Stockholms universitet

Speech Recognition with Hidden Markov Models

a study about isolated word recognition

Gonzalo Aponte Navarro

Masteruppsats i matematisk statistik
Master Thesis in Mathematical Statistics

Masteruppsats 2020:1
Matematisk statistik
Januari 2020

www.math.su.se

Matematisk statistik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm

Stockholm University

Mathematical Statistics
Stockholm University
Master Thesis **2020:1**
http://www.math.su.se

Speech Recognition with Hidden Markov Models

a study about isolated word recognition

Gonzalo Aponte Navarro*

February 2020

Abstract

Hidden Markov Models (HMM's) are a versatile class of statistical models which have been used in various contemporary speech recognition systems. In this study it was explored how these models could be applied for recognizing words in a data set with recordings of 11 different words. Feature vectors based on the Discrete Fourier Transform (DFT) were used for word modelling and recognition. Variations of the HMM's were fitted and tested on the data set. The out-of-sample error for each model was estimated by performing a 10-fold cross-validation. Moreover, the full data set was used for both training and testing. In the latter experiment, the smallest total error among the constructed word recognizers was 13.3 %. Furthermore, the study explored how HMM's could be used for detecting the presence of words in a recording. The word detector failed to detect certain segments of different words but performed well in other cases.

*Postal address: Mathematical Statistics, Stockholm University, SE-106 91, Sweden. E-mail: gonzalo.navarro@outlook.com. Supervisor: Chun-Biu Li.

Acknowledgements

I want to send my thanks to my supervisor Chun-Biu at Stockholm University, for your feedback and advice. I also want to thank my family for all the support.

Contents

1 Introduction

1.1 Purpose

This thesis is about isolated word recognition and word detection. It is essential to have a method of separating silence or noise from speech in order to fit and test a reliable word-recognizer, and in this study a simple word-detection model will be discussed. The study will consist of two main parts: a theoretical exploration of methods that have been used in previous studies and the second part consists of an application of the theory to a data set.

The data which will be studied in this project was downloaded from [1]. The data set contains recordings of the spoken digits "zero", "one",..., "nine" and the word "o". This data set has been used in a demonstration of Hidden Markov Models (HMM's) for isolated word recognition [14]. Observations from this data set will be automatically labelled by their corresponding word. Furthermore, a word-detecting HMM will be fitted and tested on audio files from this data set. We conclude by examining the performance of a collection of HMM models for word recognition.

1.2 Background

In the last years, speech recognition technology have spread through the popularization of smartphones and personal computers, but this technology has existed for several decades. There was for example a digit-recognizing system which was developed at Bell Labs in the 1950's [8]. In this project we will study speech recognition with Hidden Markov Models, which originate from the 1960's [11]. These models describe the relation between two random variables: a hidden state sequence and a sequence of observed emissions.

Speech recognition is an interdisciplinary field which covers many different subjects such as signal processing, statistics, information theory and linguistics. In a speech recognizing system there are many different parts. The speech data is typically processed into a form which is better suited for further analysis, which is known as speech parameterization or feature extraction. The first step of the processing can be done by e.g. using the Discrete Fourier Transform (DFT), technique that is fundamental in signal processing. This transformation takes an audio representation from the time domain to the frequency domain.

There are many DFT-based speech parameterizations which have been used in speech recognizing systems. These include Linear Frequency Cepstral Coefficient (LFCC), Perceptual Linear Prediction (PLP), Linear Prediction Coefficient (LPC) and Mel Frequency Cepstral Coefficient (MFCC) [7]. The MFCC in particular has been used in contemporary speech recognition systems such as the Hidden Markov Toolkit (HTK) and Kaldi, both of which also use HMM models.

HMM's will be designed to express the relation between the words and the processed data. Finally, these models are fitted and used to identify words in an audio file.

There are many ways to perform speech recognition, even within the framework of Hidden Markov Models for speech recognition. There are variations with respect to how the emissions are generated and if they are discrete or continuous. The models and methods which will be used in this project are heavily based on the instructions from Rabiner's tutorial on Hidden Markov Models for speech recognition [11] and a MATLAB implementation which has been developed by Xin and Qi [14].

In this project we will use MFCC's as the feature vectors of our word recognition system. By using the technique of Vector Quantization (VQ), these features are discretized into a predetermined number of possible values. In this study we will examine the impact that the discretization has on the performance of the HMM word-recognition system.

The feature which the word detector will use is the empirical second moment of the segments of the speech recording. Different variables which are based on the empirical second moment have been used in previous studies of speech signals and have been referred to as *energy* [4], [13]. In this study, we will discretize these values in order to implement the same class of HMM's for both word detection and word recognition.

2 Theoretical framework

This section will focus on the theory of Hidden Markov Models. We will explain the design and parameter estimation and how the models are used for *word recognition* and *word detection*. These two subjects are different but will be explored by using the same class of HMM's. Word recognition is about identifying what word is spoken while word detection is about finding the location of speech in a recording. Furthermore, this section will include a description of the feature vectors used by the HMM's for these two different applications.

2.1 The word recognition model

2.1.1 Problem formulation

First, we assume that a word recording is represented by a sequence of amplitudes $X_1, \ldots, X_N$. The amplitudes are grouped into L successive subsequences of points, known as *frames*. For each frame of order ℓ, a *feature* O_ℓ is computed. We denote the sequence of features from a speech recording as follows: $\boldsymbol{O} = (O_1, \ldots, O_L)$. In the context of HMM's, these variables are typically referred to as *emissions* or *outputs*.

Next, we will denote the unknown word to be classified by W. This word is a random variable which can assume any value in the vocabulary $\mathscr{V}$. The vocabulary $\mathscr{V}$ is known in advance, and in our case it consists of the words from the data set: "zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine" and "o". In this study we will use the problem formulation used by Rabiner [11] for the word recognition problem. Each word $w \in \mathscr{V}$ is assumed to have its own *word model*, which is a HMM with parameter collection λ_w. The word recognition problem can be summarized as follows: find the word model which maximizes the likelihood of the observed feature values $\boldsymbol{o}$. The recognized word $\widehat{W}$ is defined in the expression below.

$$\widehat{W} = \underset{w \in \mathscr{V}}{\operatorname{argmax}} \, \mathbb{P}\left(\boldsymbol{O} = \boldsymbol{o} \mid \lambda_w\right) \tag{1}$$

This word recognizer is intended to be used on recordings of words from the vocabulary $\mathscr{V}$. It will incorrectly label recordings of other sounds as being words from this vocabulary. For this reason, this word-recognizer is very limited and could be improved if it was combined with a word detector.

2.1.2 An overview of the variables

The HMM chosen for this project is the one which is defined by Rabiner [11] and has been implemented by Xin and Qi [14]. This section will be dedicated to defining the HMM and the design it will have for speech recognition. We will also show how to estimate its parameters and perform different computations which are necessary to solve the word recognition problem. We will for example explain how to compute the probability of an emission sequence under a given HMM. In other words, we will explain the steps needed to solve the optimization problem (1).

A HMM contains two variables of interest: the hidden state sequence $\boldsymbol{Q} = (Q_1, \ldots, Q_L)$ and the sequence of emissions $\boldsymbol{O} = (O_1, \ldots, O_L)$. As the names suggest, the hidden sequence $\boldsymbol{Q}$ is assumed not to be observed and it is therefore unknown or *hidden*. In contrast, the values of the emission sequence $\boldsymbol{O}$ are observable. Our HMM represents a stochastic process in which a hidden state variable Q_ℓ emitts a value O_ℓ in a way which is determined by a probability distribution. There are various assumptions which determine the properties of this stochastic process, and they will be explained later. For now, we introduce the notation of the various parameters for our class of HMM's in the table below.

Quantity	Definition
Q_ℓ	The hidden state of frame ℓ.
O_ℓ	The emission of frame ℓ.
$\mathscr{S} = \{1, \ldots, n\}$	The sample space of each hidden state Q_ℓ.
$\mathscr{E} = \{1, \ldots, m\}$	The sample space of each emission O_ℓ.
$a_{i,j} = \mathbb{P}(Q_{\ell+1} = j \mid Q_\ell = i)$	The probability of transitioning from state i to state j.
$\pi_i = \mathbb{P}(Q_1 = i)$	The initial state probability of state i.
$b_i(o_\ell) = \mathbb{P}(O_\ell = o_\ell \mid Q_\ell = i)$	The probability of state i emitting the value o_ℓ.
A	The matrix of transition probabilities $a_{i,j}$.
B	The matrix of emission probabilities $b_{i,j}$.
Π	The vector of initial state probabilities π_i.
$\lambda = (A, B, \Pi)$	The collection of parameters of a HMM.

Table 1: The parameters and quantities of a Hidden Markov Model and their notation.

2.1.3 The hidden states

The hidden states $Q_1, \ldots, Q_L$ represent some source which generates the sounds that are being observed. The state space of each of these variables is $\mathscr{S}$. The number of elements in this state space is n. There are many alternatives to decide what the hidden states are supposed to represent. One alternative is to let the hidden states represent phonemes. A phoneme is a phonetic unit which is specific to a language and can be described as the sound which distinguishes one word from another. Another alternative is to let n be approximately equal to the mean number of observed emissions O_ℓ in a word [11].

The International Phonetic Alphabet (IPA) is a phonetical system which offers a way to symbolically represent phonetic components such as phonemes. In the table below we show the words to be recognized in this thesis, together with their phonemic transcription. The phonetic representation is based on the American pronounciation of the words.

zero	z	ɪ	r	o	ʊ
one	w	ʌ	n		
two	t	u			
three	θ	r	i		
four	f	ɔ	r		
five	f	a	ɪ	v	
six	s	ɪ	k	s	
seven	s	ɛ	v	ə	n
eight	e	ɪ	t		
nine	n	a	ɪ	n	
o	o	ʊ			

Table 2: Set of phonetic symbols, without stress symbols, based on the IPA transcription.

The hidden state sequence $\boldsymbol{Q} = (Q_1, \ldots, Q_L)$ forms a Markov chain. A Markov chain is a stochastic process such that the probabilities of the next state only depends on the current state. This property can be expressed more formally as follows. For all $\ell \geq 1$ it holds that every element i_ℓ in the hidden state space $\mathscr{S}$ and every hidden state variable Q_ℓ satisfies the following.

$$\mathbb{P}(Q_{\ell+1} = i_{\ell+1} \mid Q_1 = i_1, \ldots, Q_\ell = i_\ell) = \mathbb{P}(Q_{\ell+1} = i_{\ell+1} \mid Q_\ell = i_\ell) \tag{2}$$

The matrix A contains the transition probabilities of this Markov chain. An element $a_{i,j}$ of this matrix is defined to be the probability of a transition from the state i to state j.

$$\mathbb{P}(Q_{\ell+1} = j \mid Q_\ell = i) = a_{i,j} \tag{3}$$

It follows that the transition matrix A is a square $n \times n$ matrix, where $n = |\mathscr{S}|$ as before. For our application we will let A have a design which captures the passage of time among the hidden state sequence. We will use a so-called *left-to-right model* to achieve this [11].

For a left-to-right model, the order ℓ of the expression $Q_\ell = i$ indicates that the state i was visited at frame ℓ. Such a model starts in the left-most state and this is expressed in the following way. Assume that i is the left-most state, meaning that $Q_1 = i$. The notation $\pi_i = \mathbb{P}(Q_1 = i)$ will be used.

$$\pi_i = \begin{cases} 0 \text{ if } i \neq 1 \\ 1 \text{ if } i = 1 \end{cases} \tag{4}$$

The assumption that the process goes from left to right places a restriction on the transition probabilities. The direction of the state transitions from left to right is expressed as follows.

$$a_{i,j} = 0 \text{ for } j < i \tag{5}$$

To make sure that the process does not reach the end too quickly, an extra assumption is made. We will impose a limit to how far the process can go to the right. This limit is denoted by Δ, and the restriction is expressed in the following manner.

$$a_{i,j} = 0 \text{ for } j > i + \Delta \tag{6}$$

Hence the process cannot make a transition further than Δ steps to the right. Finally, to express how the process reaches the final state n we have the next restriction.

$$a_{n,n} = 1 \tag{7a}$$
$$a_{n,i} = 0 \text{ for } i < n \tag{7b}$$

As an example, we will now show a transition matrix with the step limit $\Delta = 2$.

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & 0 & 0 \\ 0 & a_{2,2} & a_{2,3} & a_{2,4} & 0 \\ 0 & 0 & a_{3,3} & a_{3,4} & a_{3,5} \\ 0 & 0 & 0 & a_{4,4} & a_{4,5} \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

We will illustrate the possible transitions corresponding to this matrix in the figure below. The hidden states are in circles and a transition with non-zero probability is represented by an arrow. Transitions with zero probability are not visible.

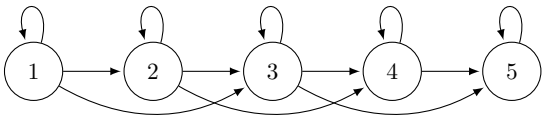

Figure 1: Possible state transitions specified by the transition matrix A.

