# Modeling and Confidence in a System for Automatic Classification of Birdsong

Fredrik Fløttum Aagaard

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 automatic classification of birdsong. The system involves several steps. In the first step the incoming acoustic signal gets decoded which gives a sequence of segments. These segments may belong all lawful classes. The system then finds the sequence of segments that gives the highest score (likelihood). In the next step the found sequence of segments gets analyzed with respect to which classes that are most frequently occurring. This information is then stored in a histogram of the various classes. In the last step the histogram gets normalized in order to eliminate the influence of the recordings total duration. The normalized histogram is then applied to a static post classifier which outputs the probability of each class. Such a N-best ranking of the probabilities has been proven to give better performance compared to the performance of using the histogram directly for decision.

In the current system the classes are modeled as a weighted sum of multidimensional normal distributions (GMM). The first part of this thesis is about investigating the performance of the system when GMM gets replaced with so called hidden Markov models (HMM). One should investigate the performance of the system when using different amounts of coefficients extracted out of the acoustic signal used for training and recognition and different lengths of the frames and corresponding window lengths used for feature extraction done by short-time stationary frequency analysis. One should also investigate the influence on the system performance when adjusting the penalty value, which is a system parameter set in the decoder which allows to add a penalty for jumping from one class to another in the recognition.

The current system assumes that the incoming acoustic signal comes from one of the classes that the system is able to recognize. This implies that signals from other (”unknown”) classes gets classified as one of the ”legal” classes, i.e. misclassified. The second part of the thesis is about implementing a so called ”Out-of-class” detector, and investigate whether such a detector deals with the unknown birdsong in a good way or not. A separate model for all the unknown classes is difficult to design because of the expected complexity and the lack of acoustic recordings. A technique based on thresholds for score (confidence) is therefore more realistic.

Assignment given: 16. January 2015
Supervisor: Magne Hallstein Johnsen, IET.

# Summary

It turns out that using a two-state-HMM model structure with appurtenant GMM-based state distributions improves the system performance compared to the use of just GMMs as model structure for each bird specie. Hence, it is reasonable to say that birdsong contains temporary information which HMMs deals with in a better way than GMMs. On average, 82.31% of the birdsong in the test sets gets classified correct using HMMs. The use of GMMs achieves an average correctness of 80.1% applying the same test sets. However, further improvement of the resulting system performance are dependent on a bigger database such that the models can be trained with more example data which cover all sorts of recording surroundings and cover more of the variances of birdsong that exists within a bird specie.

The penalty value set in the decoder which adds a ”penalty” when jumping from one bird class to another should ideally be set in a way such that the number of insertions (more segments) and deletions (less segments) of the recognition gets equal. However, it turns out that these insertions and deletions mainly concerns pause segments (silence and other sounds), and therefore does not affect the birdsong classification results noteworthy. One can also say that it is more important to not lose any information by deletions compared to the disorientation we get from added segments by insertions.

When it comes to frame length and corresponding window length used in the short-time stationary frequency analysis, it turns out that the frame length should at least be of 20 ms. The results where quite the same with the use of a frame length of 20 ms and 25 ms with corresponding window lengths of 30 ms and 40 ms, respectively. However, the use of such ”big” frame lengths, leads to less example data per model distribution which again leads to weaker models. Hence, ”big” frame lengths requires more training data. The different amounts of coefficients extracted from the acoustic signal does not vary the resulting system performance noteworthy. 15 coefficients turned out to give the best performance. It is reasonable to think that less than 12 coefficients are insufficient and that more than 19 coefficients are unnecessary.

The different bird species give different contributions to the total error. 9 out of 21 bird species gets recognized correct 100% of the time, while some few bird species gets recognized correct only 50% of the time or lower. The reason for this could be that birdsong from some bird species are very similar, i.e. have similar frequency content, making it difficult to distinguish between them. Insufficient amount or poor quality of the example data for these bird species could also be one possible reason. It turns out that the results achieved from the five different test sets used for testing the system vary a lot. One of the test sets achieves an error of 12.4%, while another test set achieves an error of 23.8%. Hence, it is reasonable to believe that the test data applied in this thesis are not fully representative to the input data applied the system later on, and conclusions of the system performance based on this test data should not be trusted blindly. This gap between the achieved results from the different test sets could also imply that the models are trained with an inadequate amount of example data, i.e. the database used in this thesis is too small. The gap between the performance of the system applying the training data and the test data implies that the system do not generalize well, this supports the thought of a too small database.

The out-of-class detector implemented in order to deal with unknown birdsong turns out to be a good idea. Setting thresholds for the log likelihood score for each recognized segment corresponding to the different bird classes from the test set makes it possible to classify 34% of the unknown birdsong as unknown. However, such an out-of-class detector requires a bigger database than we got today. In this thesis, the thresholds for the log likelihood scores are set by investigating the scores achieved from the training set and the test set. This is not optimal. With a bigger database, an untouched data set could be spared for finding these thresholds. The bigger database, the more likely it is that the thresholds get set from a data set that is representative to later input data. With a bigger database the out-of-class detector will work in a better way which also makes the total classification system better. Alternatively, a separate model for all these unknown bird classes could be designed, but this model is difficult to design because of the expected complexity and the lack of acoustic recordings, both from known and unknown bird species.

This thesis also discovers that setting the thresholds for use in the out-of-class detector is difficult. It is a compromise between allowing false accepts and false rejects. However, with the database used in this thesis it turns out that the system are very sensitive to false rejects. This is due to the way the out-of-class detector are being tested in this thesis, where only a small amount of the total birdsong files in the test set belongs to unknown bird classes. Hence, the thresholds must be set such that no false rejects are allowed while we are getting rid of as many false accepts as possible at the same time. A system that knows few bird species out of the total amount of bird species that exists in the nature are likely to face a lot of unknown birdsong. It is therefore important to set the thresholds high enough in order to avoid a lot of false accepts in this case. On the other hand, a system that knows the most of the existing bird species that exists are not very likely to face unknown birdsong. Hence, it is important that the thresholds are set low enough such that false rejects are avoided in this case.

**Sammendrag på norsk:**
Det viser seg at bruken av HMM med to tilstander hvor hver tilstand inneholder en GMM-basert tilstandsfordeling forbedrer systemets ytelse sammenlignet med bruken av GMM som modellstruktur for hver fugleklasse. Det er derfor rimelig å si at fuglesang inneholder temporær informasjon som HMM håndterer på en bedre måte enn GMM. 82.31% av fuglesangen i testsettene blir korrekt klassifisert ved bruk av HMM. Ved bruk av GMM blir 80.10% av den samme fuglesangen korrekt klassifisert. Uansett, videre forbedring av den resulterende system ytelsen er avhengig av en større database slik at modellene kan trenes med mer eksempel data som dekker alle slags opptaksomgivelser og som dekker mer av de forskjellige variantene av fuglesang som eksisterer innenfor en fugleart.

Verdien som legger til en straff når man hopper fra en fugleklasse til en annen i gjenkjenningsprosessen skal ideelt settes slik at antall innsettinger (flere segmenter) og slettinger (færre segmenter) i gjenkjenningen blir omtrent lik. Uansett, det viser seg at disse innsettingene og slettingene i hovedsak angår pause segmenter (stillhet og andre lyder), og påvirker derfor ikke klassifiserings resultatene bemerkelsesverdig. Man kan også si at det er viktigere å ikke miste noe informasjon på grunn av slettinger sammenlignet med å unngå forvirringen som oppstår fra innsettingene.

Når det kommer til rammelengder og tilhørende vinduslengder for å gjøre kort-tids stasjonær frekvensanalyse, så viser det seg at rammelengden bør være minst 20 ms. Resultatene ved bruk av rammelengder på 20 ms og 25 ms med respektive tilhørende vinduslenger på 30 ms og 40 ms var omtrentlig de samme. Uansett, ved bruk av så store rammelengder får vi mindre eksempeldata per modellfordeling som kan føre til dårligere modeller. Derfor kan man si at større rammelengder krever mer treningsdata. Det forskjellige antallet koeffisienter utvunnet fra det akustiske signalet påvirker ikke den resulterende systemytelsen i stor grad. 15 koeffisienter viser seg å gi de beste resultatene. Det er rimelig å tenke seg at mindre enn 12 koeffisienter er for lite og at mer enn 19 koeffisienter er unødvendig.

De forskjellige fugleklassene bidrar i forskjellig grad til den totale feilraten. 9 av 21 fugleklasser blir gjenkjent 100% korrekt av tilfellene, mens noen få fugleklasser blir gjenkjent korrekt bare 50% eller lavere av tilfellene. Grunnen til dette kan være at fuglesang fra noen fuglearter ligner

veldig, altså at de har lignende frekvensinnhold. Dette gjør det vanskelig å skille mellom de. For lite treningsdata eller treningsdata av dårlig kvalitet for disse fugleklassene kan også være grunn til dette. Det viser seg at resultatene oppnådd fra de fem forskjellige testsettene brukt for testing av systemet varierer mye. En av testsettene oppnår en feilrate på 12.4%, mens et annet testsett bare oppnår en feilrate på 23.8%. Derfor er det rimelig å tenke seg at testdataene brukt i denne oppgaven ikke er veldig representativ i forhold til inngangsdata som blir påtrykt systemet senere, og vi kan derfor ikke stole blindt på de oppnådde resultatene i denne oppgaven. Spriket mellom de oppnådde resultatene fra de forskjellige testsettene tilsier også at modellene er trent med en utilstrekkelig mengde av treningsdata. Altså er databasen i denne oppgaven for liten. Forskjellen på systemytelse når man påtrykker henholdsvis trenings- og testsett forteller oss at systemet generaliserer dårlig. Dette støtter også teorien om at databasen er for liten.

”Out-of-class” detektoren som er implementert for å kunne håndtere ukjent fuglesang viser seg å være en god ide. Å sette terskler for log likelihood scoren for hvert gjenkjente segment som tilhører en av de kjente fugleklassene ut i fra et av testsettene gjør det mulig å klassifisere 34% av den ukjente fuglesangen påtrykt systemet som ukjent. Uansett, en slik out-of-class detektor krever en større database enn den vi har i dag. I denne oppgaven er tersklene for log likelihood scoren satt ved å undersøke oppnådde scorer for gjenkjenningen av treningsettet og testsettet. Dette er ikke optimalt. En større database ville gjort det mulig å bruke et urørt datasett for å finne tersklene. Jo større database, jo mer sannsynlig er det at tersklene blir satt ut i fra et datasett som er representativt for senere inngangdata. Med en større database ville out-of-class detektoren fungert bedre, som igjen ville ført til en forbedring av det totale klassifiseringssystemets ytelse. Alternativt så kunne man designet en felles modell for alle de ukjente fugleklassene, men dette lar seg vanskelig gjøre på grunn av den forventede kompleksiteten og mangelen på akustiske opptak, både fra kjente og ukjente fuglearter.

En ting til som er observert ut fra denne oppgaven, er at det å sette terskler for bruk i out-of-class detektoren er vanskelig. Man må gjøre et kompromiss i forhold til å akseptere ”false accepts” og ”false rejects”. Det viser seg at systemet undersøkt i denne oppgaven med gjeldende database er veldig sensitiv for ”false rejects”. Dette er på grunn av måten out-of-class detektoren blir testet i denne oppgaven, hvor kun et fåtall av det totale antallet av påtrykte fuglesangfiler tilhører ukjente fugleklasser. Derfor må tersklene settes slik at man ikke aksepterer noen ”false rejects”, mens vi samtidig kvitter oss med så mange ”false accepts” som mulig. Et system som kjenner til få fuglearter sammenlignet med det totale antallet fuglearter som eksisterer i naturen er sannsynlig å støte på fuglesang fra ukjente fuglearter. Da må tersklene settes høye nok slik at man unngår ”false accepts”. Et system som kjenner til de fleste av det totale antallet fuglearter som eksisterer i naturen er ikke veldig sannsynlig å støte på fuglesang fra ukjente fuglearter. Da må tersklene settes lave nok slik at man unngår ”false rejects”.

# Preface

This report treats my work carried out in my final semester at the Master of Science program in Electronics, at the Department of Electronics and Telecommunications, NTNU. I would especially like to thank associate professor Magne Hallstein Johnsen, my supervisor, for a helpful support during the semester.

Trondheim, June 12, 2015
Fredrik F. Aagaard

# Table of Contents

# List of Tables

# List of Figures

# Abbreviations

| | | |
|---|---|---|
| DCT | = | Discrete cosine transform |
| EM | = | Expectation-Maximization |
| FFT | = | Fast Fourier transform |
| GMM | = | Gaussian Mixture Model |
| HMM | = | Hidden Markov Model |
| HTK | = | Hidden markov model Toolkit |
| IET | = | Department of Electronics and Telecommunication |
| MAP | = | Maximum A Posteriori |
| MFC | = | Mel-frequency cepstrum |
| MFCC | = | Mel-frequency cepstral coefficients |
| MLE | = | Maximum Likelihood Estimation |
| MLF | = | Master Label File |
| ms | = | millisecond |
| NTNU | = | Norwegian University of Science and Technology |
| UNK | = | Unknown |

Chapter 1

# Introduction

This chapter is an introduction to the report. Section 1.1 presents a motivation and the background for the thesis. Section 1.2 describes briefly the work that has been done from the start to the end while section 1.3 gives an outline of the report.

## 1.1 Motivation and background

The students at the Department of Electronics and Telecommunications (IET) at the University of Science and Technology (NTNU) are supposed to write a master thesis in their final semester of their fifth and last year of the master degree program. The goal for this master thesis is to let the students work with a well defined problem inside the area of their main profile, and give them training and experience of such work for the upcoming work career inside the technological industry.

This thesis is given by IET, and is about improving the performance of a system for automatic classification of birdsong. A company called AbleMagic are developing an application for Android and iPhone/iPad that are supposed to let the users investigate birdsong while hiking in the Norwegian nature. They asked IET for help with the signal processing needed for such a classification system, and therefore they (IET) presented a master thesis that concerns this type of problem. It should be mentioned that this thesis is independent from AbleMagic and their application, and is mainly written for IET.

The whole idea behind such an application, which is illustrated by figure 1.1 below, is to be able to answer the question ”which type of bird is singing?”. The user should take a recording of the birdsong of interest with a device and the application should then process this recording and decide which bird it is that sings. In order to obtain an application that will work satisfying, i.e. have an error rate that is low enough to make the users satisfied and happy, one has to study and find a good way of modeling the different bird species. One should also find a good way of dealing with birdsong from bird species that are unknown to the classification system. This is the essence of this thesis.

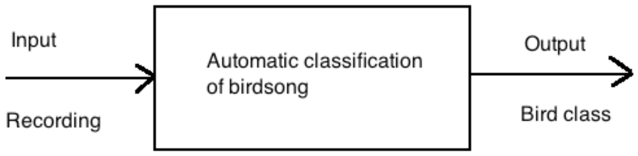


**Figure 1.1:** Idea of application

One can say that automatic classification of birdsong is quite similar to automatic classification of speech, i.e. speaker recognition. One distinguishes between deterministic and physical signals. The former are characterized in that they are described exactly by the use of mathematical formulas. Such deterministic signals does not appear in the real world. It is of course not possible to find a mathematical formula that describes birdsong. One must therefore try to find a mathematical approximation that is ”good enough”. Such an approximation is referred to as a model, both in general and in this thesis.

Most time signals have information in temporary form. For speech it is obvious that the order of sounds (phonemes) and words are essential for the meaning of the speech. Speech is also directive, meaning that if we turn around the time signal the meaning gets lost. It is reasonable to believe that the content in one single stanza of birdsong also have some temporary information while repeated stanzas probably not provide further information. Birdsong does not, as far as we know, contain a grammar similar to the one that exists for speech.

It is developed a variety of model types for physical signals. The models in this thesis are based on the most common strategy, namely that one assumes that the time signal have a stochastic nature. This is because different songs from the same bird individual will have different form on the time signal. This applies to an even greater extent between different individuals from the same bird specie. Hopefully the variation between different bird species are even bigger.

It is common to separate stochastic signals into stationary and non-stationary signals. For non-stationary signals the frequency content changes with time. Most of the physical signals, included birdsong, have logically a non-stationary nature because they contain temporary information. On the other hand, stationary signals have a fixed frequency content, and therefore it exists much better models for stationary signals compared to non-stationary signals. However, one can assume that physical signals are short-time stationary, i.e. stationary over ”short” time. This assumption is of great help in all signal processing of physical signals and such short-time stationary frequency analysis are utilized in this thesis.

One has two different scenarios in speech recognition. One can either recognize single words or whole sentences. The latter problem is much harder because one has to find the number of words, which words it is and sometimes the time limits between words. The former problem is called classification while the latter is called recognition. When it comes to birdsong one can say that this is something between classification and recognition. However, because the need of finding the number of stanzas and the start and stop time for these do not exist for the application of interest, the birdsong recognition can be simplified to classification in this thesis.

The main goal of this thesis is to optimize the system performance in order to achieve a lower total error rate than the current system achieves. The first part is about studying the system performance when modeling each bird class with HMM’s instead of GMM’s and investigating the optimal choice for a) the number of coefficients extracted from the acoustic signal used for both training and decoding, b) the segment lengths and corresponding window lengths used for both training and decoding and c) the penalty value which is a parameter that allows to add a penalty for jumping from one class to another inside the decoder. This penalty value should be adjusted such that the number of deletions and insertions of recognized segments gets pretty equal. The second part concerns the problem with unknown birdsong. The current system assumes that the incoming acoustic signal comes from one of the classes that are known. This implies that signals from other unknown birds gets misclassified, and therefore contributes to a bigger total error rate. A so-called ”out-of-class” detector is implemented and investigated in this thesis for handling unknown birdsong. This out-of-class detector is based on thresholds for the log likelihood scores given for each segment that are being recognized as one of the bird classes.

## 1.2 Work

Both training of bird class models and decoding/recognition of the birdsong of interest is done with The Hidden Markov Model Toolkit (HTK). HTK is a toolkit for building hidden Markov models and was originally developed at the Machine Intelligence Laboratory of the Cambridge University Engineering Department. It provides training tools used to estimate the parameters of a set of HMMs using example audio with corresponding labels and recognition tools for decoding audio. HTK is free to use for a server solution that is intended as a platform for this application. Detailed information of HTK and its functions can be found in [11].

First thing was to get familiar with the current system, which was handed out by my supervisor. Study the scripts for training, recognition and classification, become familiar with the catalogue structure containing birdsong recordings (database), configuration files for the different commands of HTK, different lists and proto files and get to know the HTK commands used for training and recognition. In other words just get a hands-on feeling of the whole system before doing the experiments of interest in this thesis.

The first goal of the thesis is to find a better way of modeling the bird classes in order to achieve better overall results compared to the initial results. The first experiment is to replace the existing GMM (one-state-HMM) structure with a 2-state-HMM structure. HMM is dominating the speech recognition area and similar types of physical signals. HMM is a natural choice of model structure when the physical signals contain temporary information [4]. It is therefore a suitable candidate for birdsong as well, at least if it turns out that birdsong contains different frequency content at stanza level.

Then the penalty value which is a value that can be set inside the decoder for adding a ”penalty” when jumping from one class to another in the recognition is investigated, both for the GMM setting and the HMM setting. A good choice of this penalty value will result in a reasonable amount of detected segments (stanzas + pause), while a poor choice of this value will cause many deletions or many insertions, i.e. too few segments or too many segments compared to the actual number of segments [4].

After this the best setting from the above experiments (HMM vs. GMM with their respective best penalty value) is investigated further. The number of coefficients extracted from the acoustic signals used for both training of bird models and recognition of birdsong are investigated. The segment lengths and corresponding window lengths used in both training and recognition are also investigated in order to achieve better overall results.

In order to achieve the final results of all of these experiments above, two different ways of making the decision are used and compared. One just classifying a file by picking the bird class that got recognized the most of the total duration of the respective recording and one which contains a post processor (further explanation of this in chapter 4.2.4).

Part two of the thesis concerns unknown bird classes and confidence. The current system assumes that the incoming acoustic signal comes from one of the classes that the system is able to recognize, i.e. from a class that is known. This implies that signals from other unknown classes gets classified as one of the known classes, i.e. gets misclassified. In order to deal with this problem, an out-of-class detector is implemented. This detector is based on thresholds for the log likelihood scores achieved from the recognizer for each segment of birdsong. If the decoder recognizes a segment from an unknown bird class as a known bird class with a score below some threshold for the respective bird class, it gets classified as unknown, and not the bird that it got classified as in the first place. This leads to an enhanced performance of the system, i.e. decreases the total error rate.

In the absence of recordings belonging to unknown bird species in the database, one have to improvise a bit in order to find good thresholds. The first step in this improvised method is to exclude one and one bird class from the recognition network and do recognition of the files corresponding to the missing bird class in order to force the system to recognize the segments of these files as a known bird class. This is done for all the 21 bird classes, and we get a bunch of wrong recognized segments. Next step is to recognize all birdsong files from one and one class with the complete recognition network in order to get a bunch of correct recognized segments. Then by plotting a histogram, one for each bird class, containing wrong recognized segments and correct recognized segments with corresponding log likelihood scores and number of frames (duration), one should hopefully be able to set a threshold between the wrong recognized segments and the correct recognized segments for each bird class, i.e. it is reasonable to think that wrong recognized segments have a lower log likelihood score than correct recognized segments. Finding good thresholds is actually finding the best compromise between the number of false rejects and the number of false accepts in order to achieve the best results. Two different strategies for setting these thresholds are investigated in order to get the best results. These strategies are presented in chapter 5.5.

The above procedure is first done using only files from the training set, and secondly done using only files from the test set and then compared to each other. Ideally one should have had an untouched data set for finding thresholds, but because of the lack of big enough database, this was the only opportunity. Using the test set can be seen as cheating because the same set is being used for both testing the performance of the out-of-class detector and finding thresholds, but gives a good picture of whether this out-of-class detector is a good or bad idea.

## 1.3 Outline of the report

The report is structured in the following way:

**Chapter 2 - Theory** presents the basic theory required to understand the bird classification system. Section 2.1 presents the feature extraction by explaining what MFCC is and how the frequency analysis of short-time stationary signals is done, section 2.2 presents the two different model structures GMM and HMM and how the parameters of these are estimated using the EM-algorithm. It also include an explanation of the Viterbi algorithm used for recognition.

**Chapter 3 - Database** contain a description of the database containing the audio files of birdsong used in this thesis. Section 3.1 gives some general information about the database while section 3.2 presents the structure of the database and how it is divided into different training- and test sets.

**Chapter 4 - System description and method** contains a general description of the automatic classification system used in the application. An overview of the whole system is given included a more detailed description of the training part presented in section 4.1 and a more detailed description of the classification part presented in section 4.2. This section includes the recognition part, the out-of-class detector, the use of multiple unknown classes and the post processor. The different methods and algorithms used are presented along the way together with the corresponding HTK commands and the relevant scripts used.

**Chapter 5 - Results and discussion** contains the results of the different experiments along with a corresponding discussion of these results. First, the results from the initial system using GMMs as model structure together with different penalty values are presented in section 5.1. The further results are being compared to these initial results. In section 5.2 the results achieved by using HMMs as model structure are presented together with different penalty values. Section 5.3 contains the results when investigating different frame lengths and corresponding window lengths and the different number of coefficients extracted. Section 5.4 presents the different contributions on the total error rate from the different bird species while section 5.5 contains results achieved when dealing with unknown birdsong using the out-of-class detector.

**Chapter 6 - Conclusion** presents the conclusion of the different results obtained, concerning the choice of model structure, the different choices of penalty values, number of coefficients, frame/window lengths, different contributions from different bird species, different test sets and how to deal with unknown birdsong. The final conclusion of the thesis is also given together with suggested further work.

Chapter 2

# Theory

This chapter gives the fundamental theory behind the automatic classification system. Section 2.1 presents the feature extraction by explaining what MFCC is and how the frequency analysis of short-time stationary signals are done, section 2.2 presents the two different model structures GMM and HMM and how the parameters of these are estimated using the EM-algorithm. An explanation of the Viterbi algorithm used for recognition is also included.

## 2.1 Feature extraction

When modeling the different speakers in a speaker classification system, the acoustic signal from each speaker is not used directly, but it is parameterized beforhand. This parameterization extract features that is similar for speech from the same speaker but varies between the different speakers. It is desirable that these features occur frequently, are simple to measure, does not change over time and are not dependent on the speakers health condition. They should also be unaffected of noise and be of high order. The same goes for modeling the different bird species in a bird classification system.

### 2.1.1 Mel- Frequency Cepstral Coefficients

Mel-frequency cepstral coefficients (MFCCs) are very often used as features in speech and speaker recognition systems and it is reasonable to believe that this is the best type of features for birdsong as well. 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 collectively make up an MFC. In order to approximate the human auditory system 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.

The MFCCs are commonly derived as follows:

1. Take the fast Fourier transform (FFT) of (a windowed excerpt of) the acoustic signal of interest.

2. Map the powers of the spectrum obtained above onto the mel scale, using triangular overlapping windows. These triangular filters are uniformly spaced at the mel scale, which is defined as

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

   where $f$ is the frequency in Hz. Figure 2.1 is taken from [11] and shows the filter bank. From this figure we see that the total response of the filter bank equals 1 in all bands except the first and last band.

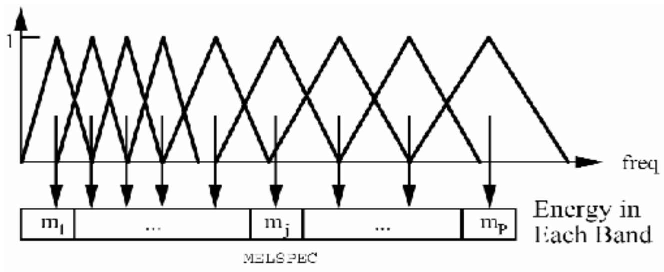


**Figure 2.1:** Mel-scale filter [11].

3. Take the logs of the powers at each of the mel frequencies.
4. Take the discrete cosine transform (DCT) of the list of mel log powers, as if it were a signal.
5. The MFCCs are the amplitudes of the resulting spectrum.

This subsection is motivated by [1] and [8].

### 2.1.2 Frequency analysis of short-time stationary signals

If one assume that a physical signal (birdsong) can be seen as short-time stationary over $K$ ms it is common to use a window of same length in order to extract a slice of the signal. The window for this usually have a rounded shape (Hamming window) that together with the choice of $K$ gives a specific frequency resolution. After the frequency analysis is done for the first frame, the window is shifted $L$ ms and the same window based frequency analysis is done for this slice of the signal. It is common to choose $L < M$ ms in order to achieve an overlap between consecutive windows which is desirable. Figure 2.2 is taken from [4] and shows the whole process for the calculation of MFCCs.

