. Bird song is non-stationary because the frequency content varies with time. Because the "production" of any physical signal has certain inertia because of mass, it is possible to assume that all physical signals are short-time stationary, i.e. stationary over a "short" time frame. Consequently there is possible to analyse the input recording segmentally. This assumption is of great use in signal processing and is also used in this thesis.

Because there is a closed set of birds represented by the system it is convenient to use model based segmentation. In this thesis there is created one Gaussian Mixture Model (GMM) for each of the bird species in addition to one model for "pause". Each of the classes are created with 32/16/8/4/2/1 mixtures. The objective of the models is to approximate the statistical properties of the birds, such that the system can distinguish between the different birds. First, the models have to be trained on labeled training data. Then these models are used in the identification on test data.

1.2 Work

The work performed in this thesis can be divided into two main tasks. As mentioned above the system created by NTNU is going to be implemented using free software. All scripts will be implemented in Perl. Next, the different parts of the system are going to be implemented in a straightforward and intuitive way. The goal is to integrate all the different scripts into a complete system leading up to a system which is ready for commercial use. The work related to this part of the thesis is presented as the basis system.

In addition to this, the possibility of adapting new training data is going to be

explored. This is of great interest since it could potentially help to improve the system performance because of the ability to create better models. If there is found a good method to do this automatically, user data can be used directly as new training data, thus it is possible to improve the models continuously. This part of the thesis is presented as the adaptation system.

One can say that the basis system is divided into two main parts, namely training and identification. Common for both parts is a feature extraction where the acoustic input signal is being parametrized to a feature vector, in this case containing Mel Frequency Cepstral Coefficients (MFCC). This feature vector is used in the training of the different GMMs, as well as in the identification. In the identification these features are used to find the model which is the best fit to the incoming acoustic signal. After this process the input file is often identified as more than one bird because of the segmentally processing of the file. It is therefore necessary with some post-processing and classification to determine which bird one should choose. It is the scripts doing the post-processing and classification that needs to be converted to Perl code.

HTK is used to perform all training and testing in this thesis, and is a developing toolkit for building and manipulating Hidden Markov Models (HMMs). The most common application area is speech recognition. It was originally developed at the Machine Learning Laboratory of the Cambridge University.

1.3 Outline of the Report

Chapter 2 presents the theory that this thesis is based on.

Chapter 3 consists of an explanation of the database used during the thesis.

Chapter 4 consists of a presentation of the basis system. There will be given an explanation of how the system works along with the system performance. In addition the initial system structure and the structure of the commercialized production model will be presented.

Chapter 5 consists of a presentation of the adaptation system. There will be given an overview of a typical adaptation experiment as well as the approach chosen in this thesis. The results and the following discussion from the adaptation experiments will also be presented in this chapter.

Chapter 6 summarizes the most important results and observations from the work performed in this thesis.

Chapter 2

Theory

This chapter provides an explanation of the theory that this thesis is based on. Frequency analysis of short-time stationary signals and the features used, mel-frequency cepstral coefficients, are covered in section 2.1. Section 2.2 covers Gaussian Mixture Models. Next the Maximum Likelihood function used as training criteria is presented in section 2.3. Section 2.4 presents the EM-algorithm used for maximum likelihood parameter estimation in the training process. Finally the Viterbi algorithm used in the identification process is presented in section 2.5.

2.1 Frequency Analysis of Short-Time Stationary Signals

With the assumption of short-time stationary signal over K msec it is common to use a window of same length to extract a segment of the signal with equal length. A common choice of such a window is the Hamming Window, also used in this thesis. The frequency analysis usually ends up in a vector of features, $\underline{x}$, representing the segment. The same analysis is performed again by moving the window L msec. Each time the analysis is performed, a new vector $\underline{x}$ is generated forming a sequence of vectors $\underline{X}$ representing the whole signal. It is common to choose $L < K$, resulting in a overlap between segments [1].

Desirable characteristics for features used in speech processing does occur frequently, are simple to measure, does not change over time and is not dependent on the speakers health condition. They should also be robust against noise and be of high order. Mel-frequency cepstral coefficients (MFCCs) are very often used as

features in speech and speaker recognition systems and are also recommended for bird song [2].

2.1.1 Mel- Frequency Cepstral Coefficients

In speech processing, or more generally, sound processing, it is common to represent the short-time power spectrum with the Mel-frequency cepstrum (MFC), and the MFCCs are coefficients that make up an MFC. In order to approximate the human auditory system's response, the MFC is used instead of the normal cepstrum. This is because the MFC have frequency bands equally spaced on the mel scale. This gives a better approximation compared to the normal cepstrum where the frequency bands are linearly spaced.

Conversion from Hertz (f) to Mel (m) is done by the following formula:

$$m(f) = 2595 \log_{10}(1 + \frac{f}{700}) \tag{2.1}$$

A plot of this relationship is shown in the figure 2.1 below.

The MFCCs are commonly derived as follows:

1. Use a window to extract a segment of the signal.
2. Take the Fourier transform the segment.
3. Take the powers of the spectrum and map them onto the mel scale. This is done using triangular overlapping windows also known as triangular filter bank.
4. Take the logs of the powers at each of the mel frequencies.
5. Take the discrete cosine transform (DCT) of the list of mel log powers, as if it were a signal.
6. The output after applying DCT is MFCC.

This procedure is illustrated in figure 2.2.

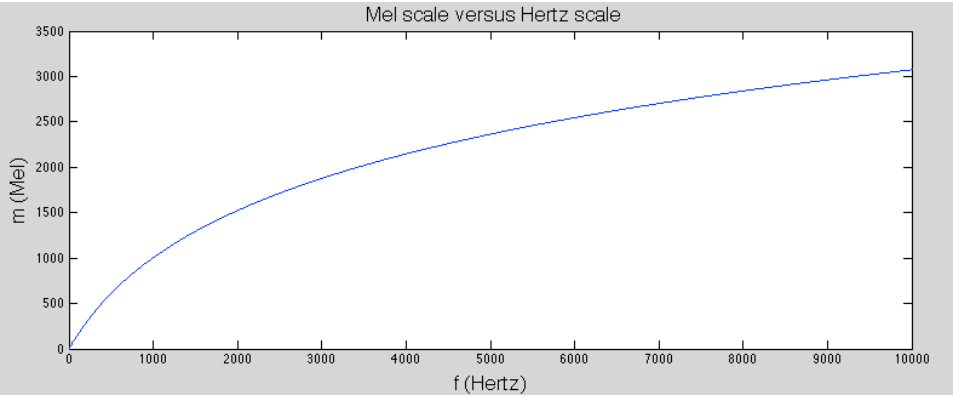


Figure 2.1: Mel vs. Hertz

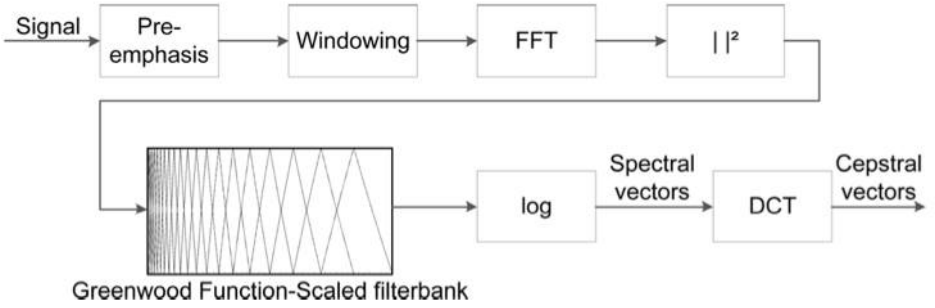


Figure 2.2: Calculation of MFCC

This section is motivated by [3].

2.2 Gaussian Mixture Models

"A Gaussian Mixture Model (GMM) is a parametric probability density function represented as a weighted sum of Gaussian component densities. GMMs are commonly used as a parametric model of the probability distribution of continuous measurements or features in a biometric system, such as vocal-tract related spectral features in a speaker recognition system. GMM parameters are estimated from training data using the iterative Expectation-Maximization (EM) algorithm or Maximum a Posteriori (MAP) estimation from a well-trained prior model.

A Gaussian mixture model is a weighted sum of M component Gaussian densities as given by the equation,

$$p(\underline{x}|\lambda) = \sum_{i=1}^{M} w_i g(\underline{x}|\underline{\mu}_i, \Sigma_i) \tag{2.2}$$

where $\underline{x}$ is a D-dimensional continous-valued data vector (MFCCs from the feature extraction in this case), w_i, $i = 1,...,M$ are the mixture weights, $g(\underline{x}|\underline{\mu}_i, \Sigma_i)$, $i = 1,...,M$ are the component Gaussian densities. Each component density is a D-variate Gaussian function of the form,

$$g(\underline{x}|\underline{\mu}_i, \Sigma_i) = \frac{1}{(2\pi)^{D/2}|\Sigma_i|^{1/2}} exp\{-\frac{1}{2}(\underline{x} - \underline{\mu}_i)'\Sigma_i^{-1}(\underline{x} - \underline{\mu}_i)\} \tag{2.3}$$

with mean vector $\underline{\mu}_i$ and covariance matrix Σ_i. The mixture weights satisfies the constraint $\sum_i^M w_i = 1$. The complete Gaussian mixture model is parametrized by the mean vectors, covariance matrices and mixture weights from all component densities. These parameters are collectively represented by the notation,

$$\lambda = \{w_i, \underline{\mu}_i, \Sigma_i\}, i = 1,...,M \tag{2.4}$$

GMMs are often used in biometric systems, most notably in speaker recognition systems, due to their capability of representing a large class of sample distributions. One of the powerful attributes of the GMM is its ability to form smooth approximations to arbitrarily shaped densities"[4].

2.3 Maximum Likelihood

Given training vectors and a GMM configuration, it is desired to estimate the parameters of the GMM, λ, which in some sense best matches the distribution of the training feature vectors. There are several techniques available for estimating the parameters of a GMM. By far the most popular and well-established method is maximum likelihood (ML) estimation. The goal of ML estimation is to find the model parameters which maximize the likelihood of the GMM given the training data. For a sequence of T training vectors $X = \{\underline{x}_1,...,\underline{x}_N\}$, the GMM likelihood, assuming independence between the vectors, can be written as,

$$p(X|\lambda) = \prod_{i=1}^{N} p(\underline{x}_i|\lambda) = L(\lambda|X) \tag{2.5}$$

This function is called the likelihood function. The likelihood is thought of as a function of the parameters λ where the data X is fixed. The maximum likelihood

problem is about finding the parameters λ that maximizes the likelihood function L described in equation 2.5. The goal is to find λ^* where

$$\lambda^* = \underset{\lambda}{\operatorname{argmax}} L(\lambda|X) \tag{2.6}$$

Often the log likelihood function is chosen instead because it is analytically easier. However, it is difficult to solve this maximization problem directly. Hence it is often necessary to use an iteratively algorithm to solve this problem, namely the Expectation-Maximization algorithm. This section is motivated by [4] and [5].

2.4 Expectation-Maximization Algorithm

"The Expectation-Maximization algorithm is is a general method of finding the maximum-likelihood estimate of the parameters of an underlying distribution from a given data set when the data is incomplete or has missing values" [5].

The basic idea of the EM algorithm is, beginning with an initial model λ, to estimate a new model $\lambda^{'}$, such that $p(X|\lambda^{'}) \geq p(X|\lambda)$. This new model $\lambda^{'}$ then becomes the initial model for the next iteration and this process is repeated until some convergence threshold is reached.

"We assume that data X is observed and is generated by some distribution. We call X the incomplete data. We assume that a complete data set exists $Z = (X; Y)$ and also assume (or specify) a joint density function:

$$p(\underline{z}|\lambda) = p(\underline{x}, \underline{y}|\lambda) = p(\underline{y}|\underline{x}, \lambda)p(\underline{x}|\lambda) \tag{2.7}$$

Where does this joint density come from? Often it "arises" from the marginal density function $p(x|\lambda)$ and the assumption of hidden variables and parameter value guesses (e.g., our two examples, Mixture-densities and Baum-Welch). In other cases (e.g., missing data values in samples of a distribution), we must assume a joint relationship between the missing and observed values. With this new density function, we can define a new likelihood function, $L(\lambda|Z) = L(\lambda|X, Y) = p(X, Y|\lambda)$, called the complete-data likelihood. Note that this function is in fact a random variable since the missing information Y is unknown, random, and presumably governed by an underlying distribution. That is, we can think of $L(\lambda) = h_{X,\lambda}(Y)$ for some function $h_{X,\lambda}(\cdot)$ where X and λ are constant and Y is a random variable. The original likelihood $L(\lambda 1X)$ is referred to as the incomplete-data likelihood function.

The EM algorithm first finds the expected value of the complete-data log-likelihood log $p(X, Y|\lambda)$ with respect to the unknown data Y given the observed data X and the current parameter estimates. That is, we define:

$$Q(\lambda, \lambda^{(i-1)}) = E[log\, p(X, Y|\lambda)|X, \lambda^{(i-1)}] \tag{2.8}$$

Where $\lambda^{(i-1)}$ are the current parameters estimates that we used to evaluate the expectation and λ are the new parameters that we optimize to increase Q. The evaluation of this expectation is called the E-step of the algorithm. Notice the meaning of the two arguments in the function $Q(\lambda, \lambda^{'})$ The first argument λ corresponds to the parameters that ultimately will be optimized in an attempt to maximize the likelihood. The second argument $\lambda^{'}$ corresponds to the parameters that we use to evaluate the expectation.

The second step (the M-step) of the EM algorithm is to maximize the expectation we computed in the first step. That is, we find:

$$\lambda^{(i)} = \underset{\lambda}{\text{argmax}} Q(\lambda, \lambda^{(i-1)}) \tag{2.9}$$

These two steps are repeated as necessary. Each iteration is guaranteed to increase the loglikelihood and the algorithm is guaranteed to converge to a local maximum of the likelihood function"[5].

2.5 Viterbi Algorithm

Given a sequence of observations, the goal of the Viterbi Algorithm is to find the state-sequence that generated it. There are many possible combinations of state-sequences, that leads to many possible paths. Figure 2.1 shows an illustration of such a state-sequence network, where the green dashed lines shows possible paths between states. The Viterbi algorithm seeks fo find the path that maximizes the probability, i.e. the connection between states that is most probable that generated the observation sequence. This most probable sequence is illustrated by the blue path in figure 2.1. For a more detailed review of the theory behind the Viterbi-algorithm, see [6].