2.1.4 Conditionally independent emissions

The following conditional independence assumption is made for the emission sequence $\boldsymbol{O} = (O_1, \ldots, O_L)$. The probability of an event under a HMM with parameter collection λ will be denoted by $\mathbb{P}_\lambda$.

$$\mathbb{P}_\lambda(\boldsymbol{O} = \boldsymbol{o} \mid \boldsymbol{Q} = \boldsymbol{q}) = \prod_{\ell=1}^{L} \mathbb{P}_\lambda(O_\ell = o_\ell \mid Q_\ell = q_\ell) = \prod_{\ell=1}^{L} b_{q_\ell}(o_\ell) \tag{8}$$

This assumption describes the relation between the hidden states and the emissions. An emission from the frame of order ℓ only depends on the hidden state which was visited at that frame. Because of this property, this kind of HMM will be referred to as a *state-emitting HMM*. This property will

affect different computations which are necessary for both estimating the parameters and computing the probabilities, as shown in the next section.

2.1.5 Calculating the emission probabilities

In order to perform the maximization procedure specified by equation (1), we need to calculate the probability of the sequence of emissions under a given parameter. One may use the Markov assumption (2) together with the conditional independence (8) to directly compute the emission probabilities. We begin by applying the law of total probability and sum over every possible hidden state sequence. When conditioning on λ, we mean that the probability is given under the HMM with the parameter collection λ.

$$\mathbb{P}_\lambda(\boldsymbol{O} = \boldsymbol{o}) = \sum_{\boldsymbol{q}} \mathbb{P}_\lambda(\boldsymbol{O} = \boldsymbol{o}, \boldsymbol{Q} = \boldsymbol{q}) \tag{9a}$$

$$= \sum_{\boldsymbol{q}} \mathbb{P}_\lambda(\boldsymbol{O} = \boldsymbol{o} \mid \boldsymbol{Q} = \boldsymbol{q}) \mathbb{P}_\lambda(\boldsymbol{Q} = \boldsymbol{q}) \tag{9b}$$

$$= \sum_{\boldsymbol{q}} \pi_{q_1} b_{q_1}(o_1) \prod_{\ell=2}^{L} b_{q_\ell}(o_\ell) a_{q_{\ell-1}, q_\ell} \tag{9c}$$

As noted by Rabiner [11], the direct computation has a high computational cost, consisting of $2L \cdot n^L$ calculations. This computational cost can be obtained by inspecting (9c). There are n^L possible state sequences $\boldsymbol{q}$ since there are n possible states at every time frame $\ell = 1, \ldots, L$. Every term in the sum requires approximately $2L$ computations, and hence the total number of calculations is approximately $2L \cdot n^L$. Rabiner describes the *forward algorithm* as a feasible alternative to the direct computation. The forward algorithm has the computational cost of $n^2 L$ calculations. We will hereafter explain how this procedure is performed.

We begin by defining the *forward variable* $\alpha_\ell(i)$. Sequences such as $o_1, \ldots, o_\ell$ will be denoted by $o_{1:\ell}$.

$$\alpha_\ell(i) = \mathbb{P}_\lambda(O_{1:\ell} = o_{1:\ell}, Q_\ell = i) \tag{10}$$

The algorithm is organized into the following steps: initialization, recursion

and termination. A proof of these formulas is given in the appendix.

$$\text{Step 1: Initialization}$$
$$\alpha_1(i) = \pi_i b_i(o_1) \tag{11a}$$
$$\text{Step 2: Recursion}$$
$$\alpha_\ell(j) = \sum_i \alpha_{\ell-1}(i) a_{i,j} b_j(o_\ell), \text{ for } \ell = 2, \ldots, L. \tag{11b}$$
$$\text{Step 3: Termination}$$
$$\mathbb{P}_\lambda(\boldsymbol{O} = \boldsymbol{o}) = \sum_i \alpha_L(i) \tag{11c}$$

The forward algorithm can be affected by *numerical underflow*. The likelihood consists of sums of products of numbers between 0 and 1. As the number of factors of such a product grows, it approaches 0. For a computer with finite precision this can result in the expression being evaluated to 0. This can cause problems in the estimation procedure.

A solution to the numerical underflow is to compute the logarithm of the probabilities. In [11] it is explained how one can use this technique together with *rescaling* the parameter estimates in order to avoid numerical underflow. An alternative is to use the Viterbi approximation of the likelihood $\mathbb{P}_\lambda(\boldsymbol{O} = \boldsymbol{o})$. The Viterbi approximation has been combined with the log-probability in Xin and Qi's word recognition demonstration [14].

2.1.6 Viterbi approximation

The Viterbi approximation is a method of approximating the probability of a given emission sequence $\boldsymbol{o}$. It is performed by first finding the most probable sequence of hidden states $\boldsymbol{q}^*$ with the *Viterbi algorithm*. Then a joint probability of the sequence $\boldsymbol{q^*}$ and the observed emissions $\boldsymbol{o}$ is computed as described below.

$$\widehat{\mathbb{P}}_\lambda(\boldsymbol{O} = \boldsymbol{o}) = \max_{\boldsymbol{q}} \{\mathbb{P}_\lambda(\boldsymbol{O} = \boldsymbol{o}, \boldsymbol{Q} = \boldsymbol{q})\} \tag{12}$$

The Viterbi algorithm is similar to the Forward Algorithm, since it is composed by similar steps: initialization, recursion and termination. The algorithm is concluded by a backtracking step. We begin by observing that the hidden sequence $\boldsymbol{q^*}$ can be expressed in the following ways.

$$\boldsymbol{q^*} = \operatorname*{argmax}_{\boldsymbol{q}} \mathbb{P}_\lambda(\boldsymbol{Q} = \boldsymbol{q}, \boldsymbol{O} = \boldsymbol{o}) \tag{13a}$$
$$= \operatorname*{argmax}_{\boldsymbol{q}} \log \mathbb{P}_\lambda(\boldsymbol{Q} = \boldsymbol{q}, \boldsymbol{O} = \boldsymbol{o}) \tag{13b}$$

The advantage of using latter form for computation is that the problem of numerical underflow is avoided, as noted by Xin and Qi [14] and Rabiner

[11]. The following problem formulation has been used by Forney [6].

$$\boldsymbol{q}^* = \underset{\boldsymbol{q}}{\operatorname{argmin}} \ -\log \mathbb{P}_\lambda(\boldsymbol{Q} = \boldsymbol{q}, \boldsymbol{O} = \boldsymbol{o}) \tag{14}$$

Under our assumptions, the last problem formulation is equivalent to the previous two. Forney's description of the Viterbi algorithm assumes a *transition-emitting HMM*. This means that an emission O_ℓ depends on the transition from Q_ℓ to $Q_{\ell+1}$ rather than just Q_ℓ. In this section, we will briefly describe the algorithm as it has been presented by Rabiner [11] for state-emitting HMM's. We begin by defining the following quantity.

$$\phi_\ell(i) = \max_{q_{1:(\ell-1)}} \ \log \ \mathbb{P}_\lambda(Q_{1:(\ell-1)} = q_{1:(\ell-1)}, Q_\ell = i, O_{1:\ell} = o_{1:\ell}) \tag{15}$$

This expression is the maximal log-probability corresponding to the hidden sequence which ends in the state i and the observed emissions $o_{1:\ell}$. The algorithm begins by initializing the values for $\phi_\ell(i)$ and then uses a recursive relation between $\phi_\ell(i)$ and $\phi_{\ell-1}(i)$ to progressively compute the maximal probabilities for hidden state sequences of various steps ℓ. For more details of the procedure, see the appendix.

2.1.7 Training the model

We will use the maximum likelihood principle to estimate the parameter of the model. Every word model has its parameters estimated independently of the other word models. We introduce the estimation procedure when given a single emission sequence $\boldsymbol{O}$. The objective of a conventional maximum-likelihood estimation would be to find the parameter which maximizes the likelihood given the complete data: $\mathbb{P}(\boldsymbol{O} = \boldsymbol{o}, \boldsymbol{Q} = \boldsymbol{q} \mid \lambda)$. Since the state sequence $\boldsymbol{Q}$ is not observed, the maximum is untractable. An alternative is to use the Baum-Welch algorithm, which can be regarded as a special case of the Expectation-Maximization (EM) algorithm [11]. The EM-algorithm deals with the missing observations of the hidden states $\boldsymbol{Q}$ by maximizing the *expected value* of the log-likelihood, given the observed emission sequence.

The computation of the expected value of the log-likelihood forms what is known as the *E-step* of the algorithm, and is defined in (16a). We formally state the steps of the EM-algorithm in the equations (16). In this section, the EM-algorithm is presented in the style of Dempster et al. [5], who have presented the algorithm in an article which Rabiner et al. have used as reference for the estimation of the HMM parameter [11].

Let $\lambda^{(t)}$ denote the estimate at iteration t and λ a HMM-parameter. The next estimate $\lambda^{(t+1)}$ is obtained by the maximization step (16b). Hereafter, the notation $p(\boldsymbol{o}, \boldsymbol{q} \mid \lambda) = \mathbb{P}(\boldsymbol{O} = \boldsymbol{o}, \boldsymbol{Q} = \boldsymbol{q} \mid \lambda)$ is used to express the likelihood. Similar notation is used for expressing the likelihood after

conditioning on the observed emission sequence $\boldsymbol{o}$.

Expectation (E-step)

$$Q\left(\lambda \mid \lambda^{(t)}\right) = \mathbb{E}\left(\log p(\boldsymbol{O}, \boldsymbol{Q} \mid \lambda) \mid \boldsymbol{O} = \boldsymbol{o}, \lambda^{(t)}\right) \tag{16a}$$

Maximization (M-step)

$$\lambda^{(t+1)} = \underset{\lambda}{\operatorname{argmax}}\, Q(\lambda \mid \lambda^{(t)}) \tag{16b}$$

As the number of iterations grow, the estimates converge to an estimate with the highest likelihood. Rabiner notes that the algorithm converges to a local maximum, and that there may be multiple local maxima [11].

The function Q which is defined in the E-step (16a) is known as *Baum's auxiliary function*, with a more explicit form given in (17a). The computation at the E-step is given by the following expression.

$$Q\left(\lambda \mid \lambda^{(t)}\right) = \sum_{\boldsymbol{q}} p\left(\boldsymbol{q} \mid \boldsymbol{o}, \lambda^{(t)}\right) \log p\left(\boldsymbol{o}, \boldsymbol{q} \mid \lambda\right) \tag{17a}$$

The EM-algorithm produces an estimate $\lambda^{(t+1)}$ with a higher likelihood than the previous one $\lambda^{(t)}$. The higher likelihood of the next estimate is obtained from the following property:

$$Q\left(\lambda^{(t+1)} \mid \lambda^{(t)}\right) \geq Q\left(\lambda^{(t)} \mid \lambda^{(t)}\right) \Rightarrow \log p\left(\boldsymbol{o} \mid \lambda^{(t+1)}\right) \geq \log p\left(\boldsymbol{o} \mid \lambda^{(t)}\right) \tag{18}$$

The property (18) is a consequence of Jensen's inequality [5]. For a more detailed explanation of the inequality (18), see the appendix. The left-hand side of (18) is a consequence of the M-step (16b).

The next estimate $\lambda^{(t+1)}$ is obtained at the M-step by applying the method of *Lagrange multipliers* in order to maximize the function $Q(\lambda \mid \lambda^{(t)})$ [11]. The constraints of this maximization procedure are the linear dependencies between the different parameters. For example, the sum of the initial state probabilities π_i equals 1. The solution to the maximization problem is given in the appendix.

We have described the estimation procedure when the data is a single emission sequence $\boldsymbol{o}$. In practice, we will use many emission sequences to estimate the parameter collection λ for the model of a single word. For this reason, we will need to use the above formulas for every observation sequence and combine them [11]. Assuming that the observed emission sequences are independent, λ is estimated by maximizing the following probability. The emission sequences will be denoted by $\boldsymbol{O}^{(1)}, \ldots, \boldsymbol{O}^{(K)}$.

$$\mathbb{P}_\lambda(\boldsymbol{O}^{(1)} = \boldsymbol{o}^{(1)}, \ldots, \boldsymbol{O}^{(k)} = \boldsymbol{o}^{(k)}) = \prod_{k=1}^{K} \mathbb{P}_\lambda(\boldsymbol{O}^{(k)} = \boldsymbol{o}^{(k)}) \tag{19}$$

Under the assumption (19), many emission sequences can be used to estimate the parameter λ by maximizing the corresponding auxiliary function Q. The expression for this estimate is given in the appendix.

2.1.8 The feature vectors

The (MFCC) feature extraction consists of many techniques that are founded on signal processing theory and models of human perception of sound [7]. Since this thesis will focus on the statistics of the HMM's, we will only briefly describe the main ideas behind the MFCC - what it represents, why it is useful for speech recognition and how it relates to the HMM's that were used in this study.

A digital audio signal can be represented as a bounded sequence of values $X_1, \ldots, X_N$, where N is a positive integer. These values will be referred to as *amplitudes*. In this thesis, the models will be trained and tested on uncompressed `wav`-files. These files are read and converted to a normalized sequence of numbers with MATLAB's `audioread` function. The values $X_1, \ldots, X_N$ are rational values in the range $[-1, 1]$. Without the normalization, the amplitudes would assume integer-values in the range from $-2^{(16-1)}$ to $2^{(16-1)} - 1$. The size of this range is determined by the quantity known as the *bit depth* or *bits per sample*, which is 16 for all the files in this study. For comparison, a recording with 4 bits per sample would take integer values in the smaller range from $-2^{(4-1)}$ to $2^{(4-1)} - 1$ before the normalization.