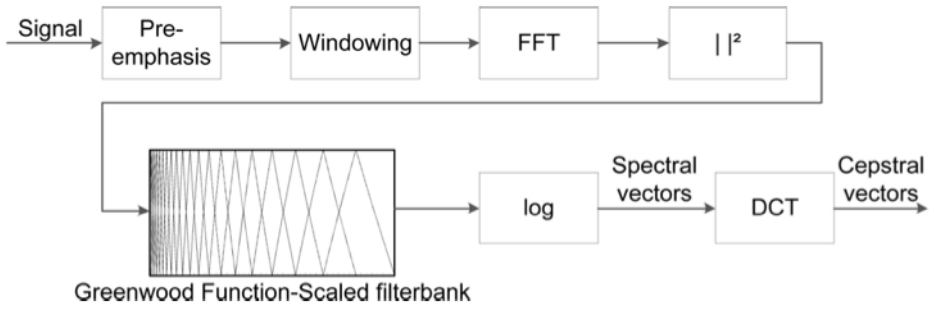


**Figure 2.2:** Frequency analysis [4].

This subsection is motivated by [4].

## 2.2 Models

The features extracted from the acoustic signals are being used to create different models for different bird classes + pause. It is especially two types of such models that are of interest in this thesis, namely Gaussian mixture models (GMMs) and hidden Markov models (HMMs).

### 2.2.1 Gaussian Mixture Models

This subsection is taken from [7]: ”Gaussian mixture models 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.

A GMM is a parametric probability density function represented as a weighted sum of Gaussian component densities. 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,

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

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

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

with mean vector $\mu_i$ and covariance matrix $\Sigma_i$. The mixture weights satisfies the constraint $\sum_i^M w_i = 1$. The complete Gaussian mixture model is parameterized by the mean vectors, covariance matrices and mixture weights from all component densities. These parameters are collectively represented by the notation,

$$\lambda = \{w_i, \mu_i, \Sigma_i\}, i = 1, ..., M \tag{2.4}$$

”

### 2.2.2 Maximum Likelihood

The following is taken from [7]: ”Given training vectors and a GMM configuration, we wish to estimate the parameters of the GMM, $\lambda$, 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 aim 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 $X = \{x_1, ..., x_T\}$, the GMM likelihood, assuming independence between the vectors, can be written as,

$$p(X|\lambda) = \prod_{t=1}^{T} p(x_T|\lambda) = L(\lambda|X) \tag{2.5}$$

.” This function is called the likelihood function. The likelihood is thought of as a function of the parameters $\lambda$ where the data $X$ is fixed. The maximum likelihood problem is about finding the parameters $\lambda$ that maximizes the likelihood function $L$ described in equation 2.5. We want to find $\lambda^*$ where

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

Unfortunately, it is not possible to solve this maximization problem directly. Hence we need to use an iteratively algorithm to solve the problem, namely the Expectation-Maximization algorithm. This section is motivated by [7] and [2].

### 2.2.3 The Expectation-Maximization algorithm

The Expectation-Maximization (EM) algorithm 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 basic idea of the EM algorithm is, beginning with an initial model $\lambda$, 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.

From [3]: ”Given a statistical model consisting of a set of observed data $X$, a set of unobserved latent data or missing values $Z$ and a vector of unknown parameters $\lambda$, along with a likelihood function $L(\lambda|X,Z) = p(X,Z|\lambda)$, the maximum likelihood estimate (MLE) of the unknown parameters is determined by the marginal likelihood of the observed data

$$L(\lambda;X) = p(X|\lambda) = \sum_{Z} p(X,Z|\lambda) \tag{2.7}$$

Here the variable $Z$ denote missing or unobserved data, containing the information about which distribution each observed data point belongs to. The EM-algorithm seeks to find the MLE of the marginal likelihood by iteratively applying the following two steps:

**Expectation (E) step:** Calculate the expected value of the log likelihood function, with respect to the conditional distribution of $Z$ given $X$ under the initial/current estimate of the parameters $\lambda^{(t)}$:

$$Q(\lambda|\lambda^{(t)}) = E_{Z|X,\lambda^{(t)}}\{\log L(\lambda;X,Z)\} \tag{2.8}$$

**Maximization (M) step:** Find the parameter that maximizes this quantity:

$$\lambda^{(t+1)} = \underset{\lambda}{\operatorname{argmax}} Q(\lambda|\lambda^{(t)}) \tag{2.9}$$

Then these two steps are iterated until a convergence threshold is reached”. This subsection is also motivated by [7] and [2].

### 2.2.4 Hidden Markov Models

From [4]: ”Hidden Markov models (HMMs) is a natural choice of modeling the different bird classes if it turns out that birdsong is a time varying process, i.e. it got temporary information”. HMM is dominating the area of speech recognition/classification and recognition/classification of similar physical signals. The general reason for using HMM as a model is to utilize the fact that a HMM contains several states with different distributions and associated transition probabilities such that we can better fit the temporary information in a stanza of birdsong [4]. In this thesis, each state of the HMM contain a GMM-based state distribution. In other words, a one-state HMM is just simply a GMM.

The following description is inspired by [5] and [11], where a more detailed description of hidden Markov models can be found. A hidden Markov model is a stochastic process model in which a discrete time signal is generated from a series of connected states. Figure 2.3 shows a HMM with $(N-2)$ states. It has to special states, one entry state and one exit state. The entry state is the state of the model before the generative process begins while the exit state is the final model state reached once the generative process terminates. $N = 3$ gives just one state and becomes a GMM, i .e. GMM is just a degenerated HMM without temporary analysis.

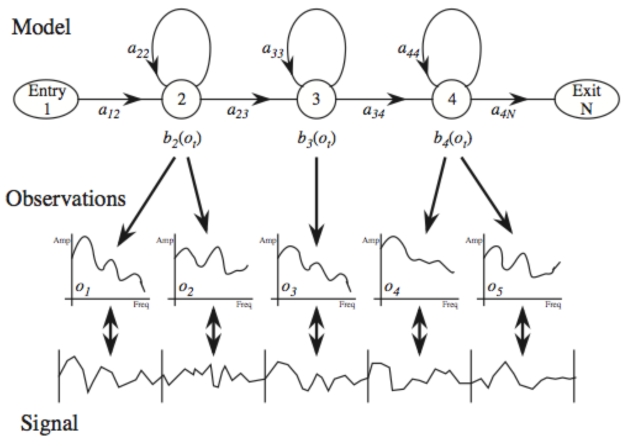


**Figure 2.3:** Illustration of a hidden Markov model [5].

For each time or frame step, the model changes state according to a set of transition probabilities $\{a_{ij}\}$. These probabilities denotes the probability for going from state $i$ to state $j$.

The resultant state generates an observation in accordance with the output probability distribution of that state. The output probability distribution function $b_j(\mathbf{o}_t)$ describes the distribution of observations produced by state $j$. It gives the probability or likelihood of state $j$ generating the observation $\mathbf{o}_t$. The output distribution may e.g. be a Gaussian distribution which is used in this thesis.

The model parameters of a HMM is estimated by using example observations (training data) of a known class. This is what is called training. The training is done similar to the estimating of GMM parameters, i.e. with a case of the EM-algorithm called Baum-Welch algorithm. For a more detailed explanation of this algorithm than what is given in chapter 2.2.3, see [11].

### 2.2.5 Recognition - Viterbi decoding

The recognition of the input applied the application is done by an algorithm called Viterbi. For a detailed mathematical description of this algorithm, see [11]. Given a sequence of observations, the goal of the Viterbi algorithm is to find the sequence of states that generated it. The Viterbi algorithm finds the most likely path of states through the recognition network. In other words, it finds the sequence of states belonging to the legal bird classes or pause that are most likely to have produced the input.

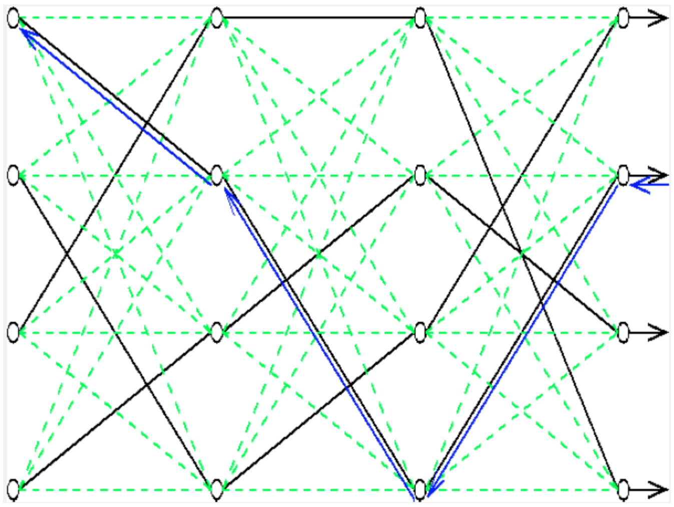


**Figure 2.4:** Illustration of possible paths

Figure 2.4 illustrates this. The green dashed lines shows possible paths between states. The Viterbi algorithm seeks to 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.

Most likely means the bird class or pause that gives the highest log likelihood score for each frame of birdsong. Consider the following: $X_s = \{x_1, ..., x_R\}$ denotes a segment of birdsong containing R frames of a fixed length. The likelihood of this segment can be written

$$Likelihood(X_s) = p(X_s|w_i) = \prod_{k=1}^{R} p(x_k|w_i) \tag{2.10}$$

where $w_i$ denotes all the legal bird classes + pause that the segment can be recognized as. The segment $X_s$ gets recognized as the class $w_i$ which maximizes the log likelihood

$$LogLikelihood(X_s) = \log(Likelihood(X_s)) = \sum_{k=1}^{R} \log(p(x_k|w_i)) \tag{2.11}$$

”The Likelihoods in the Viterbi algorithm will in the end decrease to a very small number because of the product. To avoid underflow when using computers for calculation, the Viterbi algorithm is implemented in the logarithmic domain. A recording of birdsong is acoustically closer to a HMM, the higher log likelihood it has” [6].

# Chapter 3

# Database

This chapter contains a description of the database containing the audiofiles of birdsong used in this thesis. Section 3.1 gives some general information about the database while section 3.2 presents the structure of the database and how it is divided into different training- and test sets. This will give a good overview and understanding of how the database is built up.

## 3.1 General information

The database is handed out by my supervisor while the actual recording of the birdsong is done by various sources of the company AbleMagic. The following information is given by [4]. The sampling frequency of the recordings is 16 kHz and they are saved in wav-format. The database is labeled using the program Praat, see [10] for more information of the program Praat. The labeling is done using a script that automated parts of the process. The labeling is done such that segments corresponding to song from the bird species of interest is labeled as ”song”, while the remaining segments is labeled ”pause”. This implies that the ”pause” segments is a collection of ”real” pauses, background noise, song from other bird species, chorus of birdsong etc. The label files where saved in a TextGrid-format which is a xml-format defined of Praat. Then a perl-script was used to convert the label files from the TextGrid-format to a so called Master Label

File (MLF), which HTK can use for training of models and for evaluating the results of the recognition.

## 3.2 Structure

The duration of the reorded audio files varied greatly. The shortest files was 2-3 seconds while the longest was over 25 minutes. The recordings which is intended to be applied to the application is about 30 seconds. The audio files with a duration greater than 60 seconds is therefore split into shorter audio files, where the split files have a duration up to 85 seconds. After this split, the final database used in this thesis ended up consisting of 629 files containing birdsong from 21 different bird species. The typical length of a file is 30 seconds, but some of the files deviate greatly from this. Table 3.1 shows the distribution of these 629 files over the 21 different bird species that the system are able to recognize. It also shows how many files out of the total amount that are used for training and testing in the third and fourth column respectively.

Notice from table 3.1 that the amount of recordings of the different bird species vary a lot. It exist only 14 files for Dompap, while SvartHvitFluesnapper have 76 files of birdsong. One can also note that only 126 of the totat 629 birdsong files are used for testing the system performance.

As barely mentioned earlier, the database is divided into a training set and a testing set. The training set contains the files being used as example data in order to train the models for each bird class + pause while the test set contains the files being used in order to test the system and achieve results which can give us a picture of whether such a classification system for birdsong works satisfying or not.

Because of a critical small database, one had to choose a 4:1 ratio when dividing the database into sets for training and testing respectively. This distribution is shown in table 3.1. In order to increase the confidence of the results it is made five different training- and test set which together forms a data set, i.e. five different ways of splitting the files for the bird species into a training set and a test set. Data set 1 contains different files for testing and training compared to data set 2, 3, 4 and 5. This leads to five completely

| **Class** | **Total** | **Train** | **Test** |
|---|---|---|---|
| Bjorkefink | 19 | 15 | 4 |
| Blaameis | 30 | 24 | 6 |
| Bokfink | 18 | 14 | 4 |
| Dompap | 14 | 11 | 3 |
| Granmeis | 21 | 17 | 4 |
| Gransanger | 30 | 24 | 6 |
| Grønnfink | 27 | 22 | 5 |
| Gulspurv | 23 | 18 | 5 |
| Hagesanger | 24 | 19 | 5 |
| Jernspurv | 28 | 22 | 6 |
| Lovsanger | 33 | 26 | 7 |
| Maltrost | 34 | 27 | 7 |
| Munk | 41 | 33 | 8 |
| Rodstjert | 21 | 17 | 4 |
| Rodstrupe | 32 | 26 | 6 |
| Sivspurv | 25 | 20 | 5 |
| SvartHvitFluesnapper | 76 | 61 | 15 |
| Svartmeis | 31 | 25 | 6 |
| Svarttrost | 50 | 40 | 10 |
| Toppmeis | 21 | 17 | 4 |
| Trekryper | 31 | 25 | 6 |
| Total | 629 | 503 | 126 |

**Table 3.1:** Database overview

different data sets used for testing while some of the files used for training in the different sets are similar. This let us achieve more confident results by averaging over them. Obviously, any files that are used for training are not being used for testing and vice versa. This applies to all the five data sets.

Chapter 4

# System description and method

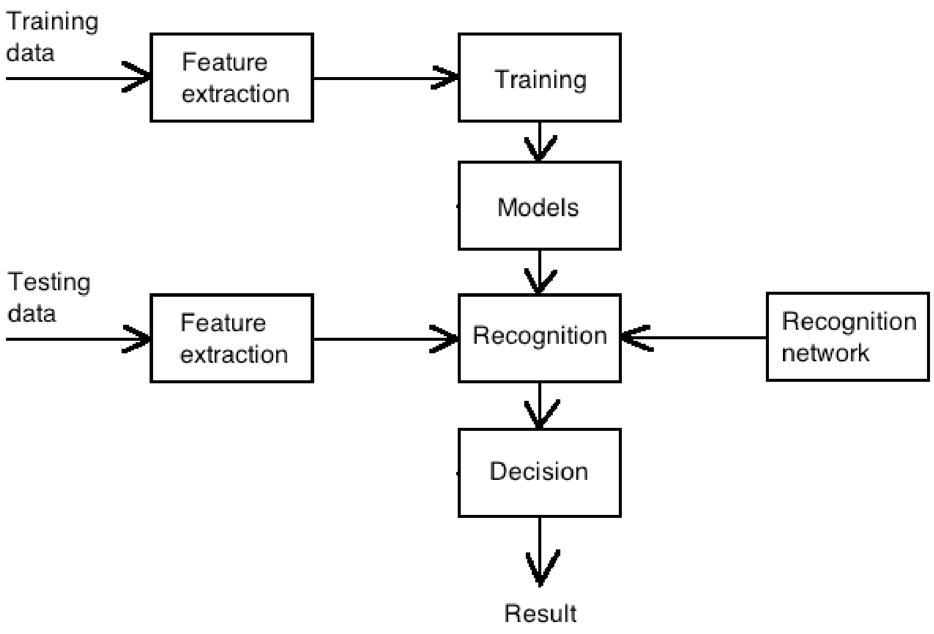


**Figure 4.1:** System

This chapter contains a general description of the automatic classification system used in the application. An overview of the whole system is given including a more detailed description of the training part presented in section 4.1 and a more detailed description of the classification part presented in section 4.2 which includes the recognition part, the out-of-class detector, the use of multiple unknown classes and the post processing. The different methods and algorithms are presented along the way together with the corresponding HTK commands and relevant scripts used.

Figure 4.1 shows the overall system which forms the basic of the application for automatic classification of birdsong. The input is typically a 10-30 seconds acoustic signal recorded by the user and the output is just an answer to the question ”What type of bird is singing?”

The process of answering this question involves several steps: training of unique models corresponding to each bird class plus pause, recognition by investigating which model the input recording matches best, i.e. choosing the bird class or pause that gives the highest log likelihood score. The following explains the whole trip from input to output.

## 4.1 Training

To be able to recognize the input data applied the application as a bird, good prior trained models for each bird class plus pause are necessary. The pause class includes all other sounds than birdsong and silence.

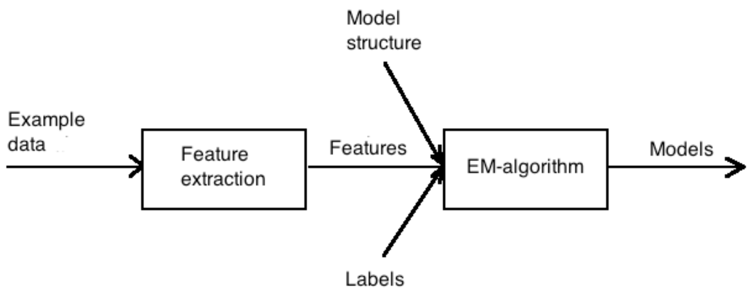


**Figure 4.2:** Training

Figure 4.2 illustrates the training process. The input is pre-labeled audio files containing birdsong from the training set for a specific bird class. A feature extraction is applied to the audio file resulting in a parameterized signal described by vectors of MFCCs. One vector contains a chosen number of MFCCs for a chosen frame length. These MFCC vectors are forming an even bigger vector which holds all the frequency information corresponding to a single class. This big MFCC vector is illustrated as ”features” in figure 4.2. The features together with corresponding labels and a model structure (GMM or HMM) is applied to the EM-algorithm, which iteratively trains the model until some convergence is reached. Given $M$ distributions (mixtures) that together forms a model (GMM/HMM), the EM-algorithm finds which MFCC vector in the big vector that belongs to which distribution/mixture of the model. The final result of the training is a model for a bird class or pause, ready to be used as a model for the upcoming recognition. This procedure is done for all the bird classes plus pause, forming a model set consisting of 22 models (21 bird species plus pause). Window based processing is utilized, meaning that the acoustic signals are split up in pieces which can be seen as stationary, and the feature extraction is done for each of these pieces.

The perl script used for training is called ”TrainModels_BootManual.pl” and can be found in Appendix A. This script uses the HTK commands:

- *HCopy*: Used for feature extraction. Converts automatically the input (training set) into MFCC vectors. To do this, a configuration file is needed which specifies all of the conversion parameters. An example of such a config file can be found in Appendix B.

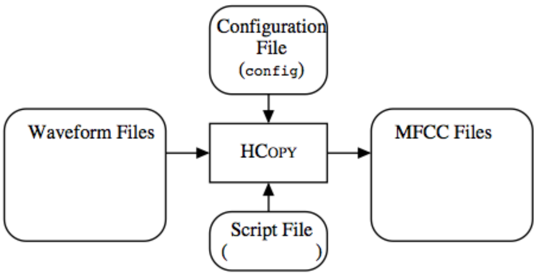


**Figure 4.3:** HCopy [11]

Figure 4.3 illustrates the flow of the command and is taken from [11].

- *HCompV*: Sets the mean and variance of every Gaussian component in a GMM/HMM definition to be equal to the global mean and variance of the training data, i.e. creates an initial model ready for further re-estimation.

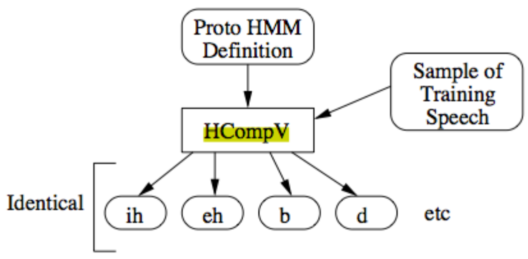


**Figure 4.4:** HCompV [11]

As seen from figure 4.4 which is taken from [11], the HCompV command needs a proto GMM/HMM Definition which holds the structure of the model (number of states, transition probabilities etc.) and sample data. The output is models for each bird plus pause with mean and variance equal to the global mean and variance.

- *HRest*: Performs the EM-algorithm (Baum-Welch re-estimation), which iteratively re-estimates the parameter values of the models. Expects an initialised GMM/HMM definition (by HCompV).

- *HHEd*: Used for cloning models in order to design models containing more mixtures.

## 4.2 Classification

The classification consists of the recognition, an out-of-class detector, a decision and a post processor. Figure 4.5 presents a block scheme of the classification process.

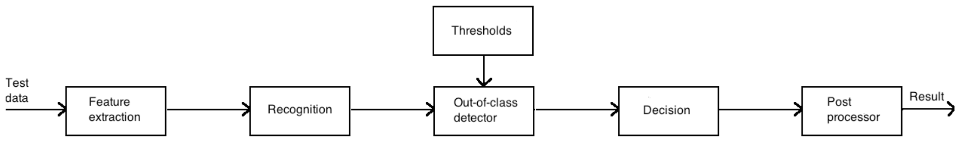
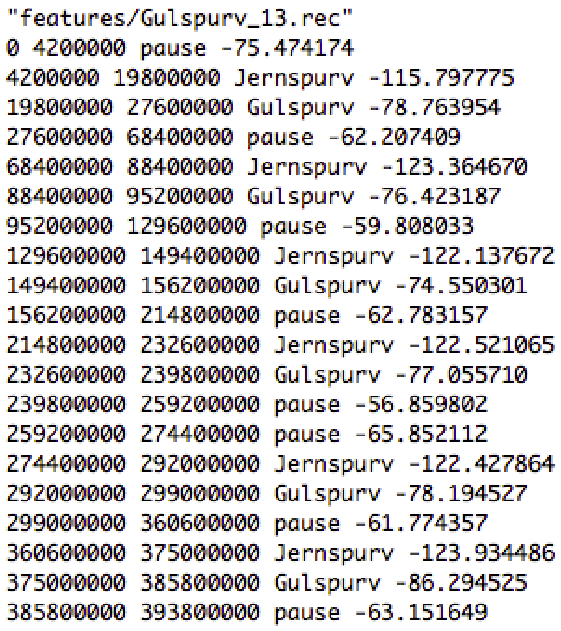


**Figure 4.5:** Classification

### 4.2.1 Recognition

Input to the recognition block is features (MFCC) of a recording of the birdsong of interest. The recognition is done by the Viterbi-algorithm, which finds the most likely path in a recognition network consisting of all possible bird classes plus pause. The Viterbi algorithm finds the most likely sequence of birds (ideally just one bird) and pauses that produced the input audio, and produces a resulting list of segments with corresponding duration, recognition label and the log likelihood score per frame. Figure 4.6 shows an example of such a list produced by the recognizer by applying birdsong from ”Gulspurv”.



```
"features/Gulspurv_13.rec"
0 4200000 pause -75.474174
4200000 19800000 Jernspurv -115.797775
19800000 27600000 Gulspurv -78.763954
27600000 68400000 pause -62.207409
68400000 88400000 Jernspurv -123.364670
88400000 95200000 Gulspurv -76.423187
95200000 129600000 pause -59.808033
129600000 149400000 Jernspurv -122.137672
149400000 156200000 Gulspurv -74.550301
156200000 214800000 pause -62.783157
214800000 232600000 Jernspurv -122.521065
232600000 239800000 Gulspurv -77.055710
239800000 259200000 pause -56.859802
259200000 274400000 pause -65.852112
274400000 292000000 Jernspurv -122.427864
292000000 299000000 Gulspurv -78.194527
299000000 360600000 pause -61.774357
360600000 375000000 Jernspurv -123.934486
375000000 385800000 Gulspurv -86.294525
385800000 393800000 pause -63.151649
```

**Figure 4.6:** Example output produced by the recognizer

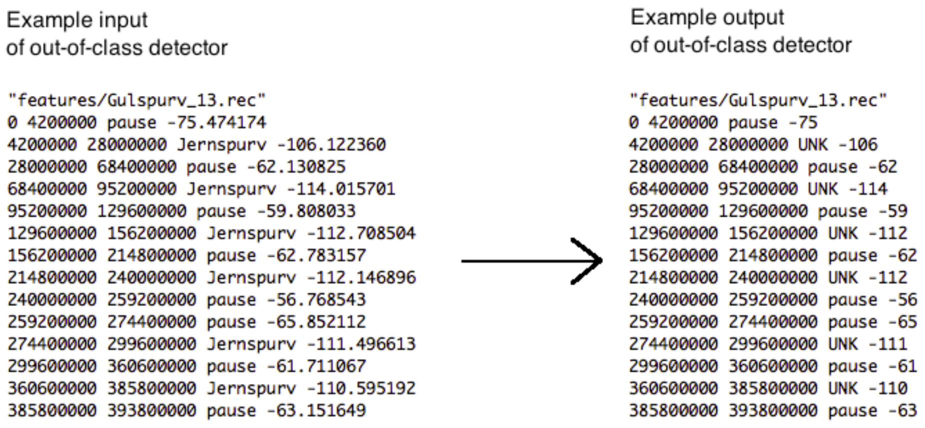
From figure 4.6 we see that the audio file containing birdsong from Gulspurv have segments that have been recognized as both Gulspurv, Jernspurv and pause.

The recognition is done using the perl script ”Decode_test5_HMM.pl” which could be found in Appendix C. The HTK commands used for recognition:

- *HParse*: Builds the recognition network consisting of all legal classes plus pause using a grammar.
- *HVite*: Performs the Viterbi-based recognition. Takes the recognition network and the pre trained model set as input and output a result file similar to the one shown in figure 4.6.
- *HResults*: Evaluates the recognition results. Takes the original Master Label File (MLF) which contains the correct labels of the test set and the MLF file generated by HVite as input and compare these two. This is a way of investigating the final results of recognition process.

### 4.2.2 Out-of-class detector

The second part of the thesis is about dealing with unknown bird classes. If birdsong from a bird class that is unknown to the system is applied, an out-of-class detector is intended to help the classifier classify this birdsong as unknown, and not as birdsong from one of the known classes. The out-of-class detector investigate the log likelihood score per frame for every recognized segment. If this score is below a fixed threshold for the actual bird class, it changes the corresponding label to ”UNK”, which is an abbreviation for ”unknown”. Figure 4.7 illustrates an example of optimal performance of the out-of-class detector.