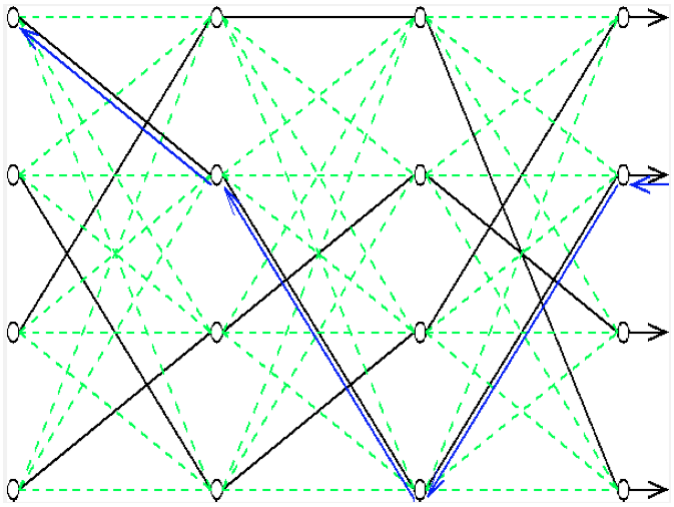


Figure 2.3: Illustration of possible paths

Chapter 3

Database

This chapter contains an explanation of the database used in this thesis. How the database is structured with respect to the number of files, length of files and the difference between training and testing data will be presented here.

The database is divided into five different data sets, which is put together to form both training and test sets. In total, the database consists of recordings from 21 different bird species. Not all birds are represented equally good, actually some of the birds have a very poor foundation of data. Able Magic who holds the rights to these recordings. All of the recordings are gathered by Able Magic from different sources. All recordings were converted to mono, decimated to a sampling frequency of 16 kHz and stored in wav-format. The length of each recording varies, and in table 3.3, it is information about the duration of the files corresponding to each of the bird species.

The training data is labeled by a program called Praat. This was done by NTNU and Able Magic. Praat is a program developed for analysis and manipulation of speech, but is also suitable for other audio recordings, in this case bird song. The labeling was done such that segments including song from the given specie were labeled as "song" and the remaining segments were labeled as "pause". Consequently, the pause segments are a collection of actual pauses, background noise and song from other birds among other things.

The labeling for a file containing Bjørkefink song is illustrated in figure 3.1. From this figure, it is seen that the pause segments do include some acoustic activity, and from listening to the file it becomes clear that the pause segment also includes bird song, but is of much lower energy. This proves that the pause class does include

other content than just pauses. This is a limitation of the database and it would have been desirable with a more detailed labeling.

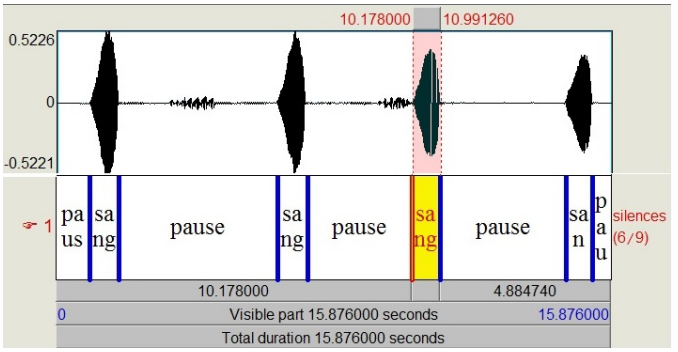


Figure 3.1: Waveform and labeling of Bjørkefink file

The label files were stored in a so called Master Label File (MLF), which can be used directly by HTK in the training of the recognition models. This file includes start and stop time and the corresponding label for each segment.

As mentioned before the amount of available recordings associated with the different species are very different. To carry out reasonable experiments it is important to have no overlap between the training and test data. Besides, it is of great importance to have enough data to train good statistical models, in addition to have a sufficient amount of test recordings to achieve reliable results. With limited access of data, this becomes a problem. As an attempt to deal with this situation and increase the confidence of the results, the database is divided into five different sets. Each of the five sets represents about 20% of the total amount of recordings, and the sum of the five sets accounts for all the recordings. This way it is possible to use four of the sets as training data and one as test data. Consequently, it is made five different combinations of data with a 4:1 ratio between training and testing, and no overlap between training and testing. Because of this, one can take the average of the results from all sets. This partitioning of the data is illustrated in figure 3.2.

Figure 3.2: Example of partitioning of the database

Table 3.1 illustrates the distribution of files between training and test set 1. From this table it is possible to see how many files there are available for both training and testing, and the sum is the total amount of available files. Svarthvit fluesnapper is represented with 76 files while Dompap has only 14 files. This illustrates how large deviation it is between the different birds with respect to the number of recordings. One reason that Svarthvit fluesnapper has so many files is the fact that some recordings have been cut into shorter recordings. The average duration, standard deviation, minimum and maximum duration related to each of the bird species can be found in table 3.3, in addition to the total duration of all recordings corresponding to each bird.

	Number of files		
Bird	Train	Test	Total
Bjørkefink	15	4	19
Blåmeis	24	6	30
Bokfink	14	4	18
Dompap	11	3	14
Granmeis	17	4	21
Gransanger	24	6	30
Grønnfink	22	5	27
Gulspurv	18	5	23
Hagesanger	19	5	24
Jernspurv	22	6	28
Lovsanger	26	7	33
Måltrost	27	7	34
Munk	33	8	41
Rødstjert	17	4	21
Rodstrupe	26	6	32
Sivspurv	20	5	25
Svarthvit fluesnapper	61	15	76
Svartmeis	25	6	31
Svattrost	40	10	50
Toppmeis	17	4	21
Trekryper	25	6	31
Total	503	126	629

Table 3.1: Information about the content of a training and test set

Bird	Mean	Stddev	Min	Max	Total	#files
Bjørkefink	33.4	14.0	2.9	61.7	633.8	19
Blåmeis	28.8	13.7	2.9	55.8	864.1	30
Bokfink	38.2	16.4	14.6	75.9	686.9	18
Dompap	41.6	14.2	19.2	70.3	582.7	14
Granmeis	44.1	15.2	28.7	75.4	925.5	21
Gransanger	33.9	12.6	5.9	60.1	1015.8	30
Grønnfink	39.9	9.6	27.4	63.8	1077.7	27
Gulspurv	33.2	13.8	2.9	61.4	762.8	23
Hagesanger	38.2	12.5	5.8	68.9	916.4	24
Jernspurv	30.8	12.6	5.9	49.4	863.1	28
Løvsanger	31.7	9.4	3.0	54.3	1045.1	33
Måltrost	36.6	11.6	5.9	66.0	1244.8	34
Munk	36.8	13.3	5.8	71.5	1509.1	41
Rødstjert	30.1	14.2	2.9	61.1	632.6	21
Rødstrupe	37.0	11.5	20.8	60.8	1183.6	32
Sivspurv	38.8	10.2	26.0	57.3	971.0	25
Svarthvit fluesnapper	34.2	8.9	2.9	62.2	2600.4	76
Svartmeis	33.8	12.7	2.9	59.8	1049.1	31
Svarttrost	35.3	12.3	2.9	85.3	1763.9	50
Toppmeis	27.9	10.3	2.9	48.1	585.2	21
Trekryper	31.9	12.8	2.9	54.8	988.8	31

Table 3.2: Statistics on recording duration corresponding to each bird. Total duration approximately 20400s, i.e approximately 5,7 hours.

Chapter 4

Basis System

This chapter contains a description of the basis bird classification system used in this project. Section 4.1 will focus on explaining the overall behaviour of the system, from input to output. Furthermore, the training and identification process will be presented in section 4.2 and 4.3. This is the two main tasks of the system, where models are trained on labeled training data so that these models can be used to classify birds in recordings without any prior knowledge. The resulting system performance is presented in section 4.4. Section 4.5 presents the initial system structure and the commercialized production model. The different methods and algorithms applied will be presented here, as well as the different scripts and the various HTK commands used.

Figure 4.1: Overview of the system with input and output

4.1 Overview

Figure 4.1 illustrates the system with input and output. When applying a recording of bird song from one of the "legal" bird species, the desired output is a label corresponding to the given bird. The goal is to classify each input file as a single bird. Often there are several labels associated with each recording and it is therefore necessary with some sort of post-processing to decide the correct label.

As mentioned in the introduction chapter there is chosen a model based segmentation approach in this system were a GMM is assigned to each class, i.e. one model for each of the birds in addition to one model for "pause". In a realistic recording, there will naturally be other sounds present. The "pause" class will be the representation for all sounds not belonging to one of the "legal" bird classes, due to the lack of labeling of the database.

The system is divided into two main parts, namely training and identification. Both parts will be discussed in the following sections. Because the database is divided into five different sets the training and identification is performed five times each.

Figure 4.2 is a block scheme describing the system where the top branch represents the training process and the bottom branch represents the identification process.

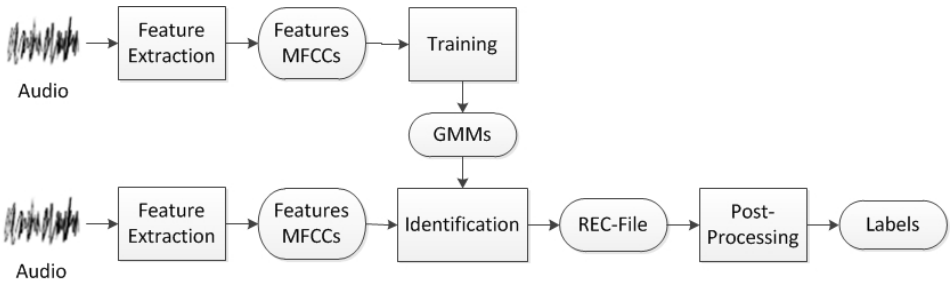


Figure 4.2: Overall description of the system

4.2 Training

Figure 4.3 shows a block scheme of the training process. As seen from the figure the training is divided into several sub tasks, briefly summarized this is the process were GMMs are created to model each of the classes.

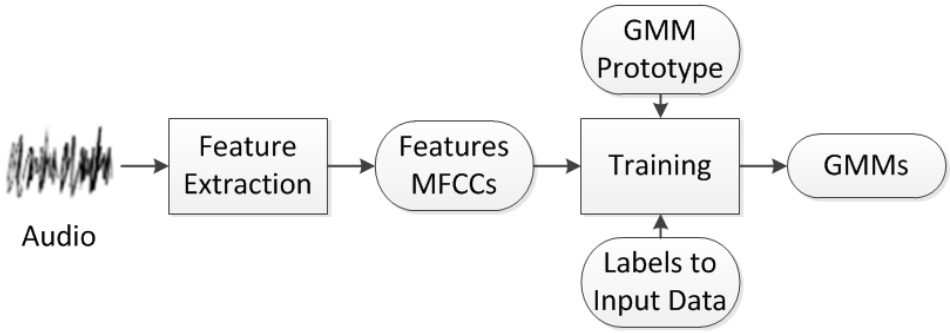


Figure 4.3: Description of the training process.

Firstly, the system is given an audio recording of bird song as input. This input is then processed segmentally because of the short-time stationary properties of bird song. There is performed a feature extraction were the signal gets represented by MFCCs. These features form a vector $\underline{x}$ for each segment, which leads to a sequence of vectors $\underline{X} = \{\underline{x}_1, \underline{x}_2, ..., \underline{x}_3\}$ representing the entire audio file. In the process of extracting these features there are used a Hamming Window of 30 msec with a step size of 20 msec resulting in an overlap between the segments. This process is described in section 2.1.

Now, the actual training process starts. Each class is now going to be modeled as a GMM. One could also have used HMMs which is dominant within the speech classification/recognition field. In this case the objective is to classify the whole recording as one single bird. The problem can be looked at as a static problem i.e. different length of the recordings are not supposed to influence the decision process. Consequently GMMs, which can be looked at as a simplified HMM were chosen to model each class. Because of simplicity reasons the GMMs were implemented in HTK as HMMs with one emitting state, but are referred to as GMMs in this thesis. Since HMMs are not used in this thesis there will not be given a detailed explanation here, but more information about the theory and its use in

speech recognition can be found in [7].

The input parameters needed to perform the training is the aforementioned features as well as a prototype GMM and labels that matches the training data. The prototype GMM contains the structure of the GMM and initial model parameters, which is mean and variance. The labeling is stored in a MLF-file which is described in chapter 3, and tells which bird is active at a given time during the recording.

The training is done by the Expectation-Maximization algorithm. This is an iterative method for finding maximum likelihood or maximum a posteriori (MAP) estimates of parameters in statistical models, where the model depends on unobserved latent variables. The EM iteration alternates between performing an expectation (E) step, which creates a function for the expectation of the log-likelihood evaluated using the current estimate for the parameters, and maximization (M) step, which computes parameters maximizing the expected log-likelihood found in the E step. These parameter-estimates are then used to determine the distribution of the latent variables in the next E step. A more detailed explanation of this process is presented in section 2.4 and 2.5.

After the training the output is a set of models. One model for each "legal" bird represented in the training data, in addition to one model for "pause".

4.3 Identification

Figure 4.4 shows a block scheme of the identification process. This part is also referred to as testing. Similar to the training process the identification is done segmentally. The identification step is where we compare the models from the training with the features from the data we want to label, i.e. the audio we want to do the identification on.

In the same way as for the training an audio recording are input to the system, and the same feature extraction is performed here. It is important to ensure that the input recordings during the identification process are different from those used for training.

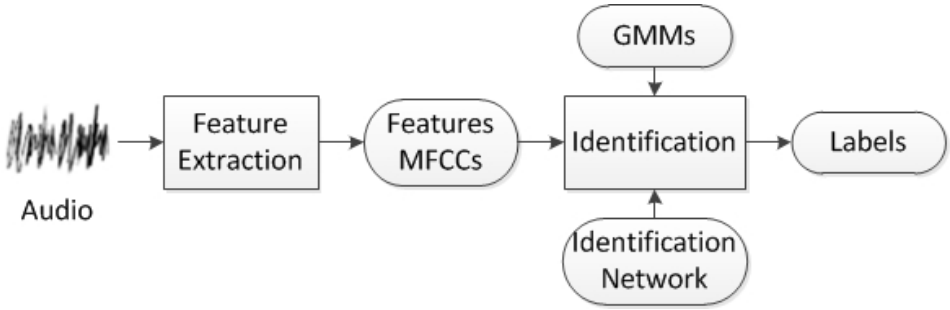


Figure 4.4: Description of the identification process.