The MFCC's are computed by first partitioning the audio recording into collections of points (frames). Every frame corresponds to a time region in the recording. Each of these frames is pre-processed and then transformed by the DFT. The justification for applying the DFT on the frames of the speech signal rather than the entire signal is that it is assumed to be *short-time stationary*. This procedure is known as the *short-time Fourier transform*, which can be visualized by a *spectrogram*. In previous studies, the frame durations have been 20-40 ms [14] and 45 ms [11]. The frames are also typically chosen to be overlapping, meaning that consecutive frames share some points. The DFT is useful because it provides information about the frequencies of the periodic waves found in the frame.

We will show a spectrogram for a simple example. We begin by generating a vector of the 2000 time points, by letting $t_1 = 0$ and $t_{n+1} = t_n + \Delta t$, where $\Delta t = 0.001$. This means that the sample rate is 1000 Hz, where 1 Hz is 1 s^{-1}. Next, we generate the non-normalized amplitudes Y_n of the audio file in the following way.

$$Y_n = 10\cos(2\pi 220 t_n) + \cos(2\pi 440 t_n) \tag{20}$$

We proceed by normalizing this sequence to X_n and can now generate a 16-bit audio file with these values. We illustrate a segment of this sequence in figure 2.

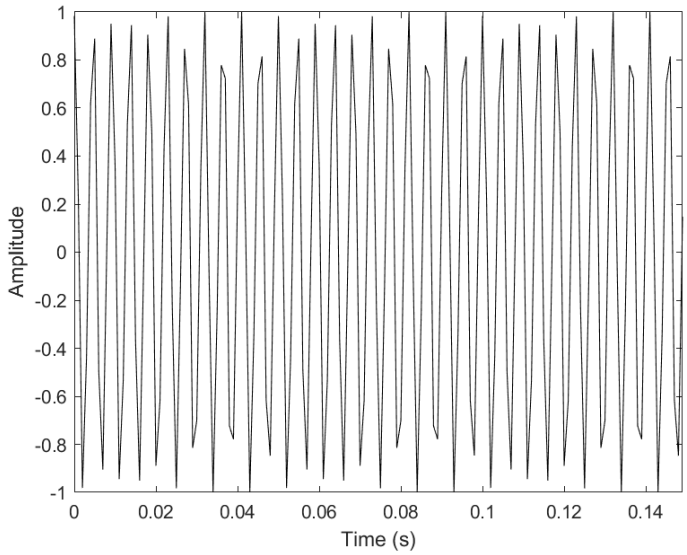

Figure 2: A segment of the waveform of the amplitude sequence X_n. The horizontal axis shows time in seconds.

Next, the audio is partitioned into non-overlapping frames of 100 points each, and the DFT is computed for each frame. The magnitude of the DFT-components of a frame reveal the frequencies that are present in the audio. In the spectrogram of figure 3, the magnitude at a particular frequency is measured in dB per Hz. A brighter color indicates a higher magnitude.

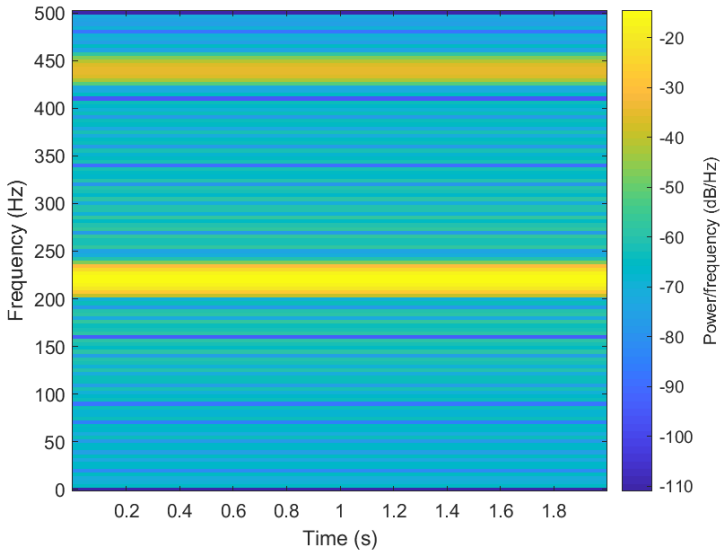

Figure 3: The short-time discrete fourier transform of the sum of cosines.

In figure 3, there are two horizontal bands whose colors represent the magnitudes of the two cosine terms from (20). The band with the brightest color extends between 200 and 250 Hz, which corresponds to the first term of (20), which has a higher magnitude than the second term. Spectrograms of speech data exhibit a more complex pattern with a mixture of different frequencies which changes as time increases. The MFCC uses the information about the frequency content that can be observed in the spectrogram.

The specific variant of MFCC's used as feature vectors for the thesis is the MATLAB demonstration found in the "Speech Recognition" chapter of Xin and Qi's literature [14]. More specifically, it is the method which has been coded in the `mfcc`-function, which is adapted from the Auditory Toolbox by Malcolm Slaney (1998). This MFCC-computation has also been described in Ganchev's literature about feature extraction, where the method has been referred to as the MFCC-FB40 method [7]. The main steps of the MFCC computation are illustrated in figure 4.

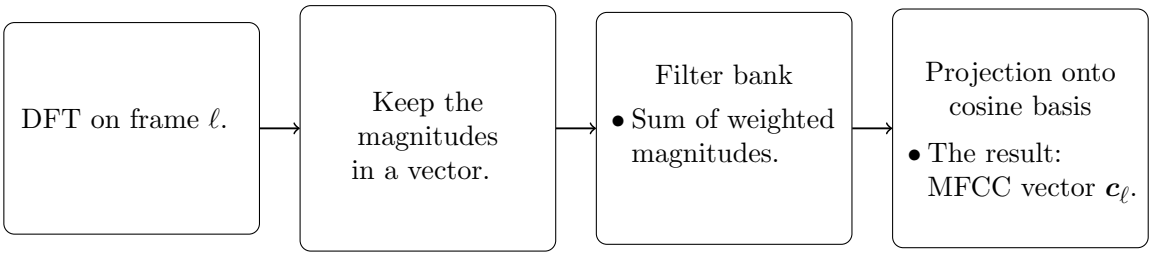

Figure 4: A summary of the MFCC computation.

The first two steps have already been described. The next step is to send the magnitude vector to a filter bank which represents the auditory system. Since Xin and Qi's literature lacks a definition for the filter bank, Ganchev's definition of the filter bank is visualized in figure 5. The filter bank produces a set of *weights* corresponding to different frequencies, with values shown on the vertical axis of figure 5. A weighted sum of the magnitude vector is produced by using the filter bank, and the logarithm of the result is taken afterwards.

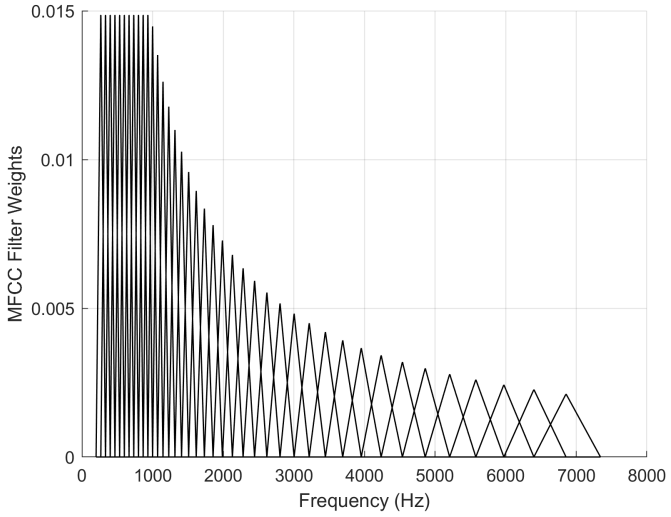

Figure 5: Filter bank weights

Each filter is represented by a triangular shape. These filters are uniformly placed on the frequency-axis for lower frequencies and are placed further apart for higher frequencies. According to Xin and Qi, the higher-frequency filters have been placed by using a log-frequency scale called the *Mel-frequency scale* or *Mel-scale*. An explicit formula for this scale has not been found in Xin and Qi's literature. According to Ganchev, this filter bank has a design that is analogous to the filter bank of the HTK MFCC-FB24 method, which uses the Mel-scale for placement of filters. The HTK MFCC-FB24 method uses the definition given in equation (21) of the Mel-scale. The frequency in hertz is denoted by f_{lin} and the Mel-scale by f_{mel}. The function defined in equation (21) is visualized in figure 6.

$$f_{\text{mel}} = 2595 \cdot \log_{10}\left(1 + \frac{f_{\text{lin}}}{700}\right) \tag{21}$$

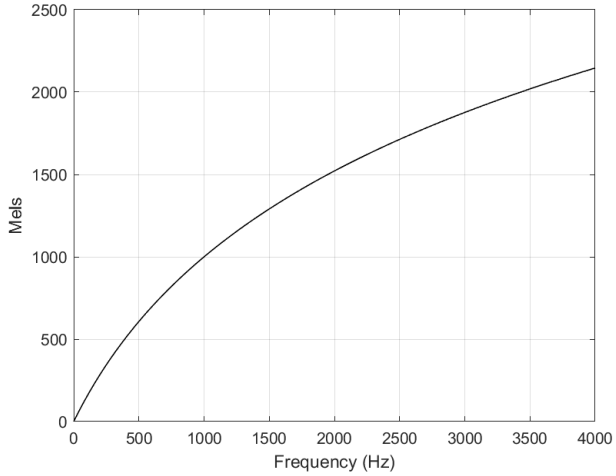

Figure 6: Mel-scale

According to Xin and Qi, the resulting values after applying the filter bank are the values $A_k,\ k = 1, \ldots, K$ where K is the total number of filters. The following (incomplete) expression is given in Xin and Qi's literature [14]. The n:th component of a MFCC-vector $\boldsymbol{c}$ is denoted by c_n in this section, and $L = 13$ represents the dimension of the MFCC-vector.

$$c_n = \sum_{k=1}^{K} (\log A_k) \cos\left(n(k - 0.5)\pi/K\right) \ \ k = 1, 2, \ldots, L \tag{22}$$

As described by Xin and Qi , the zero:th component c_0 is a measure of the *average speech power*. The next component, c_1, measures the *balance of power* for two classes of sounds: *sonorant*- and *friction*-sounds. The components c_i of higher orders $i \geq 2$, describe other properties about the frequency content of the frame. We note however, that c_0 is undefined according to the expression (22), and that the index k is defined in multiple places. Upon closer inspection of the computations coded in `mfcc`-function, we conclude that Ganchev's description the MFCC-computation in the `mfcc`-function is more accurate. As before, we will denote the outputs of the filter i by A_i for $i = 1, \ldots, K$ and present Ganchev's expression in equation (23).

$$c_n = \sqrt{\frac{2}{K}} \sum_{i=0}^{K-1} (\log_{10} A_{i+1}) \cdot \cos\left(\frac{n(i + 0.5)\pi}{K}\right) \ \ n = 0, 1, \ldots, R - 1 \tag{23}$$

In the expression (23), R is the number of unique coefficients that are computable, which in our case equals 13. Although the expressions (22), (23) differ, the authors agree that they should be interpreted as the application of the Discrete Cosine Transform on the log-output of the filter banks.

More details about the computations of the filter bank weights and MFCC-coefficients will be given in the appendix.

The rationale for using the MFCC's as feature vectors is that they use the frequency content of audio recordings, take into consideration the short-time stationarity of the data, have been used in previous studies and have been used in speech-recognition software.

Since the feature vectors have continuous values, they are not suitable for HMM's with discrete emissions. Because of this, the method of Vector Quantization (VQ) is used to transform these vectors into discrete (1-dimensional) values. If a recording is composed of L frames, the sequence of features is discretized to form the sequence of discrete emissions $\boldsymbol{O} = (O_1, \ldots, O_L)$ to be used by the HMM.

2.1.9 Discretization

We begin by assuming that we have a set of amplitude sequences $\left\{\boldsymbol{X}^{(1)}, \ldots, \boldsymbol{X}^{(R)}\right\}$. For all $r = 1, \ldots, R$, a sequence of amplitudes $\boldsymbol{X}^{(r)}$ is grouped into L_r frames, which are then transformed into a sequence of feature vectors $\boldsymbol{c}^{(r)} = \left(\boldsymbol{c}_1^{(r)}, \ldots, \boldsymbol{c}_{L_r}^{(r)}\right)$. The VQ algorithm works by first generating a set of vectors $\{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m\}$, known as a *codebook*, from the training data, which is the set of vectors $\boldsymbol{c}_\ell^{(r)}$ such that $\ell = 1, \ldots, L_r$ and $r = 1, \ldots, R$.

The size m, of the codebook, is chosen in advance. We assume that we want to discretize the sequence of MFCC-vectors $\boldsymbol{c} = (\boldsymbol{c}_1, \ldots, \boldsymbol{c}_L)$ corresponding to the amplitude sequence $\boldsymbol{X}$, which has been grouped into L frames. The codebook is used to discretize every element $\boldsymbol{c}_\ell$ of the sequence $\boldsymbol{c}$, where ℓ denotes the order of the frame from $\boldsymbol{X}$. The discretization is performed by labelling feature vectors by the closest vector from the codebook. The main steps of the VQ procedure are illustrated in figure 7.