Example input
of out-of-class detector

```
"features/Gulspurv_13.rec"
0 4200000 pause -75.474174
4200000 28000000 Jernspurv -106.122360
28000000 68400000 pause -62.130825
68400000 95200000 Jernspurv -114.015701
95200000 129600000 pause -59.808033
129600000 156200000 Jernspurv -112.708504
156200000 214800000 pause -62.783157
214800000 240000000 Jernspurv -112.146896
240000000 259200000 pause -56.768543
259200000 274400000 pause -65.852112
274400000 299600000 Jernspurv -111.496613
299600000 360600000 pause -61.711067
360600000 385800000 Jernspurv -110.595192
385800000 393800000 pause -63.151649
```

Example output
of out-of-class detector

```
"features/Gulspurv_13.rec"
0 4200000 pause -75
4200000 28000000 UNK -106
28000000 68400000 pause -62
68400000 95200000 UNK -114
95200000 129600000 pause -59
129600000 156200000 UNK -112
156200000 214800000 pause -62
214800000 240000000 UNK -112
240000000 259200000 pause -56
259200000 274400000 pause -65
274400000 299600000 UNK -111
299600000 360600000 pause -61
360600000 385800000 UNK -110
385800000 393800000 pause -63
```

**Figure 4.7:** Out-of-class detector

In this example, Gulspurv has been deleted from the recognition network meaning that birdsong from Gulspurv is unknown to the system in this case. The segments of this recording of Gulspurv gets recognized as Jernspurv, but because of log likelihood scores below the fixed threshold for Jernspurv, the segments gets labeled as ”UNK”, and later on classified as unknown instead of Jernspurv.

These fixed thresholds mentioned above are found by investigating the log likelihood scores for wrong recognized segments and correct recognized segments of birdsong from each bird class. In this thesis these thresholds are found from both the training set and the test set. Neither of them are optimal. An untouched data set for finding these thresholds should have been used, together with recordings from bird classes that are unknown to the system. In the absence of both an untouched data set (a data set not used for training or testing) and birdsong from unknown bird classes, an improvised method for finding thresholds and testing the performance of the out-of-class detector is used. The following describes the method for finding thresholds from the training set (only files from one of the five training sets are used):

1. For class $i = 1 : 21$:
    (a) Remove the model belonging to class $i$ from the recognition network and generate the recognition network ”anti_i” consisting of 20 classes.
    (b) Recognize all birdsong files from class $i$ with the recognition network ”anti_i” and generate the result file ”anti_i.rec”, consisting of only wrong recognized segments.
2. For class $j = 1 : 21$:
    (a) Find all segments from the 21 ”anti_i” rec-files which has been recognized as class $j$ and place in a file called ”wrongclass_j”. One line is the segment information: ”correct class”, #frames, log likelihood/frame
    (b) Find min_j and max_j of the log likelihood score per frame.
3. For class $j = 1 : 21$:
    (a) Recognize all birdsong files from class $j$ with the full recognition network (21 classes) and generate the result file ”cor-

rect_j.rec", consisting of hopefully a lot of correct recognized segments.

4. For class $j = 1 : 21$:
   (a) Find all segments in "correct_j.rec" which is recognized as class $j$ and place in a file called "correctclass_j". One line is the segment information: "correct class", #frames, log likelihood/frame
   (b) Find min_j and max_j of the log likelihood score per frame.
5. For class $j = 1 : 21$:
   (a) Find the smallest min_j of step 2b and 4b and find the biggest max_j of step 2b and 4b.
   (b) Define the bin sizes $bin\_j = (max\_j - min\_j)/30$ by using the values found in 5a.
   (c) Define the bin boundaries $bing\_j[k] = min\_j + bin\_j * (k - 1)$ for $k = 1 : 31$
   (d) Make histograms, i.e.
      i. Find the total number of frames in the file "wrongclass_j" that satisfies:
      $bing\_j[k] < loglikelihood/frame < bing\_j[k + 1]$ for $k = 1 : 30$
      ii. Find the total number of frames in the file "correctclass_j" that satisfies:
      $bing\_j[k] < loglikelihood/frame < bing\_j[k + 1]$ for $k = 1 : 30$
6. For class $j = 1 : 21$:
   (a) Plot the two histograms created above in the same figure. One shows wrong recognized segments with the related log likelihood scores and one shows correct recognized segments with the related log likelihood scores.
   (b) Investigate how much the histograms overlap. Set threshold for the log likelihood between them. Ideally should all the wrong recognized segments have a lower log likelihood score compared to the correct recognized segments.

In the absence of unknown birdsong in the database, one of the five test sets are used in order to test the out-of-class detector. This is done by doing classification of the test set 21 times, and letting one by one bird class

being unknown to the system at each time. The final results are achieved by averaging over all the 21 different cases.
Setting thresholds from the test, i.e. from the same data that are being used for investigating the performance of the out-of-class detector afterwards, gives a good picture of whether such an out-of-class detector is a good idea or not. The resulting performance using the same data for both finding thresholds and testing gives an upper limit of how good the out-of-class detector could ideally work with the data set of interest. The method of finding thresholds from the test set is done quite similar as the way they where found from the training set. The following describes this method:

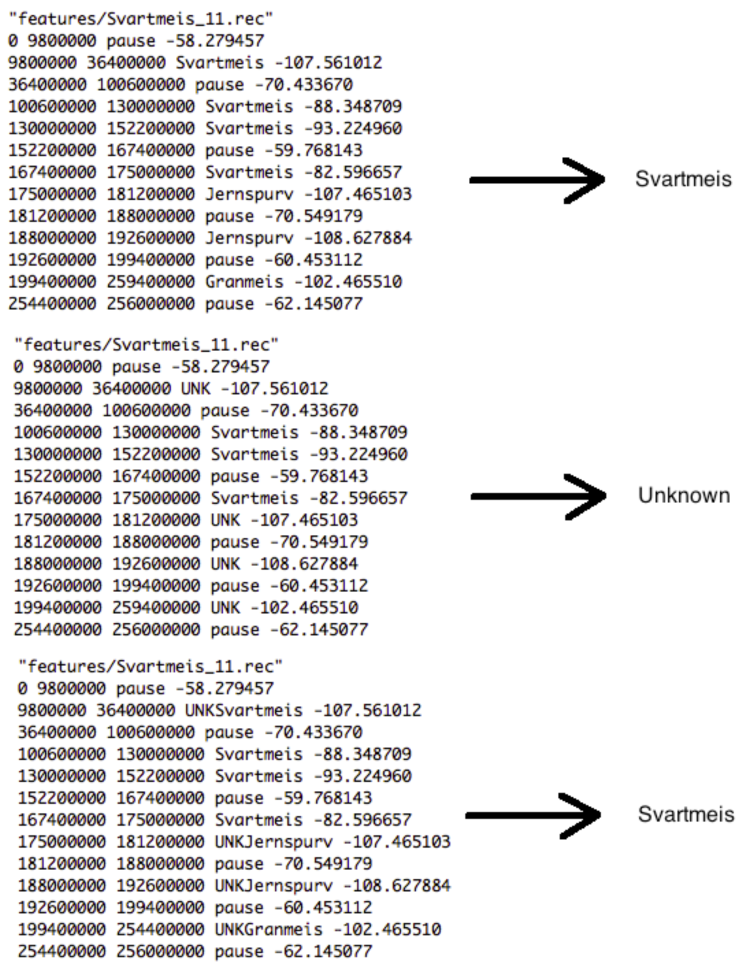
1. Repeat the steps 1ab, 2a, 3a, 4a from the method described above related to thresholds found from training set, but now using only the birdsong files from one of the five test sets.
2. For class $j = 1 : 21$:
   (a) Define an area around the threshold found from the training set and portion this area such that you get 50 values/thresholds uniformly spread in this area.
   (b) For each threshold value $T\_jk$, $k = 1 : 50$, do the following:
      i. Check the log likelihood score for each segment in ”wrongclass_j”.
      If $loglikelihood/frame < T\_jk$ you have a correct reject.
      If $loglikelihood/frame > T\_jk$ you have a false accept.
      ii. Check the log likelihood score for each segment in ”correctclass_j”.
      If $loglikelihood/frame < T\_jk$ you have a false reject.
      If $loglikelihood/frame > T\_jk$ you have a correct accept.
   (c) Plot the number of false accepts vs. the number of false rejects as function of the various $T\_jk$.
   (d) Select the threshold which gives the lowest amount of false accepts and false rejects.

Testing the performance of the out-of-class detector with thresholds found from the test set are done similar as it was done with thresholds found from the training set.

The matlab scripts designed for finding the thresholds and for replacing the recognition labels for the segments to ”UNK” when the log likelihood score per frame is below the found thresholds for each bird class are found in Appendix D.

### 4.2.3 Multiple unknown classes

One problem arises in the out-of-class detector when labeling the segments ”UNK” when the log likelihood score per frame for a segment is below the respective threshold. Consider the following example illustrated by figure 4.8.



```
"features/Svartmeis_11.rec"
0 9800000 pause -58.279457
9800000 36400000 Svartmeis -107.561012
36400000 100600000 pause -70.433670
100600000 130000000 Svartmeis -88.348709
130000000 152200000 Svartmeis -93.224960
152200000 167400000 pause -59.768143
167400000 175000000 Svartmeis -82.596657
175000000 181200000 Jernspurv -107.465103
181200000 188000000 pause -70.549179
188000000 192600000 Jernspurv -108.627884
192600000 199400000 pause -60.453112
199400000 259400000 Granmeis -102.465510
254400000 256000000 pause -62.145077
```

```
"features/Svartmeis_11.rec"
0 9800000 pause -58.279457
9800000 36400000 UNK -107.561012
36400000 100600000 pause -70.433670
100600000 130000000 Svartmeis -88.348709
130000000 152200000 Svartmeis -93.224960
152200000 167400000 pause -59.768143
167400000 175000000 Svartmeis -82.596657
175000000 181200000 UNK -107.465103
181200000 188000000 pause -70.549179
188000000 192600000 UNK -108.627884
192600000 199400000 pause -60.453112
199400000 259400000 UNK -102.465510
254400000 256000000 pause -62.145077
```

```
"features/Svartmeis_11.rec"
0 9800000 pause -58.279457
9800000 36400000 UNKSvartmeis -107.561012
36400000 100600000 pause -70.433670
100600000 130000000 Svartmeis -88.348709
130000000 152200000 Svartmeis -93.224960
152200000 167400000 pause -59.768143
167400000 175000000 Svartmeis -82.596657
175000000 181200000 UNKJernspurv -107.465103
181200000 188000000 pause -70.549179
188000000 192600000 UNKJernspurv -108.627884
192600000 199400000 pause -60.453112
199400000 254400000 UNKGranmeis -102.465510
254400000 256000000 pause -62.145077
```

**Figure 4.8:** Illustration of multiple unknown classes

The recognition file on top in figure 4.8 shows that a birdsong file from Svartmeis gets recognized with segments corresponding to both Svartmeis, Jernspurv and Granmeis. However, the resulting classification without the out-of-class detector of the file is correct, i.e. it gets classified as Svartmeis. In the middle example, the same recognition file is applied to the out-of-class detector which leads to segments getting labeled as ”UNK” when the log likelihood score per frame is lower than the threshold for the actual bird class. Then a decision on which class that occur most frequently (longest duration compared to the total length of the birdsong file) is made. It turns out that the birdsong file that belongs to Svartmeis now gets classified as ”Unknown”, which is wrong. This is because the segments labeled UNK have a together longer duration than the segments labeled Svartmeis. The bottom example shows a solution for this, which is to create 21 different unknown classes, one for each bird, such that segments initially recognized as different bird species and later labeled as ”UNK” do not melt together and beat the true bird class in the decision.

The matlab script designed for replacing the recognition labels to ”UNKbirdclass” when the log likelihood score per frame is below the found thresholds for each bird class are found in Appendix D.

### 4.2.4 Post processing and decision

The Viterbi-based decoder discussed in subsection 4.2.1 provides a sequence of segments of varying lengths, each one with a corresponding log likelihood score. This gives a noisy picture of the reality, i.e. a file gets recognized with segments labeled as different bird species. Figure 4.6 illustrates the problem of deciding which bird the system should classify the recording as. Both Gulspurv and Jernspurv have been recognized as six segments. This implies that the decision is impossible to do if one should count the frequency of a bird class on segment level. This implies that long and short segments are both weighted equally. A method that counts the number of occurrences of the different bird species on frame level takes the lengths of the segments into account, and a decision of which bird the birdsong comes from can be made.

The matlab script designed for making the decision on a frame level is found in Appendix E.

However, the method mentioned above uses only the frequency of the N best classes. Most of the classes (all other classes than Gulspurv and Jernspurv in the example illustrated in figure 4.6) does not appear in the resulting result file after the recognition of one single file, and have therefore a frequency equal zero. One have chosen to investigate whether the distribution of the durations for every single file contains discriminating information. When recognizing all the files corresponding to the 21 bird species (the whole test set) it results in one histogram for each single file containing information about the occurrences of the different bird species and their total number of frames (duration). Now we do some post processing in order to make the decision. Each of the obtained histograms gets normalized in order to compensate for the various amount of frames, i.e. various file lengths. The normalized histogram acts as a vector-estimate of the posteriori probabilities $P_A = \{P_1(w_1|X), ..., P_C(w_C|X)\}$ where $C$ is the number of classes and $X$ is the sequence of MFCC-vectors of the recording. The probability vector $P_A$ is then used as input to a linear classifier with so-called softmax based output nodes. The output values will then, hopefully, represent a better estimate of the posteriori probabilities $P_B(w_i|X)$ for each class. This is what it is done in the post processing block in figure 4.5. Then a decision is made by investigating these $P_B$ probabilities by picking the class with the highest probability. For a more detailed description of such a post processor, see chapter 3 in [9].

The script for using the post processor for making the decision is found in Appendix E. Note that one has to run the script for making the decisions on frame level before running the script using the post processor for making the decision. This is because the histograms generated by the script for making decision on frame level acts as input to the post processor.

Chapter 5

# Results and discussion

This chapter presents the results of the different experiments along with a corresponding discussion. First the results from the initial system are presented. The choice of the initial GMM structure, penalty values, number of coefficients, frame and window length etc. is proposed by [4]. Secondly the different ways of modeling the bird species are presented with the corresponding results. All these results corresponds to a system where all the recordings comes from a known bird. After this the system performance when dealing with unknown bird classes are presented together with the strategies of finding good and "optimal" thresholds for the out-of-class detector.

## 5.1 GMM

The initial case, from [4], is when the bird classes are modeled with GMMs of 32 mixtures. The feature extraction is done on frames of 20 ms with corresponding window lengths of 30 ms. The number of MFCCs extracted from one frame is 15 + one extra coefficient appended as the energy coeffficient. It is also included coefficients for the derivative (delta) and the double derivative (acceleration) of the features that together forms a vector of $(15 + 1) \times 3 = 48$ coefficients. The penalty value is set to $P = -40$.

| Data set | Best | 2-best | 3-best |
|---|---|---|---|
| Training | 9.42 | 4.29 | 3.54 |
| Test | 24.17 | 13.56 | 10.05 |

**Table 5.1:** GMM - Initial average error percentage without post processor

Table 5.1 shows the resulting error percentage when the decision is done on frame level, i.e. without the post processor. These results are an average of the five different training/test sets. It also includes the 2-best and 3-best results, meaning that if the correct bird class is one of the two (2-best) or three (3-best) best suggestions from the histograms it does not increase the error percentage.

Table 5.2 shows the results of the same setting, but now with the post processor.

| Data set | Best | 2-best | 3-best |
|---|---|---|---|
| Training | 3.93 | 0.87 | 0.40 |
| Test | 19.91 | 10.84 | 7.50 |

**Table 5.2:** GMM - Initial average error percentage with post processor

As can be seen out of table 5.1 and 5.2 the system performance gets improved with the use of the post processor. The average error of the test sets decreases from 24.17% to 19.91% when looking at only the best candidate from the histograms. This is an improvement of 4.26% which makes the system able to recognize about 80% of the birdsong correct instead of 75%.

| $P = -40$ | $P = -60$ | $P = -80$ | $P = -100$ |
|---|---|---|---|
| 19.91 | 19.90 | 20.56 | 20.87 |

**Table 5.3:** GMM - Average error percentage with different penalty values

Table 5.3 shows the average results for the test sets when calculating the error percentage for the 1-best case and adjusting the penalty value. It can be seen from this table that the different penalty values does not affect the system performance much.

It can also be seen from table 5.1 and 5.2 that the difference between the results achieved using the test sets versus the training sets are big. Because of this one can say that the system ”recognizes” the training set, meaning that if one uses the same data set for training and testing one achieves results that are way better than what is realistic. The performance achieved by applying the training set gives an upper limit of how good the classification system could work. The difference in performance achieved between the test set and the training set implies that the system do not generalize well. This leads to poorer performance of the test set.

## 5.2 HMM

Now the bird classes are modeled with a 2-state-HMM where each state contains a GMM of 32 mixtures. The feature extraction is done similar to the one for the GMM case, i.e. frames of 20 ms with corresponding window lengths of 30 ms and $(15 + 1) \times 3 = 48$ coefficients. The penalty value is set to $P = -40$.

| Data set | Best | 2-best | 3-best |
|---|---|---|---|
| Training | 2.46 | 1.03 | 0.39 |
| Test | 19.74 | 11.64 | 7.82 |

**Table 5.4:** 2-state-HMM - Average error percentage without post processor

Table 5.4 shows the resulting error percentage without the use of the post processor. From this we can see that we achieve an average error percentage of the test sets of 19.74$\%$ for the 1-best case, which is an improvement of 4.43$\%$ compared to the similar GMM setting.

| Data set | Best | 2-best | 3-best |
|---|---|---|---|
| Training | 1.67 | 0.24 | 0.08 |
| Test | 18.32 | 10.84 | 8.12 |

**Table 5.5:** 2-state-HMM - Average error percentage with post processor

Table 5.5 shows the system performance when the post processor is included when using HMM. Including the post processor leads to an improvement of the 1-best case by 1.42% compared to the case without a post processor, for the test sets. In comparison, the post processor improved the 1-best case by 4.26% using GMM. It looks like the post processor make a bigger difference when GMMs are used as model structure.

| $P = -40$ | $P = -60$ | $P = -80$ | $P = -100$ |
|---|---|---|---|
| 18.32 | 18.33 | 18.96 | 17.69 |

**Table 5.6:** HMM - Average error percentage with different penalty values

Table 5.6 shows the average results for the test sets when calculating the error percentage for the 1-best case and adjusting the penalty value. It can be seen from this that the different penalty values does not affect the system performance much (less than 1% improvement). However, the best result is achieved with a penalty value of -100. With this we get an average error percentage of 17.69%. Comparing the test set result of the best setting using GMM with the best setting using 2-state-HMM, shows that the system performance improves from an average error percentage of 19.90% to an average error percentage of 17.69%.

A 3-state-HMM where also investigated, but the training program terminated due to a big mismatch between the models and the incoming training data.

## 5.3 Number of coefficients, frame and window lengths

From the above experiments, the best best results are achieved by using a 2-state HMM with a GMM of 32 mixtures corresponding to each state, $(15 + 1) \times 3 = 48$ coefficients, frame length of 20 ms, window length of 30 ms, penalty value of -100 and using the post processor. This setting will further be referred to as ”best setting”. This best setting is now investigated by looking at various lengths of frames and corresponding window lengths and the number of coefficients used in the feature extraction.

| Data set | Best | 2-best | 3-best |
|---|---|---|---|
| Training | 2.43 | 0.48 | 0.12 |
| Test | 21.17 | 15.30 | 10.98 |

**Table 5.7:** Average error percentage with frame length of 15 ms and window length of 25 ms

| Data set | Best | 2-best | 3-best |
|---|---|---|---|
| Training | 1.03 | 0.24 | 0 |
| Test | 17.68 | 11.64 | 9.24 |

**Table 5.8:** Average error percentage with frame length of 25 ms and window length of 40 ms

Table 5.7 and 5.8 shows the results for the best setting from the previous experiments, but now using a shorter and longer frame length and corresponding window length, respectively. The use of 25/40 ms is better than the use of 15/25 ms. However, none of them achieves noticeable improvements compared to the previous 20/30 ms case.

| Data set | Best | 2-best | 3-best |
|---|---|---|---|
| Training | 1.71 | 0.28 | 0.08 |
| Test | 18.63 | 11.32 | 8.61 |

**Table 5.9:** Average error percentage with $(12 + 1) \times 3 = 39$ coefficients

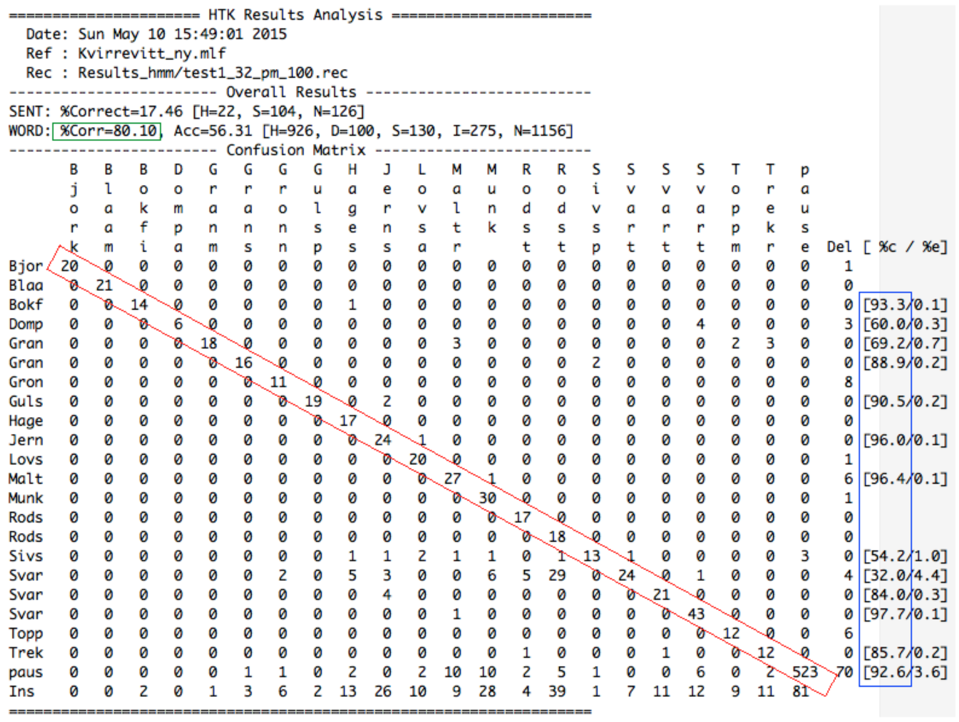
| Data set | Best | 2-best | 3-best |
|---|---|---|---|
| Training | 1.11 | 0.20 | 0.12 |
| Test | 18.62 | 9.39 | 7.32 |

**Table 5.10:** Average error percentage with $(19 + 1) \times 3 = 60$ coefficients

Table 5.9 and 5.10 shows the results for the best setting from the previous experiments, but now using less and more coefficients in the feature extraction, respectively. The use of 39 coefficients and 60 coefficients gives pretty much the same system performance, and does not vary a lot from the previous best setting.

## 5.4 Different bird species and different test sets

The different bird species tends to give different contributions to the overall result of the system.



```
===================== HTK Results Analysis ======================
  Date: Sun May 10 15:49:01 2015
  Ref : Kvirrevitt_ny.mlf
  Rec : Results_hmm/test1_32_pm_100.rec
----------------------- Overall Results -------------------------
SENT: %Correct=17.46 [H=22, S=104, N=126]
WORD: %Corr=80.10, Acc=56.31 [H=926, D=100, S=130, I=275, N=1156]
----------------------- Confusion Matrix ------------------------
        B   B   B   D   G   G   G   G   H   J   L   M   M   R   R   S   S   S   S   T   T   p
        j   l   o   o   r   r   r   u   a   e   o   a   u   o   o   i   v   v   v   o   r   a
        o   a   k   m   a   a   o   l   g   r   v   l   n   d   d   v   a   a   a   p   e   u
        r   a   f   p   n   n   n   s   e   n   s   t   k   s   s   s   r   r   r   p   k   s
        k   m   i   a   m   s   n   p   s   s   a   r       t   t   p   t   t   t   m   r   e Del [ %c / %e]
Bjor   20   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1
Blaa    0  21   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
Bokf    0   0  14   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0 [93.3/0.1]
Domp    0   0   0   6   0   0   0   0   0   0   0   0   0   0   0   0   0   0   4   0   0   0   3 [60.0/0.3]
Gran    0   0   0   0  18   0   0   0   0   0   0   3   0   0   0   0   0   0   0   2   3   0   0 [69.2/0.7]
Gran    0   0   0   0   0  16   0   0   0   0   0   0   0   0   0   2   0   0   0   0   0   0   0 [88.9/0.2]
Gron    0   0   0   0   0   0  11   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   8
Guls    0   0   0   0   0   0   0  19   0   2   0   0   0   0   0   0   0   0   0   0   0   0   0 [90.5/0.2]
Hage    0   0   0   0   0   0   0   0  17   0   0   0   0   0   0   0   0   0   0   0   0   0   0
Jern    0   0   0   0   0   0   0   0   0  24   1   0   0   0   0   0   0   0   0   0   0   0   0 [96.0/0.1]
Lovs    0   0   0   0   0   0   0   0   0   0  20   0   0   0   0   0   0   0   0   0   0   0   1
Malt    0   0   0   0   0   0   0   0   0   0   0  27   1   0   0   0   0   0   0   0   0   0   6 [96.4/0.1]
Munk    0   0   0   0   0   0   0   0   0   0   0   0  30   0   0   0   0   0   0   0   0   0   1
Rods    0   0   0   0   0   0   0   0   0   0   0   0   0  17   0   0   0   0   0   0   0   0   0
Rods    0   0   0   0   0   0   0   0   0   0   0   0   0   0  18   0   0   0   0   0   0   0   0
Sivs    0   0   0   0   0   0   0   0   1   1   2   1   1   0   1  13   1   0   0   0   0   3   0 [54.2/1.0]
Svar    0   0   0   0   0   0   2   0   5   3   0   0   6   5  29   0  24   0   1   0   0   0   4 [32.0/4.4]
Svar    0   0   0   0   0   0   0   0   0   4   0   0   0   0   0   0   0  21   0   0   0   0   0 [84.0/0.3]
Svar    0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0   0   0  43   0   0   0   0 [97.7/0.1]
Topp    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  12   0   0   6
Trek    0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   1   0   0  12   0   0 [85.7/0.2]
paus    0   0   0   0   0   1   1   0   2   0   2  10  10   2   5   1   0   0   6   0   2 523  70 [92.6/3.6]
Ins     0   0   2   0   1   3   6   2  13  26  10   9  28   4  39   1   7  11  12   9  11  81
=================================================================
```

**Figure 5.1:** Recognition matrix

Figure 5.1 shows the results of the recognition for one of the five test sets where ”the best setting” where used in order to train the models. It shows a matrix consisting of all the bird species that are included in the test files and known to the system. The red diagonal corresponds to all the segments that are correct recognized while the other numbers in the matrix corresponds to segments that are recognized wrong. One can see from the blue column on the right that 9 out of 21 bird species gets recognized 100% correct, 6 birds species gets recognised between 88%-98% correct while Dompap (60%), Granmeis (69.2%), Sivspurv (54.2%) and Svarthvit Fluesnapper (32%) contributes to lower the overall recognition results to 80.1% correct recognitions (926 out of 1156 segments correct). The amount of insertions (275) and deletions (100) indicates that the penalty value ($p = -100$)

maybe is too low. However, this penalty value turned out to give the best classification performance and most of the insertions and deletions corresponds to ”pause” and does not affect the results anyway.