The actual decoding is done by a Viterbi-decoder based on the Viterbi-algorithm described in section 2.6. The decoder finds the most likely sequence of birds and pause that produced the input audio. This is equivalent to finding the most likely path through the identification network shown in figure 4.5. For each detected segment the decoder provides all the classes with a log-likelihood score and the chosen bird is the one with the highest score. This decoder provides an opportunity to give a punishment for moving from one class to another in the given recognition network. A good choice of this value will result in a reasonable number of detected segments. A poor choice of this value will result in either a lot of insertions or deletions.

Output from the decoder is a REC-file corresponding to each input file, containing a series of labeled time intervals with corresponding log-likelihood score. A section from such a REC-file is displayed in table 4.1. Typically each input file is identified as several birds. Since the objective is to classify each file as one bird it is not interesting to look at $p(\underline{x}_j|w_i)$, but rather $p(\underline{X}|w_i)$. Where w_i represents the different classes. Therefore it is necessary with some kind of post-processing in order to classify the whole file as one bird.

Each line in table 4.1 represents an identified section of the file, and can consist of several segments. The two first columns are respectively start and stop time in msec for the classified class. Furthermore, the last two columns are chosen class and log-likelihood score averaged over number of segments of length 20 msec. In this case there is five sections identified as Bjørkefink and two sections identified as Munk. To decide which bird the file should be classified as it is therefore necessary with some kind of post processing. In this case it is sufficient to count the number of occurrences, but if the number of labels corresponding to Bjørkefink and Munk were equal it would be unclear which of the the birds one should choose.

"features/Bjorkefink_22.rec"			
0	9600000	pause	-75.62
9600000	18000000	Bjorkefink	-61.97
18000000	28400000	pause	-101.52
28400000	45600000	Munk	-101.52
45600000	62400000	pause	-78.87
62400000	71000000	Bjorkefink	-70.08
71000000	90400000	pause	-100.89
90400000	102600000	Munk	-100.89
102600000	109600000	Bjorkefink	-66.52
109600000	145400000	pause	-72.24
145400000	153400000	Bjorkefink	-62.61
153400000	158600000	pause	-69.89

Table 4.1: Example of REC-file after identification.

Because of this a more robust way of classifying is to count the number of segments recognized as each bird and pick the bird that occurs most frequent. This information is then stored in a histogram including all C classes. As a result, this method of classification enables the possibility of representing the $N > 1$ most likely classes that matches the given file, i.e. most frequent occurring classes. This can be implemented as an N-best list as output instead of just one single bird.

The system has also the possibility of using another classifier in addition to using the results from the histogram directly. In this case, the histogram is normalized such that the length of each file does not affect the result. Furthermore the histogram is applied as an input vector to a linear post-classifier with soft max output nodes. The vector works as estimates of the posteriori probabilities $\underline{P}_1 = \{P_1(w_1|\underline{X}), ..., P_C(w_C|\underline{X})\}$, where C is the number of classes, w is the models and X is the sequence of MFCC vectors representing a input recording. This way the output values from the classifier will represent another estimate of the a posteriori probabilities $\underline{P}_2(w_i|\underline{X})$ for each class. It follows that the chosen class is the one with highest probability, eventually a ranked list of the N classes with highest probability. This strategy resulted in a slightly better performance compared to using the results from the histogram directly.

Mathematically this classifier is described as:

$$\underline{z} = \mathbf{Q}\underline{P}_1 \tag{4.1}$$

$$\underline{P}_2(w_i|\underline{X}) = \frac{e^{z(i)}}{\sum_k e^{z(k)}}, i = 1, ..., C \tag{4.2}$$

Where $\underline{z}$ is an internal vector with dimension C, k is the number of segments corresponding to a detected class and $\mathbf{Q}$ is a matrix with dimension $C \times C$. This matrix $\mathbf{Q}$ has to be trained [1]. More information about linear classifiers can be found in [8].

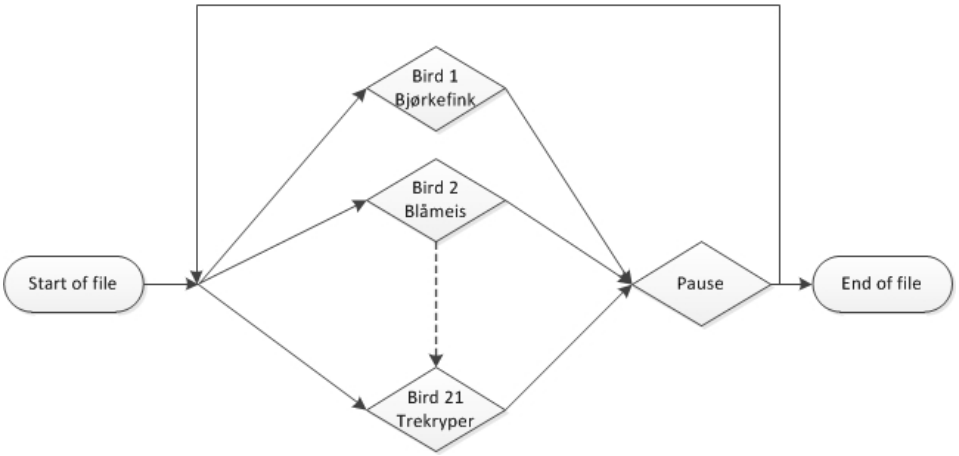


Figure 4.5: Identification network.

4.4 Results and Discussion

Table 4.2 is a representation of the system performance for the basis system. The result are presented as an 3-best list with the corresponding error rate for each case. This enables the possibility of checking if the correct bird is among the three birds with the highest likelihood after the identification. The results from using only the histogram classifier and from using the linear post-classifier in combination with the histogram are presented in this table. There is an improvement in the error rate of about 1% when looking at the 1-best case, for the 2-best case there is an improvement of about 1% and 3.5% for the 3-best case. These results will not be discussed further, since the goal of this part of the thesis was to create a production model and not focus on the system performance. The results presented in table 4.2 are the results obtained by the production model, but are identical to the results from the initial system developed by NTNU.

		80% Models Histogram Clasifer	80% Models Post-Clasifier
Test Set 1	1 Best	24.603	23.810
	2 Best	15.079	15.079
	3 Best	7.937	8.730
Test Set 2	1 Best	24.409	22.835
	2 Best	18.898	11.811
	3 Best	17.323	7.087
Test Set 3	1 Best	23.387	18.548
	2 Best	16.129	12.098
	3 Best	14.516	7.258
Test Set 4	1 Best	11.628	9.302
	2 Best	6.202	6.202
	3 Best	3.876	3.876
Test Set 5	1 Best	15.447	17.886
	2 Best	8.943	10.569
	3 Best	7.317	7.317
Average	1 Best	19.895	18.476
	2 Best	13.050	11.152
	3 Best	10.194	6.854

Table 4.2: Error rates for the different test sets and average error rate, when using histogram classifier and linear post-classifier.

4.5 System Structure

In this section the system structure will be presented. All the different scripts used to build the system will be presented and the different HTK functions used in the implementation will be explained. The process of commercializing the system into a production model will also be part of this section.

Figure 4.6 represents the system structure, with the different scripts and the relationship between them. The input and output from each script is also illustrated by this figure.

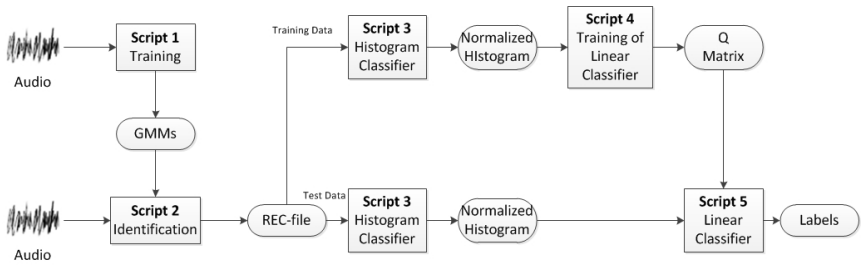


Figure 4.6: System structure.

Initially there was five separate scripts which together formed the whole system:

1. **Script 1**

   This is the script performing the training of the models. It has been used two different training scripts during this project. The first one had very poor performance, consequently another script was provided by NTNU. This new script had the same structure and applied the same HTK functions as the first script. The reason why this script performed better is not clear at this point.

   In the feature extraction the function **HCopy** were used. This function converts the input, in this case, the waveform to MFCCs. To initialize identical prototype GMMs by setting mean and variance equal to the global mean and variance **HCompV** is used. In the re-estimation of the GMM models **HRest** has been used. This is where the actual training of the models is performed. This function re-estimates the initial/previous values using the EM-algorithm. **HHed** is a function used to manipulate the GMMs. This is used to update the number of mixtures. The usual process is to modify a set

of GMMs in stages using **HHed** and then re-estimating the parameters of the modified set using **HRest** after each stage. In this script it is possible to adjust the number of mixtures used and also the number of iterations in the EM-algorithm for each number of mixtures.

Input to the script is a file list containing all the names and locations of the audio files used for training, a MLF-file containing information about the labeling of each audio file, a list of all classes (Bjorkefink, Blaameis, ... , pause), the files containing the model-parameters and their initial values and a configuration file where the sourceformat, target, segment length, window size, window type and number of coefficients is set.

2. **Script 2**

This is the script performing the Viterbi-decoding. This script takes the identification network, a file list including the names and locations of all the files included in the test set, a dictionary, the trained models from program 1, a list of all legal classes and a configuration file as input. To build the identification network the function **HParse** is used. To perform the actual decoding the **HVite** function was used. **HVite** matches the input audio file against the identification network and produces a transcription. In this case the output is a REC-file.

3. **Script 3**

This script generates the histogram mentioned in section 4.3. This was initial written in Matlab, and has been converted to a Perl script. This script takes the REC-file obtained by program 2 as input and creates the mentioned histogram. A normalized version of this histogram is the output if it is chosen to use the linear classifier. The histogram can also be used as a classifier directly, in this case the output will be a N-best list representing the most frequent occurring classes.

This script is used to calculate a normalized histogram for REC-files corresponding to both training sets and test sets. The histograms calculated

for training sets are used in the training for the post classifier, and the histogram calculated for test sets can either be used as labels or input to the post-classifier.

Figure 4.7 and 4.8 illustrates how the histogram script works. Figure 4.7(a) is the REC-file associated with a file containing song from a Gransanger. From this REC-file it is seen that this file has been recognized as tree different birds, namely Gransanger(6), Lovsanger(11) and Munk(13). Figure 4.7(b) represents the corresponding histogram which displays the duration of each recognized bird in msec. The number on the x-axis corresponds to what place the affected birds have in the class list. In the same way figure 4.8(a) and 4.8(b) represents a REC-file with the corresponding histogram. In this REC-file there is only one bird recognized, consequently there is only one bar in the histogram plot. For cases like this there is no need for additional processing.

4. **Script 4**

This is the training script of the linear post-classifier. The script was initially written in Matlab, and has been converted to a Perl script. The input to this classifier is the normalized histogram from program 3. The output is the matrix **Q** mentioned in equation 4.1.

5. **Script 5**

This script is the post-classifier, and has also been converted from Matlab to Perl. The normalized histogram and the trained **Q** matrix from script 4 are input. The output is a vector of probabilities, one value for each of the classes. These probabilities tells which bird most likely was represented in the given audio file.

More information about the different HTK commands and HTK in general can be found in [9].

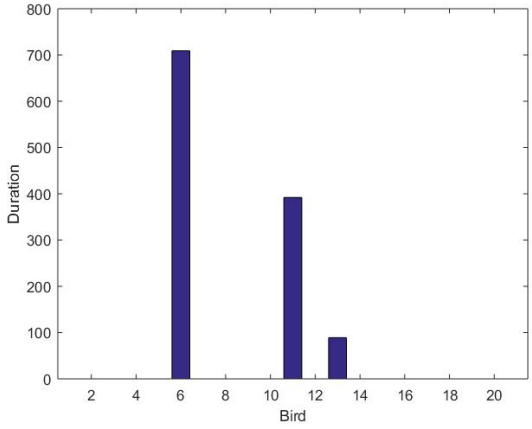
In the initial system developed by NTNU these five scripts were implemented individually. To begin with the focus was on getting a functioning system. This was naturally since the system started out as a feasibility study. When it became clear that the system worked well enough to proceed with the project, the focus shifted to commercializing the system and improving the system performance. A part of this process and the focus of this part of the thesis was to convert all scripts to Perl code, and restructuring them.

After converting all the scripts to Perl code and put them together as a complete system, this resulted in three scripts compared to the initial five. The goal was to create a simple and straightforward structure. This resulted in one script taking care of the training process consisting of script 1. Another script is handling the whole identification process. This script combines script 2,3 and 5. The last script is the training script for the post-classifier and combines script 2,3 and 4. All the three scripts can be found in appendix A,B and C (also included as attachments).