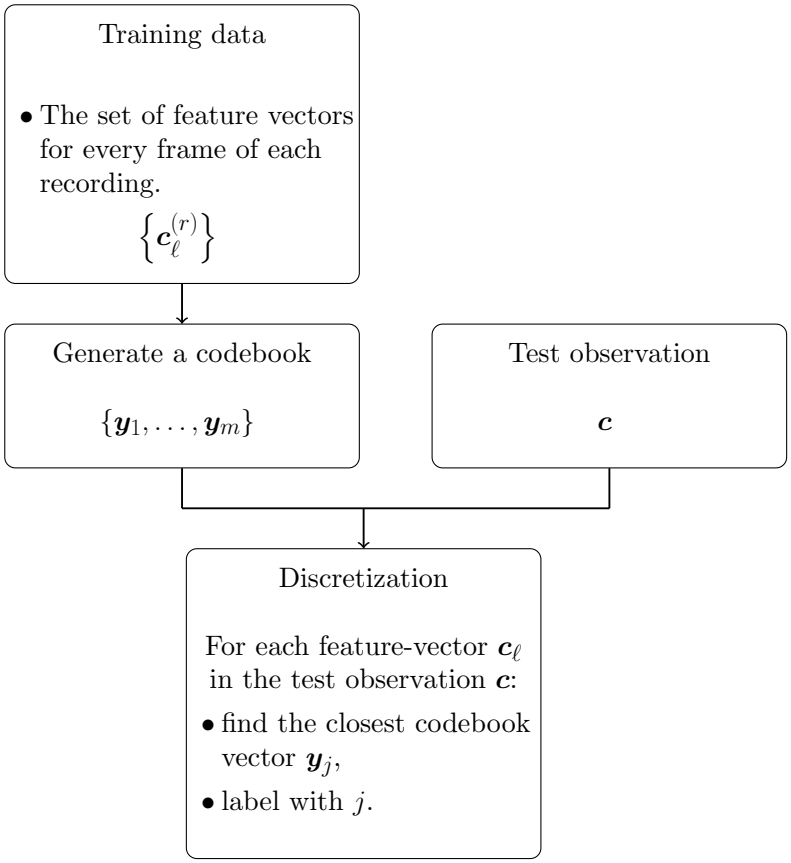

Figure 7: A summary of the VQ procedure.

As noted in the previous section, the components of the MFCC-vector $\boldsymbol{c}_\ell$ represents different properties of a frame for an audio recording. The discretized value of the feature vector $\boldsymbol{c}_\ell$ is denoted by o_ℓ and represents the outcome of an emission O_ℓ for a word model HMM. We observe that number of states in the hidden state space $\mathscr{E}$ equals the chosen codebook size m, and hence the codebook size affects the number of parameters in the word model HMM.

The procedure which generates the codebook is a variation of the *K-means clustering* algorithm, which has been implemented in the `vqsplit`-function from Xin and Qi's literature. The rationale for using this procedure is that it has been applied to sets of audio recordings in previous works in speech recognition [11], [14]. This method uses *binary splitting* and produces codebooks of sizes which are powers of two.

This method starts by generating a small codebook with the *K-means clustering* algorithm and then progressively splits the codebook vectors so that a larger codebook can be generated. For this reason, the codebook size m must be defined as a power of two before performing the VQ-procedure.

The binary splitting method has been outlined in Rabiner's speech recognition literature [12]. Rabiner has noted that it is advantageous to perform the codebook generation in stages with the binary splitting method.

We will proceed by describing how a codebook $\{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m\}$ is created with the binary-splitting procedure. As noted before, the size m is chosen in advance and is a power of 2. The distance measure which will be used in the clustering algorithm is the Euclidean distance d, which will be used to calculate the distance between MFCC-vectors.

The K-means clustering algorithm can briefly be described as follows: partition a data set into clusters by generating a set of cluster centers. The clusters are such that the distance between every datapoint and its cluster center is minimized. The set of cluster centers is the codebook which is used in the VQ-algorithm.

1. Generate a codebook of size $m = 1$. The codebook vector $\boldsymbol{y}_1$ is generated from the training data with the K-means clustering algorithm.

2. First, let $0.01 \leq \epsilon \leq 0.05$. Split each element $\boldsymbol{y}_k$ in the codebook in the following manner.

$$\boldsymbol{y}_k^+ = \boldsymbol{y}_k(1 + \epsilon) \tag{24a}$$
$$\boldsymbol{y}_k^- = \boldsymbol{y}_k(1 - \epsilon) \tag{24b}$$

   We observe that the codebook size is twice as large now than it was at the previous iteration.

3. Obtain a new codebook by applying the K-means clustering algorithm on the set of vectors which were generated in step 2. This step does not change the codebook size.

4. Repeat the steps 2 to 3 until the codebook of the desired size has been generated.

The discretization procedure works in the following manner. Suppose that we have a feature vector $\boldsymbol{c}_\ell$, corresponding to some frame ℓ of a recording. We want to discretize it with our codebook $\{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m\}$. The discretized version of this vector is then given by the following expression.

$$j = \underset{1 \leq k \leq m}{\operatorname{argmin}}\ d(\boldsymbol{c}_\ell, \boldsymbol{y}_k) \tag{25}$$

In figure 8 we visualize the results of the discretization procedure applied to a simulated sequence of 2-dimensional vectors. The red balls represent *codebook vectors* $\boldsymbol{y}_1, \boldsymbol{y}_2$ and the digits represent the discretized values. The discretized values correspond to the index of the closest codebook vector.

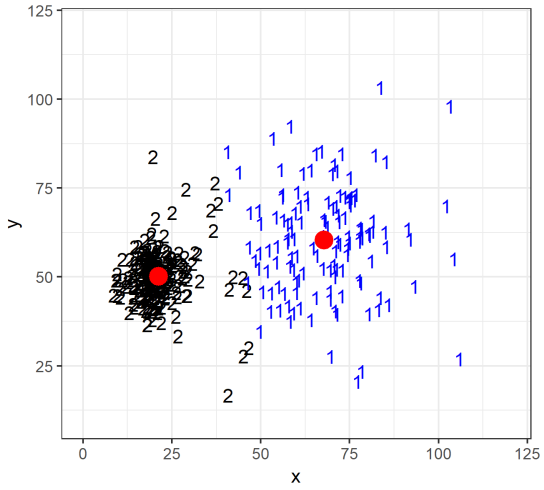

Figure 8: Discretized values of a simulated example.

2.2 The word detection model

2.2.1 Problem formulation

The word detection model is also a HMM, and thus the theory from the previous section about HMM's is also valid for the word detection model. A difference between these models is that the features and the hidden states represent other quantities. As before, we assume that we have a sequence of emissions $\boldsymbol{O} = (O_1, \ldots, O_L)$ which have been generated by a sequence of hidden states $\boldsymbol{Q} = (Q_1, \ldots, Q_L)$. For the sake of simplicity, the approach to detecting words is based on identifying silent segments from word recordings.

A hidden state variable is a binary random variable which is 1 if the frame is too silent and 0 otherwise. The emissions represent a quantity known as the *energy* of a frame.

The problem of locating the presence of a word in a recording can be expressed as follows. Given a sequence of emissions, what is the hidden sequence $\boldsymbol{q}^*$ with the highest probability? The formal expression of this sequence is given below.

$$\boldsymbol{q}^* = \underset{\boldsymbol{q}}{\operatorname{argmax}}\, \mathbb{P}(\boldsymbol{Q} = \boldsymbol{q} \mid \boldsymbol{O} = \boldsymbol{o}) \tag{26}$$

A word is then identified in the recording by locating the corresponding 0's in the optimal sequence $\boldsymbol{q}^*$. This problem is solved by the Viterbi Algorithm which was described in a previous section.

2.2.2 The feature values

For the word detection model, the audio files are grouped into non-overlapping frames with a time duration of 10 ms. The feature value for a frame is the

second moment of a frame, and it is referred to as the *energy* in this thesis. Other quantities that are related to the second moment have been used in other speech studies to detect words [13] or classify speech as voiced or unvoiced [4]. The energy is large for loud audio segments and small for silent ones. Because of this property, this feature may be suitable for detecting silent segments in speech recordings. For a frame with index ℓ and size L, the energy is formally defined as follows. As before, the sequence $X_1, X_2, \ldots$ is the waveform or sequence of amplitudes that represents a recording.

$$E_\ell = \frac{\sum_{j=L(\ell-1)+1}^{L\ell} X_j^2}{L} \tag{27}$$

This quantity can assume many rational values in the unit interval [0,1], and is therefore unfeasible for a HMM with discrete emissions. This problem can be solved by discretizing the energy, which will be explained in the next section.

There is a diversity of energy levels among the phonemes in spoken words and it can be challenging to separate them from silence and noise which can be present in the recording. The concept of silence can be somewhat ambiguous as it can include low-energy noise. An evaluation of the performance of our word-detector is to examine if the segments labelled with "silence" exclude any identifiable speech.

2.2.3 Discretization

The discretizer maps the energy to integers corresponding to segments of the unit interval. These segments are obtained by progressively halving the unit interval. The largest discrete value is a pre-defined positive integer u corresponding to the interval $[1/2, 1]$. Smaller values correspond to segments of the interval $[0, 1/2]$. If the value u is set to be high, the lower interval is partitioned into more segments. The idea behind progressively halving the interval in this fashion is to associate very high discrete values to regions of the recording with high energy values.

Some information is lost by the discretization procedure, and it is to an extent regulated by the parameter u. For this reason, this parameter should be not be too low since the partitioning of the lower half becomes very coarse. At the low setting, $u = 1$, a discretized value only indicates if the energy is higher than $1/2$.

The discretizer is composed by the following steps.

1. Transformation with respect to the pre-defined upper limit u. By using the base-2 logarithm, the idea of progressive halving is properly described in a practical, algorithmic form.

$$Y_\ell = u + \log_2(E_\ell)$$

2. Extract the positive part. The positive part of a negative number is defined to equal 0. This definition is extended to include the special case of $\log_2(0) = -\infty$, whose positive part is defined to be 0.

$$Z_\ell = Y_\ell^+$$

3. Apply the ceiling function.

$$O_\ell = \lceil Z_\ell \rceil$$

This discretizer is illustrated for the case when the upper limit is $u = 4$ in the figure below.

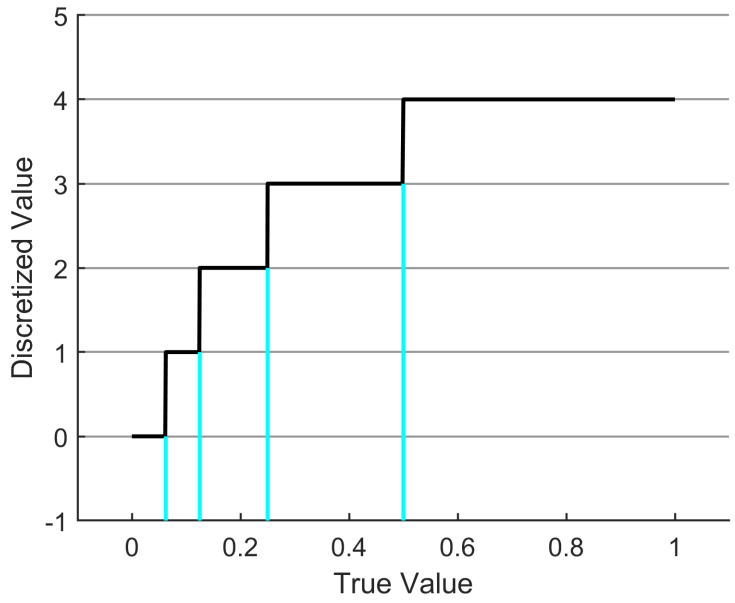

Figure 9: In black: the logarithmic discretizer.

2.2.4 Model design

We now proceed to describe the design of the HMM which will model the presence of a word. A spoken word has a certain duration and energy values in its frames. We will make the very simple assumption that a word is present in the non-silent segments. The hidden states in our HMM will only assume two possible values:

$$Q_\ell = \begin{cases} 1 \text{ if frame } \ell \text{ is too silent.} \\ 0 \text{ otherwise.} \end{cases}$$

Furthermore, the model will allow transitions between the two states and back to the same state, as shown in the figure below.

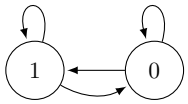

Figure 10: Possible state transitions for the word-detection model.

The duration of a word can be represented by how long the process stays in the non-silent state 0. The number of times d that a frame remains in a state i is given by the geometric distribution [11]. Let $a_{i,i}$ denote the probability of a transition from state i to itself.

$$p(d) = a_{i,i}^{d-1}(1 - a_{i,i}) \tag{28}$$

For the sake of simplicity, we will assume that the process always starts in a silent segment. This means that we will only need to estimate the emission matrix and the state transition matrix.

We can choose to estimate the parameter collection $\lambda = (\Pi, A, B)$ of the word-detection model in two different ways. The first method is the EM-algorithm. For this method, it is assumed that the discretized energies represent the sequence of emissions $\boldsymbol{O}$ and that the hidden states $\boldsymbol{Q}$ are not observed.