The trend according to which bird species that the system easily recognizes correct or not seems to yield all the different test sets and all the different ways of modeling the bird species.

| Test set 1 | Test set 2 | Test set 3 | Test set 4 | Test set 5 | Average |
|---|---|---|---|---|---|
| 23.81 | 18.55 | 12.40 | 19.35 | 23.02 | 19.43 |

**Table 5.11:** Error percentage for the five different test sets for the ”best setting” without post processor

From table 5.11 it can be seen that the classification results varies a lot between the five different test sets. This implies that one should be careful making conclusions of the system performance based on results achieved from our test sets.

## 5.5 Out-of-class detector

Finding the thresholds that are being used in the out-of-class detector is done by investigating the log likelihood scores from the training set and the test set. These methods are described in detail in section 4.2.2.

### 5.5.1 Thresholds from training set

In the absence of an untouched data set for finding thresholds for the log likelihood scores for each bird used in the out-of-class detector, one of the five training sets are used as input for the recognition in order to set the thresholds.

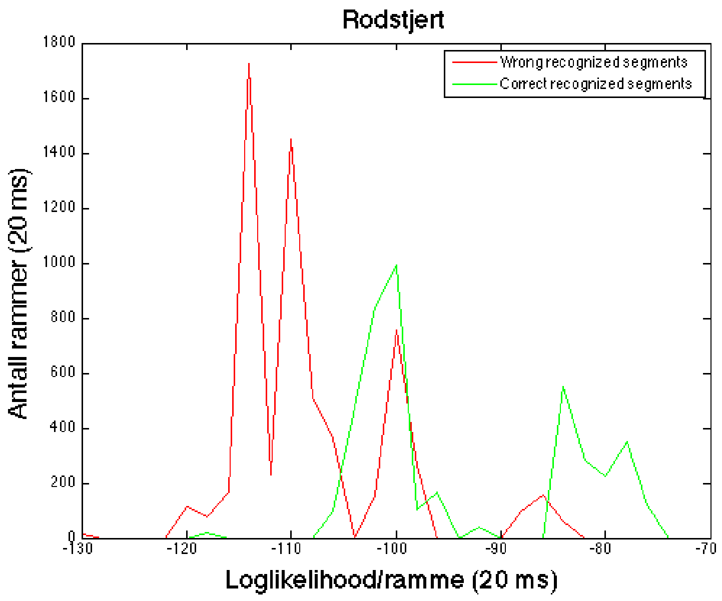


**Figure 5.2:** Trivial case of setting threshold

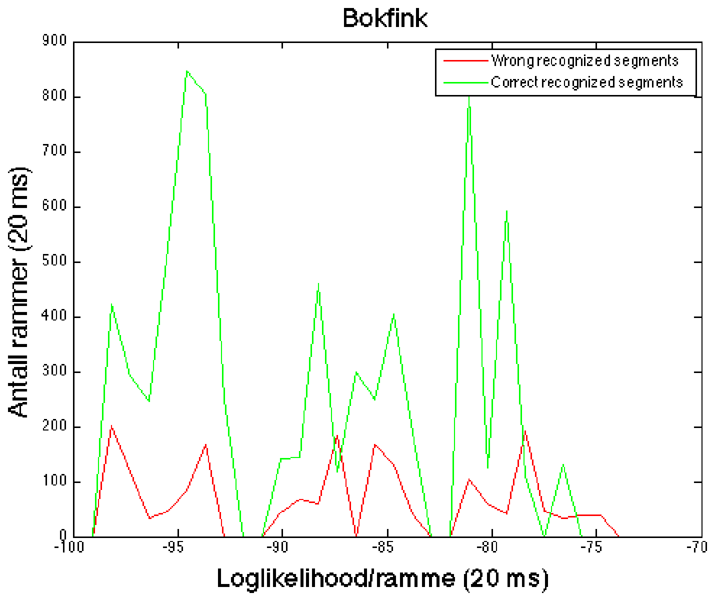


**Figure 5.3:** Difficult case of setting threshold

Figure 5.2 and 5.3 illustrates a trivial case and a difficult case of setting thresholds, respectively. The red curve corresponds to segments which is recognized wrong, i.e. recognized as a bird when the birdsong belongs to another bird. The green curve corresponds to segments which is recognized correct. By choosing a threshold for the log likelihood score per frame of -105 for the bird class ”Rodstjert” in figure 5.2, we get rid of a lot false acceptance (red curve) without eliminating any correct acceptance (green curve). On the other hand, figure 5.3 illustrates that setting any threshold higher than -100 for the bird class ”Bokfink” would affect the correct acceptance which is undesirable. All the plots similar to the ones presented above in figure 5.2 and 5.3 for the other bird classes can be found in Appendix F.

Two different strategies of setting thresholds are investigated. With strategy 1 the thresholds are chosen in a way such that no false rejects are allowed, i.e. the thresholds is set such that the whole green curve lies to the right of the threshold. With strategy 2 the thresholds are chosen in a way such that some few false rejects are allowed if one can get rid of a lot of false accepts, i.e. the thresholds is set such that as much of the red curve as possible lies to the left of the threshold while accepting a bit of the green curve to do the same. The chosen thresholds is presented in table 5.12.

| Bird class | Strategy 1 | Strategy 2 |
|---|---|---|
| Bjorkefink | -82.0 | -82.0 |
| Blaameis | -101.5 | -97 |
| Bokfink | -99.2 | -99.2 |
| Dompap | -92.8 | -80.0 |
| Granmeis | -94.2 | -94.2 |
| Gransanger | -99.9 | -99.9 |
| Grønnfink | -102.0 | -102.0 |
| Gulspurv | -99.3 | -99.3 |
| Hagesanger | -105.8 | -103.5 |
| Jernspurv | -114.0 | -104.5 |
| Lovsanger | -109.0 | -102.0 |
| Maltrost | -109.0 | -99.6 |
| Munk | -106.0 | -99.4 |
| Rodstjert | -120.0 | -104.0 |
| Rodstrupe | -109.0 | -101.5 |
| Sivspurv | -112.0 | -92.0 |
| SvartHvitFluesnapper | -114.0 | -106.0 |
| Svartmeis | -116.1 | -96.0 |
| Svarttrost | -105.0 | -110.2 |
| Toppmeis | -98.5 | -95.0 |
| Trekryper | -109.5 | -96.5 |

**Table 5.12:** Thresholds for the log likelihood score per frame chosen using one of the training sets

In the absence of a data set containing birdsong from unknown bird classes, the testing of the out-of-class detector and the corresponding system performance is done by running classification of one of the five test sets 21 times. Each time a new bird is taken out of the recognition network such that the birdsong files that belongs to this bird is unknown for the system. After this an average of all these 21 different classification results are taken. In order to say something about whether the out-of-class detector works good the result is compared with the average result for classification of the same test set 21 times (one and one bird unknown) but now without the out-of-class detector. This implies that all the files that belongs to the unknown bird gets classified wrong. The results are shown in table 5.13 below.

| Data set | Without | Strategy 1 | Strategy 2 |
|---|---|---|---|
| Test set 1 | 26.9085 | 27.3998 | 35.2608 |

**Table 5.13:** Average error percentage without/with out-of-class detector with thresholds from training set

From table 5.13 above it can be seen that the results achieved with the out-of-class detector are worse than the results achieved without the out-of-class detector. This yields both with thresholds set by strategy 1 and strategy 2. This implies that the thresholds found from the training set does not work properly. Hence, one can say that setting thresholds from the same set that are being used for training the models is a bad idea. It can also be seen from table 5.13 that strategy 1 gives much better results than strategy 2, hence the system is very vulnerable with respect to false rejects.

### 5.5.2 Thresholds from test set

The same procedure for finding threshold is done, but now using the test set that are also being used in order to test the system performance afterwards. This could be seen as a way of ”cheating” because we set the thresholds from the same set that we use for testing. However, it could help us to find out whether such an out-of-class detector could improve the system performance or not when dealing with unknown birdsong.

Figure 5.4 and 5.5 illustrates a trivial case and a difficult case of setting thresholds, respectively. The blue curve corresponds to false accepts while the pink curve corresponds to false rejects. By choosing a threshold for the log likelihood score per frame of -107 for the bird class ”Rodstjert” in figure 5.4, we get rid of a lot of false accepts (from 38 to 18) without introducing any false rejects. On the other hand, figure 5.5 illustrates that setting any threshold higher than -100 for the bird class ”Sivspurv” would introduce false rejects while the win by decreasing the amount of false accepts is not very big. All the plots similar to the ones presented above in figure 5.4 and 5.5 for the other bird classes can be found in appendix F.

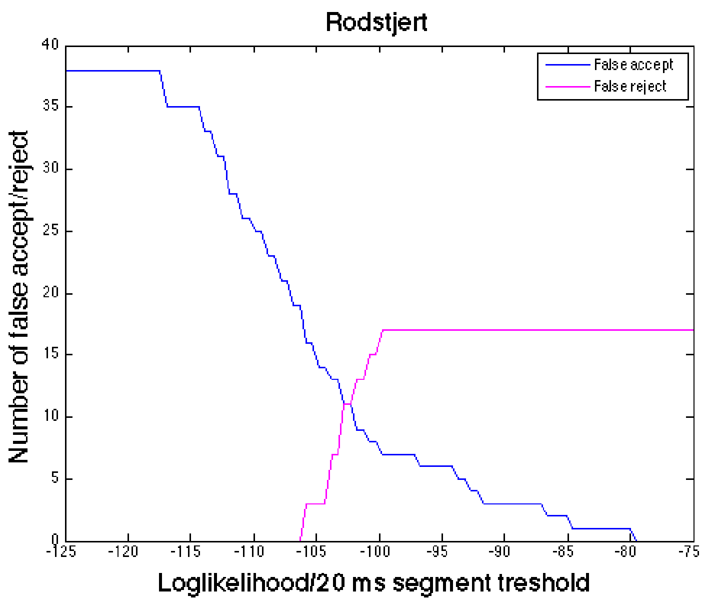


**Figure 5.4:** Trivial case of setting threshold

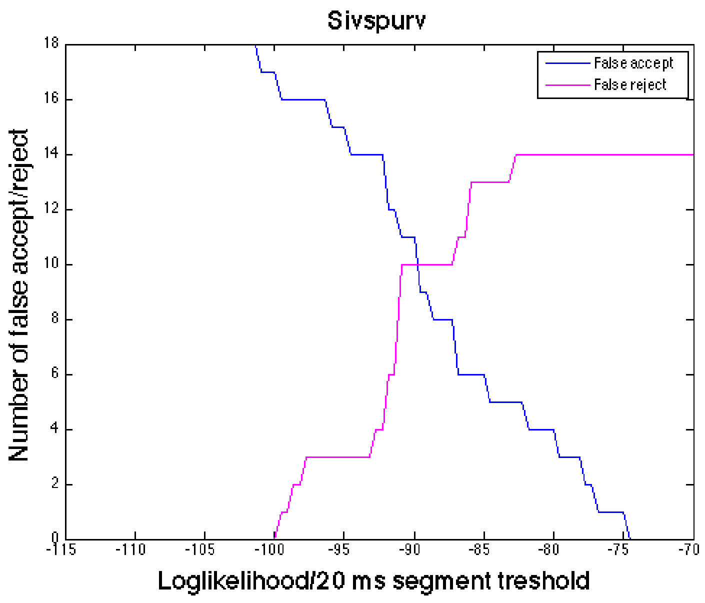


**Figure 5.5:** Difficult case of setting threshold

Similar as for the training set, two different strategies of setting thresholds are investigated. Strategy 1 does not allow any false rejects. This implies that the result from these thresholds must be at least as good as the result achieved without the out-of-class detector. Strategy 2 allow some few false rejects if the threshold decreases the amount of false accepts a lot. The chosen thresholds is presented in table 5.14.

| Bird class | Strategy 1 | Strategy 2 |
|---|---|---|
| Bjorkefink | -70.0 | -70.0 |
| Blaameis | -96.5 | -96.5 |
| Bokfink | -94.1 | -94.1 |
| Dompap | -77.4 | -75.5 |
| Granmeis | -95.0 | -95.0 |
| Gransanger | -94.1 | -94.1 |
| Grønnfink | -85.1 | -75.1 |
| Gulspurv | -115.0 | -105.0 |
| Hagesanger | -105.3 | -103.1 |
| Jernspurv | -107.1 | -105.2 |
| Lovsanger | -106.1 | -104.1 |
| Maltrost | -110.5 | -94.6 |
| Munk | -107.3 | -104.2 |
| Rodstjert | -106.4 | -104.3 |
| Rodstrupe | -101.4 | -97.2 |
| Sivspurv | -100.1 | -100.1 |
| SvartHvitFluesnapper | -109.2 | -109.2 |
| Svartmeis | -108.6 | -108.6 |
| Svarttrost | -102.4 | -100.1 |
| Toppmeis | -96.4 | -94.5 |
| Trekryper | -101.2 | -95.1 |

**Table 5.14:** Thresholds for the log likelihood score per frame chosen using one of the test sets

The testing of the out-of-class detector with threshold found from the test set and the corresponding system performance is done similar as it was done for the thresholds found from the training set. The results are shown in table 5.15 below.

| Data set | Without | Strategy 1 | Strategy 2 |
|---|---|---|---|
| Test set 1 | 26.9085 | 25.3212 | 30.3477 |

**Table 5.15:** Average error percentage without/with out-of-class detector with thresholds from test set

From table 5.15 it can be seen that with the thresholds set by strategy 1, the system improves from an average error percentage of 26.91% to an average percentage of 25.32%. This is an improvement of 1.59%. This number can be misleading. Remember that for each of the 21 tests that are run, only one bird class provides birdsong that is unknown to the system. Each bird class have an average of (Total files in test set 1/Number of bird classes = 126/21 = 6) six birdsong files in the test set. Hence only 6 of the total 126 files forming the test set are unknown. Ideally, the out-of-class detector classifies all six files that belongs to the unknown bird class as unknown. Without the out-of-class detector, 6 out of 126 (4.76%) gets classified wrong (in average). So with an ideal out-of-class detector, the average error percentage can only be improved by 4.76%, meaning that all the unknown birdsong gets classified as unknown. With strategy 1, $1.59\%/4.76\% = 33.4\%$, of the unknown birdsong gets classified as unknown in average, and not as one of the known bird species in the system.

### 5.5.3 Multiple unknown classes

Implementing multiple unknown classes avoids segments classified as ”UNK” to melt together and beat the true bird class in the decision.

| Thresholds from | Strategy | Joint unknown class | Multiple unknown classes |
|---|---|---|---|
| Training set | 1 | 27.40 | 27.04 |
| Training set | 2 | 35.26 | 34.39 |
| Test set | 1 | 25.32 | 25.28 |
| Test set | 2 | 30.35 | 29.70 |

**Table 5.16:** Average error percentage with joint unknown class and multiple unknown classes used in the out-of-class detector

From table 5.16 it can be seen that introducing multiple unknown classes for the case using strategy 1 to find thresholds from the test set decreases the average error percentage from 25.32% to 25.28%. Hence, 35 out of 100 unknown birdsong recordings gets classified as unknown instead of 34 out of 100 as for using one unknown class.

Chapter 6

# Conclusion

This chapter presents the conclusion of the different results obtained, concerning the choice of model structure, the different choices of penalty values, number of coefficients, frame/window lengths, different contributions from different bird species, different test sets and how to deal with unknown birdsong. The final conclusion of the thesis is also given together with suggested further work.

## 6.1 GMM vs. HMM

The best result using GMMs as models for the bird classes where achieved using 32 mixtures, frames of 20 ms with corresponding window lengths of 30 ms, 15 MFCCs, a penalty value in the decoder of $p = -60$ and using the post processor for making the decision. This setup led to an average error of 19.90% for the 1-best case for the test sets.

Replacing GMMs with 2-state-HMMs where each state contains a GMM of 32 mixtures, using frames of 25 ms with corresponding window lengths of 40 ms and 15 MFCCs in the feature extraction, a penalty value in the decoder of $p = -100$ and using a post processor for making the decision, led to an average error of 17.68% for the 1-best case for the test sets which was the best result using HMMs as models for the bird classes.

From this it can be said that using HMMs instead of GMMs for modeling the different bird classes, the resulting system performance improves by 2.22%. This is clearly a noticeable improvement. Because HMM works better than GMM, it is reasonable to say that birdsong does contain some temporary information which HMMs are able to deal with in a better way than GMMs. On the other hand, using a 2-state-HMM doubles the amount of distributions that needs to be trained for each bird class. This leads to less training data (less frames of birdsong) per distribution. In other words, a 2-state-HMM requires a bigger training set than just a GMM.

## 6.2 Penalty value, number of coefficients and frame length

The different penalty values set in the decoder for adding penalties for jumping from one bird class to another in the recognition of a file does not affect the resulting system performance much. The different penalty values investigated with the GMM setup gives resulting error percentages that vary less than 1%. The same goes for the HMM setup. A reason for this is because the deletions and insertions caused by the different penalty values mainly concerns pause segments (silence and other sounds), and therefore does not affect the resulting system performance noteworthy. The purpose of this penalty value is to make the amount of insertions (more segments) and deletions (fewer segments) pretty equal, but it turned out that the best penalty value was the one who gave more insertions than deletions. This implies that it is more important to avoid losing information by deleted segments than it is to avoid added segments by insertions.

The number of coefficients extracted from each frame used for modeling the different bird species does not affect the resulting performance much. Using 12, 15 and 19 MFCCs achieves an average error of 18.63%, 17.69% and 18.62% respectively, using the same setup. However, the less MFCCs that are being extracted, the system gets faster and requires less memory. If the resulting system performance does not improve at all with more coefficients extracted, the best choice is the lowest amount of coefficients.

The different lengths of the frames with corresponding window lengths used in the short-time stationary frequency analysis gave some difference of the system performance. When using frames of 15 ms and windows of 25 ms the average error became 21.17%. Using frames of 20 ms and windows of 30 ms improved the system performance to an average error of 17.69%, while frames of 25 ms and windows of 40 ms achieved an average error of 17.68% for the same setup. By this it can be said that the frames should be at least 20 ms with a corresponding window of 30 ms. However, when the supply of training data is limited, bigger frame lengths and window lengths leads to fewer frames (less training data) per distribution. This could affect the quality of the models, but the database used in this thesis seems to be big enough to handle frames of 20-30 ms.

## 6.3 Different contributions and different test sets

Birdsong from some of the bird classes gets recognized wrong more frequently than birdsong from other bird classes, and contributes to an increase of the overall error rate to a greater extent than others. There are several reasons for this. One reason is that these bird classes produces birdsong that are difficult to distinguish from other birdsong, i.e. the frequency content are similar such that the models gets similar to one or several models that belongs to other birds. Another reason could be that the database used in this thesis provides recordings of these birds containing birdsong of poor quality. Big deviations in frequency content between the different recordings that belongs to one bird specie leads to poorer models and also a bigger mismatch between the files used for testing and training.

It turns out that the average error percentage varies a lot when investigating the system performance using five different test sets (from 23.81% to 12.40%). This gap between the achieved results from the different test sets could imply that the models are trained with an inadequate amount of example data, i.e. the database used in this thesis is too small. Because of this variation in the achieved results from the different test sets, one can say that the test data used in this thesis are not very representative of the later input data. Another thing that is worth mentioned, is that the difference between the system performance when the training sets are used for classification and the test sets are used for classification is big. The performance of the system using the training set as test set gives an upper limit/indication of which results that are possible to obtain. Because of big difference of the system performance between testing with the training set and the test set, one can say that the system do not generalize well and it substantiates the above argument of a too small test database.

It is reasonable to believe that a bigger test database would lead to better overall results because big deviations from the normal birdsong within a specie does not affect the total achieved result of the test set greatly. A bigger database would also help us create better models for each bird specie such that as much as possible of the variances of birdsong within a bird specie gets covered.

## 6.4 Unknown birdsong

Ideally, the out-of-class detector should get rid of all the false acceptance while not increasing the number of false rejects. It turns out that setting thresholds for the log likelihood score per frame for each bird class is not straight forward. The log likelihood scores for segments that are recognized correct and the log likelihood scores for segments that are recognized wrong tends to melt in to each other, making the job of setting an optimal threshold difficult.

When finding thresholds from the training set it is reasonable to believe that these thresholds gets set to high. This is because the system ”remembers” this data from the training process. When recognizing the same files that are used for training the log likelihood scores for the correct recognized segments is more likely to get high. When using the test set for investigating the out-of-class detector it is reasonable to believe that recognitions of these files achieves lower log likelihood scores, and therefore it arises many false rejects when the thresholds are set too high. However, it turned out that the thresholds found from the training set tended to be lower than the thresholds found from the test set, using the same strategy. This is somehow strange, and hard to explain.

The resulting system performance when dealing with unknown birdsong with thresholds in the out-of-class detector found from the training set gets poorer than without the use of the out-of-class detector. The best strategy for finding thresholds gives an average error percentage of 27.4, while the average error percentage achieved without the out-of-class detector is 26.9. This is mainly because of a too big gap/mismatch between the birdsong in the training set used for finding thresholds and the test set used for investigating the system performance afterwards. It can be said that setting thresholds from the training set is a bad idea, both because the models are trained with the same data set and because the birdsong in test set does not resemble the birdsong in the training set.

Setting thresholds from the test set, i.e. the same data set that are used later for investigating the resulting system performance could be seen as a way of ”cheating”, but gives a good picture of whether such an out-of-class detector is a good or bad idea in order to deal with unknown birdsong. Using strategy 1 (don’t allow any false rejects) improves the average error from 26.91% to 25.32%. Hence, 33.4% of the average six unknown birdsong files out of the total 126 birdsong files in the test set gets classified as unknown. This is clearly an improvement, and it shows that the out-of-class detector could be a good idea as long as the thresholds are set correctly.

Using the training set or the test set for finding good thresholds is not ideal. One should have had a third untouched data set for this. Hence, a bigger total database is needed. In order for the out-of-class detector to work satisfying, the data set used for finding thresholds must be similar to the input data that is likely to show up as recordings applied to the application in the future.

Setting thresholds with strategy 1 gives a much better performance than setting thresholds with strategy 2, both for thresholds found from the test set and the training set. Our system, database and way of investigating the out-of-class detector are very sensitive when it comes to false rejects. It is of much greater interest to avoid false rejects than getting rid of false accepts. The best way of setting thresholds is therefore by getting rid of as many false accepts as possible without introducing any false rejects. The reason for this is because when the out-of-class detector gets tested, only an average of 6 out of the total 126 files in the test set contains birdsong from unknown bird species at a time. It is therefore of greater importance that the 120 files corresponding to known bird species don’t get classified wrong because of arising false rejects than that all of the 6 files corresponding to unknown bird species gets classified as unknown. Without the out-of-class detector we achieve an error of 26.9%, and we know that in average 6 out of 126 files corresponding to the unknown birdsong gets classified wrong. Its only when the thresholds are set in a way that no false rejects arises that the out-of-class detector improves this result. When the thresholds are set in a way that we get rid of a lot of false accepts, but simultaneously accepts some false rejects, the resulting performance of the out-of-class detector gets poorer than without using it at all. Hence, when setting thresholds for the out-of-class detector one has to take into account whether it is most likely that the input data comes from a known bird specie or from an unknown bird specie. If the system ”knows” most of the bird species that exists in the nature, thresholds should be set in a way such that as few as possible false rejects arises, i.e. low. On the other hand, if the system only ”knows” a small part of the existing bird species, i.e. it is likely that the input recording belongs to an unknown bird, the thresholds should be set high in order to avoid false accepts.

Out of this thesis, it is shown that the use of an out-of-class detector is a good idea for dealing with unknown birdsong. Creating a common model for all unknown bird classes are difficult and the model itself gets very complex. However, the use of an out-of-class detector, and to achieve satisfying performance, requires a much bigger database such that one can set thresholds from an untouched data set which is likely to be representative to later input data. Bigger database also provides better thresholds because any great deviations from the normal does not affect the resulting threshold to a large extent.

Introducing multiple unknown classes could be a well functioning solution if the recognized segments labeled as unknown tends to melt together at beat the real bird class in the decision when known birdsong is applied the system.

## 6.5 Bottom line

HMM is better than GMM and an out-of-class detector have a good potential of dealing with unknown birdsong. A bigger database is strongly needed. Both for creating better models for each bird specie and for setting thresholds in the out-of-class detector. Bigger database = better performance of the automatic classification system for birdsong.

## 6.6 Further work

Suggested further work in order to improve the system:

- Collect much more data from all the bird species and with different recording situations. Hopefully this leads to better models and let us create a data set only for setting thresholds for the out-of-class detector.

- Label the new data.

- Expand the amount of bird classes in order to avoid that birdsong comes from unknown bird species.

- Adaptation: Utilize the recordings submitted by the users. Use these recordings as new training data or for data set used for finding thresholds for the out-of-class detector.

- Re-label the database. Current database are labeled with just birdsong or ”pause”. Split up the class ”pause” into different classes. One class for background noise, one for noise from cars, one for noise from waterfalls etc, and train these as separate models.

# Bibliography

[1] Aizawa, K., Nakamura, Y., Satoh, S., 2004. Advances in multimedia information processing.