The reason why there is one separate script performing the training of the linear post-classifier, is because the script is dependent on doing an identification with the training data. As seen in figure 4.6 the input to this script is the normalized histogram from the mentioned identification. To perform the identification in the first place the training script has to create the models going to be used in the identification.

```
"features/Gransanger_8.rec"
0 22000000 pause -74.423302
22000000 38400000 Gransanger -96.915222
38400000 43400000 Lovsanger -105.328690
43400000 60000000 Gransanger -97.242882
60000000 65200000 Lovsanger -104.869881
65200000 77800000 Gransanger -94.947716
77800000 113000000 pause -64.549911
113000000 131600000 Gransanger -95.877617
131600000 136400000 Lovsanger -106.995148
136400000 150600000 Gransanger -94.426048
150600000 162400000 Lovsanger -103.516815
162400000 170200000 Gransanger -92.576035
170200000 183200000 Lovsanger -101.383636
183200000 198000000 Gransanger -96.794891
198000000 203000000 Lovsanger -104.247551
203000000 208800000 Gransanger -97.692528
208800000 262600000 pause -71.315475
262600000 273200000 Gransanger -94.728294
273200000 306800000 Lovsanger -99.348923
306800000 312800000 Gransanger -93.838799
312800000 357400000 pause -71.366447
357400000 375800000 Gransanger -96.892570
375800000 393600000 Munk -106.170471
393600000 407000000 pause -75.914391
```

(a) Rec-file

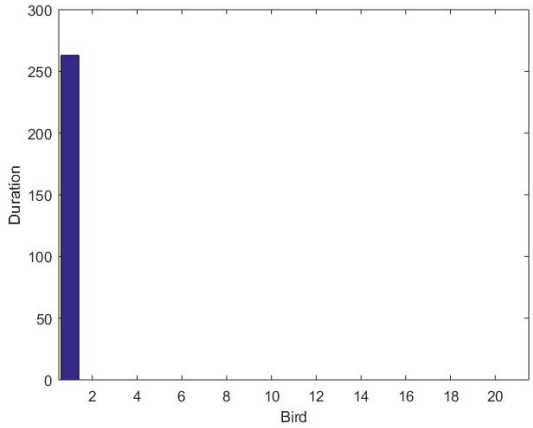


(b) Histogram

Figure 4.7: Rec-file and corresponding histogram for a Gransanger file.

```
"features/Bjorkefink_5.rec"
0 21200000 pause -56.481956
21200000 30000000 Bjorkefink -66.939262
30000000 96200000 pause -57.100235
96200000 105000000 Bjorkefink -61.741383
105000000 182000000 pause -55.012318
182000000 190600000 Bjorkefink -62.302761
190600000 263200000 pause -54.472038
263200000 272000000 Bjorkefink -61.761753
272000000 322800000 pause -67.229744
322800000 331800000 Bjorkefink -64.020370
331800000 388400000 pause -61.584763
388400000 397000000 Bjorkefink -69.598885
397000000 427400000 pause -63.503448
```

(a) Rec-file



(b) Histogram

Figure 4.8: Rec-file and corresponding histogram for a Bjørkefink file.

Chapter 5

Adaptation System

This chapter is a presentation of the experiments performed with regards to adapting more training data. This is done to improve the already existing models achieved by the basis system. Section 5.1 gives a general description of the adaptation process and typical usage of this technique. Further section 5.2 describes the approach chosen in this thesis and the difference from a typical adaptation experiment. Finally the results from the experiments are presented and discussed in section 5.3.

5.1 Adaption Overview

Because of the similarities in the treatment of bird song and speech, the adaptation process is described by looking at speech recognition. Within speech recognition speaker adaptation is widely used. The goal of this process is to modify a speakers acoustic model given the speakers acoustics characteristics by introducing new training data specific to this speaker. This is an approach that tries to deal with speaker specific variations as speaking styles and accents. The goal of the speaker adaptation is to reduce the mismatch between the models and the test data. It is normal to distinguish between supervised and unsupervised adaptation. For supervised adaptation the labeling of the adaptation data is known. This is not the case for the unsupervised case were the labeling of the adaptation data has to be estimated, i.e. using the recognition output. There exist several methods for speaker adaptation. Some of the most common methods are MAP, MLLR and VTLN. These methods will not be discussed further in this thesis, but more information about them and the adaptation process in general can be found in [10] and [11].

5.2 Adaptation Experiment

This part is related to the adaptation approach chosen in this thesis. The goal is to improve already existing models by adapting additional data to the initial training sets. This is different from what is seen as a typical adaptation situation described in 5.1 were the goal is to adapt the models to the different birds/speakers. The goal of this experiment is to improve the overall system performance by adapting more data. This data is not directed against any birds in particular, but representing all birds. This is an effort of trying to improve the system performance by increasing the amount of training data and hence get better models. The goal of this experiment is to find out if this process can be done automatically, by using the output from the system to label the data. This represents an unsupervised adaptation process mentioned in section 5.1.

As mentioned in chapter 3 the database is divided into 5 different sets which is put together to form training and test sets. The adaptation experiment where the adaptation data is going to be labeled automatically is implemented by using 60% of the total database for training instead of 80%. This way 20% of the database is set aside for testing and another 20% for the actual adaptation. The automatic labeling is performed by the histogram classifier. Still, there are five different partitions. The only difference from the original setting is that a part of what used to form the training set is now used as data going to be adapted. Consequently, the adaptation and the test sets are the same, but for each training set it is used adaptation and test sets that are different from each other. Figure 5.1 represents one partition of the data used in this experiment. By doing the adaptation experiment with the manually labeled data the results will be the same as in section 4.4 and represented in table 4.2.

Figure 5.1: Example of partitioning of the database used in the adaptation experiment.

The adaptation experiment when the adaptation data is labeled automatically is conducted this way:

1. Train models with the 60% data sets.
2. Run the identification procedure on the generated models from step 1 with the adaptation sets, and use the histogram classifier to label each file.
3. Use the resulting REC-files from the identification in step 2 together with the results from the histogram classifier to form new MLF-files. This is done by setting the segments in the REC-file which is identified as something else than the classifier says to "unknown". This way these segments will not be a part of the data that will be adapted to the existing training data. Labels which are not in the class list given to HTK will simply be ignored.
4. Form new training sets consisting of the 60% training sets combined with the 20% adaptation sets, except the segments that are discarded. These training sets will contain between 60% - 80% of the total database dependent on how much of the adaptation sets that will be set to "unknown". The new training sets needs a new MLF-file and list-file. The MLF are formed by joining the MLF from step 3 with the MLF corresponding to the initial 60% training sets. The list is formed by joining the list used for the 60% training sets with the list for the adaptation sets.
5. Train models with the new data sets from step 4.
6. Run the identification procedure on the new models with the test sets, and use the histogram classifier to label each file.
7. Finally compare the results when doing the identification on the 80% models, the 60% models and the (60+20)% (adaptation) models.

5.3 Results and Discussion

This section is a presentation of the results obtained by the different experiments performed with regards to adaptation. The resulting discussion related to the outcome of these experiments are also presented here.

		80% Models	60% Models	(60 + 20)% Models
Test Set 1	1 Best	24.603	24.603	28.571
	2 Best	15.079	16.667	18.254
	3 Best	7.937	14.286	12.698
Test Set 2	1 Best	24.409	24.409	23.622
	2 Best	18.898	16.535	17.322
	3 Best	17.323	15.748	15.748
Test Set 3	1 Best	23.387	30.645	37.903
	2 Best	16.129	23.387	27.419
	3 Best	14.516	20.968	23.387
Test Set 4	1 Best	11.628	23.256	22.480
	2 Best	6.202	12.403	13.178
	3 Best	3.876	9.302	10.078
Test Set 5	1 Best	15.447	20.325	22.764
	2 Best	8.943	13.821	13.008
	3 Best	7.317	10.569	10.569
Average	1 best	19.895	24.648	27.068
	2 Best	13.050	16.563	17.837
	3 Best	10.194	14.175	14.496

Table 5.1: Error rates for the different test sets and average error rate.

		60% Models
Adaptation Set 1	1 Best	24.194
	2 Best	16.935
	3 Best	13.710
Adaptation Set 2	1 Best	21.429
	2 Best	15.873
	3 Best	8.730
Adaptation Set 3	1 Best	15.504
	2 Best	7.752
	3 Best	5.426
Adaptation Set 4	1 Best	27.642
	2 Best	15.447
	3 Best	12.195
Adaptation Set 5	1 Best	25.197
	2 Best	19.685
	3 Best	16.535
Average	1 Best	22.793
	2 Best	15.139
	3 Best	11.319

Table 5.2: Error rates for the different adaptation sets and average error rate.

Table 5.1 is a comparison of the results obtained by doing identification with the same test sets with different models. Were the 80% models refers to models trained with 80% of the available data, while the 60% models refers to the models trained with 60% of the available data and the (60 + 20)% models refers to the models trained with the 60% training sets in addition to data from the adaptation sets automatically labeled by the system. These models are the adaptation models obtained by following the steps on the previous page. What can be seen from this table is that the results from the 80% models serves as an upper bound on the system performance. The average error rate is about 7% higher for the adaptation models compared to the 80% models when looking at the 1-best case. The difference is about 5% for the 2-best case and 4.5% for the 3-best case. The adaptation models also achieve a worse performance than the 60% models with an increase in the error rate of about 2.5% for the 1-best case.

Table 5.2 shows the results from the identification performed with the adaptation sets on the 60% models, in other words this is a measure of how well the labeling of the new training data has been done. The reason why these results are different

from the results from the test sets on the 60% models is that the different sets are recognized on different models. From this table it seen that approximately 23% of the data going to be adapted gets misclassified.

Another observation is that the performance varies widely for the different test sets in table 5.1 and adaptation sets in table 5.2. By looking at the adaptation models in table 5.1 it is over 15% higher error rate for test set 3 compared to test set 4 when looking at the 1-best case. In the same way there is a 12% difference between adaptation set 3 and 4. This implies that the variation in the bird song and the quality of the recordings has very great influence on the system performance. This shows that it is very important to have good recordings, and indicates that the process of adapting more data automatically can be difficult to implement without a quality control of the adapted data. This will be especially important for user submitted recordings, which may be of very bad quality dependent on surroundings and recording device among other things. There is reason to believe that this sensitivity will decrease with increasing amount of training data, since the models will be better when the data covers a greater area of the birds characteristics. With the current system and available data it seems like being "lucky" with the input recordings plays an important role in the performance. Based on this, one should be careful about drawing firm conclusion from the results.

The goal of the adaptation experiment was to find a way of improving already existing models. In other words, the desired performance from the adaptation models should have been between the performance of the 80% models which serves as an upper measure of how well the system can perform, and the 60% models. For both the 80% models and the 60% models all training data is labeled correctly. This is not the case for the adaptation models, but with an error of labeling a file wrong in the adaptation sets at approximately 23%, it is somewhat surprising that the performance has not improved. This is an indication that the mistakes being made are very damaging and that the system is sensitive to misclassifications. It looks like the system performance is not good enough to do this adaptation automatically.

Figure 5.2 shows how the new MLF-file has been manipulated after the adaptation experiment. In this figure there is two different MLF-files corresponding to two different input files. One of the files has been correctly classified as Bjørkefink and the other one has been misclassified as Rødstrupe, while the correct bird is Blåmeis. This figure is a good illustration on one of the issues related to this automatic adaptation of data. From the file which has been classified correctly, it is seen that three segments are still set as unknown. This is reducing the potential amount of adaptation data. For the other file most segments has been set as unknown, while three segments are set as Rødstrupe. These three segments are further included in the training of a new Rødstrupe model instead of a Blåmeis model. This is affecting the Rødstrupe model by including Blåmeis attributes, and it is affecting the Blåmeis model by making the Rødstrupe model more alike the Blåmeis model.

Another reason that may contribute to the bad performance is a mismatch between the labeling of the input file and the recognized segment borders in the REC-file. This will potentially have an impact on the training of the new models. After the identification the resulting REC-file list all recognized segments with start and stop time. If the identification has been done badly it is possible to have overlapping segments compared to the actual recording. The result of such a mismatch can for example be inclusion of pause data in the class of an arbitrary bird class. Potentially this can lower the quality of the models. The pause model can be affected by including bird song in the training of a new adapted model. The different bird models can be affected by including data, initially labeled as pause in the same process.

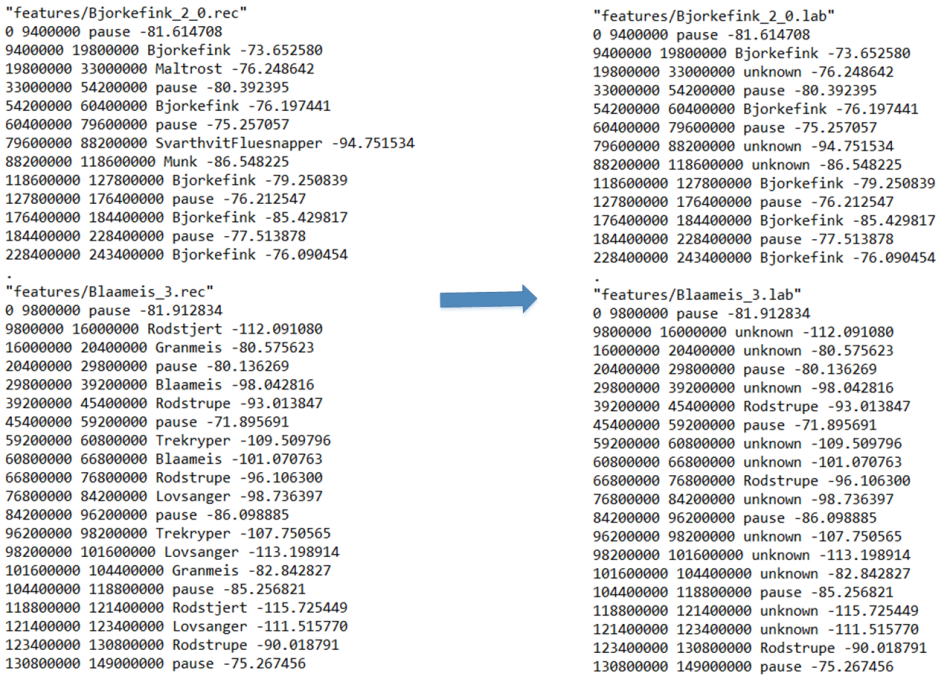


Figure 5.2: Manipulation of MLF-files

As an effort to investigate why the adaptation do not achieve the desired improvement, it is conducted a more thorough analysis of the individual performance of the different birds, trying to find a trend in which species performs good and bad. Table 5.3 represents the number of misclassifications for each bird summed up over all five test sets, and table 5.4 represents the corresponding error rate. What is seen from these tables is the same trend as for the overall performance shown in table 5.1. Table 5.5 represents both number of misclassifications and error rate for the adaptation sets on the 60% models.

It would then be of interest to look at the results obtained by the adaptation sets on the 60% models and compare these results against the ones obtained by the test sets on the 60% and (60+20)% models. What is expected from this comparison is that species performing good for the adaptation sets will have the greatest increase in performance when comparing the result from the 60% and (60+20)% models. Species performing badly for the adaptation set are expected to come badly out of this comparison. This is because the good classes will have more correct labeled training data and the poor classes have more misclassifications resulting in wrong labeling. Despite this, it is difficult to see a clear trend substantiating this theory. An explanatory reason for this is that the database is too small, and the amount of adaptation data is too small to achieve the desired improvement.