The second method is a conventional maximum likelihood estimation. Before computing the estimate, the frames of the recordings are labelled as "silent" or "non-silent". These labels represent the sequence of states $\boldsymbol{Q}$ which are not hidden anymore. In other words, this method assumes that we have the observed the complete data: the states $\boldsymbol{Q}$ and emissions $\boldsymbol{O}$. The maximum likelihood estimate is a solution to a maximization problem with constraints. This is a standard optimization problem which can be solved with the method of Lagrange multipliers. The problem formulation is the following.

$$\underset{\lambda}{\text{maximize}}\ f(\lambda) = \log p(\boldsymbol{o}, \boldsymbol{q} \mid \lambda) \tag{29a}$$

$$\text{s.t. } g(\lambda) = 1 - \sum_{j=1}^{n} \pi_j = 0 \tag{29b}$$

$$h_i(\lambda) = 1 - \sum_{j=1}^{n} a_{i,j} = 0 \quad \text{for } 1 \leq i \leq n \tag{29c}$$

$$k_i(\lambda) = 1 - \sum_{j=1}^{m} b_i(j) = 0 \quad \text{for } 1 \leq i \leq n \tag{29d}$$

The complete solution for this problem has been given in lecture slides for a course in Machine Learning at Carnegie-Mellon University [2]. The resulting parameter estimates are then based on counts of transitions and emissions. For more details on the solution to the problem (29), we refer to the appendix. We observe that the maximum likelihood estimator is designed for a recording. For information about the pre-labelling and how to estimate the parameters by using several recordings, see the Methods section.

In this study, we will estimate the parameters of the word-detection model by pre-labelling the data and performing a conventional maximum likelihood estimation.

3 Data

The data set that was used in this project is a subset of the TIDIGITS data set and was downloaded from Jack Xin's website [1]. This data is used in a speech recognition demonstration which has been described in Xin and Qi's speech recognition literature [14]. In this demonstration, a HMM is trained and tested for isolated word recognition. The data set consists of recordings of the 11 words which were listed in table (2): "zero", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine", "o".

The audio files were of the `wav` format and contained recordings of 39 men and 57 women. The recordings were organized by the test subjects they belonged to and by the sex of the speaker. Each test subject was identifiable by its own initials. Many subjects had several recordings of the same word. Some information about the audio files is summarized in the table below.

Group	**Sample rate**	**Channels**	**Observations per word**
Male: Test_Ryan	44100 Hz	2	0-1
Male: Other	8000 Hz	1	74-77
Female: All	8000 Hz	1	114

Table 3: TIDIGITS - general information.

A recording has k channels if it is represented by k sequences of amplitudes. In our case, we only take the first channel and ignore the others if a recording has multiple channels. In the male data, there were recordings for a man labelled as "Test_Ryan" who had different characteristics compared to the others. This man had 1 recording for the every word except for "o", a much higher sample rate than the other subjects and 2 channels. The male recordings had a duration of between 0.18925 and 1.2 seconds, while the female recordings were between 0.24 and 1.49 seconds. In this study, we will only study the set of male recordings, which includes both male groups that have been described in table 3.

3.1 Energy values

In this section we will present plots of the amplitudes together with the frame energy for some word recordings. The plots contain the amplitudes X_n marked in black, rescaled by the maximum frame energy of the recording. The blue line represents the frame energy. Since a frame is a collection of points, the corresponding frame energy of these points is the same. This explains why the frame energy is piecewise constant in the plots.

In figure 11, we see the energy plot for a recording of the word "two". The energy is very low for the phoneme "t", which ends by index $n = 1000$. The energy is much higher for the phoneme "u", which reaches a maximum value in the region $n \in [1000, 1500]$, a local maximum around $n = 2000$ and then gradually decreases in magnitude.

The boundaries between the phonemes are sometimes not noticeable by visual inspection. In the next figure, 12, the phonemes "f" and "v" are difficult to locate visually. These phonemes are also not as audible as the phonemes "a", "ɪ". For many of the recordings, the segments with highest energy are those containing vocals. Those with low mean energy tend to be either consonants or silence, but variations do exist. In figure 13, the word "six" has low energy for the phonemes "s" and "x". Furthermore, the phoneme "x" is separated from the others by a silent segment, and is located around the region of $n \in [3000, 4500]$, while the others are in $n \in [250, 1250]$. Finally, for the plot in figure 14, the final phoneme "t" is separated from "e" and "ɪ" by a silent segment. These are just a few examples which show the variation of the energy that may be present among the phonemes from the same word.

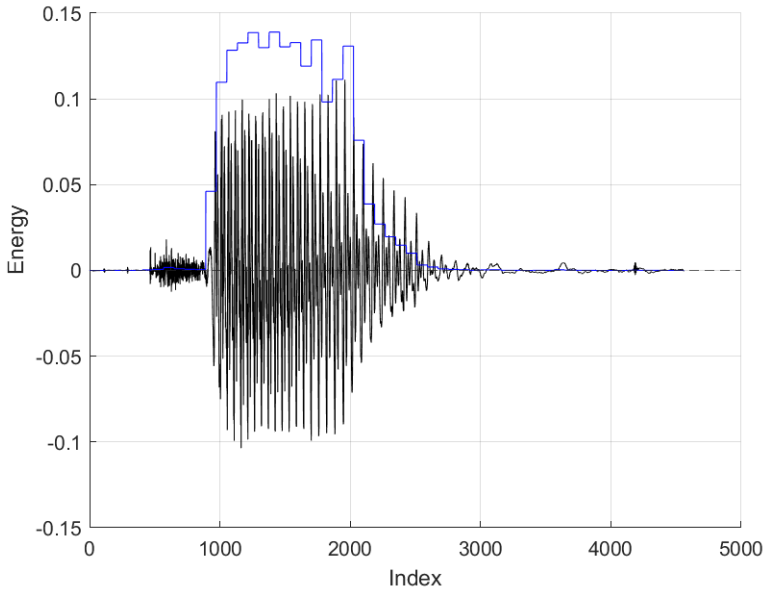

Figure 11: Word: "two". Male subject: AJ. File: `2A_endpt.wav`.

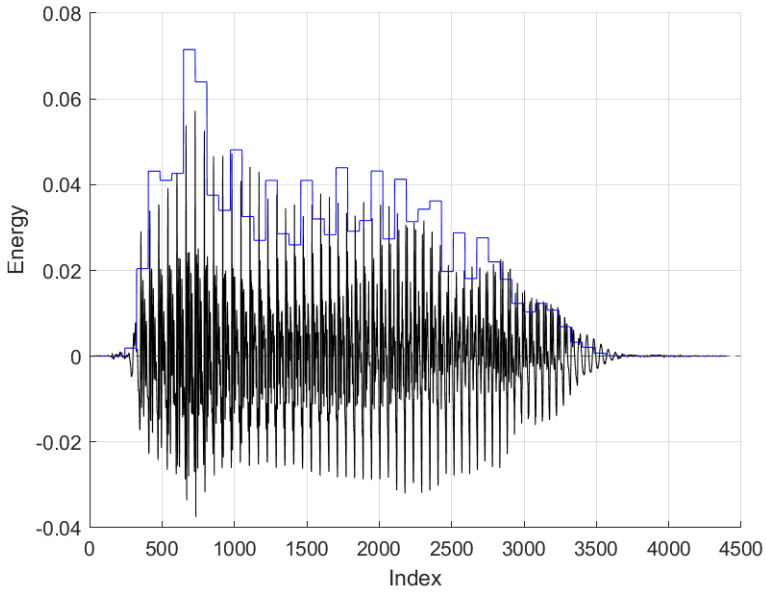

Figure 12: Word: "five". Male subject: AE. File `5A_endpt.wav`.

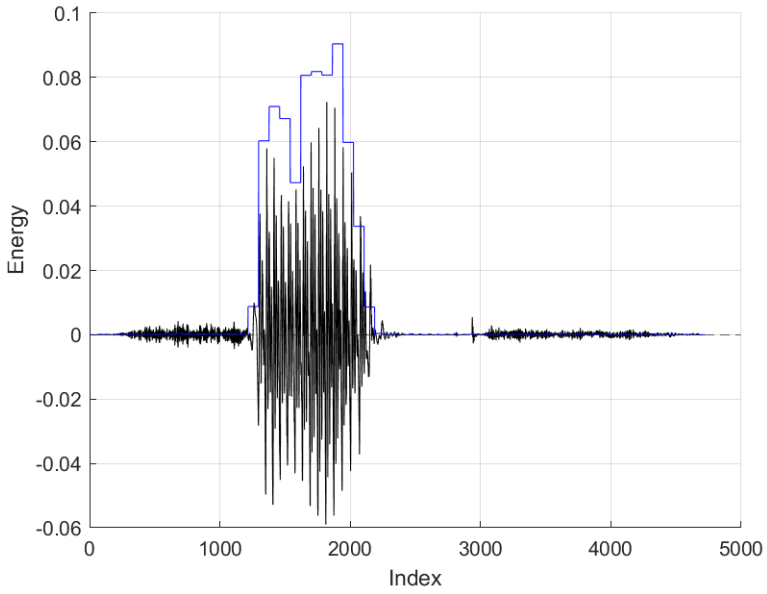

Figure 13: Word: "six". Male subject: AE. File: `6A_endpt.wav`.

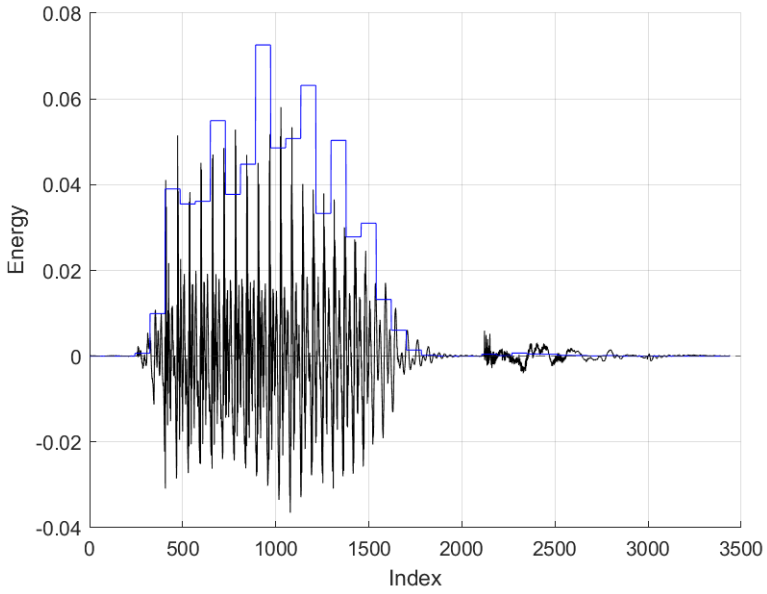

Figure 14: Word: "eight". Male subject: AE. File: `8A_endpt.wav`.

4 Methods

4.1 The word recognizer

In this thesis, the MATLAB adaptation of the HMM algorithms from Xin and Qi's demonstration [14] has been modified to function for different codebook sizes. The code was also updated to work with MATLAB version R2018b.

4.1.1 Processing and parameter estimation

The observations of the male subject "Test_Ryan" were processed by first using MATLAB's `resampling`-function from the Signal Processing Toolbox to resample it at the same sample rate as the other test subject recordings, which is 8000 Hz. The `mfcc`-function from Xin and Qi's implementation was used to pre-process and extract the feature values from the audio files.

The models were trained by using `hmmtrain` from MATLAB's Statistics and Machine Learning Toolbox. This function implements Baum-Welch estimation of the parameters and it is given an initial estimate of the parameters. These initial estimates were chosen to be such that the transition matrix A has equally large non-zero elements within each row. For example, a row with three non-zero elements is given $1/3$ as an initial estimate of these non-zero elements. The initial estimate of the emission matrix B is designed by using a similar principle: every emission gets an equal probability of being emitted from any state. Finally, the likelihoods under the word models were approximated by using the `drecm`-function which uses a modification of the Viterbi approximation for improved numerical stability.

4.1.2 Model comparison and parameter restrictions

For this study, we compared two classes of state-emitting HMM's with discrete emissions. Both of these classes of HMM's were left-right models, as described in section 2.3.1, with step limit $\Delta = 2$. This was expressed by setting zeros in the initial estimates of the transition matrix accordingly.

The first class of HMM's which were studied were modifications of the model used by Xin and Qi [14]. These models are such that every word model HMM has a hidden state space of size 5. We refer to the speech recognition system of these word models as the *5-state word recognizer*. For the second class, each word model has the size of the hidden state space $\mathscr{S}$ determined by the number of phonemes in the corresponding word. This is called the *phoneme-sized word recognizer*. The phonemes that form these words are those given in table 2. For example, "nine" has 4 phonemes and would thus have 4 states in its hidden state space. The word "o" has 2 phonemes and therefore has 2 states in its hidden state space. The size of $\mathscr{E}$, the space of emitted values, equals the codebook size. For both of these

classes of models, the codebook sizes are the following values: 2, 4, 8, 16, 32, 64, 128, 256.

In Xin and Qi's demonstration, the HMM models were fitted to the words "zero", ... , "nine" spoken by 33 subjects. In this study, we have included the recordings of "o" as well. This study is composed of three parts. In the first part we examine which model performs the best when fitted on the male subjects from Xin and Qi's training data. This data was used to train the two classes of models and these were applied to the other male subjects. Then, 10-fold cross validation was performed to further investigate if the model can make appropriate classifications when other recordings are used for training and testing the model. Finally, the entire male data was used for both training and testing the phoneme-sized and 5-state word recognizers.

4.1.3 Performance metrics

The performance of a word recognizer can be illustrated by a classification table, which shows the number of counts of each possible classification for every observation. We begin by presenting a binary classifier inspired by Fisher's famous tea drinker experiment [3]. In this classical experiment, a tea drinker is supposed to classify 8 cups of milk and tea by the order in which these liquids were poured into the cups. If a cup was first filled with milk, it would be classified as "milk" and otherwise by "tea". The outcomes of this experiment are typically illustrated by a classification table. We modify this thought experiment by letting the problem be about word recognition. The words to be identified are "milk" and "tea", and the tea-drinker is a word-recognizer. The results of the the classifications from this hypothetical tea-drinker is visualized by the colored classification table in figure 15. The cells are colored by the counts, with a darker color for larger counts. An ideal classifier would only have non-zero counts on the main diagonal, and zeros everywhere else. Such a classifier would have a dark diagonal and light-colored cells everywhere else, such as in the figure below.