[2] Bilmes, J. A., 1998. A gentle tutorial of the em algorithm and its application to parameter estimation for gaussian mixture and hidden markov models.
URL http://crow.ee.washington.edu/people/bulyko/papers/em.pdf

[3] Dempster, A. P., Laird, N. M., Rubin, D. B., 1977. Maximum likelihood from incomplete data via the em algorithm.
URL http://www.jstor.org/stable/2984875?seq=1#page_scan_tab_contents

[4] Johnsen, M. H., Svenden, T., Perkis, A., 2015. Whatbird - dokumentasjon.

[5] Odell, J. J., 1995. The use of context in large vocabulary speech recognition.

[6] Onshus, I., 2011. Indexing of audio databases.

[7] Reynolds, D., 2010. Gaussian mixture models.
URL https://www.ll.mit.edu/mission/cybersec/publications/publication-files/full_papers/0802_Reynolds_Biometrics-GMM.pdf

[8] Sahidullah, M., Saha, G., 2012. Design, analysis and experimental evaluation of block based transformation in mfcc computation for speaker recognition.

[9] Theodoridis, S., Koutroumbas, K., 1999. Pattern Recognition.

[10] van Lieshout, P., 2003. PRAAT Short Tutorial - A Basic Introduction.

[11] Young, S., Evermann, G., Gales, M., Hain, T., Kershaw, D., Liu, X., Moore, G., Odell, J., Ollason, D., Povey, D., Valtchev, V., Woodland, P., 2006. The HTK Book.
URL `http://htk.eng.cam.ac.uk/prot-docs/htkbook.pdf`

# Appendix

## A: Perl script for training: TrainModels_BootManual.pl

```
#!/usr/bin/perl
#
# TrainModels_BootManual
#
use Getopt::Std;

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


# Various files needed
$PhoneMlf="lists/Kvirrevitt.mlf"; # Phone level MLF from
    initial segm.
$list="lists/train1.scp"; # database information file
$Proto="lists/proto.pcf"; # Specifications for the HMM
    prototype
$Hcomp_conf="$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/state2_$startDir";
$tgtdir="$m/state2_$nmix";
&makedir($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);
$cmd="$t/MakeProtoHMMSet $w/proto.pcf";
&run("$cmd");
```

```
# 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/state2_$nmix";
&makedir($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/state2_${nmix}";

while ($mmix < $maxmix) {

    $newmix=2*$mmix;
    $targetdir="$m/state2_${newmix}";
    &makedir($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: HCopy config file: hcopy.cfg

```
# Input file format (16 kHz WAV format)
SOURCEKIND     = WAVEFORM
SOURCERATE     = 625
SOURCEFORMAT   = WAV

# Output file format
 TARGETKIND      = MFCC_E_D_A_Z
 TARGETFORMAT    = HTK
 TARGETRATE      = 200000
 WINDOWSIZE      = 300000.0
 USEHAMMING      = TRUE
 PREEMCOEF       = 0.97
#USEPOWER        = TRUE
 NUMCHANS        = 26
 CEPLIFTER       = 22
 NUMCEPS         = 15
# SILFLOOR        = 50

SAVECOMPRESSED = FALSE
SAVEWITHCRC    = FALSE
```

## C: Perl script for recognition: Decode_test5_HMM.pl

```
#!/usr/bin/perl

# Script for classifying the test sets using the
   corresponding gmm models with $mix mixtures.
# The resulting *.rec file is used for normalized histogram
    generation which therafter is classified using the
   corresponding postprocessor.

# Choose number of mixtures and penalty for jumping between
    class models
$mix = 32; $wmix = "32";
$wordpen = -100; $wp = "pm_100";

 # Assuming gmm models are trained and placed in ... :
```

```
my @modeldirs = ("models1", "models2", "models3", "models4
    ", "models5");

# Using the following configure file
$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
$cl_list = "class_pause_ny.list";
$dict = "birds_ny.dict";
$network = "birds_ny.net";

$labmlf = "Kvirrevitt_ny.mlf";

# Establish folder to put rec-files
$resultDir = "Results_hmm";
unless(-d "$resultDir") {system("mkdir $resultDir");}
$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 $modeldir (@modeldirs){

    $set = $set +1;
    $setind = $set +1;
    $testset = $datasets[$set];
    $modeldir = $modeldirs[$set];
    $mmf = "$modeldir/state2_$mix/state2_$mix.mmf";
    print "speakerlist : $testset\n";
    # printf "Results are placed in $recmlf and $resultfile
        \n";
    $recmlf = $resultDir."/test".$setind."_".$wmix."_".$wp.
        ".rec";


    system "HVite -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";
}

# In the following should the normalized histograms be
   generated (now in the Matlab m-file called
   Frame_hist_all.m).
# Also histogram based classification (1,2,3 best) of the
   test set is presently done in the same m-file.

# Her should the testing using the postprocessor be done (
   now in Matlab m-file called Tren_test_post.m)
```

## D: Matlab scripts for finding thresholds from training set

### 1: Generate wrongclass_j files

```
%Script generating the 21 wrongclass_j files

classfile = 'class_pause_ny.list';
class = struct('memb',{{}});
fclass = fopen(classfile, 'r');
Nclass = 0;

while 1
    line = fgetl(fclass);
    if ~ischar(line), break, end
    Nclass = Nclass + 1;
    class.memb(Nclass) = {line};
end

Nclass_birds = Nclass - 1;
load('setsizes.txt','-ascii');

for i = 1:21
    fasitklasse = class.memb{i};
    recnr = num2str(i);
    recfilename = strcat('Results_hmm/test_anti_',recnr,'
       .rec');
    frec = fopen(recfilename,'r');
    mlfline = fgetl(frec);
    Nfiles = setsizes(i);
    for j=1:Nfiles
        fileline = fgetl(frec);
        fileline = fgetl(frec);
        while(~(strcmp(fileline,'.')))
            scanfile = sscanf(fileline, '%d %d %s %d');
            recog_bird = char(scanfile(3:length(scanfile)
                -1))';
            printfilename = strcat('Results_hmm/
                test_misklass_',...
                 recog_bird,'.rec');
            ffile = fopen(printfilename,'a');
            antallrammer = num2str((scanfile(2)-scanfile(1)
                )/200000);
            logscore = num2str(scanfile(length(scanfile)));
            %string = strcat(fasitklasse,antallrammer,
                logscore);
            %fwrite(ffile,string);
            fwrite(ffile,fasitklasse);
            fwrite(ffile,' ');
            fwrite(ffile,antallrammer);
            fwrite(ffile,' ');
            fwrite(ffile,logscore);
            fprintf(ffile,'\r\n');
            fclose(ffile);
            fileline = fgetl(frec);
        end
    end
    fclose(frec);
end
```

## 2: Generate correctclass_j files

```
%Script generating the 21 correctclass_j files

classfile = 'class_pause_ny.list';
class = struct('memb',{{}});
fclass = fopen(classfile, 'r');
Nclass = 0;
```

```
while 1
    line = fgetl(fclass);
    if ~ischar(line), break, end
    Nclass = Nclass + 1;
    class.memb(Nclass) = {line};
end

Nclass_birds = Nclass - 1;
load('setsizes.txt','-ascii');

for i = 1:21
    fasitklasse = class.memb{i};
    recnr = num2str(i);
    recfilename = strcat('Results_hmm/test_korrekt_',recnr,
        '.rec');
    frec = fopen(recfilename,'r');
    mlfline = fgetl(frec);
    Nfiles = setsizes(i);
    for j=1:Nfiles
        fileline = fgetl(frec);
        fileline = fgetl(frec);
        while(~(strcmp(fileline,'.')))
            scanfile = sscanf(fileline, '%d %d %s %d');
            recog_bird = char(scanfile(3:length(scanfile)
                -1))';
            if (strcmp(recog_bird,fasitklasse)==1)
            printfilename = strcat('Results_hmm/
                test_korrklass_',...
                 fasitklasse,'.rec');
            ffile = fopen(printfilename,'a');
            antallrammer = num2str((scanfile(2)-scanfile(1)
                )/200000);
            logscore = num2str(scanfile(length(scanfile)));
            fwrite(ffile,fasitklasse);
            fwrite(ffile,' ');
            fwrite(ffile,antallrammer);
            fwrite(ffile,' ');
            fwrite(ffile,logscore);
            fprintf(ffile,'\r\n');
            fclose(ffile);
            end
            fileline = fgetl(frec);
        end
    end
    fclose(frec);
```

```
end
```

## 3: Find min_j and max_j for the files wrongclass_j

```
%Script finding min and max log likelihood per frame in the
    wrongclass_j files

minmax_misklass = zeros(21,2);
classfile = 'class_pause_ny.list';
class = struct('memb',{{}});
fclass = fopen(classfile, 'r');
Nclass = 0;

while 1
    line = fgetl(fclass);
    if ~ischar(line), break, end
    Nclass = Nclass + 1;
    class.memb(Nclass) = {line};
end

for i = 1:21
    classname = class.memb{i};
    recfilename = strcat('Results_hmm/misklass_',classname,
        '.rec');
    frec = fopen(recfilename,'r');
    fileline = fgetl(frec);
    scanfile = sscanf(fileline, '%s %d %d');
    minmax_misklass(i,1) = scanfile(length(scanfile));
    minmax_misklass(i,2) = scanfile(length(scanfile));
    while(~feof(frec))
        fileline = fgetl(frec);
        scanfile = sscanf(fileline, '%s %d %d');
        minormax = scanfile(length(scanfile));
        if (minormax < minmax_misklass(i,1))
            minmax_misklass(i,1) = minormax;
        end
        if (minormax > minmax_misklass(i,2))
            minmax_misklass(i,2) = minormax;
        end
    end
    fclose(frec);
end
```

```
minmax_misklass
```

## 4: Find min_j and max_j for the files correctclass_j

```
%Script finding min and max log likelihood per frame in the
    correctclass_j files

minmax_korrklass = zeros(21,2);
classfile = 'class_pause_ny.list';
class = struct('memb',{{}});
fclass = fopen(classfile, 'r');
Nclass = 0;

while 1
    line = fgetl(fclass);
    if ~ischar(line), break, end
    Nclass = Nclass + 1;
    class.memb(Nclass) = {line};
end

for i = 1:21
    classname = class.memb{i};
    recfilename = strcat('Results_hmm/korrklass_',classname
        ,'.rec');
    frec = fopen(recfilename,'r');
    fileline = fgetl(frec);
    scanfile = sscanf(fileline, '%s %d %d');
    minmax_korrklass(i,1) = scanfile(length(scanfile));
    minmax_korrklass(i,2) = scanfile(length(scanfile));
    while(~feof(frec))
        fileline = fgetl(frec);
        scanfile = sscanf(fileline, '%s %d %d');
        minormax = scanfile(length(scanfile));
        if (minormax < minmax_korrklass(i,1))
            minmax_korrklass(i,1) = minormax;
        end
        if (minormax > minmax_korrklass(i,2))
            minmax_korrklass(i,2) = minormax;
        end
    end
    fclose(frec);
end
```

```
minmax_korrklass
```

## 5: Generate histograms for wrongclass_j and correctclas_j for finding thresholds from training set

```
%Script generating histograms

min_max=zeros(21,2);
%Verdier settes manuelt
min_max(1,1) = -82; min_max(1,2) = -48;
min_max(2,1) = -101; min_max(2,2) = -55;
min_max(3,1) = -100; min_max(3,2) = -73;
min_max(4,1) = -94; min_max(4,2) = -57;
min_max(5,1) = -97; min_max(5,2) = -54;
min_max(6,1) = -113; min_max(6,2) = -77;
min_max(7,1) = -102; min_max(7,2) = -68;
min_max(8,1) = -109; min_max(8,2) = -67;
min_max(9,1) = -118; min_max(9,2) = -77;
min_max(10,1) = -120; min_max(10,2) = -77;
min_max(11,1) = -124; min_max(11,2) = -86;
min_max(12,1) = -120; min_max(12,2) = -65;
min_max(13,1) = -121; min_max(13,2) = -71;
min_max(14,1) = -130; min_max(14,2) = -70;
min_max(15,1) = -114; min_max(15,2) = -67;
min_max(16,1) = -130; min_max(16,2) = -68;
min_max(17,1) = -119; min_max(17,2) = -72;
min_max(18,1) = -118; min_max(18,2) = -60;
min_max(19,1) = -119; min_max(19,2) = -75;
min_max(20,1) = -115; min_max(20,2) = -53;
min_max(21,1) = -115; min_max(21,2) = -59;

%Definerer bin storrelser
bin=zeros(21,1);
for j = 1:21
    bin(j) = (min_max(j,2) - min_max(j,1))/30;
end

%Definerer bin-grenser
bing=zeros(21,31);
for j = 1:21
    for k = 1:31
        bing(j,k) = min_max(j,1) + bin(j)*(k-1);
    end
end

%Histogram for misklass_j
%Finner totalt antall rammer i filen misklass_j og
   korrklas_j som har
%likelihood/ramme score som ligger mellom de forskjellige
   bin-grensene
classfile = 'class_pause_ny.list';
class = struct('memb',{{}});
fclass = fopen(classfile, 'r');
Nclass = 0;

while 1
    line = fgetl(fclass);
    if ~ischar(line), break, end
    Nclass = Nclass + 1;
    class.memb(Nclass) = {line};
end

result_misklass = zeros(21,30);
for i = 1:21
    classname = class.memb{i};
    recfilename = strcat('Results_hmm/misklass_',classname,
       '.rec');
    frec = fopen(recfilename,'r');
    while(~feof(frec))
        fileline = fgetl(frec);
        scanfile = sscanf(fileline, '%s %d %d');
        llr = scanfile(length(scanfile));
        r = scanfile(length(scanfile)-1);
        for k = 1:30
            if llr > bing(i,k) && llr < bing(i,k+1)
                result_misklass(i,k) = result_misklass(i,k)
                    + r;
            end
        end
    end
    fclose(frec);
end

result_korrklass = zeros(21,30);
for i = 1:21
    classname = class.memb{i};
    recfilename = strcat('Results_hmm/korrklass_',classname
        ,'.rec');
    frec = fopen(recfilename,'r');
    while(~feof(frec))
        fileline = fgetl(frec);
        scanfile = sscanf(fileline, '%s %d %d');
        llr = scanfile(length(scanfile));
        r = scanfile(length(scanfile)-1);
        for k = 1:30
            if llr > bing(i,k) && llr < bing(i,k+1)
                result_korrklass(i,k) = result_korrklass(i,
                    k) + r;
            end
        end
    end
    fclose(frec);
end

for k = 1:21
    bird = class.memb{k};
    figure;
    plot(bing(k,1:30),result_misklass(k,:),'Color','Red');
    xlabel('Loglikelihood/ramme (20 ms)','fontsize',18);
    ylabel('Antall rammer (20 ms)','fontsize',18);
    title(bird,'fontsize',18);
    hold on;
    plot(bing(k,1:30),result_korrklass(k,:),'Color','Green'
        );
    legend('Wrong recognized segments','Correct recognized
        segments');
end
```

## 6: Plot false reject vs. false accept for finding thresholds from test set

```
%Script for checking log likelihood per frame for each line
     in "test_wrongclass_j.rec"
%and log likelihood per frame for each line in "
    test_correctclass_j.rec" against 50
%uniformly spread thresholds in an area of interest. Plots
    false accept against
%false reject in order to find optimal thresholds.
```

```
t_1 = linspace(-83,-48,100);
t_2 = linspace(-102,-55,100);
t_3 = linspace(-100,-73,100);
t_4 = linspace(-95,-60,100);
t_5 = linspace(-98,-56,100);
t_6 = linspace(-103,-77,100);
t_7 = linspace(-103,-70,100);
t_8 = linspace(-110,-67,100);
t_9 = linspace(-120,-77,100);
t_10 = linspace(-120,-80,100);
t_11 = linspace(-120,-87,100);
t_12 = linspace(-115,-65,100);
t_13 = linspace(-120,-75,100);
t_14 = linspace(-125,-75,100);
t_15 = linspace(-115,-70,100);
t_16 = linspace(-115,-70,100);
t_17 = linspace(-120,-75,100);
t_18 = linspace(-120,-60,100);
t_19 = linspace(-120,-75,100);
t_20 = linspace(-115,-55,100);
t_21 = linspace(-115,-60,100);

treshold_matrix = zeros(21,100);

treshold_matrix(1,:) = t_1;
treshold_matrix(2,:) = t_2;
treshold_matrix(3,:) = t_3;
treshold_matrix(4,:) = t_4;
treshold_matrix(5,:) = t_5;
treshold_matrix(6,:) = t_6;
treshold_matrix(7,:) = t_7;
treshold_matrix(8,:) = t_8;
treshold_matrix(9,:) = t_9;
treshold_matrix(10,:) = t_10;
treshold_matrix(11,:) = t_11;
treshold_matrix(12,:) = t_12;
treshold_matrix(13,:) = t_13;
treshold_matrix(14,:) = t_14;
treshold_matrix(15,:) = t_15;
treshold_matrix(16,:) = t_16;
treshold_matrix(17,:) = t_17;
treshold_matrix(18,:) = t_18;
treshold_matrix(19,:) = t_19;
treshold_matrix(20,:) = t_20;
treshold_matrix(21,:) = t_21;

correct_reject=zeros(21,100);
false_accept=zeros(21,100);
false_reject=zeros(21,100);
correct_accept=zeros(21,100);

classfile = 'class_pause_ny.list';
class = struct('memb',{{}});
fclass = fopen(classfile, 'r');
Nclass = 0;

while 1
    line = fgetl(fclass);
    if ~ischar(line), break, end
    Nclass = Nclass + 1;
    class.memb(Nclass) = {line};
end

for i = 2:21 %fra 2 fordi test_misklass_bjorkefink.rec ikke
    eksisterer.
    classname = class.memb{i};
    recfilename = strcat('Results_hmm/test_misklass_',
       classname,'.rec');
    frec = fopen(recfilename,'r');
    while(~feof(frec))
        fileline = fgetl(frec);
        scanfile = sscanf(fileline, '%s %d %d');
        llr = scanfile(length(scanfile));
        for k = 1:100
            if llr < treshold_matrix(i,k)
                correct_reject(i,k) = correct_reject(i,k) +
                    1;
            else
                false_accept(i,k) = false_accept(i,k) + 1;
            end
        end
    end
    fclose(frec);
end

for i = 1:21
    classname = class.memb{i};
    recfilename = strcat('Results_hmm/test_korrklass_',
       classname,'.rec');
    frec = fopen(recfilename,'r');
    while(~feof(frec))
        fileline = fgetl(frec);
        scanfile = sscanf(fileline, '%s %d %d');
        llr = scanfile(length(scanfile));
        for k = 1:100
            if llr < treshold_matrix(i,k)
                false_reject(i,k) = false_reject(i,k) + 1;
            else
                correct_accept(i,k) = correct_accept(i,k) +
                    1;
            end
        end
    end
    fclose(frec);
end

for k = 1:21
    bird = class.memb{k};
    figure;
    plot(treshold_matrix(k,:),false_accept(k,:),'color','
       blue');
    xlabel('Loglikelihood/20 ms segment treshold','fontsize
       ',18);
    ylabel('Number of false accept/reject','fontsize',18);
    title(bird,'fontsize',18);
    hold on;
    plot(treshold_matrix(k,:),false_reject(k,:),'color','
       magenta');
    legend('False accept','False reject');
end
```

## 7: Replace recognition label with ”UNK”

```
%Script generating "tot_unk_anti_i.rec". Read the files "
   tot_anti_i.rec"
%and replaces the recognition labels with UNK if the log
   likelihood score
%per frame is below the fixed threshold for the actual bird
    class.
```

```
%22x1 Matrise med terskler (inkl. terskel for pause som er
   satt til -1000)
terskel=zeros(22,1);
terskel(1,1)=-82.0; terskel(1,2)=-97.0;
terskel(1,3)=-99.5; terskel(1,4)=-80.0;
terskel(1,5)=-95.0; terskel(1,6)=-102.0;
terskel(1,7)=-102.0; terskel(1,8)=-100.0;
terskel(1,9)=-103.5; terskel(1,10)=-104.5;
terskel(1,11)=-102.0; terskel(1,12)=-96.6;
terskel(1,13)=-99.4; terskel(1,14)=-104.0;
terskel(1,15)=-101.5; terskel(1,16)=-92.0;
terskel(1,17)=-106.0; terskel(1,18)=-96.0;
terskel(1,19)=-105.0; terskel(1,20)=-95.0;
terskel(1,21)=-96.5; terskel(1,22)=-1000;

classfile = 'class_pause_ny.list';
class = struct('memb',{{}});
fclass = fopen(classfile, 'r');
Nclass = 0;

while 1
    line = fgetl(fclass);
    if ~ischar(line), break, end
    Nclass = Nclass + 1;
    class.memb(Nclass) = {line};
end

for i = 1:21

    recnr = num2str(i);
    sourcefilename = strcat('Results_hmm/tot_anti_',recnr,'
       .rec');
    destinationfilename = strcat('Results_hmm/tot_unk_anti_
       ',recnr,'.rec');
    fsource = fopen(sourcefilename,'r');
    mlfline = fgetl(fsource);
    fdestination = fopen(destinationfilename,'w');
    fwrite(fdestination,mlfline); %write top line (#!MLF!#)
    fprintf(fdestination,'\r\n');
    Nfiles = 126;

    for j=1:Nfiles

        fileline = fgetl(fsource);
        fwrite(fdestination,fileline);
        fprintf(fdestination,'\r\n');

        fileline = fgetl(fsource);

        while(~(strcmp(fileline,'.')))

        scanfile = sscanf(fileline,'%d %d %s %d');
        start = num2str(scanfile(1));
        stop = num2str(scanfile(2));
        recog_bird = char(scanfile(3:length(scanfile)-1))';
        logscore = num2str(scanfile(length(scanfile)));
        antallrammer = num2str((scanfile(2)-scanfile(1))
            /200000);
        for u = 1:22
            if strcmp(recog_bird,class.memb{u})
                terskel_index = u;
            end
        end
            if str2num(logscore) < terskel(1,terskel_index)
                ;

                fwrite(fdestination,start);
                fwrite(fdestination,' ');
                fwrite(fdestination,stop);
                fwrite(fdestination,' ');
                fwrite(fdestination,'UNK');
                fwrite(fdestination,' ');
                fwrite(fdestination,logscore);
                fprintf(fdestination,'\r\n');
            else
                fwrite(fdestination,start);
                fwrite(fdestination,' ');
                fwrite(fdestination,stop);
                fwrite(fdestination,' ');
                fwrite(fdestination,recog_bird);
                fwrite(fdestination,' ');
                fwrite(fdestination,logscore);
                fprintf(fdestination,'\r\n');
            end
            fileline = fgetl(fsource);
        end
        fprintf(fdestination,'.');
        fprintf(fdestination,'\r\n');
    end
    fclose(fsource);
    fclose(fdestination);
end
```

## 8: Replace recognition label with ”UNKbirdclass”

```
%Script generating "tot_mult_unk_anti_i.rec". Read the
   files "tot_anti_i.rec"
%and replaces the recognition labels with UNKbirdclass (21
   different UNK classes
%if the log likelihood score per frame is below the fixed
%threshold for the actual bird class.