Bird	Number of Files	Misclassifications		
		80% Models	60% Models	(60+20)% Models
Bjørkefink	19	5	5	8
Blåmeis	30	6	9	11
Bokfink	18	2	2	3
Dompap	14	6	7	8
Granmeis	21	6	9	9
Gransanger	30	1	3	1
Grønnfink	27	0	0	1
Gulspurv	23	7	13	14
Hagesanger	24	0	4	2
Jernspurv	28	2	2	2
Løvsanger	33	12	9	16
Måltrost	34	11	11	11
Munk	41	5	7	6
Rødstjert	21	3	4	6
Rødstrupe	32	3	4	6
Sivspurv	25	7	11	9
Svarthvit Fluesnapper	76	26	28	28
Svartmeis	31	12	12	15
Svarttrost	50	4	3	0
Toppmeis	21	7	10	9
Trekryper	31	2	2	5

Table 5.3: Number of misclassification related to each bird for the test sets.

Bird	Number of Files	Error Rate		
		80% Models	60% Models	(60+20)% Models
Bjørkefink	19	26.3	26.3	42.1
Blåmeis	30	20.0	30	36.7
Bokfink	18	11.1	11.1	16.7
Dompap	14	42.9	50.0	57.1
Granmeis	21	28.6	42.9	42.9
Gransanger	30	3.3	10.0	3.3
Grønnfink	27	0.0	0.0	3.7
Gulspurv	23	30.4	56.5	60.9
Hagesanger	24	0	16.7	8.3
Jernspurv	28	7.1	7.1	7.1
Løvsanger	33	36.4	27.3	48.5
Måltrost	34	32.4	32.4	32.4
Munk	41	12.2	17.1	14.6
Rødstjert	21	14.3	19.0	28.6
Rødstrupe	32	9.4	12.5	18.8
Sivspurv	25	28.0	44.0	36.0
Svarthvit Fluesnapper	76	34.2	36.8	36.8
Svartmeis	31	38.7	38.7	48.4
Svarttrost	50	4.0	6.0	0.0
Toppmeis	21	33.3	47.6	42.9
Trekryper	31	6.5	6.5	16.1

Table 5.4: Error rate related to each bird for the test sets.

		60% Models	
Bird	Number of Files	Misclassifications	Error Rate
Bjørkefink	19	6	31.6
Blåmeis	30	8	26.7
Bokfink	18	2	11.1
Dompap	14	8	57.1
Granmeis	21	5	23.8
Gransanger	30	6	20.0
Grønnfink	27	1	3.7
Gulspurv	23	9	39.1
Hagesanger	24	3	12.5
Jernspurv	28	2	7.1
Løvsanger	33	14	42.2
Måltrost	34	12	35.3
Munk	41	3	7.3
Rødstjert	21	4	19.0
Rødstrupe	32	2	6.3
Sivspurv	25	10	40.0
Svarthvit Fluesnapper	76	24	31.6
Svartmeis	31	11	35.5
Svarttrost	50	3	6.0
Toppmeis	21	9	42.9
Trekryper	31	1	3.2

Table 5.5: Number of misclassifications and corresponding error rate to each bird for the adaptation sets.

To illustrate this it is chosen to look at the five birds with the best results from the adaptation sets shown in table 5.5, and compare them against the same birds from table 5.3 and 5.4. This comparison is summarized in table 5.6 with error rates and number of misclassifications in parenthesis. The most important in this table is to compare column three and four. Since the table illustrates the birds with the best performance there are initially few misclassifications done, and not that much room for improvements. This means that it is difficult to draw definitive conclusions from these results. Still the number of misclassification for Trekryper, Grønnfink and Rødstrupe has increased and this is surprising with regards to how well the results from the adaptation sets are. This means that most of the files corresponding to these birds have been labeled correctly. The result is particularly surprising for Trekryper and Rødstrupe where the number of misclassifications are relatively high with error rates of respectively 16.1% and 18.8% for the adaptation models. For Grønnfink the number of misclassifications has gone from zero to one. One should not read too much into that, since it is naturally that the models change when introducing new data and the error rate is still low with 3.7%.

In the same way table 5.7 illustrates the five birds with the poorest performance from the adaptation sets. The expected outcome for these birds is a smaller improvement when comparing column 3 and 4 than for the birds with higher performance. Since the overall improvement is negative for the adaptation models, it is expected that the birds having the fewest correct classifications for the adaptation sets are the ones that performs worst. From the table it is seen that Dompap, Løvsanger and Sivspurv gets a decline in performance. For Løvsanger the number of misclassifications has increased from 9 to 16, which is an increase in the error rate of almost 44%. However, both Toppmeis and Sivspurv gets a decrease in the number of misclassifications. It is therefore difficult to draw a conclusion between which birds have the most correct labeled adaptation data, and the birds with the greatest improvement after the adaptation experiment. When Sivspurv with just 40% of its adaptation data correct labeled gets an increase of 8% it becomes evident that the system is very unreliable and that it is very important to be "lucky" with the adapted data. In general we can see that the changes in performance between the 60% models and the (60+20)% models are greater in table 5.7 compared to table 5.6. This can probably be explained by the fact that there was a lot more misclassifications for the 60% models in this case.

Bird	Adaptation 60% Models	Test 60% Models	Test (60+20)% Models
Trekryper	3.2 (1)	6.5 (2)	16.1 (5)
Grønnfink	3.7 (1)	0.0 (0)	3.7 (1)
Svarttrost	6.0 (3)	6.0 (3)	0 (0)
Rødstrupe	6.3 (2)	12.5 (4)	18.8 (6)
Jernspurv	7.1 (2)	7.1 (2)	7.1 (2)

Table 5.6: Error rate and number of misclassification for the five birds performing best for the adaptation sets.

Bird	Adaptation 60% Models	Test 60% Models	Test (60+20)% Models
Dompap	57.1 (8)	50.0 (7)	57.1 (8)
Toppmeis	42.9 (9)	47.6 (10)	42.9 (9)
Løvsanger	42.2 (14)	27.3 (9)	48.5 (16)
Sivspurv	40.0 (10)	44.0 (11)	36.0 (9)
Gulspurv	39.1 (9)	56.5 (13)	60.9 (14)

Table 5.7: Error rate and number of misclassification for the five birds performing worst for the adaptation sets.

As an attempt to remove some of the segments that has been misclassified in the REC-file corresponding to the adaptation sets, each bird class are given a log-likelihood threshold shown in table 5.8. In the REC-files there is a corresponding log-likelihood value to each identified class as described in chapter 4. If this value is below the given birds threshold, this segment is set to unknown and will not be a part of the adapted data when training the adaptation models. The reason why this is done is to remove the most severe mistakes, which is believed to corrupt the models the most. The corresponding log-likelihood to each class in the REC-file is an indication on how reliable the identification of this class has been done. This is the reasoning behind setting these thresholds.

These thresholds were found by plotting the log-likelihood values associated with both the adaptation sets from running the identification on the 60% models and running the identification with the same training sets used to train these 60% models. It is only the log-likelihoods with the lowest scores that is included in the histograms, since these are the most likely to have been misclassified. Since the training data is labeled it is possible to plot only the correct classifications from these sets. This is not the case for the adaptation sets where the histogram classifier has to be trusted. A histogram plot is generated for each bird specie with data from each of the five training and adaptation sets displayed in the same figure. As a result of this it is possible to set a threshold if the log-likelihoods corresponding to the adaptation sets is different from the ones corresponding to the training sets which is known to be correct classifications.

Bird	Threshold
Bjørkefink	Keep all
Blåmeis	-95
Bokfink	-103
Dompap	Remove all
Granmeis	Keep all
Gransanger	-100
Grønnfink	-100
Gulspurv	-110
Hagesanger	Keep all
Jernspurv	-110
Løvsanger	-113
Måltrost	-105
Munk	-108
Rødstjert	-108
Rødstrupe	-105
Sivspurv	Keep all
Svathvit Fluesnapper	-108
Svartmeis	-107
Svarttrost	-105
Toppmeis	-98
Trekryper	-96

Table 5.8: Log-likelihood thresholds corresponding to each of the birds.

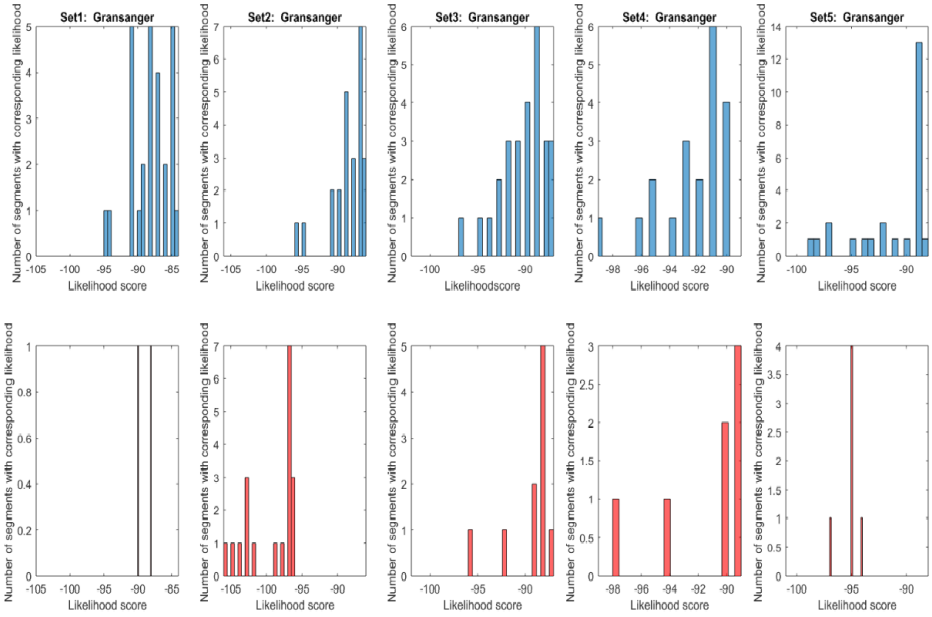


Figure 5.3: Histogram plot of log-likelihood values corresponding to Gransanger identifications.

Figure 5.3 is an example of such a histogram plot for the Gransanger class. All five test sets are plotted in the same figure making it easier to see an overall trend. It is desired to set a threshold which is applicable for all sets. The goal is to set a threshold excluding the segments from the adaptation sets with the lowest score, but also falling outside of the region the log-likelihoods from the training sets span over since the segments included in the histograms for the training sets are all correct classifications. These values are therefore an indication on what log-likelihood values represents good classifications. In this particular case for the Gransanger the threshold is set to -100. This choice is illustrated well by looking at set 2 in figure 5.3, where some segments from the adaptation set are excluded while keeping all of the training segments.

By comparing the results from the identification done with and without the thresholds as shown in table 5.9, it is possible to see that the system performance has slightly improved when using the thresholds. The goal of the thresholds was to eliminate the recognized segments with the lowest log-likelihood scores and hence the most unreliable segments. There is an improvement of about 1.5% for the average error rate for the 1-best case and an improvement of about 1% for the 2-best and 3-best case. With these thresholds applied there has not been removed many segments, so that the improvement is small is not surprising. Despite this, the improvement shows that the misclassifications in the adaptation data is one of the reasons why this adaptation experiments provides such bad results.

It seems like the combination of too little available data and not good enough system performance makes the process of automatically adapting data very difficult to perform in a satisfactory manner. In fact the adaptation experiments have led to poorer results compared to the results obtained by the 60% models. This is mainly a result of the limited access to data. As a result of this the system is very sensitive to misclassified adaptation data. This is illustrated well by the threshold experiment. There is reason to believe that this adaptation process will achieve better results with a larger database. This is supported by the fact that the results obtained by the 80% models are that much better. This proves that an increase in correct labeled training data results in better models and better system performance. This indicates that the adaptation process should be performed manually with the current system performance and access to data. It has not been done any experimenting with the partition of data with respect to training/testing/adaptation. It is possible that another relationship here could have gotten better results, but the main problem is the small database.

		(60+20)% Models No Thresholds	(60+20)% Models Thresholds
Test Set 1	1 Best	28.571	28.571
	2 Best	18.254	16.666
	3 Best	12.698	12.698
Test Set 2	1 Best	23.622	23.622
	2 Best	17.322	17.322
	3 Best	15.748	16.535
Test Set 3	1 Best	37.903	31.451
	2 Best	27.419	25.806
	3 Best	23.387	20.967
Test Set 4	1 Best	22.480	25.581
	2 Best	13.178	13.178
	3 Best	10.078	9.302
Test Set 5	1 Best	22.764	18.699
	2 Best	13.008	12.195
	3 Best	10.569	8.130
Average	1 Best	27.068	25.585
	2 Best	17.837	17.034
	3 Best	14.496	13.527

Table 5.9: Error rates for the different test sets and average error rate, when comparing identification performed with and without log-likelihood thresholds.

Chapter 6

Conclusion

6.1 Basis System

The initial system has been adapted to a commercial setting by converting all scripts to Perl code and put them together to a complete system. In this process the system has been changed from consisting of five different individual scripts, to a more straightforward and intuitive system consisting of three different scripts. In this new structure one script is aimed at the training process, another one for the identification process and a third one dealing with the training of the linear post-classifier.

6.2 Adaptation System

By performing the identification with the adaptation models there are an 2.5% increase in the error rate compared to when using the 60% models. Compared to the 80% models there are an 7% increase in the error rate. The desired outcome of adapting data was to get models lying between the 60% and 80% models with regards to performance. This is not the case, and shows that the system is very sensitive to misclassifications in the adapted data. With the current performance the amount of misclassified segments in the adaptation data lies around 23%. With this accuracy on the labeling of the new training data it is somewhat surprising that the the adaptation models performs this bad. It is an indication on that the database is too small.

Another thing shown by the identification results is that the different test sets gets very different results. This is common for the results obtained by all the models.

It seems like the quality of the recordings are very crucial for the system performance, and it indicates that it is important to be "lucky" with the input recordings to achieve the best results.