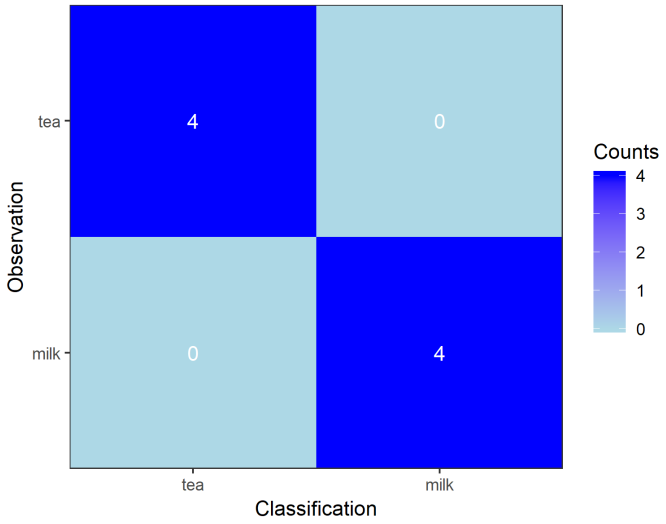

Figure 15: Classification table

A more realistic outcome of such an experiment might have given a classification table which is more difficult to interpret, such as one with counts scattered both on the main diagonal and elsewhere. For such cases, different statistical tests are available to test the independence between the classifications and the observations. In this simplified example, there are only two kinds of errors and the results are very clear.

By inspecting the classification tables of word recognizers we can evaluate its performance. In this study, there are recordings of 11 words to be classified and so many kinds of errors can occur. In this study we will calculate two types of errors, the total error and the error specific to a word. These errors can be formally expressed by using the *zero-one loss function*. If the classification $\widehat{W}$ is incorrect, the loss function returns 1, otherwise it is 0.

$$L(W, \widehat{W}) = \begin{cases} 1 \text{ if } W \neq \widehat{W} \\ 0 \text{ else} \end{cases} \tag{30}$$

Hence the total classification error of a sample of N recordings would be equal to the following. We will denote the true word of the j:th recording by W_j. The number of incorrect classifications is the sum of the off-diagonal cell counts, and the total number of recordings the sum of all cell counts.

$$\text{Error} = \frac{\text{Number of incorrect classifications}}{\text{Total number of recordings}} \tag{31a}$$

$$= \frac{\sum_{j=1}^{N} L(W_j, \widehat{W}_j)}{N} \tag{31b}$$

Next, we will present the classification error of a word v. We will let $W_j(v)$ denote the j:th recording of the word v, and $\widehat{W_j}(v)$ its classification. For the word v there are N_v recordings in total, which equals the *row sum* of the cell counts for word v in the classification table.

$$\text{Error}(v) = \frac{\text{Number of times a recording of } v \text{ is incorrectly classified}}{\text{Total number of recordings of the word } v} \tag{32a}$$

$$= \frac{\sum_{j=1}^{N_v} L(W_j(v), \widehat{W}_j(v))}{N_v} \tag{32b}$$

In previous studies [11], the total classification error has been reported as a metric of HMM classification performance. Since it is possible for the total error to be low while some words have a high error $\text{Error}(v)$, we will use the maximal word error as our main metric of the classifiers performance. A classifier with low total error but high errors for some words may produce severely undesirable results. We will seek to find the model which minimizes the *maximal word error*, which is defined below.

$$\max_{v \in \mathscr{V}} \text{Error}(v) \tag{33}$$

4.1.4 Cross validation

The cross validation procedure used in this thesis is described and illustrated in [9]. It is the *K-fold cross validation procedure*, which is used to estimate the out-of-sample error. This procedure works in the following way. We randomly partition the whole data set into K sets (called *folds*). We then select a fold as our validation set, and the other folds as the training set. For the next step, a new validation set is chosen and all the other folds (including the previously chosen validation set) form the training set. This step is iterated until the last fold, K, is chosen as the validation set. We illustrate this procedure in the figure below.

fold 1	fold 2	fold 3	fold 4	fold 5
Training	Training	**Validation**	Training	Training

Figure 16: 5-fold cross validation

In our case, every word model requires observations of its corresponding word. Because of this, the procedure described above is performed by

partitioning the set of observations of a given word into K folds. This way, there is always a validation and training set for every given word at every iteration of the cross validation.

The out-of-sample error is estimated with the cross validation. Suppose that all points of the data set are labelled by the integers $1, 2, \ldots, N$. Let $\mathcal{T}(k) = \{V_k, T_k\}$ be a partition of the observation labels into a validation set V_k and training set T_k. The set V_k is the set of labels corresponding to fold k and T_k is the rest of the labels. Also, let m be the codebook size.

First, the test error $\text{Error}_{\mathcal{T}(k)}(m)$ a the set $\mathcal{T}(k)$ is calculated. Finally, the average of these errors for every fold $k = 1, \ldots, K$, denoted by Error, is calculated. These two quantities are described in equations (34). Let W_j denote the observation labelled by j and $\widehat{W}_j^{(k)}(m)$ the result of the word recognizer applied to observation j when it is trained with the training set T_k and has codebook size m. Futhermore, let N_k be the number of observations in the validation set V_k.

$$\text{Error}_{\mathcal{T}(k)}(m) = \frac{1}{N_k} \sum_{j \in V_k} L\left(W_j, \widehat{W}_j^{(k)}(m)\right) \tag{34a}$$

$$\text{Error}(m) = \frac{1}{K} \sum_{k=1}^{K} \text{Error}_{\mathcal{T}(k)}(m) \tag{34b}$$

We can measure the impact that the codebook size m has on the error rate by computing the error $\text{Error}(m)$ for different codebook sizes m.

4.2 The word detector

We begin by explaining how the parameters of the word detector were estimated by using multiple recordings. First, recording was pre-labelled and then each sequence of hidden states was concatenated to form a long sequence of hidden states corresponding to all recordings. The same principle was applied to the emission sequences. In this manner, the maximum likelihood estimator which was designed for a single recording could be used on the entire training data.

We will use a heuristic principle for pre-labelling silence in the start and end of speech recordings for training the word detection model. First, we explore the properties of the following cumulative sum.

$$S_n = \sum_{j=1}^{n} |X_j| \tag{35}$$

The sequence $S_1, S_2, S_3, \ldots$ is monotonically increasing, and it increases in regions where the sound is loud and grows slower or is constant where there is silence. Therefore, if the final part of the recording is silent, the sequence

should grow much slower or remain constant at that segment of the recording. To locate the point of where the sequence grows too slowly, we will divide the cumulative sum by an expression such that we obtain the following sequence.

$$\widetilde{S}_n = \frac{\sum_{j=1}^{n} |X_j|}{1 + \log(n)} \tag{36}$$

The sequence $\widetilde{S}_n$ has been designed such that it decreases when S_n increases too slowly but tends to increase when S_n increases. If there is silence close to the end of the recording, this sequence will have a maximum where the cumulative sum S_n increases slowly. The amplitudes X_n to the right of this maximal $\widetilde{S}_n$ are then labelled as "silence". If this maximum is not unique, we pick the maximum of highest index n. This same principle can be used to locate silence in the beginning of the recording. By reversing the original amplitude sequence $X_1, X_2, X_3, \ldots$, we can use the same expressions and methods as previously described.

Another candidate for penalizing a slow growth of S_n is for example division by n, resulting in a sample mean S_n/n. A general problem for these methods is how one penalizes slow growth while not classifying low-energy speech sounds as "silent". Finally, another altenative is to introduce a threshold for the energy to separate silence from non-silence. The choice of a suitable threshold needs to be such that low-energy speech is not classified as "silence".

In figures 17 and 18 we show the result of applying the silence removal based on sequences $\widetilde{S}_n$ and S_n/n respectively. The methods have been applied to a recording of the word "two" from a male subject with initials GR. The file is called `2A_endpt.wav`.

In both figures, the blue lines represent the penalized cumulative sums, which have been normalized to be visible on the plot. The vertical black lines represent the detected boundary of the silence at the start and end of the recording. For example, in figure 17 we see that the sequence $\widetilde{S}_n$, in the right direction, has a maximum when the time is close to 0.4 seconds. The sequence $\widetilde{S}_n$ in the left direction has a maximum slightly before 0.05 seconds. The location of the maximal values of the sequences are marked with the black vertical lines.

In this specific case of the recorded word "two", the performance of the silence removal at the extremities is better when using the sequence $\widetilde{S}_n$. In figure 18, we see that the left boundary is placed inside the region corresponding to the spoken word. This is bad because it means that the "t" sound is classified as silence. Furthermore, a part of the vowel-sound of "two" is also classified as silence. Having n as a denominator penalizes slow growth of S_n more than $1 + \log(n)$. The consequence is that the sequence S_n/n reaches its maximum quicker than $\widetilde{S}_n$ does. In this case, the maximum is reached too quickly for the sequence S_n/n in both directions.

In this study we used the sequence $\widetilde{S}_n$ for pre-labelling the data for the word-detection model. To further inspect if this method is appropriate, it has been tested some other recordings in our data set, with decent results. We note that this is only a heuristic method and not necessarily the best one.

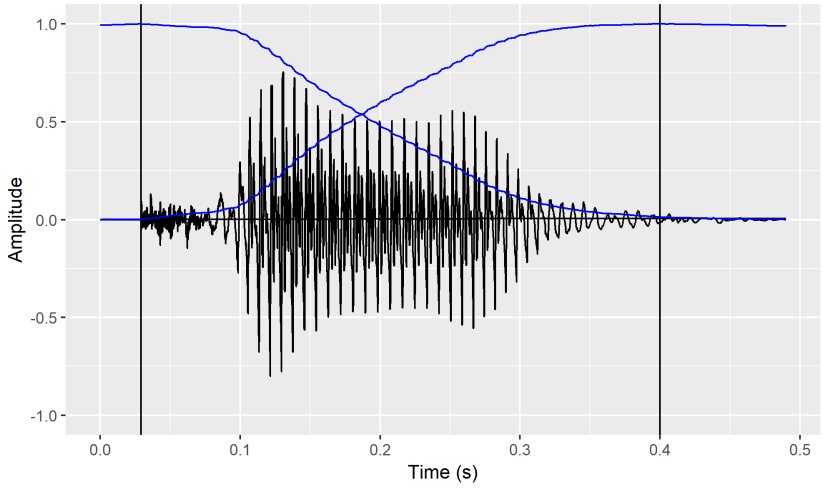

Figure 17: Silence remover, using the sequence $\widetilde{S}_n = S_n/(1 + \log(n))$.

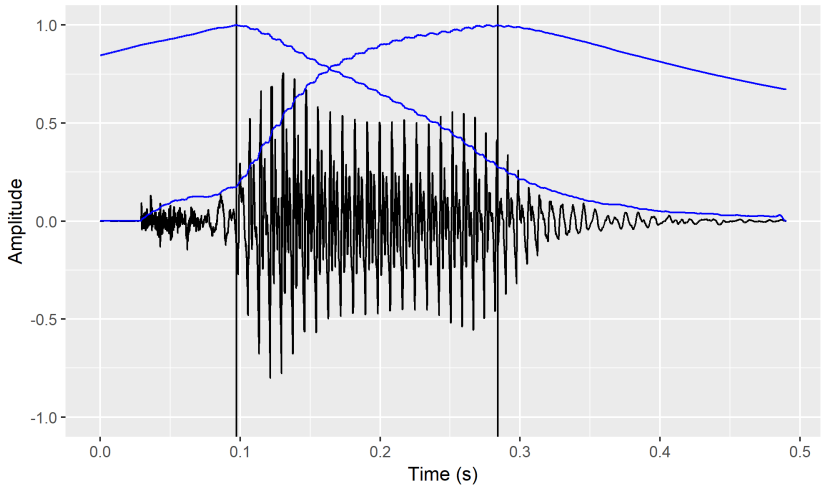

Figure 18: Silence remover, using the sequence S_n/n.

5 Results

The results and analysis of the word recognition experiments has been divided into separate subsections. The first subsections *describe* the results

for each of the three experiments and an *error analysis* is given in another subsection. Figures which illustrate different kinds of errors will be shown to *describe* the results.

The different word recognizers were tested and trained using only the male TIDIGITS data set. We have divided the model training and testing into three sections. In the first section, we show the models which were fitted to Xin and Qi's male training subjects. In the next section, we describe the estimate the out-of-sample error by cross-validation. Lastly, training and testing of the models on the full TIDIGITS male data is described. The codebook sizes which were tested in this study were of the following sizes: $\{2, 4, 8 \ldots, 256\}$. For all of the following three cases, we begin by showing the total errors and the maximum word error for each codebook size. Next, we show the word errors for the models with lowest total error and lowest maximum word-error. In the plots, the word "zero" is denoted by "Z".

5.1 Xin and Qi's training set

In this section, we describe the performance of the models fitted on the training data consisting of the 33 male subjects that Xin and Qi have used for training a HMM speech recognizer [14]. In Xin and Qi's implementation, the word "o" was excluded from the analysis, but we have decided to include it to the study. The remaining male observations were used for testing these models in this study.