%22x1 Matrise med terskler (inkl. terskel for pause som er
   satt til -1000)
terskel=zeros(22,1);
terskel(1,1)=-82.0; terskel(1,2)=-97.0;
terskel(1,3)=-99.5; terskel(1,4)=-80.0;
terskel(1,5)=-95.0; terskel(1,6)=-102.0;
terskel(1,7)=-102.0; terskel(1,8)=-100.0;
terskel(1,9)=-103.5; terskel(1,10)=-104.5;
terskel(1,11)=-102.0; terskel(1,12)=-96.6;
terskel(1,13)=-99.4; terskel(1,14)=-104.0;
terskel(1,15)=-101.5; terskel(1,16)=-92.0;
terskel(1,17)=-106.0; terskel(1,18)=-96.0;
terskel(1,19)=-105.0; terskel(1,20)=-95.0;
terskel(1,21)=-96.5; terskel(1,22)=-1000;

classfile = 'class_pause_ny.list';
class = struct('memb',{{}});
fclass = fopen(classfile, 'r');
Nclass = 0;

while 1
    line = fgetl(fclass);
    if ~ischar(line), break, end
    Nclass = Nclass + 1;
    class.memb(Nclass) = {line};
end

for i = 1:21

    recnr = num2str(i);
```

```
sourcefilename = strcat('Results_hmm/tot_anti_',recnr,'
    .rec');
destinationfilename = strcat('Results_hmm/
    tot_mult_unk_anti_',...
    recnr,'.rec');
fsource = fopen(sourcefilename,'r');
mlfline = fgetl(fsource);
fdestination = fopen(destinationfilename,'w');
fwrite(fdestination,mlfline); %write top line (#!MLF!#)
fprintf(fdestination,'\r\n');
Nfiles = 126;

for j=1:Nfiles

    fileline = fgetl(fsource);
    fwrite(fdestination,fileline);
    fprintf(fdestination,'\r\n');

    fileline = fgetl(fsource);

    while(~(strcmp(fileline,'.')))

    scanfile = sscanf(fileline,'%d %d %s %d');
    start = num2str(scanfile(1));
    stop = num2str(scanfile(2));
    recog_bird = char(scanfile(3:length(scanfile)-1))';
    logscore = num2str(scanfile(length(scanfile)));
    antallrammer = num2str((scanfile(2)-scanfile(1))
        /200000);
    for u = 1:22
        if strcmp(recog_bird,class.memb{u})
            terskel_index = u;
        end
    end
        if str2num(logscore) < terskel(1,terskel_index)
            ;

            fwrite(fdestination,start);
            fwrite(fdestination,' ');
            fwrite(fdestination,stop);
            fwrite(fdestination,' ');
            %unknr = num2str(terskel_index);
            unkname = lower(class.memb{terskel_index});
            unkstring = strcat('UNK',unkname);
            fwrite(fdestination,unkstring);
```

```
                fwrite(fdestination,' ');
                fwrite(fdestination,logscore);
                fprintf(fdestination,'\r\n');
            else
                fwrite(fdestination,start);
                fwrite(fdestination,' ');
                fwrite(fdestination,stop);
                fwrite(fdestination,' ');
                fwrite(fdestination,recog_bird);
                fwrite(fdestination,' ');
                fwrite(fdestination,logscore);
                fprintf(fdestination,'\r\n');
            end
            fileline = fgetl(fsource);
        end
        fprintf(fdestination,'.');
        fprintf(fdestination,'\r\n');
    end
    fclose(fsource);
    fclose(fdestination);
end
```

# E: Matlab scripts for decision

## 1: Decision made on frame level (without post processor)

```
%Script for making decision on the most (3most) frequently
   occuring bird class.
%Generates histograms for input to post processor
%Expects file called setsizes.txt to exist. Ascii file with
    matrix 5x2
%giving sizes for train/test for the five datasets

% set ={1,2,3,4,5} dataset choice while type ={1,2} (choose
    train or respectively test set)
typestruct = struct('typer',{{}});
typestruct.typer(1) = {'train'};
typestruct.typer(2) = {'test'};
mix = 32; pm = 100;

% mix = number of mixtures
% pm is used penalty value in HVite
pm = abs(pm);
% mix and pm is only used for naming the rec-file of
    interest

% number of 100 nsec elements in a frame of 20 msec
scale = 20*10^4;

classfile = 'class_pause_ny.list'; % the handy file with
    legal classes ...
fclass = fopen(classfile,'r');
class = struct('memb',{{}});
fclass = fopen(classfile,'r');
Nclass = 0;
while 1
    line = fgetl(fclass);
    if ~ischar(line), break, end
    Nclass = Nclass +1;
    class.memb(Nclass) = {line};
end

Nclass_birds = Nclass -1;  % last modell named pause is not
     a class ...

load('setsizes.txt','-ascii');

% set ={1,2,3,4,5} dataset choice while type ={1,2} (
    respectively train and test set)
Perr_avg = zeros(2,3);

for type = 1:2
    for set = 1:5

        Nfiles = setsizes(set,type);
        Perr =zeros(1,3);

        % designing rec-file name
        mixnum = num2str(mix);
        pmnum =num2str(pm);
        setchar=num2str(set);
        typechar = char(typestruct.typer(type));
        recfile = strcat('Results_hmm/',typechar,setchar,'_
            ',mixnum,'_pm_',pmnum,'.rec');
        filename = strcat('Results_hmm/',typechar,setchar);
        disp([recfile '  ' filename])
```

```
frame_distr = zeros(Nfiles,Nclass);
file_class = zeros(Nfiles,1);
file_rec = zeros(Nfiles,3);
num_hit = file_class;
conf = zeros(Nclass, Nclass);

frec = fopen(recfile,'r');
line2 = fgetl(frec); % MLF-line

for kfile = 1: Nfiles

    % Read rec-file
    trec =[];
    sind_old2 = 1;
    linex =fgetl(frec); % filename
    for i = 1: Nclass_birds
        filefind = strfind(linex,char(class.memb(i)
            ));
        if(length(filefind) > 0)
            file_class(kfile) = i;  % correct class
            num_hit(kfile) = num_hit(kfile) +1;  %
                all elements should be 1 at the end
                ...
        end
    end

    line2 = fgetl(frec);
    while(length(line2) > 4) % until dot - i.e.
        file end
        ss2 = sscanf(line2,'%d %d %s');
        cl2 =  char(ss2(3:length(ss2)-1))';
        for i = 1: Nclass
            if (strcmp(char(class.memb(i)),cl2) ==
                1)
                 tars2 = i;
            end
        end
        sind2 = floor(ss2(2)/scale);  % last frame
        seglen2 = length(sind_old2+1:sind2);
        sind_old2 = sind2;
        trec = [trec; tars2*ones(seglen2,1)];
        line2 = fgetl(frec);
    end % end acoustic file
```

```
        %if(rem(kfile-1,25) == 0)
            % disp([kfile length(trec) length(tlab)])
        %end

        for i =1:Nclass
            jnum = find(trec == i);
            frame_distr(kfile,i) = length(jnum);
        end
        [y,ind] = sort(frame_distr(kfile,1:Nclass_birds
           ),'descend');
        file_rec(kfile,:) = ind(1:3);

    end % end rec-file

    fclose(frec);

    conf2 = conf;
    conf3 = conf;
    for i = 1:Nclass
        for j = 1:Nclass
            conf(i,j) = length(find((file_class==i)&(
               file_rec(:,1)==j)));
        end
    end

    numcorr2 = 0; numcorr3 = 0;
    for k = 1: Nfiles
        j = file_class(k);
        i2 = find(file_rec(k, 1:2)==j); numcorr2 =
           numcorr2 + length(i2);
        i3 = find(file_rec(k, 1:3)==j); numcorr3 =
           numcorr3 + length(i3);
    end

    disp('Percentage error')
    numcorr = sum(diag(conf));
    totnum = sum(sum(conf));
    Perr(1) = 100*(1 -numcorr/totnum);
    Perr(2) = 100*(1-numcorr2/totnum);
    Perr(3) = 100*(1-numcorr3/totnum);
    disp(Perr)
    Perr_avg(type,:) = Perr_avg(type,:) + Perr/5;
    classhist= frame_distr(:,1:Nclass_birds);
    target = file_class;
    classhistn = classhist;
```

```
        for i =1:Nfiles
            classhistn(i,:) =classhist(i,:)/sum(classhist(i
               ,:));
        end

        % filename = strcat('Results_hmm/',typechar,setchar
           );
        save(filename,'classhistn','target','Perr','
           Nclass_birds')

    end
end

disp(Perr_avg)
```

## 2: Post processing. Decision made on output of this post processor

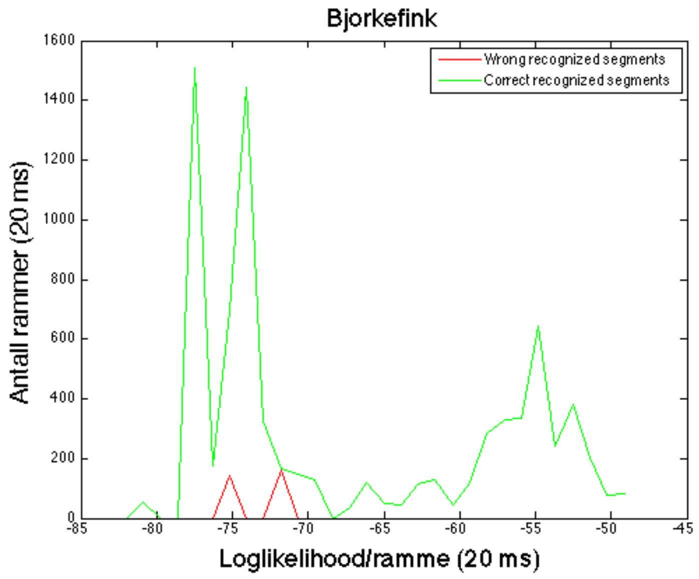
```
%Script taking histograms from "frame_hist_all.m" as input
   and do a post processing
%before making the decision

clear
typestruct = struct('typer',{{}});
typestruct.typer(1) = {'train'};
typestruct.typer(2) = {'test'};
Perr_avg = zeros(2,3);

for set = 1:5

    setchar=num2str(set);
    disp(['set nr ' setchar])
    typechar = char(typestruct.typer(1));  % trainset
    filename1 = strcat('Results_hmm/',typechar,setchar);
    load(filename1)
    trainhist = classhistn';
    [Nclass,Ntrain] = size(trainhist);
    traintar = target';
    trainhist = [trainhist; (0*trainhist(1,:)+1)];
    Ttrain = zeros(Nclass,Ntrain);

    typechar = char(typestruct.typer(2));   % testset
    filename2 = strcat('Results_hmm/',typechar,setchar);
    load(filename2)
    testhist = classhistn';
    [Nclass,Ntest] = size(testhist);
    testhist = [testhist; (0*testhist(1,:)+1)];
    testtar = target';
    Ttest = zeros(Nclass,Ntest);

    filename_ut = strcat('Results_hmm/post_set',setchar);
    for cl = 1:Nclass
        i=[]; j =[];
        i = find(traintar == cl);
        Ttrain(cl,i) = Ttrain(cl,i) + 1;
        j = find(testtar == cl);
        Ttest(cl,j) = Ttest(cl,j) + 1;
    end

    % Training and testing start ...
    Itershow = 100;
    Totiter = 400;
    Edpl = []; Etpl = []; rdpl = []; rtpl = [];

    % Initialiserer nettverksparametre

    w=.01*(rand(Nclass ,Nclass +1 )-.5) ;

    % Definerer parameter-gradienter (lik 0)

    dw=0*w;  dwo=0*w;  wo =w;

% Definerer individuelle steg-faktorer for hver parameter

    sinit=1;
    sw=dw+10*sinit;

% Definerer diverse (ikke alt er i bruk). Studer program !

    ud=.15;
    alphax=.5/Ntrain;
    mink=.5;

    Edvec=[];
    Etvec=[];
    nbvec=[];
    Ed_old=-1000;
    Et_old=-1000;
    rt_old=0;


% velg ulinearitet paa utgang
% nonlin_ut=0 -> logsig; nonlin_ut=1 -> softmax (kontakt
   meg)



% Gjoer Totiter iterasjoner (kan muligens reduseres noe ..)

    iter=0;
    while(iter <= Totiter)
            % Definerer forvekslingsmatriser for hhv 1best,
                2best og 3best

        cdev1=zeros(Nclass,Nclass); cdev2=cdev1; cdev3=
           cdev1;
        ctest1=cdev1; ctest2=cdev1; ctest3=cdev1;

    % Nullstiller deriverte

        dw=0*w;   dwo=0*w;
        iter=iter+1;

    % Forover beregninger

        zd = w*trainhist;
        zt = w*testhist;
        bd=exp(zd);
        bd=bd./(ones(Nclass,1)*sum(bd));
        bt=exp(zt);
        bt=bt./(ones(Nclass,1)*sum(bt));

        % Beregner trenings og test kriterier
        Ed=sum(sum(Ttrain.*log(bd)));
        Et=sum(sum(Ttest.*log(bt)));


        Ed=100*Ed/Ntrain;
        Et=100*Et/Ntest;
```

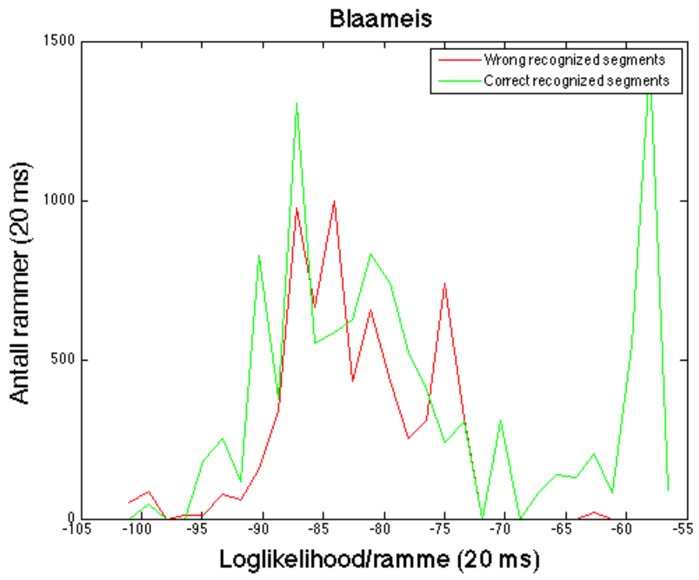
```
% Beregner forvekslingsmatriser
    classdev = zeros(Ntrain,3); classtest = zeros(Ntest
        ,3);
    [y,ind] = sort(bd,'descend');
    classdev = ind(1:3,:);
    [y,ind] = sort(bt,'descend');
    classtest = ind(1:3,:);

    for i=1:Nclass
        for j=1:Nclass
            cdev1(i,j) = length(find((traintar==i)&(
                classdev(1,:)==j)));
            ctest1(i,j) = length(find((testtar==i)&(
                classtest(1,:)==j)));
        end
    end

    numcorr1d = sum(diag(cdev1)); numcorr1t = sum(diag(
        ctest1));
    totnumd = sum(sum(cdev1)); totnumt = sum(sum(ctest1
        ));

    numcorr2d = 0; numcorr3d = 0; numcorr2t = 0;
        numcorr3t = 0;
    for k = 1: Ntrain
        j = traintar(k);
        i2 = find(classdev(1:2,k)==j); numcorr2d =
            numcorr2d + length(i2);
        i3 = find(classdev(1:3,k)==j); numcorr3d =
            numcorr3d + length(i3);
    end

    for k = 1: Ntest
        j = testtar(k);
        i2 = find(classtest(1:2,k)==j); numcorr2t =
            numcorr2t + length(i2);
        i3 = find(classtest(1:3,k)==j); numcorr3t =
            numcorr3t + length(i3);
    end

    Perrd = zeros(1,3); Perrt = Perrd;
    Perrd(1) = 100-100*numcorr1d/totnumd;
    Perrd(2) = 100-100*numcorr2d/totnumd;
    Perrd(3) = 100-100*numcorr3d/totnumd;
    Perrt(1) = 100-100*numcorr1t/totnumt;
    Perrt(2) = 100-100*numcorr2t/totnumt;
    Perrt(3)= 100-100*numcorr3t/totnumt;

% disp([Perrd Perrt])

% Kun hvis kontinuerlig plot er oenskelig

    Edvec=[Edvec Ed];
    Etvec=[Etvec Et];
    nbvec=[nbvec iter];

% Beregner antall riktige (1best) i prosent

    rd=100*sum(diag(cdev1))/sum(sum(cdev1));
    rt=100*sum(diag(ctest1))/sum(sum(ctest1));

% Skriv til skjerm hver Itershow iterasjon (valgbar)

    Edpl = [Edpl Ed]; Etpl = [Etpl Et]; rdpl =[rdpl rd
       ]; rtpl = [rtpl rt];

    if(rem(iter-1,Itershow)==0)
        iterm=rem(iter,100);
        iter100=(iter-iterm)/100;

        format bank
        disp([Perrd Perrt])
        subplot(211), plot(1:iter,Edpl,'r',1:iter,Etpl,
           'b')
        subplot(212), plot(1:iter,rdpl,'r',1:iter,rtpl,
           'b')
        pause(3)
    end


    % Gjem system og resultat for beste testresultat

    if(rt>rt_old)
        save(filename_ut,'w' ,'cdev1','ctest1','rt','rd
           ');
        rt_old=rt;
    end
```

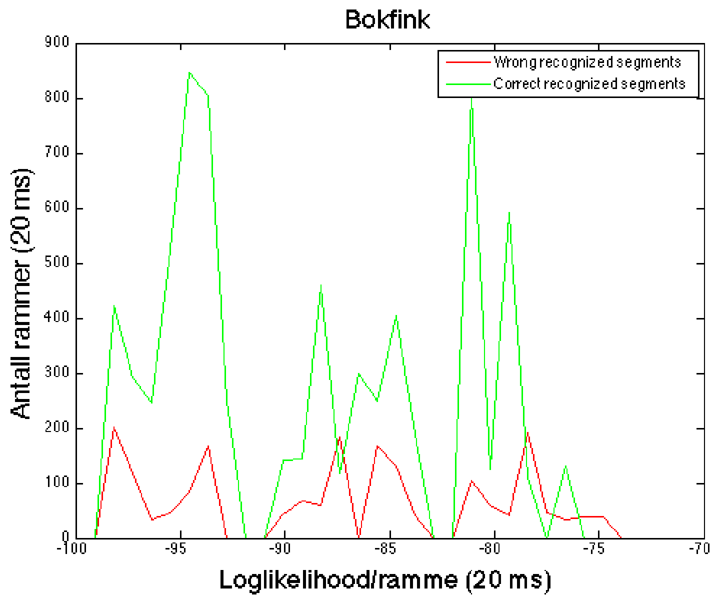
```
        % Oppdatering kun hvis kryssentropi ikke har minket
            med mer enn 10% (skal normalt oke)


         %  if(Ed -Ed_old > 0)
          if(Ed_old == Ed_old)

            % Bakoverforplantning av feil og beregning av
               delta-parametre

            dbd = -(bd - Ttrain);
            dw = alphax*dbd*trainhist';

            % Multiplikasjon med steg-faktorene

            dw=sw.*dw;

            % Oppdatering av stegfaktorene

            sw=sw+ud*(sw.*sign(dwo.*dw));


            % Gjem systemparametre fra forrige gang (trengs
                hvis feil neste gang
            % oeker med mer enn 10%)

            wo=w;


            % Oppdater parametre

            % w=w+dw;
            w = 0.999*w +dw;

            % Lagring av delta-parametre (trengs til aa
               beregne nye
            % stegfaktorer i neste iterasjon)

            dwo=dw;

            % Lagring av feilmaal (trengs for feilmaalsjekk
                i neste iterasjon)

            Ed_old=Ed;
            Et_old=Et;
        % Hvis feilmaal har oeket med mer enn 10% :

        else

            %disp('readjust')

            % Hent tilbake parametre fra forrige iterasjon

            w=wo;

            % Nullstill gamle deriverte

            dwo=0*w;

            % Reduser stegfaktorer

            sw=mink*sw;


        end

    end

    Perr_avg(1,:) = Perr_avg(1,:) + Perrd/5;
    Perr_avg(2,:) = Perr_avg(2,:) +  Perrt/5;