If adaptation is going to be carried out in a way that helps improve the system performance this has to be done manually, by going through the adaptation data making sure that it is labeled correctly. In a commercial setting where this system is implemented as an application for smart phones the adaptation data can be recordings from the users of the applications. This data can be taken in different environments and be of bad quality. This quality control will be very important with regards to the systems sensitivity to different recordings. After collecting enough data and labeled it manually, the adaptation process can most likely be done automatically because of a more robust system. When the database gets larger the system will not be as sensitive to misclassifications. This is proven by the fact that the 80% models gets better results than the 60% models. This also substantiates the approach of making better models by adding more training data, and shows that a major problem is related to the misclassifications.

The threshold experiment also substantiated the fact that the system is sensitive to misclassifications. The goal of this experiment was to eliminate some of the most severe misclassifications from the adaptation data. When introducing the thresholds the system performance increased slightly. The thresholds did not exclude very many segments corresponding to each bird and therefore it was not expected with a great improvement, but the purpose was rather to prove that the automatic labeling was not done sufficiently well, and that the most sever mistakes were critical for the result.

All the experiments performed with regards to adaptation are showing that the current system with regards to performance and the amount of data which is currently available is not suited for automatically adaptation of training data. It is possible that there could have been a better partition of the data with regards to training/test/adaptation, but the bottom line is that the database is too small to perform this adaptation in a sufficient way.

6.3 Further Work

Based on this report and the work performed in this thesis it is apparent that there are several improvements that can be done to improve the system. The most important thing will be to collect more data. This data should cover all the different species and different recording conditions. Ideally, the recordings should come from different sources and situations. All this to make sure that the recordings capture a lot of different aspects of the birds characteristics, in contrast to having a lot of data which is almost similar. This will make the corresponding models more robust.

The collected data then has to be labeled in order to be used in training of the models. Based on the results achieved by the adaptation experiments in this report it is evident that this has to be done manually. After a large enough amount of data has been collected the system performance will increase and the need of manually labeling will diminish. It is difficult to give an exact number of how much more data is needed before the labeling can be done automatically. In the process of labeling there is also room for improvements in splitting up the "pause" class into background noise and other noise sources when this is present.

Bibliography

[1] Johnsen, Svendsen and Perkis. WhatBird - Documentation, 2015

[2] Neural Information Processing Scaled for Bioacoustics, 2013.

[3] Gupta, Jaafar, Fatimah wan Ahmad and Bansal. Feature Extraction Using MFCC, 2013:
http://airccse.org/journal/sipij/papers/4413sipij08.pdf

[4] Douglas Reynolds, Gaussian Mixture Models:
https://www.ll.mit.edu/mission/cybersec/publications/publication-
full_papers/0802_Reynolds_Biometrics-GMM.pdf

[5] Jeff A. Blimes, A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models, 1998:
http://crow.ee.washington.edu/people/bulyko/papers/em.pdf

[6] G. David Forney, The viterbi Algorithm, 1973:
http://www.systems.caltech.edu/EE/Courses/EE127/EE127A/
handout/ForneyViterbi.pdf

[7] Zoubin Ghahramani, An Introduction to Hidden Markov Models and Bayesian Networks, 2001.
http://mlg.eng.cam.ac.uk/zoubin/papers/ijprai.pdf

[8] Theodoridis and Koutoumbas, Pattern Recognition, First Edition, Chapter 3.

[9] Young, Evermann, Gales, Hain, Kershaw, Liu, ...Woodland. The HTK Book v3.4.1, 2015.

[10] Cui and Alwan, Robust Speaker Adaptation by Weighted Model Averaging Based on the Minimum Description Length Criterion, 2007:
http://www.seas.ucla.edu/spapl/paper/aslp_mdl.pdf

[11] Steve Renals, Speaker Adaptation, 2008:
http://www.inf.ed.ac.uk/teaching/courses/asr/2008-9/asr-adapt-1x2.pdf

Appendix

A: TrainingProcess.pl

```
#!/usr/bin/perl


use Getopt::Std;

# Directory structure for training
$d="./config";                          # input data files, part of
    installation
$w="./work_60_20_threshold_";                           # directory for
    generated index files,
                                        # edit scripts, etc.
$t="./tools";                           # sub-scripts, part of installation
$m="./models_60_20_threshold_";                         # output HMM root
    directory
$f="features";          # feature file root directory


# Various files needed
#$PhoneMlf="lists/Kvirrevitt_new.mlf";  # Phone level MLF from initial
    segm.
$PhoneMlf="MLF/train5adaption5_threshold.mlf";
$list="lists/train5.scp";               # database information file
#$Proto="lists/proto_s1_m1_dc.pcf";  # Specifications for the HMM
    prototype
$Proto="lists/proto.pcf";  # Specifications for the HMM prototype
# $Hcomp_conf="$d/hcompv_mfc.cfg";
# $Hrest_conf="$d/herest_mfc.cfg";
$Hcomp_conf="$d/hcompv_mfc.cfg";
$Hrest_conf="$d/herest_mfc.cfg";

# Parameters for training
$nmix=1;                                # Number of mix components for
    initialization
$maxmix=32;                             # Max number of mixture components
$moniter1=20;                            # Iterations of HInit and HRest (1
    mix)
$moniter2=40;                            # Iterations of HRest (> 1 Gaussian
    mix)


# Read initial MLF and directory to put models (optional)
```

```
getopts('p:');
$partition=2;
if ($opt_p){$partition=$opt_p;}

# Check/create subdirectories...

foreach $tmp ($t,$d) {die "Directory $tmp not found\n" unless (-d $tmp);}
$ENV{'PATH'} = "./$t".":".$ENV{'PATH'};          # put tools directory
    first in path
require("$t/MMF_subs.pl");                       # load common subroutine
    libraries
require("$t/common.pl");




$list="lists/train"."$partition".".scp";
$m="$m"."$partition";
$w="$w"."$partition";
$Dict="$w/Kvirrevitt.dict";       # Dictionary
$Phones="$w/Kvirrevitt.lis";      # Class inventory
$Trainset="$w/trainset.scp";        # List of training files

# Then we are ready to start processing...

print "Trainmodels_Bootstrap started ",`date`,"\n";
goto skip;        # restart the script by moving the skip label to the
                  # point where it stopped

skip:             # start processing again here
foreach $tmp ($m,$w){&makedir($tmp)};

$cmd="cp $list $Trainset";
&run("$cmd");


open(PHL,"$PhoneMlf") || die "Could not open $PhoneMlf\n";
%phlist=();
while(<PHL>){
    chomp;
    if ($_ =~ /^[\"\.\#]/) {next;}
    ($start,$end,$phone)=split(' ');
    $phone=~ s/^\s+//;
    $phone=~ s/\s+$//;
    $phlist{$phone}=$phlist{$phone}+1;
}
close(PHL);

open(PHN,">$Phones") || die "Could not open $Phones\n";
open(DIC,">$Dict") || die "Could not open $Dict\n";
foreach $phone (sort keys %phlist) {
    if ($phone eq "sil"){next;}
    print PHN "$phone\n";
    print DIC "$phone $phone\n";
}
close(PHN);
close(DIC);
```

```
$startDir=0;

print "HInit/HCompV based on initial segmentation\n";
#-------------------------------------------------------------

$srcdir="$m/state1_$startDir";
$tgtdir="$m/state1_$nmix";
&makedir($srcdir);
&makedir($tgtdir);
open(PRTIN,"$Proto") || die "Could not open HMM prototype spec file,
    $Proto for reading\n";
open(PRTOUT,">$w/proto.pcf") || die "Could not open HMM prototype spec
    file, $Proto for writing\n";
while(<PRTIN>){
        chomp;
        if ($_ =~ /^outDir/){
                print PRTOUT "outDir: $srcdir\n";
        } else {
                print PRTOUT "$_\n";
        }
}
close(PRTIN);
close(PRTOUT);
#$cmd="cat $Proto | sed 's/proto\/$srcdir2\/' > $w/proto.pcf";
#print "$cmd\n";
#&run("$cmd");
$cmd="$t/MakeProtoHMMSet $w/proto.pcf";
&run("$cmd");
#&MkMacro($ProtoMMF,"$srcdir/$Macro");

# Initialize with a) global variance, b) <20 iterations of HInit
# For all phones in $Phones:
open(PHONES,"$Phones") || die "Could not open $Phones\n";
while(<PHONES>){
        chomp;
        $Ph=$_;
        $cmd="HCompV -A -C $Hcomp_conf -M $srcdir -S $Trainset $srcdir/$Ph
              >> dump";
        &run("$cmd");
}
close(PHONES);

print "Baum-Welch reestimation using HRest\n";
#-------------------------------------------------------------

$tgtdir="$m/state1_$nmix";
&makedir($tgtdir);

# For all phones in $Phones:

open(PHONES,"$Phones") || die "Could not open $Phones\n";
while(<PHONES>){
        chomp;

        &run("HRest -A -T 1 -w 2.0 -i $moniter1 -I $PhoneMlf -l $_ -t " .
```

```
            "-C $Hrest_conf -H $srcdir/$_ -M $tgtdir -S $Trainset " .
            "$tgtdir/$_ >> dump");
}
close(PHONES);
$tgthmm="$tgtdir.mmf";
&MkMMF2($tgthmm,$Phones,$tgtdir,A);


print "Training mixture monophones ".`date`;
#-----------------------------------------

sub monomixup {
    local($nmix,$sourcehmm,$targethmm,$phone) = @_;
    &printfile("$w/mix${nmix}.hed", "MU $nmix {*.state[2-4].mix}");
    # print " ...updating to $nmix mixtures in $targethmm\n";
    &run("HHEd -A -H $sourcehmm -w $targethmm $w/mix${nmix}.hed ".
         "Modelnames/$phone");
    # print " ...finished ".`date`;
}

$mmix=$nmix;
$sourcedir="$m/state1_${nmix}";

while ($mmix < $maxmix) {

    $newmix=2*$mmix;
    $targetdir="$m/state1_${newmix}";
        &makedir($targetdir);
    $mmix=$newmix;
    print "training $targethmm for $newmix mixtures\n";
        open(PHONES,"$Phones") || die "Could not open $Phones\n";
        while(<PHONES>){
                chomp;
                $targethmm="$targetdir/$_";
                $sourcehmm="$sourcedir/$_";
#               &run("echo $_ > $w/temp");
        &monomixup($newmix,$sourcehmm,$targethmm,$_);
#               &run("rm -f $w/temp");
                &run("HRest -A -T 1 -i $moniter2 -I $PhoneMlf -l $_ -t " .
                     "-C $Hrest_conf -H $targethmm -S $Trainset " .
                     "-M $targetdir $targethmm >> dump");
        }
        close(PHONES);
        $tgthmm="$targetdir.mmf";
        &MkMMF2($tgthmm,$Phones,$targetdir,"A");
        $sourcedir=$targetdir;
}

print "TrainModels finished ",`date`;
```