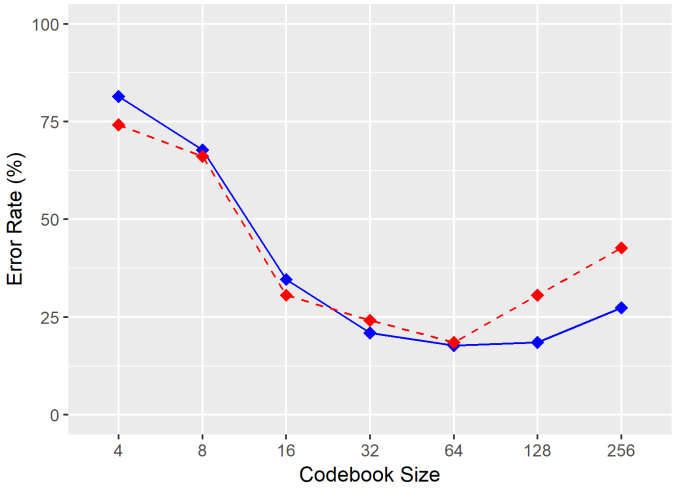

Figure 19: Total classification errors. In red: phoneme-sized word recognizer. In blue: the 5-state word recognizer.

We observe that the phoneme-sized word recognizer has a slightly lower classification error for codebook-sizes up to 16. There is a minimum for

both word recognizers at the codebook size of 64, with the 5-state word recognizer having the error of 17.7 % and the other having 18.5 % error. In the next figure, the maximal word error for each codebook size is shown. We observe that there is a minimum at codebook size 128 for the 5-state word recognizer. We also observe that although the total error is lower than 50 % for recognizers with codebooks larger than 8, the maximal word error is larger than 50 % for every codebook size except for 128.

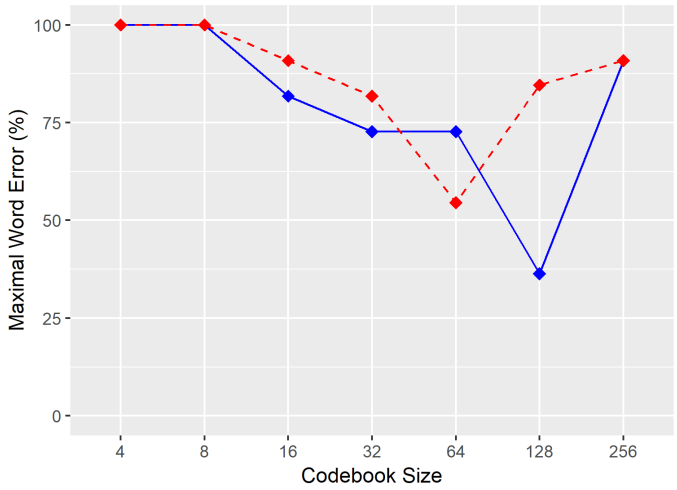

Figure 20: Maximal word classification errors. In red: phoneme-sized word recognizer. In blue: the 5-state word recognizer.

In the next figure, we show the error rates of two 5-state classifiers: the one with smallest maximal word error (in brown) and the one with smallest total error (in black). We see that the classifier with lowest total error has the very high error rate of 72.7 % for the word "nine".

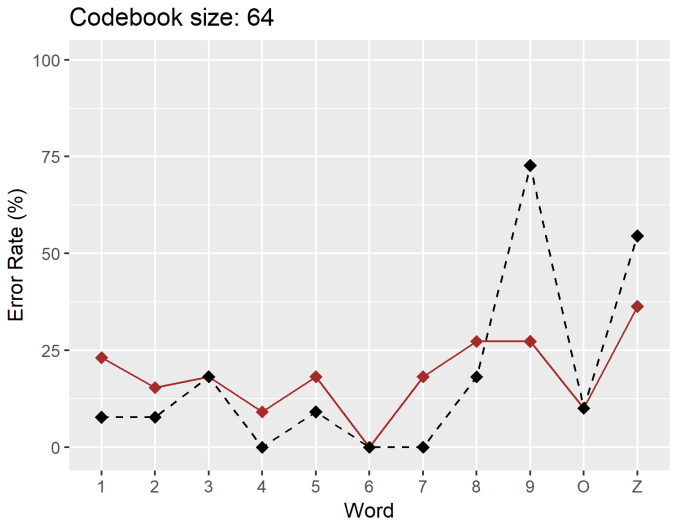

Figure 21: In brown: classifier with lowest maximal word error. In black: classifier with lowest total error.

5.2 Cross-validation results

For the cross-validation, we divided the full male data set into 10 approximately equally sized folds. The error rates within these folds were calculated and then the average μ of these values was calculated to estimate the total error. In the figures below we have visualized the estimated error with error bars. The midpoint represents the estimated mean μ of the errors while the upper and lower horizonal segments were obtained by adding and subtracting the estimated standard deviation σ.

We observe that, like in the previous case, the lowest total errors are obtained for codebook size 64. In the next figure, we observe that the smallest maximal word error which equals 46.7% is obtained for a 5-state recognizer with a size 32 codebook. Apart from this recognizer and the 5-state recognizer with the 64-sized codebook, every other classifier has the maximal word error over 50 %.

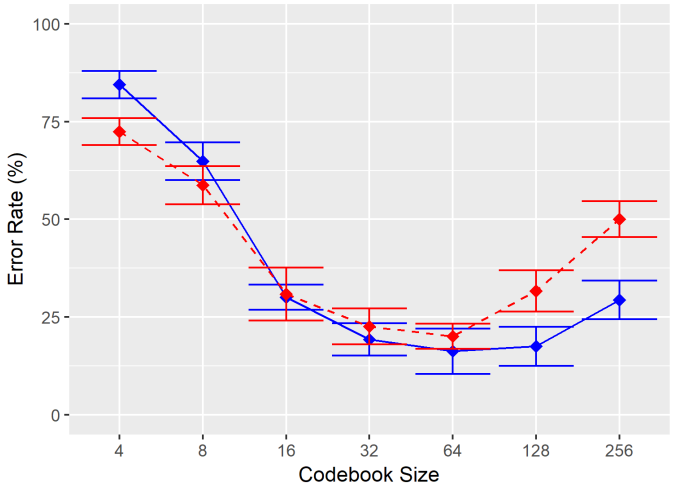

Figure 22: Cross validation results of the error rates. In red: phoneme-sized word recognizer. In blue: the 5-state word recognizer.

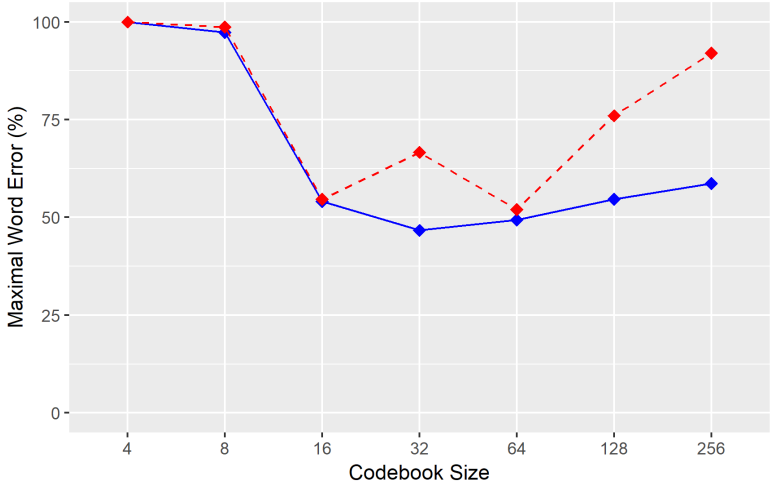

Figure 23: Cross validation results of the error rates. In red: phoneme-sized word recognizer. In blue: the 5-state word recognizer.

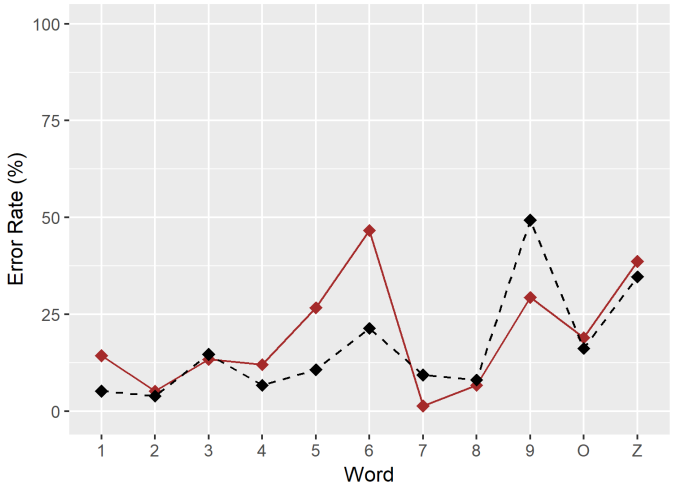

Figure 24: In brown: classifier with lowest maximal word error. In black: classifier with lowest total error.

5.3 The entire male data set

In this section, we have used the entire male data set for fitting and testing. The model with the lowest total error and lowest max-word-error were of the same dimensions as those obtained from the cross-validation: both were 5-state word recognizers with codebook sizes 64 and 32 respectively. There were convergence issues in the training procedure (EM-algorithm estimation) for both kinds of word recognizers, when the codebook size was 4. For an overview of the all the results of the experiments, see table 4 below.

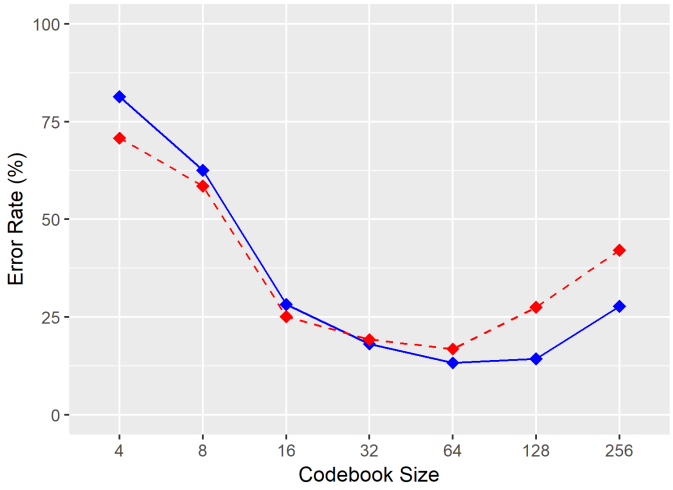

Figure 25: Total classification errors. In red: phoneme-sized word recognizer. In blue: the 5-state word recognizer.

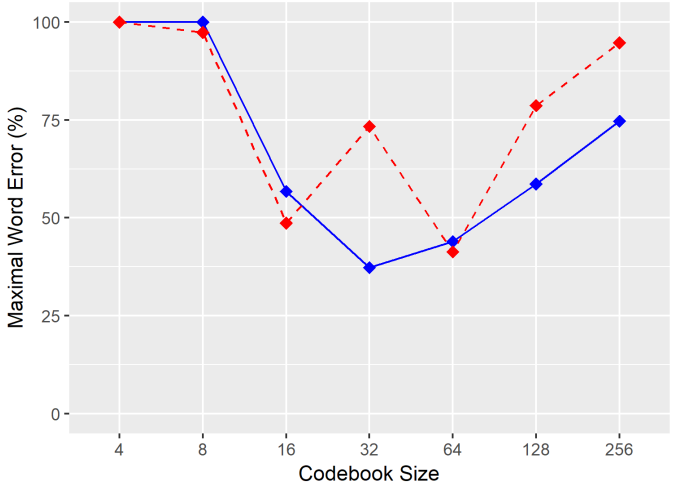

Figure 26: Classification errors by word. In red: phoneme-sized word recognizer. In blue: the 5-state word recognizer.

5.3 The entire male data set

Lowest total error	**Lowest max-word-error**	**Case**
Codebook size 64 Type: 5 state Total error: 17.7 % Max. word error: 72.7 %	Codebook size: 128 Type: 5 state Total error: 18.5 % Max. word error: 36.4 %	Xin and Qi's training data
Codebook size: 64 Type: 5 state Total error: 16.3 % Max. word error: 49.3 %	Codebook size: 32 Type: 5 state Total error: 19.3 % Max. word error: 46.7 %	Cross-Validation
Codebook size: 64 Type: 5 state Total error: 13.3 % Max. word error: 44 %	Codebook size: 32 Type: 5 state Total error: 18.1 % Max. word error: 37.3 %	Entire male data set

Table 4: Summary of the results

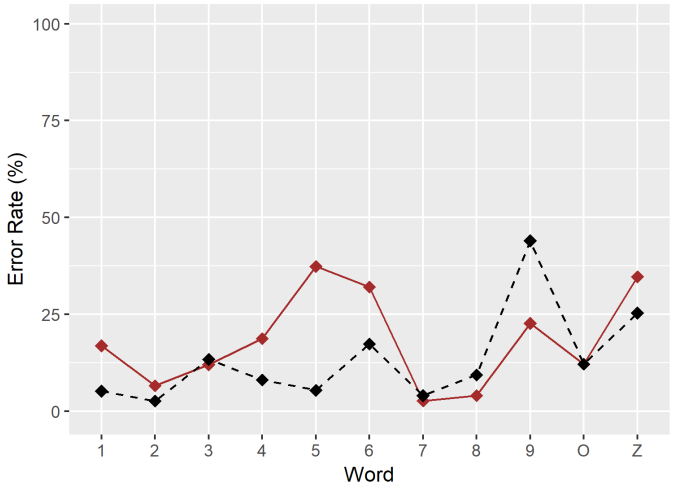

Figure 27: In brown: classifier with lowest maximal word error. In black: classifier with lowest total error.