end

disp(Perr_avg)
```

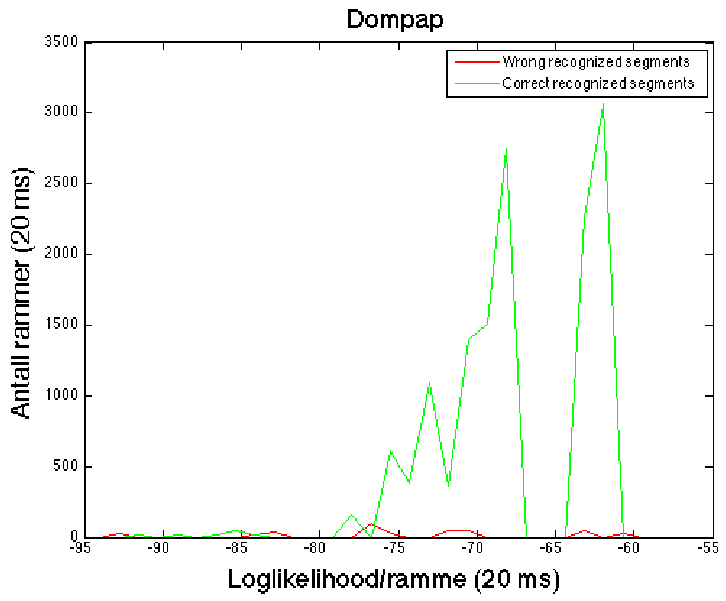
# F: Plots for finding thresholds

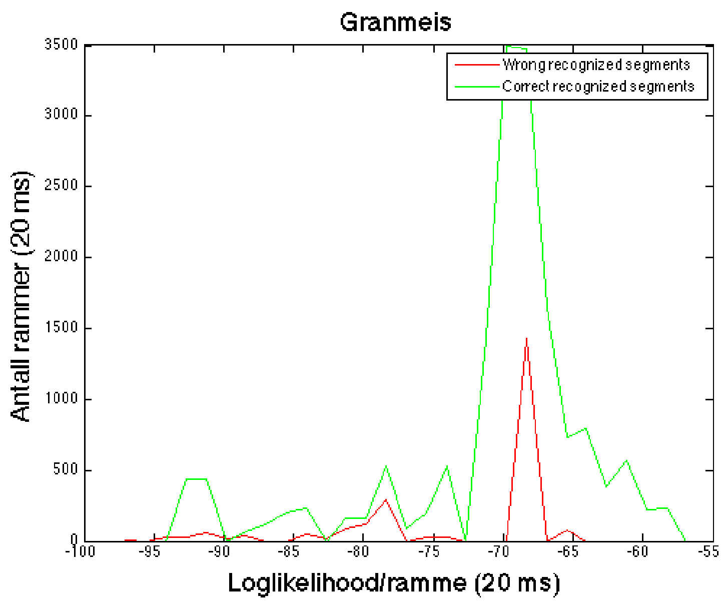
## 1: From training set

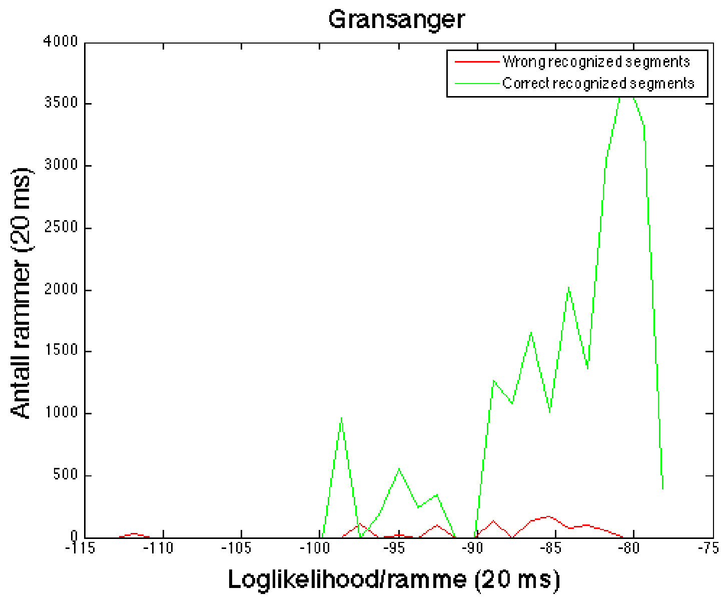


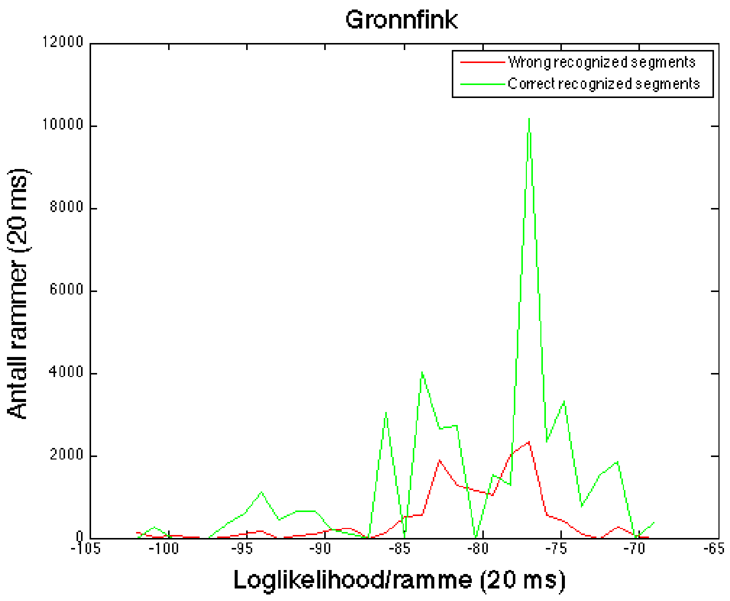


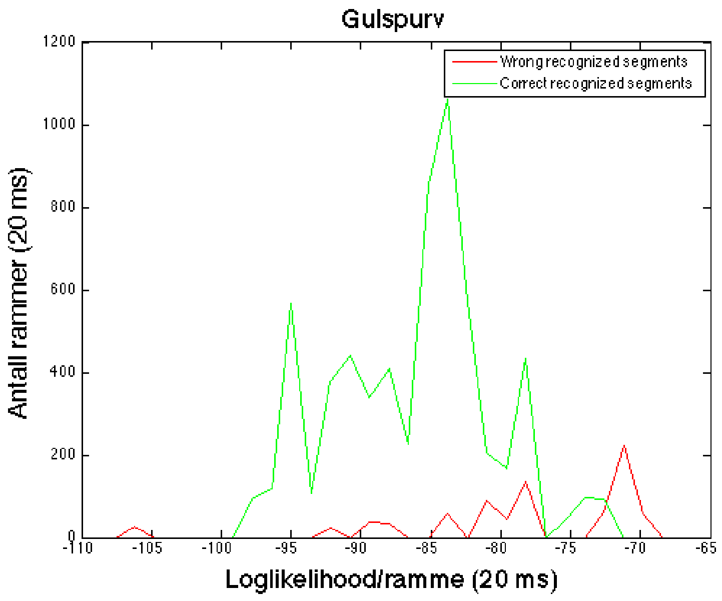

Bokfink
Wrong recognized segments
Correct recognized segments
Antall rammer (20 ms)
Loglikelihood/ramme (20 ms)
900
800
700
600
500
400
300
200
100
0
-100
-95
-90
-85
-80
-75
-70

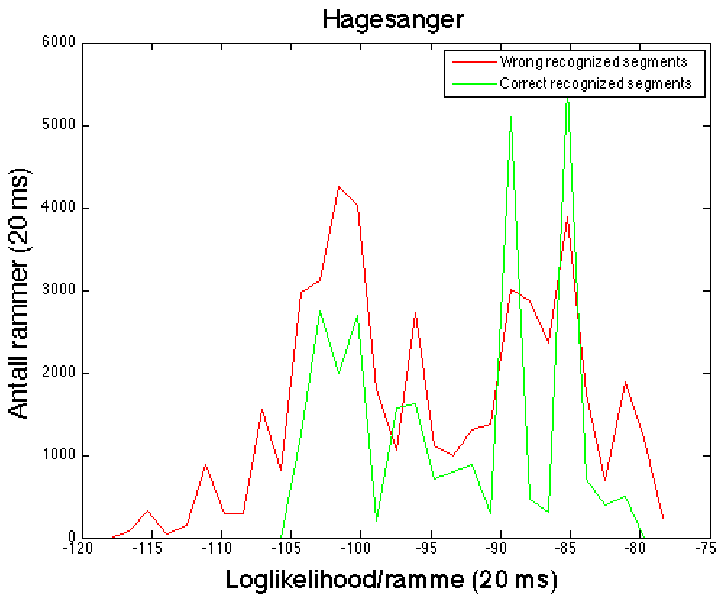

Dompap
Wrong recognized segments
Correct recognized segments
Antall rammer (20 ms)
Loglikelihood/ramme (20 ms)
3500
3000
2500
2000
1500
1000
500
0
-95
-90
-85
-80
-75
-70
-65
-60
-55

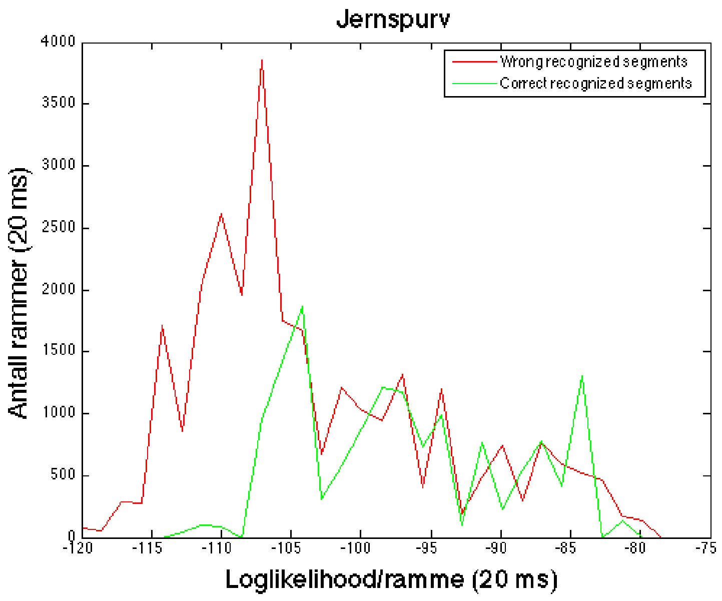

Granmeis
Wrong recognized segments
Correct recognized segments
Antall rammer (20 ms)
Loglikelihood/ramme (20 ms)
3500
3000
2500
2000
1500
1000
500
0
-100
-95
-90
-85
-80
-75
-70
-65
-60
-55

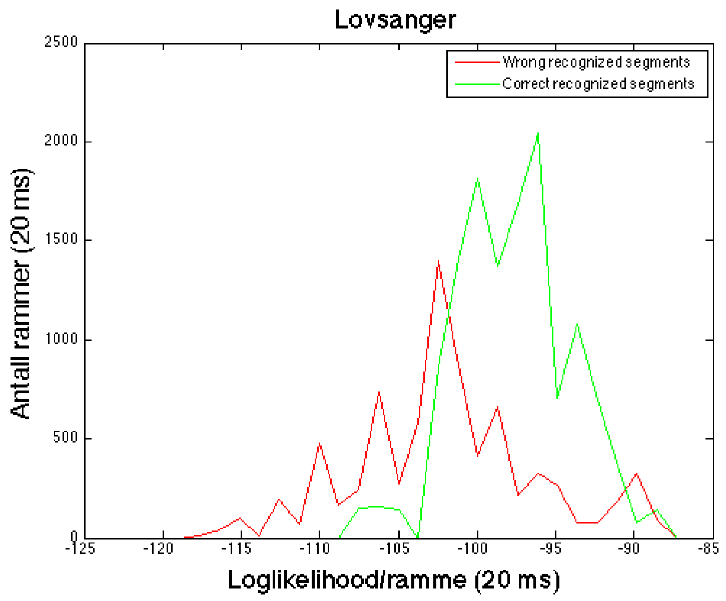

Gransanger
Wrong recognized segments
Correct recognized segments
Antall rammer (20 ms)
Loglikelihood/ramme (20 ms)
4000
3500
3000
2500
2000
1500
1000
500
0
-115
-110
-105
-100
-95
-90
-85
-80
-75

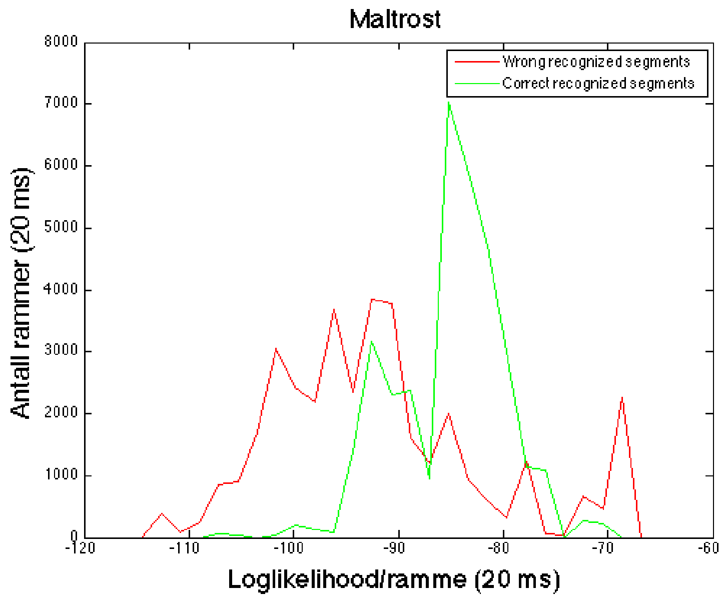

Gronnfink
Wrong recognized segments
Correct recognized segments
Antall rammer (20 ms)
Loglikelihood/ramme (20 ms)
12000
10000
8000
6000
4000
2000
0
-105
-100
-95
-90
-85
-80
-75
-70
-65

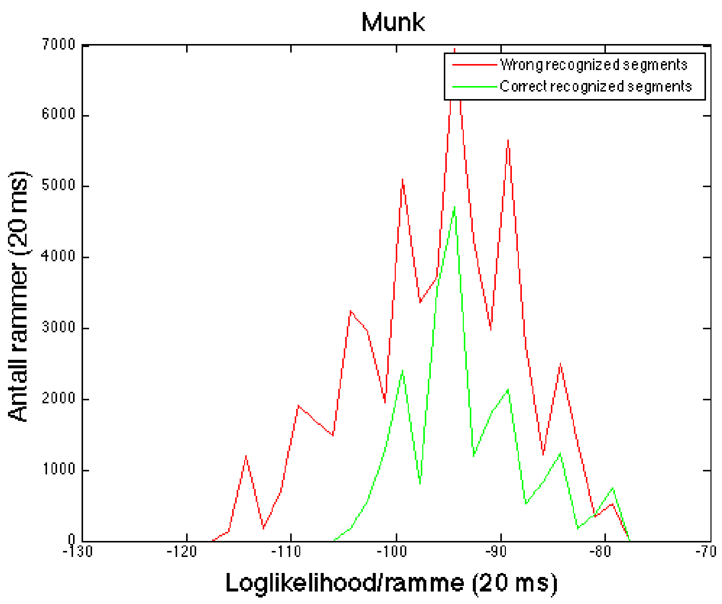

Gulspurv
Wrong recognized segments
Correct recognized segments
Antall rammer (20 ms)
Loglikelihood/ramme (20 ms)
1200
1000
800
600
400
200
0
-110
-105
-100
-95
-90
-85
-80
-75
-70
-65

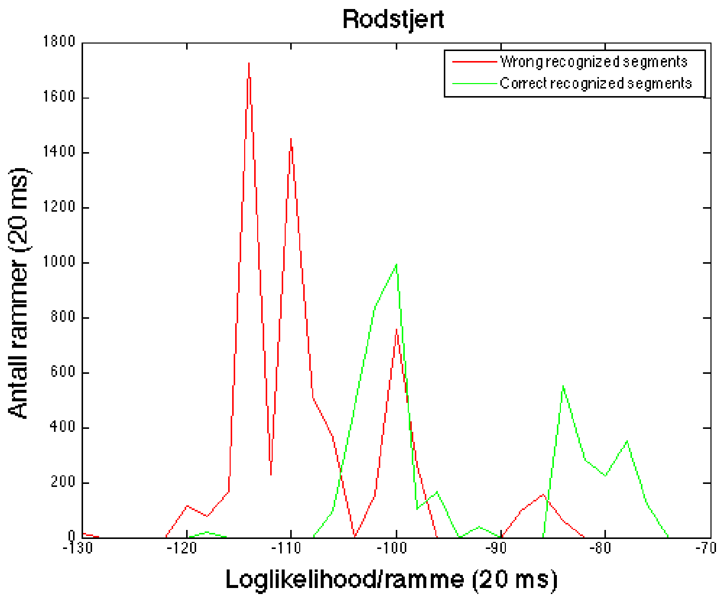

Hagesanger
Wrong recognized segments
Correct recognized segments
Antall rammer (20 ms)
Loglikelihood/ramme (20 ms)
6000
5000
4000
3000
2000
1000
0
-120
-115
-110
-105
-100
-95
-90
-85
-80
-75

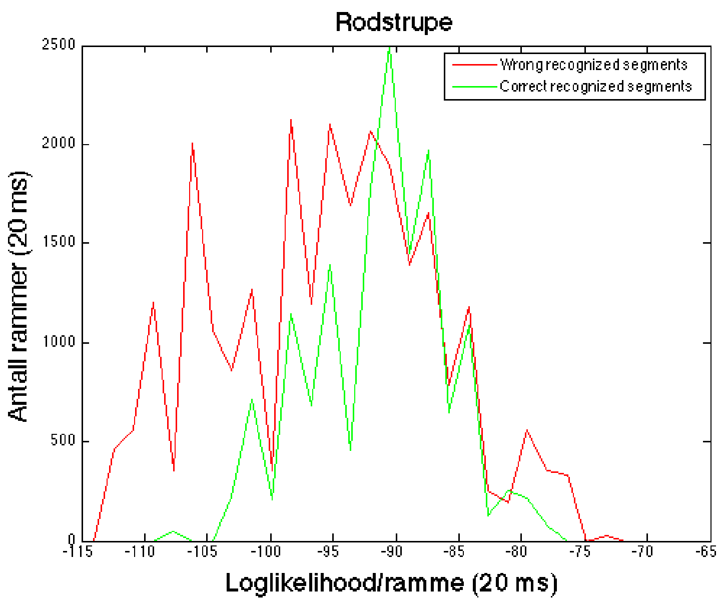

Jernspurv
Wrong recognized segments
Correct recognized segments
Antall rammer (20 ms)
Loglikelihood/ramme (20 ms)
4000
3500
3000
2500
2000
1500
1000
500
0
-120
-115
-110
-105
-100
-95
-90
-85
-80
-75

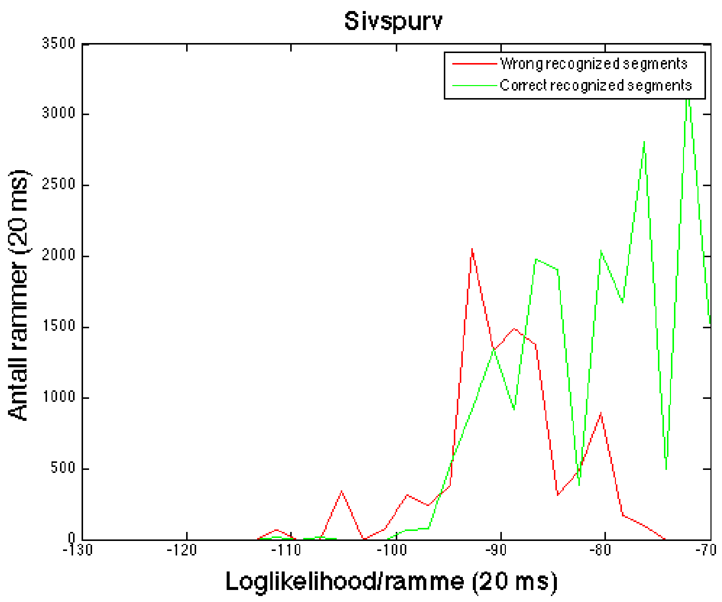

Lovsanger
Wrong recognized segments
Correct recognized segments
Antall rammer (20 ms)
Loglikelihood/ramme (20 ms)
2500
2000
1500
1000
500
0
-125
-120
-115
-110
-105
-100
-95
-90
-85

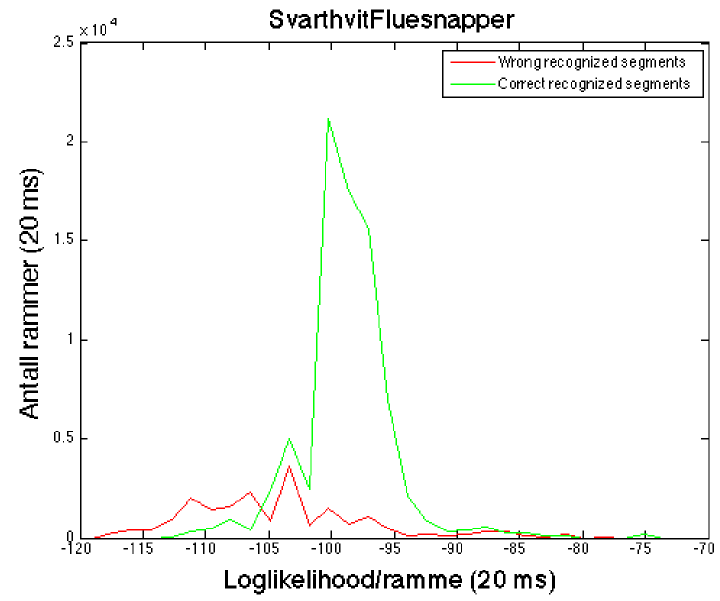

Maltrost
Wrong recognized segments
Correct recognized segments
Antall rammer (20 ms)
Loglikelihood/ramme (20 ms)
8000
7000
6000
5000
4000
3000
2000
1000
0
-120
-110
-100
-90
-80
-70
-60

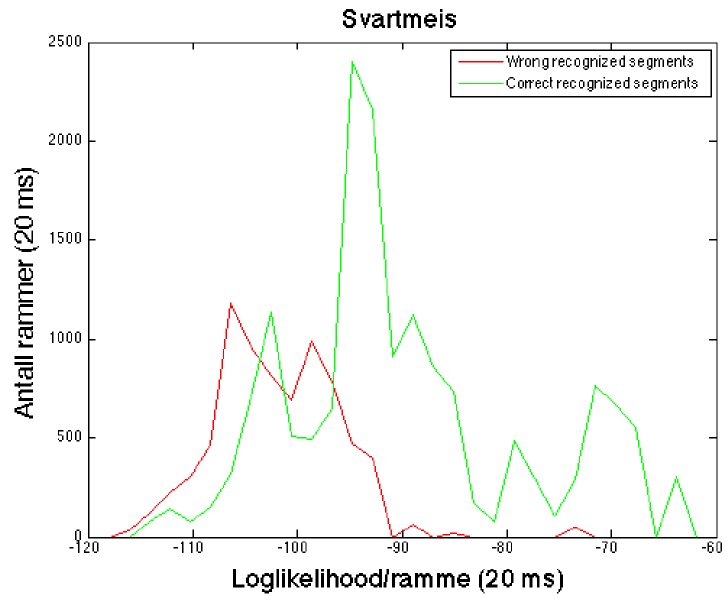

Munk
Wrong recognized segments
Correct recognized segments
Antall rammer (20 ms)
Loglikelihood/ramme (20 ms)
7000
6000
5000
4000
3000
2000
1000
0
-130
-120
-110
-100
-90
-80
-70

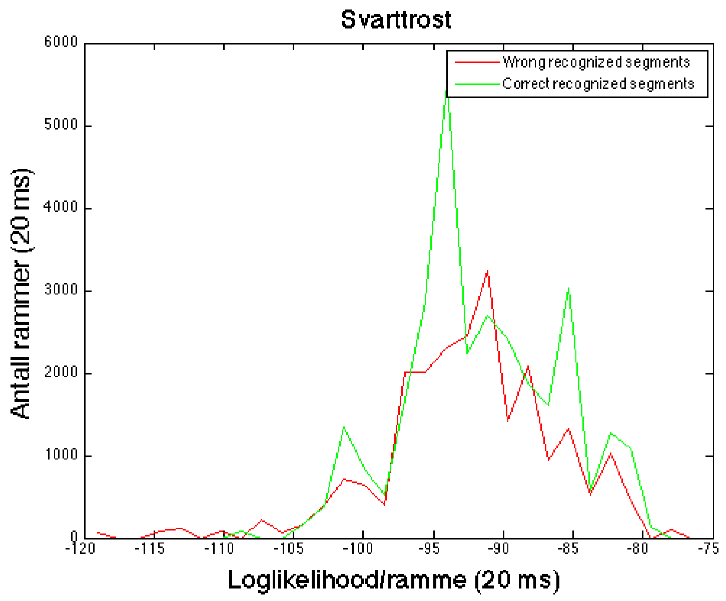

Rodstjert
Wrong recognized segments
Correct recognized segments
Antall rammer (20 ms)
Loglikelihood/ramme (20 ms)
1800
1600
1400
1200
1000
800
600
400
200
0
-130
-120
-110
-100
-90
-80
-70


Rodstrupe
Wrong recognized segments
Correct recognized segments
Antall rammer (20 ms)
Loglikelihood/ramme (20 ms)
2500
2000
1500
1000
500
0
-115
-110
-105
-100
-95
-90
-85
-80
-75
-70
-65

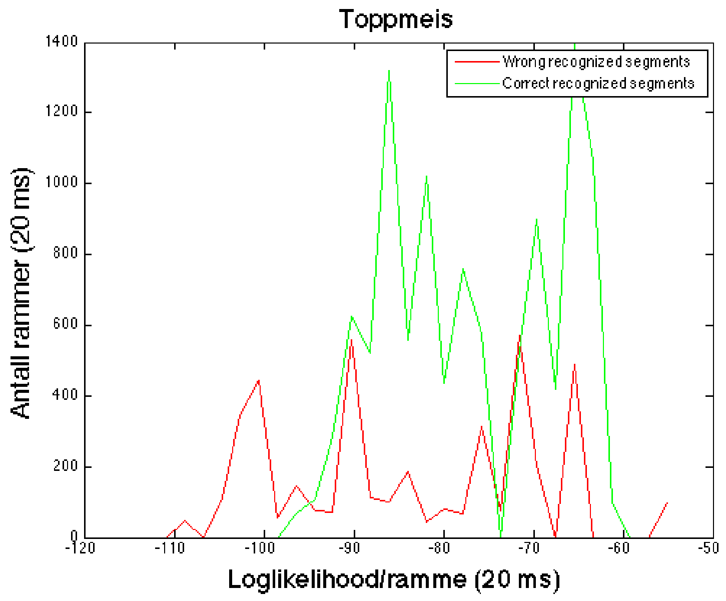

Sivspurv
Wrong recognized segments
Correct recognized segments
Antall rammer (20 ms)
Loglikelihood/ramme (20 ms)
3500
3000
2500
2000
1500
1000
500
0
-130
-120
-110
-100
-90
-80
-70


SvarthvitFluesnapper
x 10^4
2.5
2
1.5
1
0.5
0
-120
-115
-110
-105
-100
-95
-90
-85
-80
-75
-70
Wrong recognized segments
Correct recognized segments
Antall rammer (20 ms)
Loglikelihood/ramme (20 ms)

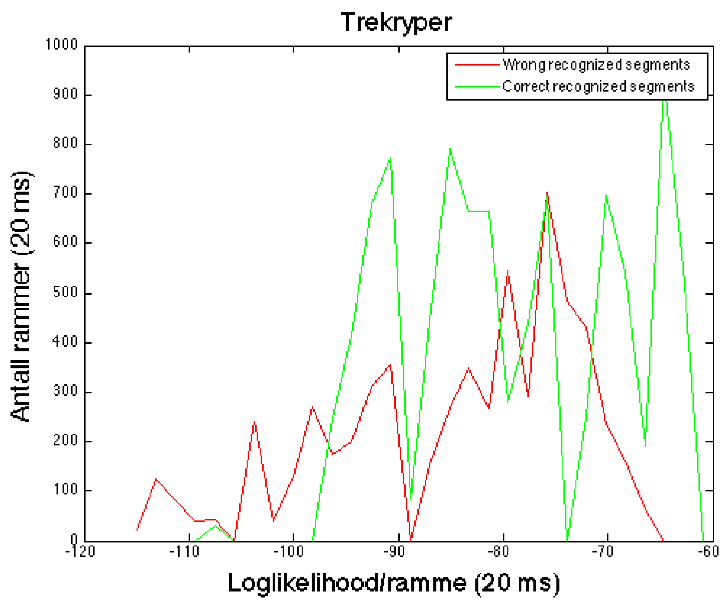

Svartmeis
2500
2000
1500
1000
500
0
-120
-110
-100
-90
-80
-70
-60
Wrong recognized segments
Correct recognized segments
Antall rammer (20 ms)
Loglikelihood/ramme (20 ms)



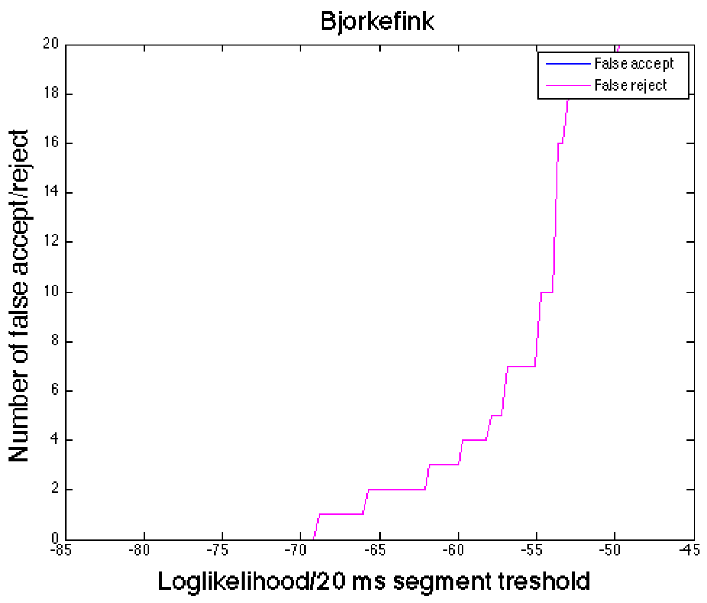
Svarttrost
Wrong recognized segments
Correct recognized segments
Antall rammer (20 ms)
Loglikelihood/ramme (20 ms)
6000
5000
4000
3000
2000
1000
0
-120
-115
-110
-105
-100
-95
-90
-85
-80
-75

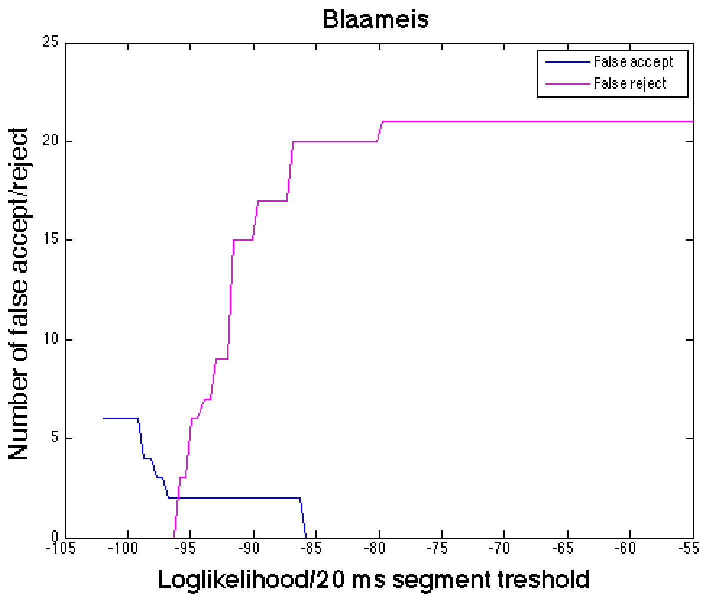


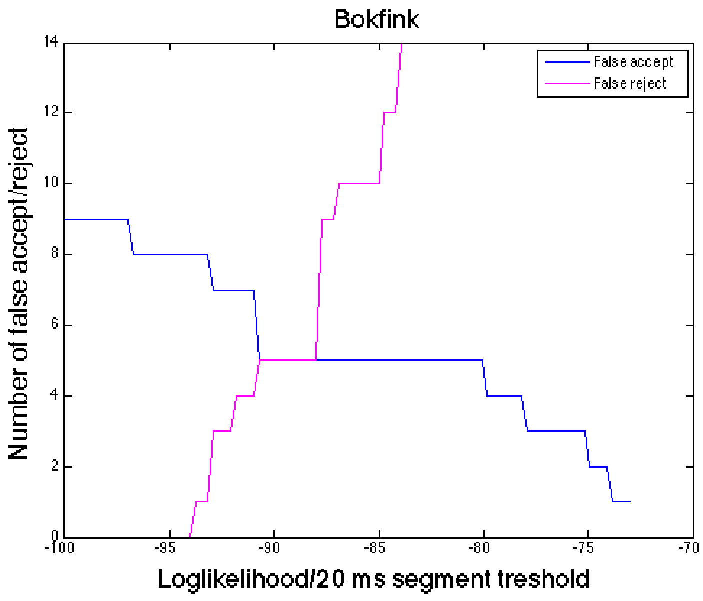
Toppmeis
Wrong recognized segments
Correct recognized segments