B: IdentificationProcess.pl

```
#!/usr/bin/perl

use strict;
use warnings;
use POSIX;

##################################################
#------------------------------------------------
#-----------Viterbi-decoding HTK-----------------
#------------------------------------------------
##################################################

# Choose number of mixtures and penalty for jumping between class models
my $mix = 32; my $wmix = "32";
my $wordpen = -80; my $wp = "pm_80";

 # Assuming gmm models are trained and placed in ... :
my @modeldirs = ("models1", "models2", "models3", "models4", "models5");

# Using the following configure file
my $cnfg = "mhj_20_mfc.cfg";

# and the following test sets :
my @datasets= ("lists/test1.scp",  "lists/test2.scp", "lists/test3.scp", "
    lists/test4.scp", "lists/test5.scp" );

# Class_list, dictionary and network must exist
my $cl_list = "class_pause_ny.list";
my $dict = "birds_ny.dict";
my $network = "birds_ny.net";

my $labmlf = "Kvirrevitt_ny.mlf";

# Establish folder to put rec-files
my $resultDir = "Results_hmm";
unless(-d "$resultDir") {system("mkdir $resultDir");}
my $resultfile = $resultDir."/dummy.res";
system "rm $resultfile ";
system "touch $resultfile";

# Loop over the five training/test-set partitions of the data.
my $set = -1;
foreach my $modeldir (@modeldirs){

        $set = $set +1;
    my $setind = $set +1;
        my $testset = $datasets[$set];
        $modeldir = $modeldirs[$set];
    my $mmf = "$modeldir/state1_$mix.mmf";
        print "speakerlist : $testset\n";
        # printf "Results are placed in $recmlf and $resultfile\n";
    my $recmlf = $resultDir."/test".$setind."_".$wmix."_".$wp.".rec";
```

```
        system "HVite -o N -C $cnfg  -A -t 0.0 -s 0 -p $wordpen -H $mmf -y
             rec -S $testset -i $recmlf -w $network $dict $cl_list >>
            $resultfile";
    system "HResults -p  -A -I $labmlf $cl_list $recmlf >> $resultfile";
}



##################################################
#-------------------------------------------------
#-------------Histogram program-------------------
#-------------------------------------------------
##################################################

my $pm = 80;
my $mix = 32;
my $set = 4;
my $type = 'test';
my $modelName;
my @modelList = ();
my $Nclass = 0;
my $classfile = 'class_pause_ny.list'; # list of all legal classes

open(LIST,"$classfile") || die "Could not open $classfile for reading\n";
while(<LIST>){
        chomp;
        $modelName = $_;
        push (@modelList,$modelName);
        $Nclass = $Nclass + 1;
}
close(LIST);


my $Nclass_birds = $Nclass - 1; # pause is not a valid class

my $Best_1 = 0;
my $Best_2 = 0;
my $Best_3 = 0;

for ($set = 1; $set <= 5; $set++){ #all five sets

my $recfile = "Results_hmm/test$set\_$mix\_pm\_$pm.rec";

open (REC,"$recfile") || die "Could not open $recfile";

my $line1 = <REC>; # first line, MLF line
my $linelen;
my $seglen;
my @sent = ();
my $start;
my $end;
my $label;
my $counter2 = -1;
my $counter1 = -1;
#$counter1 = -1;
```

```
my $tars2;
my $sind2;
my $sind_old2;
my $title;
my @trec;
my $scale = (20 * (10**4));
my @frame_distr;
my @file_class;
my @num_hit = @file_class;
my $length_trec = 0;
my $jnum;

while ($line1 = <REC>){
        chomp($line1);
        $linelen = length($line1);
        if ($line1 =~ /\.rec\"/)
        {
                $sind_old2 = 0;
                @trec = ();
                $length_trec = 0;
                $title = $line1;
                $counter1 = $counter1 + 1;   # count number of files
                for (my $i = 0; $i < $Nclass; $i++){
                        $frame_distr[$i][$counter1] = 0;
                        my $str = $title;
                        my $substr = $modelList[$i];
                        if (index($str, $substr) != -1) {
                                $counter2 = $counter2 + 1;
                                $file_class[$counter1] = $i;
                                $num_hit[$counter1] += 1;
                        }
                }
        }
        elsif ($linelen > 4)
        {
                chomp($line1);
                ($start, $end, $label, ) = split(/ /,$line1);  #
                    extracting info from rec-file
                for (my $i = 0; $i < $Nclass; $i++){
                        if ($label eq $modelList[$i]){
                                $tars2 = $i;  #comparing against list of
                                    birds
                        }
                }
                $sind2 = floor($end / $scale);  # end of file
                $seglen = $sind2 - $sind_old2; #length of each recognized
                    bird
                #$sind_old2 = $sind2;
                my @ones = (1) x $seglen;
                foreach (@ones){
                        $_ = $_ * $tars2;
                        $length_trec = $length_trec + 1;
                }
                push (@trec, @ones);
                for (my $j = 0; $j < $Nclass_birds; $j++){
                        my $counter_trec = 0;
                        for (my $i = $sind_old2; $i < $length_trec; $i++){
```

```
                                if ($trec[$i] == $j){
                                        $counter_trec = $counter_trec + 1;
                                }
                        }
                        $frame_distr[$j][$counter1] += $counter_trec;
                }
                $sind_old2 = $sind2;
        }
}

close (REC);

#----------------------------------------
#-------Normalized Histogram-------------
#----------------------------------------

my @frame_distr_sum;
my @frame_distr_norm;


for (my $i = 0; $i <= $counter1; $i++){
        $frame_distr_sum[$i] = 0;
        for (my $j = 0; $j < $Nclass_birds; $j++){
                $frame_distr_sum[$i] = $frame_distr_sum[$i] + $frame_distr
                    [$j][$i];
        }
}

my $sum;

for (my $i = 0; $i <= $counter1; $i++){
        $sum = 0;
        for (my $j = 0; $j < $Nclass_birds; $j++){
                $frame_distr_norm[$j][$i] = $frame_distr[$j][$i] /
                    $frame_distr_sum[$i];
                $sum = $sum + $frame_distr_norm[$j][$i];
        }
}

##########################################
#----------------------------------------
#------------Classification--------------
#----------------------------------------
##########################################

#---------------------------------------------------------
#----------------------Importing relevant data------------
#---------------------------------------------------------

for (my $i = 0; $i <= $counter1; $i++){
        $frame_distr_norm[$Nclass_birds][$i] = 1;
}

my @Ttest = ();

for (my $i = 0; $i <= $counter1; $i++){
        for (my $j = 0; $j < $Nclass_birds; $j++){
```

```
                $Ttest[$j][$i] = 0;
                if ($file_class[$i] == $j){
                        $Ttest[$j][$i] = $Ttest[$j][$i] + 1;
                }
        }
}

#---------------------------------------
#----Subroutine matrix multiplication----
#---------------------------------------

sub matrix_multiply {
    my ($r_mat1,$r_mat2)=@_;
    my ($r_product);
    my ($r1,$c1)=matrix_count_rows_cols($r_mat1);
    my ($r2,$c2)=matrix_count_rows_cols($r_mat2);


    die "matrix 1 has $c1 columns and matrix 2 has $r2 rows>"
        . " Cannot multiply\n" unless ($c1==$r2);
    for (my $i=0;$i<$r1;$i++) {
        for (my $j=0;$j<$c2;$j++) {
            my $sum=0;
            for (my $k=0;$k<$c1;$k++) {
                $sum+=$r_mat1->[$i][$k]*$r_mat2->[$k][$j];
            }
            $r_product->[$i][$j]=$sum;
        }
    }
    $r_product;
}

sub matrix_count_rows_cols {
    my ($r_mat)=@_;
    my $num_rows=@$r_mat;
    my $num_cols=@{$r_mat->[0]};
    ($num_rows,$num_cols);
}

#---------------------------------------
#----------Defining w matrix-----------
#---------------------------------------

my @w;
my $line;
my $i = 0;
my $textfile = "classificationMatrix$set.txt";

open (TEXT,"$textfile") || die "Could not open $textfile";

while ($line = <TEXT>){
        chomp($line);

        (my $class1, my $class2, my $class3, my $class4, my $class5, my
            $class6, my $class7, my $class8, my $class9, my$class10, my
            $class11, my $class12, my $class13, my $class14, my $class15,
            my $class16, my $class17, my $class18, my $class19, my
```

```
            $class20, my $class21, my $class22 ) = split(/ /,$line);


        #print "\n";

                $w[$i][0] = $class1;
                $w[$i][1] = $class2;
                $w[$i][2] = $class3;
                $w[$i][3] = $class4;
                $w[$i][4] = $class5;
                $w[$i][5] = $class6;
                $w[$i][6] = $class7;
                $w[$i][7] = $class8;
                $w[$i][8] = $class9;
                $w[$i][9] = $class10;
                $w[$i][10] = $class11;
                $w[$i][11] = $class12;
                $w[$i][12] = $class13;
                $w[$i][13] = $class14;
                $w[$i][14] = $class15;
                $w[$i][15] = $class16;
                $w[$i][16] = $class17;
                $w[$i][17] = $class18;
                $w[$i][18] = $class19;
                $w[$i][19] = $class20;
                $w[$i][20] = $class21;
                $w[$i][21] = $class22;

                $i = $i +1;

}

close(TEXT);


#-----------------------------------------------
#-----Initializing misclassification matrix------
#-----------------------------------------------

my @ctest1;
my @ctest2;
my @ctest3;

for (my $i = 0; $i < $Nclass_birds; $i++){
        for (my $j = 0; $j < $Nclass_birds; $j++){
                $ctest1[$i][$j] = 0;
                $ctest2[$i][$j] = 0;
                $ctest3[$i][$j] = 0;
        }
}


#-----------------------------------------------
#---------Forward calculations------------------
#-----------------------------------------------

my @zt;
```

```
my @bt;

@zt = @{ matrix_multiply(\@w, \@frame_distr_norm) };

for (my $i = 0; $i <= $counter1; $i++){
        for (my $j = 0; $j < $Nclass_birds; $j++){
                $bt[$j][$i] = exp($zt[$j][$i]);
        }
}

my @bt_sum;

for (my $i = 0; $i <= $counter1; $i++){
        $bt_sum[$i] = 0;
        for (my $j = 0; $j < $Nclass_birds; $j++){
                $bt_sum[$i] = $bt_sum[$i] + $bt[$j][$i];
        }
}

my @sum;
my @bt_norm;

for (my $i = 0; $i <= $counter1; $i++){
        for (my $j = 0; $j < $Nclass_birds; $j++){
                $bt_norm[$j][$i] = $bt[$j][$i] / $bt_sum[$i];
        }
}


#-------------------------------------------------
#------Calculating misclassification matrices------
#-------------------------------------------------

#####Sorting bd matrix
my @bt_norm_sorted;
my @bt_norm_sorted_temp;


for (my $i = 0; $i <= $counter1; $i++){
        @bt_norm_sorted_temp = sort {$b->[$i] <=> $a->[$i]} @bt_norm;
        for (my $j = 0; $j < $Nclass_birds; $j++){
                $bt_norm_sorted[$j][$i] = $bt_norm_sorted_temp[$j][$i];
        }
}

######Finding indexes of the sorted matrix
my @file_rec;

for (my $i = 0; $i <= $counter1; $i++){
        $file_rec[0][$i] = -1;
        $file_rec[1][$i] = -1;
        $file_rec[2][$i] = -1;
        for (my $j = 0; $j < $Nclass_birds; $j++){
                if ($bt_norm_sorted[0][$i] == $bt_norm[$j][$i] &&
                    $bt_norm_sorted[0][$i] > 0.01){   #If the score is low
                    , do not show all 3best
                        $file_rec[0][$i] = $j;
```

```
                }
                if ($bt_norm_sorted[1][$i] == $bt_norm[$j][$i] &&
                    $bt_norm_sorted[1][$i] > 0.01){
                        $file_rec[1][$i] = $j;
                }
                if ($bt_norm_sorted[2][$i] == $bt_norm[$j][$i] &&
                    $bt_norm_sorted[2][$i] > 0.01){
                        $file_rec[2][$i] = $j;
                }
        }
}

#######Display wich bird is recognized
my @recognized;

for (my $i = 0; $i <= $counter1; $i++){
        for (my $j = 0; $j < $Nclass_birds; $j++){
                for (my $k = 0; $k < 3; $k++){
                        $recognized[$k][$i] = $file_rec[$k][$i];
                        if ($recognized[$k][$i] == -1){
                                $recognized[$k][$i] = "No bird recognized
                                    ";
                        }
                        else {
                                $recognized[$k][$i] = "$modelList[
                                    $recognized[$k][$i]]";
                        }
                }
        }
}

###########Calculating misclassification matices
my @classtest;
my @cdev;

for (my $i = 0; $i <= $counter1; $i++){
        $classtest[0][$i] = $file_rec[0][$i];
        $classtest[1][$i] = $file_rec[1][$i];
        $classtest[2][$i] = $file_rec[2][$i];
}

my $count_file_class1 = 0;
my $count_file_class2 = 0;
my $count_file_class3 = 0;


##Mis matrix

for (my $i = 0; $i <= $counter1; $i++){
        $count_file_class1 = 0;
        for (my $j = 0; $j < $Nclass_birds; $j++){
                for (my $k = 0; $k < $Nclass_birds; $k++){
                        if ($file_class[$i] == $j && $file_rec[0][$i] ==
                            $k){
                                $count_file_class1 +=1;
                                $ctest1[$j][$k] = $ctest1[$j][$k] +
                                    $count_file_class1;
```

```
                        }
                }
        }
}


####Number of correct classifications

$count_file_class1 = 0;
$count_file_class2 = 0;
$count_file_class3 = 0;

for (my $i = 0; $i <= $counter1; $i++){
        if ($file_rec[0][$i] == $file_class[$i]){
                $count_file_class1 +=1;
        }
        if ($file_rec[1][$i] == $file_class[$i]){
                $count_file_class2 +=1;
        }
        if ($file_rec[2][$i] == $file_class[$i]){
                $count_file_class3 +=1;
        }
}

my $numcorr1d = $count_file_class1;
my $numcorr2d = $count_file_class1 + $count_file_class2;
my $numcorr3d = $count_file_class1 + $count_file_class2 +
    $count_file_class3;

my @perr1;
my @perr2;
my @perr3;
my @dzd;

for (my $i = 0; $i < 3; $i++){
        if ($count_file_class1 != 0){
                my $score1 = 100 * (1 - ($numcorr1d / ($counter1 + 1)));
                $perr1[$set] = "$score1";
        }
        if ($count_file_class2 != 0){
                my $score2 = 100 * (1 - ($numcorr2d / ($counter1 + 1)));
                $perr2[$set] = "$score2";
        }
        if ($count_file_class3 != 0){
                my $score3 = 100 * (1 - ($numcorr3d / ($counter1 + 1)));
                $perr3[$set] = "$score3";
        }

}

print "\n";
print "\n";

print "$perr1[$set], $perr2[$set], $perr3[$set]\n \n";

$Best_1 = $Best_1 + $perr1[$set]/5;
$Best_2 = $Best_2 + $perr2[$set]/5;
```

```
$Best_3 = $Best_3 + $perr3[$set]/5;

print "\n";
print "\n";


open(my $fh, '>', 'adap_recognized.list');
for (my $i = 0; $i <= $counter1; $i++){
        print $fh $recognized[0][$i];
        print $fh "\n";
}

}

print "Average error rate: \n";
print "$Best_1    $Best_2     $Best_3 \n";  #average  error
```