Antall rammer (20 ms)
Loglikelihood/ramme (20 ms)
1400
1200
1000
800
600
400
200
0
-120
-110
-100
-90
-80
-70
-60
-50

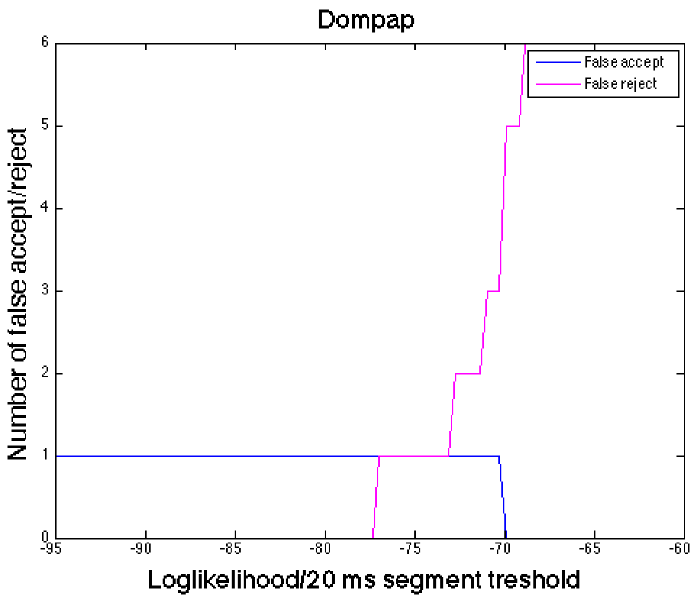


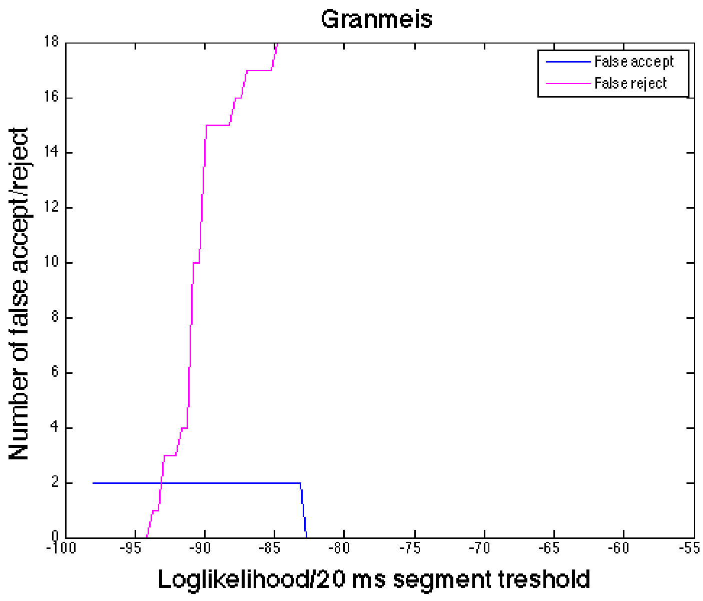
## 2: From test set

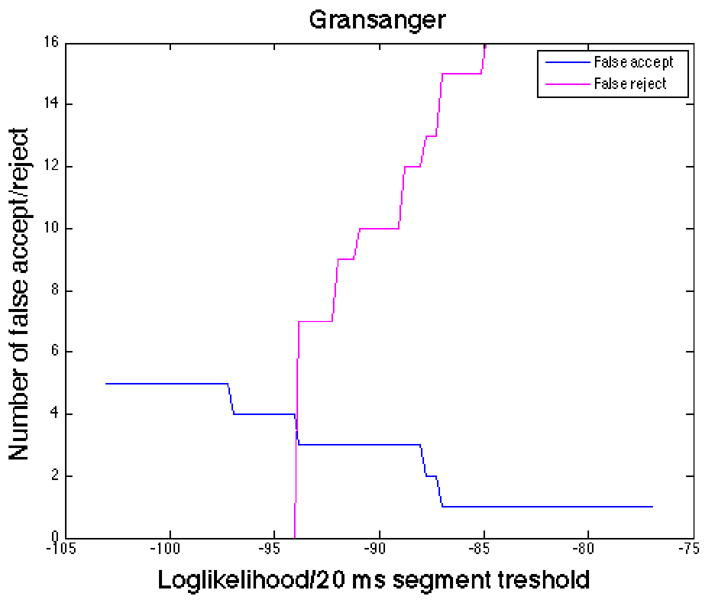


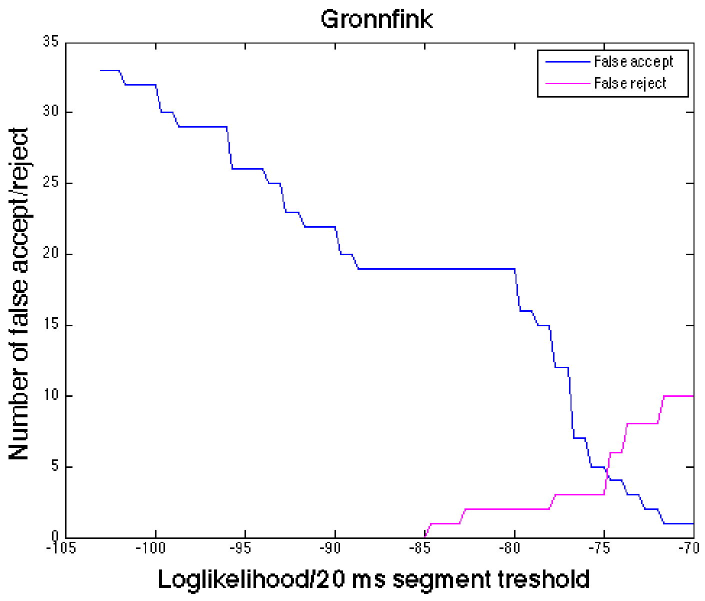

Blaameis
Number of false accept/reject
Loglikelihood/20 ms segment treshold
False accept
False reject
25
20
15
10
5
0
-105
-100
-95
-90
-85
-80
-75
-70
-65
-60
-55

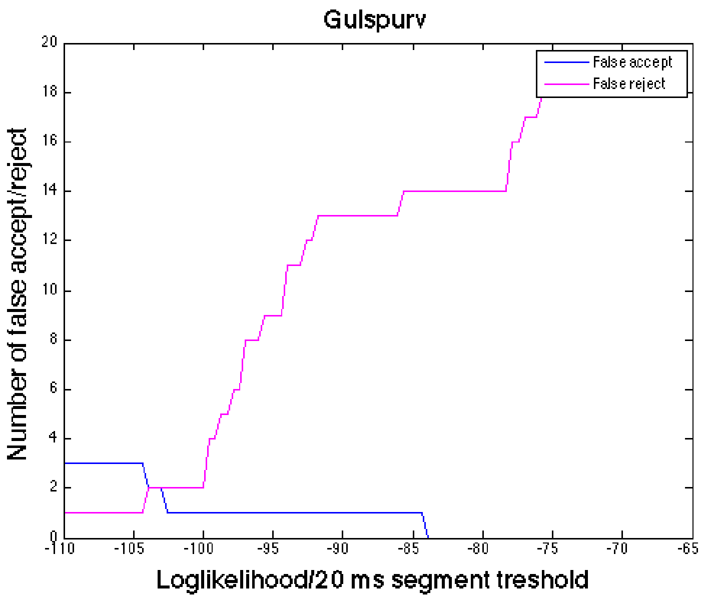

Bokfink
Number of false accept/reject
Loglikelihood/20 ms segment treshold
False accept
False reject
14
12
10
8
6
4
2
0
-100
-95
-90
-85
-80
-75
-70

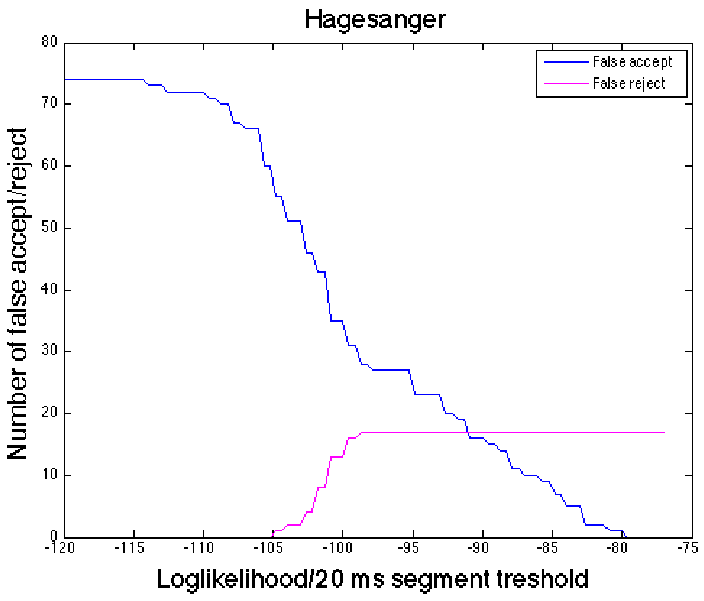

Dompap
Number of false accept/reject
Loglikelihood/20 ms segment treshold
False accept
False reject
0
1
2
3
4
5
6
-95
-90
-85
-80
-75
-70
-65
-60

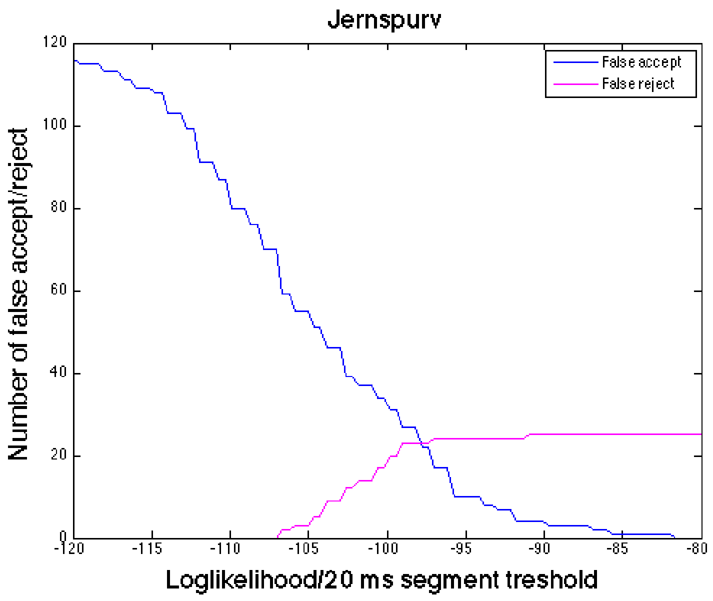

Granmeis
Number of false accept/reject
Loglikelihood/20 ms segment treshold
False accept
False reject
0
2
4
6
8
10
12
14
16
18
-100
-95
-90
-85
-80
-75
-70
-65
-60
-55

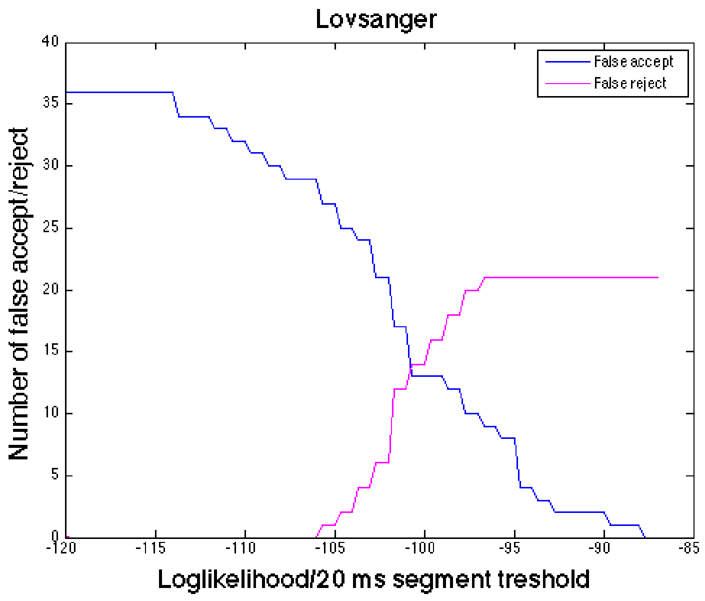

Gransanger
Number of false accept/reject
Loglikelihood/20 ms segment treshold
False accept
False reject
16
14
12
10
8
6
4
2
0
-105
-100
-95
-90
-85
-80
-75

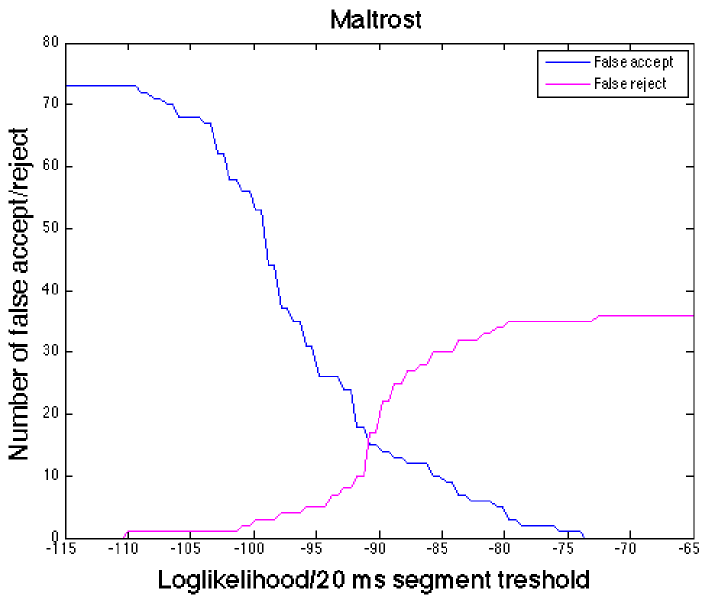

Gronnfink
Number of false accept/reject
Loglikelihood/20 ms segment treshold
False accept
False reject
35
30
25
20
15
10
5
0
-105
-100
-95
-90
-85
-80
-75
-70

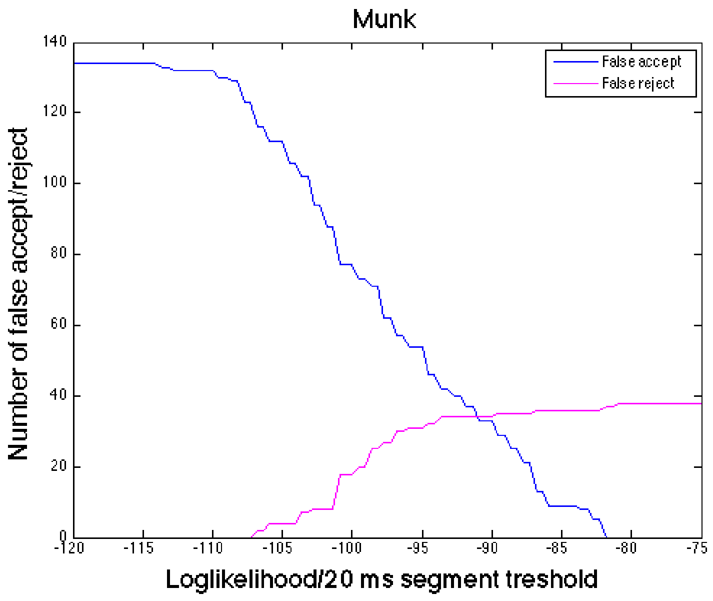

Gulspurv
Number of false accept/reject
Loglikelihood/20 ms segment treshold
False accept
False reject
20
18
16
14
12
10
8
6
4
2
0
-110
-105
-100
-95
-90
-85
-80
-75
-70
-65

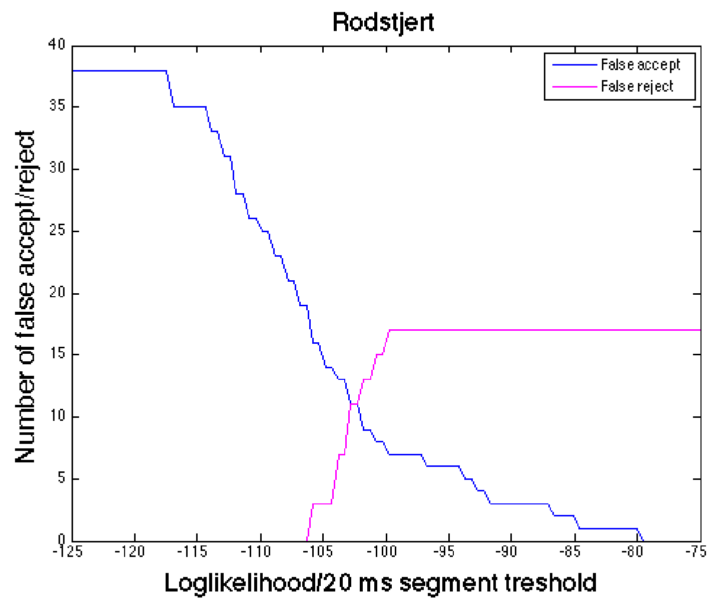

Hagesanger
Number of false accept/reject
Loglikelihood/20 ms segment treshold
False accept
False reject
80
70
60
50
40
30
20
10
0
-120
-115
-110
-105
-100
-95
-90
-85
-80
-75

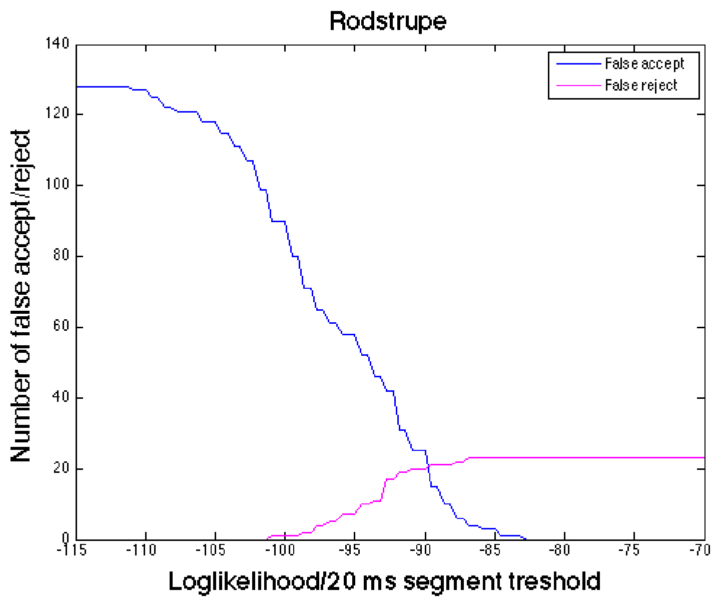

Jernspurv
Number of false accept/reject
Loglikelihood/20 ms segment treshold
False accept
False reject
120
100
80
60
40
20
0
-120
-115
-110
-105
-100
-95
-90
-85
-80

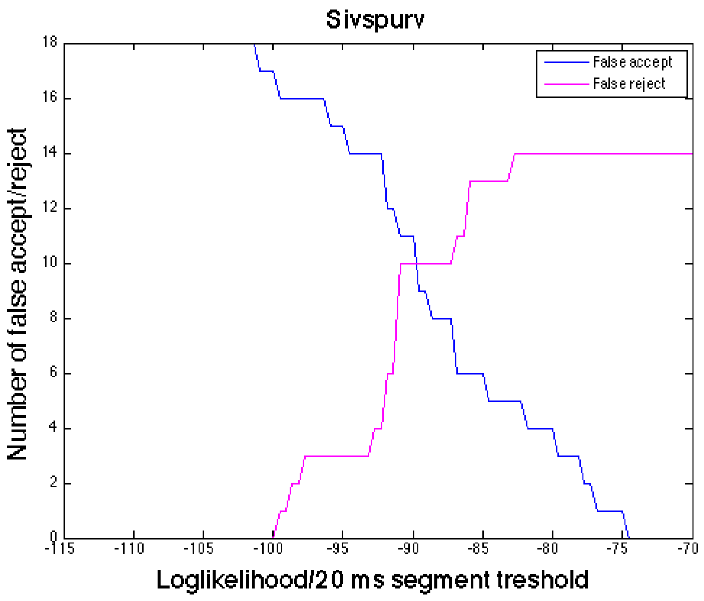

Lovsanger
Number of false accept/reject
Loglikelihood/20 ms segment treshold
False accept
False reject
40
35
30
25
20
15
10
5
0
-120
-115
-110
-105
-100
-95
-90
-85

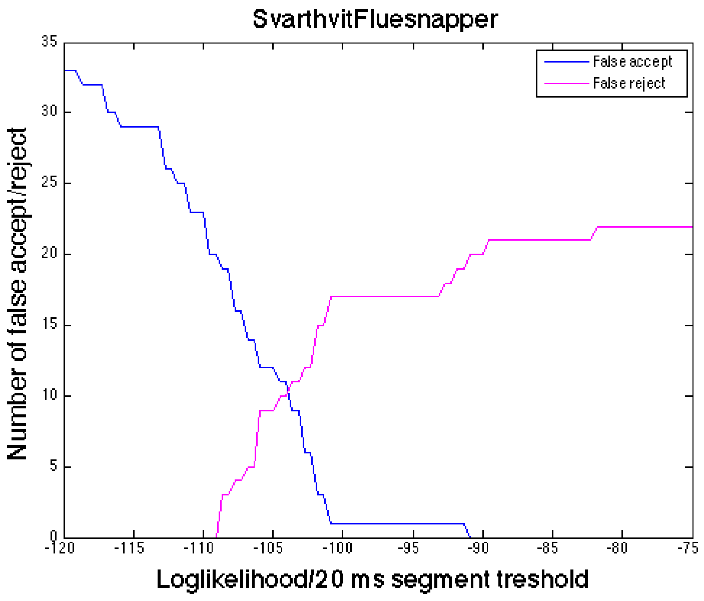

Maltrost
Number of false accept/reject
Loglikelihood/20 ms segment treshold
False accept
False reject
0
10
20
30
40
50
60
70
80
-115
-110
-105
-100
-95
-90
-85
-80
-75
-70
-65

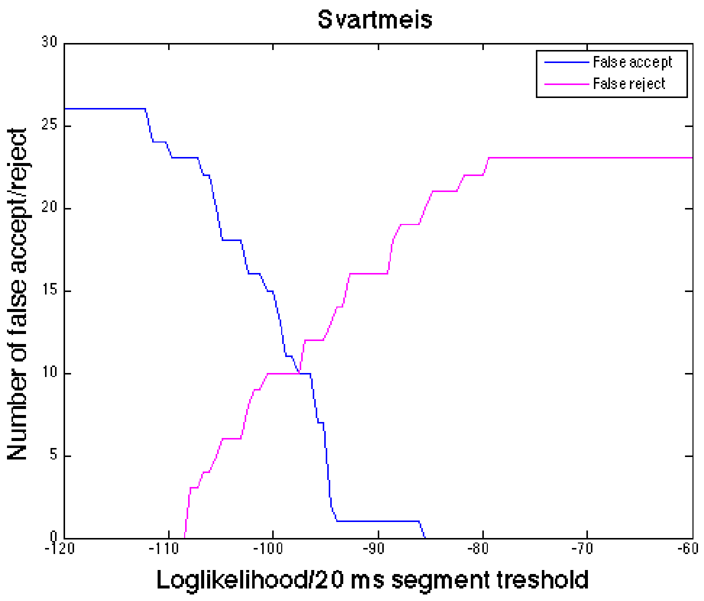

Munk
Number of false accept/reject
Loglikelihood/20 ms segment treshold
False accept
False reject
0
20
40
60
80
100
120
140
-120
-115
-110
-105
-100
-95
-90
-85
-80
-75

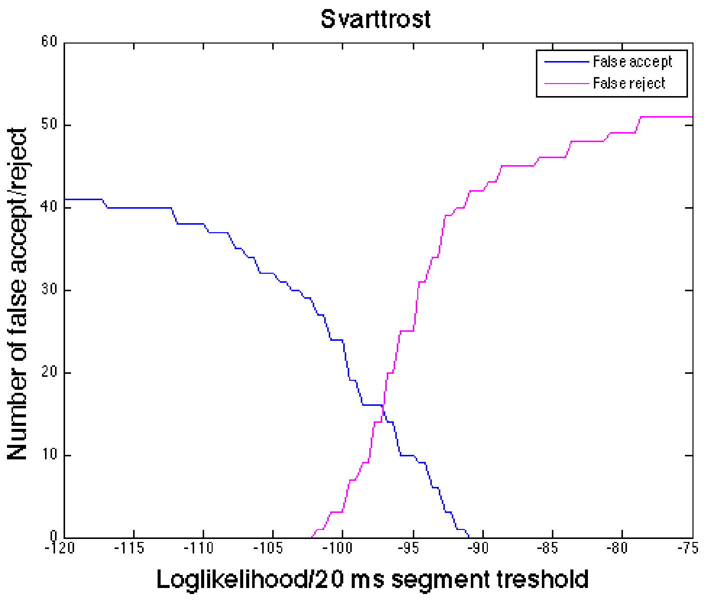

Rodstjert
Number of false accept/reject
Loglikelihood/20 ms segment treshold
False accept
False reject
40
35
30
25
20
15
10
5
0
-125
-120
-115
-110
-105
-100
-95
-90
-85
-80
-75

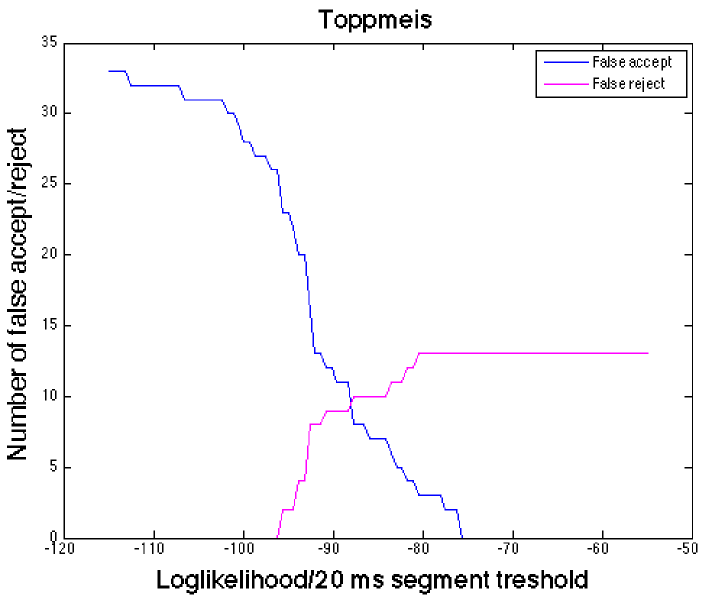

Rodstrupe
Number of false accept/reject
Loglikelihood/20 ms segment treshold
False accept
False reject
140
120
100
80
60
40
20
0
-115
-110
-105
-100
-95
-90
-85
-80
-75
-70

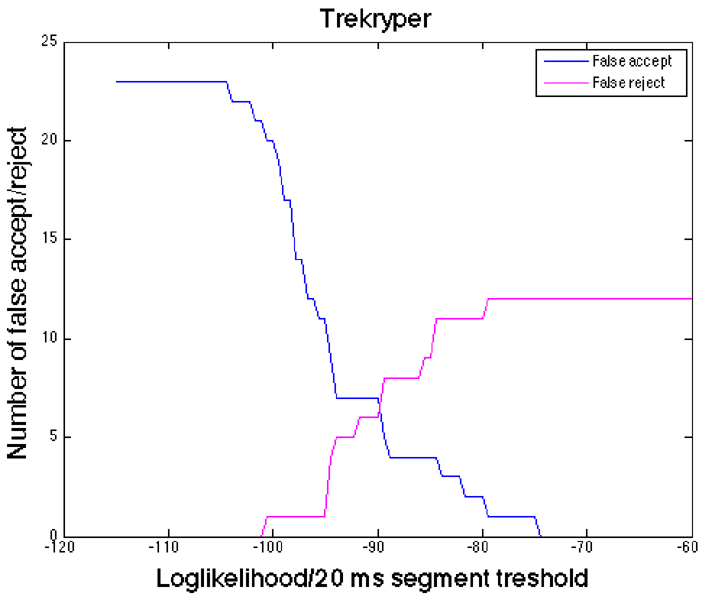

Sivspurv
False accept
False reject
Number of false accept/reject
Loglikelihood/20 ms segment treshold
0
2
4
6
8
10
12
14
16
18
-115
-110
-105
-100
-95
-90
-85
-80
-75
-70


SvarthvitFluesnapper
False accept
False reject
Number of false accept/reject
Loglikelihood/20 ms segment treshold
0
5
10
15
20
25
30
35
-120
-115
-110
-105
-100
-95
-90
-85
-80
-75


Svartmeis
Number of false accept/reject
Loglikelihood/20 ms segment treshold
False accept
False reject
30
25
20
15
10
5
0
-120
-110
-100
-90
-80
-70
-60


Svarttrost
Number of false accept/reject
Loglikelihood/20 ms segment treshold
False accept
False reject
60
50
40
30
20
10
0
-120
-115
-110
-105
-100
-95
-90
-85
-80
-75


Toppmeis
False accept
False reject
Number of false accept/reject
Loglikelihood/20 ms segment treshold
35
30
25
20
15
10
5
0
-120
-110
-100
-90
-80
-70
-60
-50


Trekryper
False accept
False reject
Number of false accept/reject
Loglikelihood/20 ms segment treshold
25
20
15
10
5
0
-120
-110
-100
-90
-80
-70
-60