C: TrainingClassifier.pl

```
#!/usr/bin/perl

use strict;
use warnings;
use POSIX;

##################################################
#-------------------------------------------------
#-------------Histogram program-------------------
#-------------------------------------------------
##################################################

my $pm = 80;
my $mix = 32;
my $set = 2;
my $type = 'train';
my $modelName;
my @modelList = ();
my $Nclass = 0;

my $classfile = 'class_pause_ny.list'; # list of all legal classes

open(LIST,"$classfile") || die "Could not open $classfile for reading\n";
while(<LIST>){
        chomp;
        $modelName = $_;
        push (@modelList,$modelName);
        $Nclass = $Nclass + 1;
}
close(LIST);


my $Nclass_birds = $Nclass - 1; # pause is not a valid class

my $Best_1 = 0;
my $Best_2 = 0;
my $Best_3 = 0;

my @perr1;
my @perr2;
my @perr3;

for ($set = 1; $set <= 5; $set++){ #all five sets

my $recfile = "Results_hmm/train$set\_$mix\_pm\_$pm.rec";

my $linelen;
my $seglen;
my @sent = ();
my $start;
my $end;
my $label;
my $counter2 = -1;
my $counter1 = -1;
```

```
my $tars2;
my $sind2;
my $sind_old2;
my $title;
my @trec;
my $scale = (20 * (10**4));
my @frame_distr;
my @file_class;
my @num_hit = @file_class;
my $length_trec = 0;
my $jnum;

#-----------------------------------------------
#----------------Reading MLF file----------------
#-----------------------------------------------


open (REC,"$recfile") || die "Could not open $recfile";

my $line1 = <REC>; # first line, MLF line

while ($line1 = <REC>){
        chomp($line1);
        $linelen = length($line1);
        if ($line1 =~ /\.rec\"/)      #New file
        {
                $sind_old2 = 0;
                @trec = ();
                $length_trec = 0;
                $title = $line1;
                $counter1 = $counter1 + 1;   # count number of files
                for (my $i = 0; $i < $Nclass; $i++){
                        $frame_distr[$i][$counter1] = 0;
                        my $str = $title;
                        my $substr = $modelList[$i];
                        if (index($str, $substr) != -1) {
                                $counter2 = $counter2 + 1;
                                $file_class[$counter1] = $i;
                                $num_hit[$counter1] += 1;
                        }
                }
        }
        elsif ($linelen > 4)
        {
                chomp($line1);
                ($start, $end, $label, ) = split(/ /,$line1);  #
                    extracting info from rec-file
                for (my $i = 0; $i < $Nclass; $i++){
                        if ($label eq $modelList[$i]){
                                $tars2 = $i;  #comparing against list of
                                    birds
                        }
                }
                $sind2 = floor($end / $scale);  # end of file
                $seglen = $sind2 - $sind_old2; #length of each recognized
                    bird
                #$sind_old2 = $sind2;
```

```
                my @ones = (1) x $seglen;
                foreach (@ones){
                        $_ = $_ * $tars2;
                        $length_trec = $length_trec + 1;
                }
                push (@trec, @ones);
                for (my $j = 0; $j < $Nclass_birds; $j++){
                        my $counter_trec = 0;
                        for (my $i = $sind_old2; $i < $length_trec; $i++){
                                if ($trec[$i] == $j){
                                        $counter_trec = $counter_trec + 1;
                                }
                        }
                        $frame_distr[$j][$counter1] += $counter_trec;
                }
                $sind_old2 = $sind2;
        }
}

close (REC);

#-----------------------------------------------------------------------
#----------------------Normalized Histogram-----------------------------
#-----------------------------------------------------------------------

my @frame_distr_sum;
my @frame_distr_norm;


for (my $i = 0; $i <= $counter1; $i++){
        $frame_distr_sum[$i] = 0;
        for (my $j = 0; $j < $Nclass_birds; $j++){
                $frame_distr_sum[$i] = $frame_distr_sum[$i] + $frame_distr
                    [$j][$i];
        }
}

my $sum;

for (my $i = 0; $i <= $counter1; $i++){
        $sum = 0;
        for (my $j = 0; $j < $Nclass_birds; $j++){
                $frame_distr_norm[$j][$i] = $frame_distr[$j][$i] /
                    $frame_distr_sum[$i];
                $sum = $sum + $frame_distr_norm[$j][$i];
        }
}

##################################################
#------------------------------------------------
#----------------Classification------------------
#------------------------------------------------
##################################################

#-------------------------------------------------------------
#---------------Importing relevant data-----------------------
#-------------------------------------------------------------
```

```
for (my $i = 0; $i <= $counter1; $i++){
        $frame_distr_norm[$Nclass_birds][$i] = 1;
}

my @Ttrain = ();

for (my $i = 0; $i <= $counter1; $i++){
        for (my $j = 0; $j < $Nclass_birds; $j++){
                $Ttrain[$j][$i] = 0;
                if ($file_class[$i] == $j){
                        $Ttrain[$j][$i] = $Ttrain[$j][$i] + 1;
                }
        }
}

#-----------------------------------------------------------------------
#-----------------Subroutine matrix multiplication--------------------
#-----------------------------------------------------------------------

sub matrix_multiply {
    my ($r_mat1,$r_mat2)=@_;
    my ($r_product);
    my ($r1,$c1)=matrix_count_rows_cols($r_mat1);
    my ($r2,$c2)=matrix_count_rows_cols($r_mat2);

    die "matrix 1 has $c1 columns and matrix 2 has $r2 rows>"
        . " Cannot multiply\n" unless ($c1==$r2);
    for (my $i=0;$i<$r1;$i++) {
        for (my $j=0;$j<$c2;$j++) {
            my $sum=0;
            for (my $k=0;$k<$c1;$k++) {
                $sum+=$r_mat1->[$i][$k]*$r_mat2->[$k][$j];
            }
            $r_product->[$i][$j]=$sum;
        }
    }
    $r_product;
}

sub matrix_count_rows_cols {
    my ($r_mat)=@_;
    my $num_rows=@$r_mat;
    my $num_cols=@{$r_mat->[0]};
    ($num_rows,$num_cols);
}

#-------------------------------------------------
#-----------Subroutine transpose matrix-----------
#-------------------------------------------------

sub pivot {
    my @src = @_;

    my $max_col = 0;
    $max_col < $#$_ and $max_col = $#$_ for @src;
```

```
    my @dest;
    for my $col (0..$max_col) {
        my @new_row;
        for my $row (0..$#src) {
            push @new_row, $src[$row][$col] // '';
        }
        push @dest, \@new_row;
    }

    return @dest;
}

#----------------------------------------
#----------Subroutine sign----------------
#----------------------------------------

sub sign {
    if ($_[0]*$_[1] > 0){
        return 1;
    }
    elsif ($_[0]*$_[1] < 0){
        return -1;
    }
    else{
        return 0;
    }
};

#--------------------------------
#------Training starts-----------
#--------------------------------

my $totiter = 1000;

#---------------------------------------
#----------Defining w matrix-----------
#---------------------------------------

my @w;
my $line;
my $i = 0;
my $textfile = "w.txt";  #Initially using the same w matrix

open (TEXT,"$textfile") || die "Could not open $textfile";

while ($line = <TEXT>){
        chomp($line);

        (my $class1, my $class2, my $class3, my $class4, my $class5, my
            $class6, my $class7, my $class8, my $class9, my$class10, my
            $class11, my $class12, my $class13, my $class14, my $class15,
            my $class16, my $class17, my $class18, my $class19, my
            $class20, my $class21, my $class22 ) = split(/ /,$line);

                $w[$i][0] = $class1;
                $w[$i][1] = $class2;
                $w[$i][2] = $class3;
```

```
                $w[$i][3] = $class4;
                $w[$i][4] = $class5;
                $w[$i][5] = $class6;
                $w[$i][6] = $class7;
                $w[$i][7] = $class8;
                $w[$i][8] = $class9;
                $w[$i][9] = $class10;
                $w[$i][10] = $class11;
                $w[$i][11] = $class12;
                $w[$i][12] = $class13;
                $w[$i][13] = $class14;
                $w[$i][14] = $class15;
                $w[$i][15] = $class16;
                $w[$i][16] = $class17;
                $w[$i][17] = $class18;
                $w[$i][18] = $class19;
                $w[$i][19] = $class20;
                $w[$i][20] = $class21;
                $w[$i][21] = $class22;

                $i = $i +1;

}

close(TEXT);

#-------------------------------------------------
#-------Defining gradients and stepfactors--------
#-------------------------------------------------

#my @w;
my @wo;
my @dw;
my @dwo;
my @sw;
my $sinit = 10;

for (my $i = 0; $i <= $Nclass_birds; $i++){
        for (my $j = 0; $j < $Nclass_birds; $j++){
                #$w[$j][$i] = 0.01 * (rand() - 0.5);
                $dw[$j][$i] = 0;
                $dwo[$j][$i] = 0;
                $wo[$j][$i] = 0;
                $sw[$j][$i] = 1 * $sinit;
        }
}


#-----------------------------------------
#--------Defining various variables-------
#-----------------------------------------

my $ud = 0.15;
my $alpha = 0.5 / ($counter1 + 1);
my $Ed_old = -1000;

#-----------------------------------------
```

```
#-------------Totiter itrations--------------
#--------------------------------------------

my $iter = 0;
my $rd_old = -1;
my @cdev1;
my @cdev2;
my @cdev3;
my @zd;
my @bd;
my @bd_sum;

while ($iter <= $totiter){

        #########Initializing misclassification matrix
        for (my $i = 0; $i < $Nclass_birds; $i++){
                for (my $j = 0; $j < $Nclass_birds; $j++){
                        $cdev1[$i][$j] = 0;
                        $cdev2[$i][$j] = 0;
                        $cdev3[$i][$j] = 0;
                }
        }

        #####reseting derivatives
        for (my $i = 0; $i <= $Nclass_birds; $i++){
                for (my $j = 0; $j < $Nclass_birds; $j++){
                        $dw[$j][$i] = 0;
                        $dwo[$j][$i] = 0;
                        #$wo[$j][$i] = 0;
                }
        }

        $iter = $iter + 1; #increasing counter

        ########Forward calculations
        @zd = @{ matrix_multiply(\@w, \@frame_distr_norm) };

        my @sum;
        my @bd_norm;

        for (my $i = 0; $i <= $counter1; $i++){
                $bd_sum[$i] = 0;
                for (my $j = 0; $j < $Nclass_birds; $j++){
                        $bd[$j][$i] = exp($zd[$j][$i]);
                        $bd_sum[$i] = $bd_sum[$i] + $bd[$j][$i];
                }
        }

        for (my $i = 0; $i <= $counter1; $i++){
                for (my $j = 0; $j < $Nclass_birds; $j++){
                        $bd_norm[$j][$i] = $bd[$j][$i] / $bd_sum[$i];
                }
        }

        #####Training criteria
        my @tb;
        my $Ed = 0;
```

```
for (my $i = 0; $i <= $counter1; $i++){
        for (my $j = 0; $j < $Nclass_birds; $j++){
                $tb[$j][$i] = $Ttrain[$j][$i] * log($bd_norm[$j][
                    $i]);
                $Ed = $Ed + $tb[$j][$i];
        }
}

$Ed = (100 * $Ed) / ($counter1 + 1);
#print "$Ed, ";

#####Sorting bd matrix
my @bd_norm_sorted;
my @bd_norm_sorted_temp;

for (my $i = 0; $i <= $counter1; $i++){
        @bd_norm_sorted_temp = sort {$b->[$i] <=> $a->[$i]}
            @bd_norm;
        for (my $j = 0; $j < $Nclass_birds; $j++){
                $bd_norm_sorted[$j][$i] = $bd_norm_sorted_temp[$j
                    ][$i];
        }
}

######Finding indexes of the sorted matrix
my @file_rec;

for (my $i = 0; $i <= $counter1; $i++){
        $file_rec[0][$i] = 0;
        $file_rec[1][$i] = 0;
        $file_rec[2][$i] = 0;
        for (my $j = 0; $j < $Nclass_birds; $j++){
                if ($bd_norm_sorted[0][$i] == $bd_norm[$j][$i]){
                        $file_rec[0][$i] = $j;
                }
                if ($bd_norm_sorted[1][$i] == $bd_norm[$j][$i]){
                        $file_rec[1][$i] = $j;
                }
                if ($bd_norm_sorted[2][$i] == $bd_norm[$j][$i]){
                        $file_rec[2][$i] = $j;
                }
        }
}



###########Calculating misclassification matrices

my @classdev;
my @cdev;

for (my $i = 0; $i <= $counter1; $i++){
        $classdev[0][$i] = $file_rec[0][$i];
        $classdev[1][$i] = $file_rec[1][$i];
        $classdev[2][$i] = $file_rec[2][$i];
}
```

```
my $count_file_class1 = 0;
my $count_file_class2 = 0;
my $count_file_class3 = 0;

##Initializing
for (my $j = 0; $j < $Nclass_birds; $j++){
        for (my $k = 0; $k < $Nclass_birds; $k++){
                $cdev[$j][$k] = 0;
        }
}

##Mis matrix

for (my $i = 0; $i <= $counter1; $i++){
        $count_file_class1 = 0;
        for (my $j = 0; $j < $Nclass_birds; $j++){
                for (my $k = 0; $k < $Nclass_birds; $k++){
                        if ($file_class[$i] == $j && $file_rec[0][
                            $i] == $k){
                                $count_file_class1 +=1;
                                $cdev[$j][$k] = $cdev[$j][$k] +
                                    $count_file_class1;
                                #print $cdev[$j][$k];
                        }
                        #print "\n";
                }
        }
}

####Number of correct classifications 1best, 2best and 3best

$count_file_class1 = 0;
$count_file_class2 = 0;
$count_file_class3 = 0;

for (my $i = 0; $i <= $counter1; $i++){
        if ($file_rec[0][$i] == $file_class[$i]){
                $count_file_class1 +=1;
        }
        if ($file_rec[1][$i] == $file_class[$i]){
                $count_file_class2 +=1;
        }
        if ($file_rec[2][$i] == $file_class[$i]){
                $count_file_class3 +=1;
        }
}

my $numcorr1d = $count_file_class1;
my $numcorr2d = $count_file_class1 + $count_file_class2;
my $numcorr3d = $count_file_class1 + $count_file_class2 +
    $count_file_class3;

my @dzd;

$perr1[$set] = 100 * (1 - ($numcorr1d / ($counter1 + 1)));
$perr2[$set] = 100 * (1 - ($numcorr2d / ($counter1 + 1)));
```

```
        $perr3[$set] = 100 * (1 - ($numcorr3d / ($counter1 + 1)));

        print "$perr1[$set], $perr2[$set], $perr3[$set]\n";

        if ($Ed_old == $Ed_old){

                ##Hide the old parameters
                @wo = @w;
                @dwo = @dw;
                $Ed_old = $Ed;

                ##Bakward loop of error and calculation of delta-
                    parameters and updating step factor
                for (my $i = 0; $i <= $counter1; $i++){
                        for (my $j = 0; $j < $Nclass_birds; $j++){
                                $dzd[$j][$i] = -($bd_norm[$j][$i] -
                                    $Ttrain[$j][$i]); ###skal kanskje
                                    bruke bd_sorted
                                $dzd[$j][$i] = $dzd[$j][$i] * $alpha;
                        }
                }

                my @frame_distr_norm_trans = pivot(@frame_distr_norm);

                @dw = @{ matrix_multiply(\@dzd, \@frame_distr_norm_trans)
                    };

                for (my $i = 0; $i <= $Nclass_birds; $i++){
                        for (my $j = 0; $j < $Nclass_birds; $j++){
                                $dw[$j][$i] = $dw[$j][$i] * $sw[$j][$i];
                                $sw[$j][$i] = $sw[$j][$i] + ($ud * ($sw[$j
                                    ][$i] * sign($dwo[$j][$i], $dw[$j][$i
                                    ])));
                                $w[$j][$i] = (0.999 * $w[$j][$i]) + $dw[$j
                                    ][$i];
                        }
                }
        }
}


$Best_1 = $Best_1 + $perr1[$set]/5;
$Best_2 = $Best_2 + $perr2[$set]/5;
$Best_3 = $Best_3 + $perr3[$set]/5;

open(my $fh, '>', "classificationMatrix$set.txt");
for (my $i = 0; $i < $Nclass_birds; $i++){
        for (my $j = 0; $j <= $Nclass_birds; $j++){
                print $fh $w[$i][$j];
                print $fh " ";
        }
        print $fh "\n";
}
close $fh;
print "done\n";

}
```

```
print "$Best_1     $Best_2     $Best_3 \n";
```