We will now illustrate the classifications of the word recognizer with the lowest maximal word error for the entire male data set in the classification table below. In the rows we have the group of observations corresponding to a given word and in the columns we have the classifications made. In each cell is the count of the number of classifications made for a given word, with a darker shade of blue for high counts.

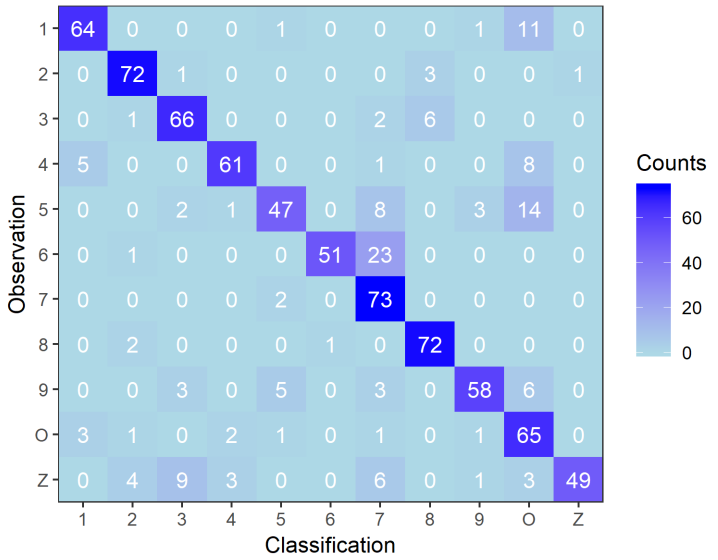

Figure 28: Classification table plot for a 5-state word recognizer with codebook size 32 fitted on the whole data set.

5.4 Error analysis of word recognition

In this section we will explore the mechanisms that produce the classification errors. An error occurs when an observation $\boldsymbol{o}$ corresponding to a word v does not get the highest likelihood under the corresponding HMM with parameter collection λ_v. We observe that each parameter collection λ_v is *separately* estimated for every word v and therefore the *classification error* is not necessarily minimized when by the HMM parameter estimation.

The *codebook* generated for the discretization of the feature vectors can be overfitted in a way analogous to an overfitted simple linear regression model. This is because the codebook is generated by minimizing distances between feature vectors and the codebook vectors. If a codebook is overfitted to the training data, the discretization might perform worse for the validation data.

There are 38 phonetic symbols in table 2, out of which 20 are unique. These symbols represent the different speech sounds that are present in our data set. If a hidden state of a word model HMM is assumed to produce the sounds that are represented by these phonetic symbols, then having a codebook which is too large is undesirable. Having a codebook with the same size as number of phonetic symbols might also be inadequate due to the possible variations in pronounciation that arise due to for example age or differences in sex.

The discretized values o_ℓ are not necessarily unique to a specific word model, and hence different word models might be capable of emitting the

same values. For this reason, it is possible that certain emission sequences $\boldsymbol{o}$ can receive the highest likelihood even if it corresponds to the incorrect word. This can for example occur for words that sound similar and have similar feature vectors.

To investigate why e.g. the word "nine" gets such a high maximal word error under the various word recognizers as shown in figures 21, 24 and 27 it is appropriate to investigate the errors for this word under a specific word model. In figure 28, we can see that recordings of the word "nine" are incorrectly classified 17 times out of 75 times for the final model. The most frequent incorrect classifications of recordings of "nine" are those with "o" and "five". If we take a look at the incorrect classifications of recordings of "five" in figure 28, we see that the most frequent is "nine". A possible explanation of some errors is that there is a similarity in the feature vectors, since both words share the phonemes a and ɪ as shown in the list of phonetic symbols in table 2. For comparison, the word "six" is misclassified as "seven" quite often, and both share the same first sound "s". It is also noteworthy that "seven" is not misclassified as "six" at all.

5.5 Word detection results

For this analysis, the upper limit of the discretizer was set to 20. Some emission values had 0 counts and this caused errors when using the Viterbi Algorithm. To solve this problem, the word detection model was trained by using the `Pseudoemissions` argument from the `hmmestimate`-function of MATLAB's Statistics and Machine Learning Toolbox. The pseudocounts-setting solves the problem by assigning the emission probabilities, corresponding to low-count-emissions, with pre-defined numbers.

The results of the word detector were evaluated by listening to the segments classified as silence, and by visual inspection of waveform. In the figures below, we present some plots of the word-detection results for a couple of examples of the word "six" and "eight". The black line represents the waveform and the blue line represents the most probable hidden state sequence.

As noted before, the phonemes for the letters "s", "x" and "t" have low energy. In the following plots it can be observed that these low-energy phonemes were sometimes incorrectly labelled as silence. The figures 29 and 30 show waveforms of the word "six", where the "x" is separated from the rest of the word by a short silence. In figure 29, the "x" is incorrectly classified as silence. After these two plots, two waveforms of the word "eight" are shown in figures 31 and 32. The phoneme of "t" is separated from the rest of the word by silence in both cases. In figure 31, the "t" is incorrectly labelled as silence. However, the "t" is detected in figure 32. These results are not ideal if the word detector is to be used together with a word-classifier, because valuable information might be lost.

There are many possible origins of the errors which the word detector produces. Since the model always assumes that we start at a silent state, this will affect the hidden sequence which is detected by the Viterbi Algorithm. Another error-producing mechanism is that the word detection model was trained on pre-labelled data with a method which is *only* designed to label silence at the start and the end of a recording. If a silent segment exists within a spoken word, the heuristic pre-labelling procedure labels it as non-silent, with a 0. Even though the training data was pre-labelled in this manner, the word detector was still able to locate silence *within* a word, as shown in figures 30 and 32. In conclusion, the pre-labelling procedure has labelled both silent segments within a word as non-silent and silent segments at the extremes as silent and it is reasonable to assume that this is a factor behind the inconsistent word-detection.

Another factor behind the word-detection is the discretization parameter u which determines the number of discrete energy levels. As with any discretization procedure, this removes some information which might be valuable. Figures 29 to 32 suggest that consonants tend to have much lower energy levels than vowels. Because of the difference in energy levels between consonants and vowels, it is reasonable to let u be large *enough* for the discretization to assign separate values to: pure silence, i.e. when the frame energy is *zero*, and to a frame of a *low-level energy speech sound*. One must be careful to not let the parameter u be too large as it would break the principle of parsimony by producing too many parameters to estimate for the word-detection HMM.

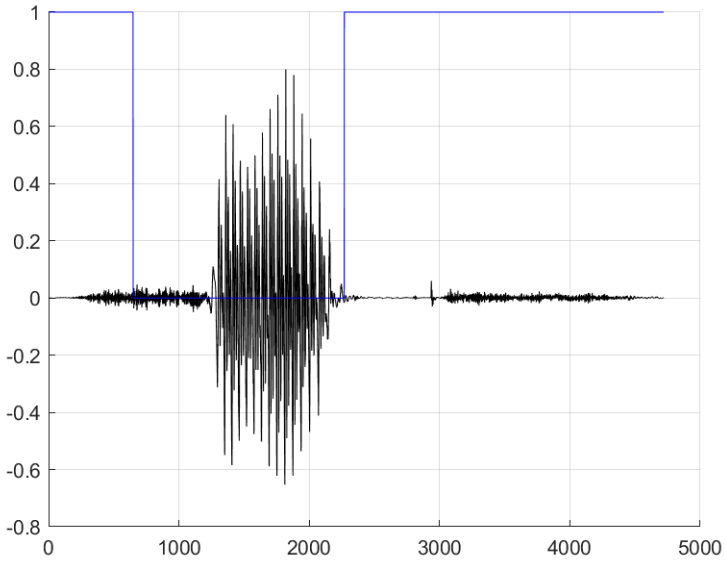

Figure 29: Word: “six”. Male subject: AE. File: `6A_endpt.wav`

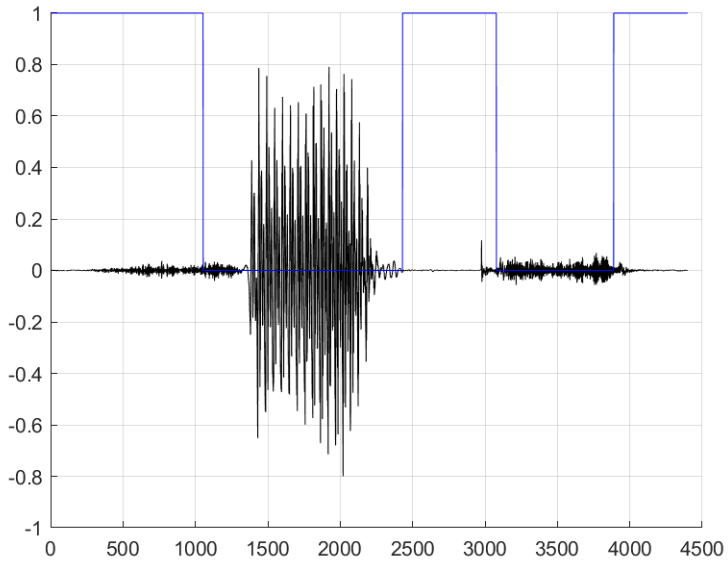

Figure 30: Word: "six". Male subject: EL. File: `6B_endpt.wav`

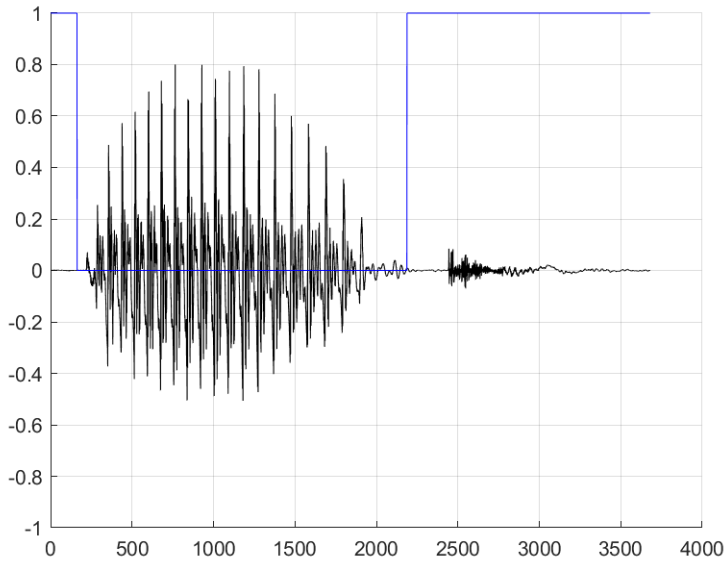

Figure 31: Word: "eight". Male subject: GT. File: `8A_endpt.wav`

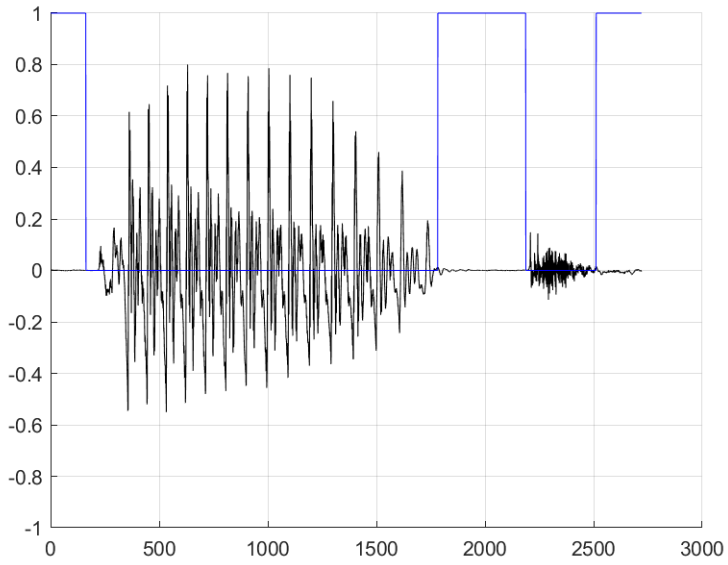

Figure 32: Word: "eight". Male subject: GT. File: `8B_endpt.wav`

6 Discussion and conclusion

In this study we used the theory of HMM's to solve two types of problems: word recognition and word detection. The models were trained and tested on the recordings of the male subjects set from the TIDIGITS data. We have examined the performance of different word recognizers with respect to the total error and error by word. We conclude that letting the word models have a size 5 of the hidden state space produced word recognizers with lower total and maximal word error.

Ideally, the final model has minimal classification error and as few parameters as possible to avoid overfitting. In our case, we note that the model with lowest numbers of parameters among the models in table 4 is the 5-state system with codebook size 32. Although the total error was estimated to be 19.3 % in the cross-validation, this system had the high word error rate of 46.7 % for the word "six".

Compared to word recognition, it is not as straightforward to evaluate the performance of the word detector since the data is only heuristically pre-labelled at the tails. It is very time-consuming to manually inspect several files, and part of the problem is to find an optimal way of separating the word segments from silent segments automatically. A possible way of evaluating the word detector is to combine it with a word recognizer and investigate if there is any improved performance. After all, the idea of the word detector is to detect the segments with valuable information for word

recognition. The performance of the word detector could be investigated by changing the discretization range of the energy levels. Another alternative is to study the effect of not pre-labelling the training data, i.e. to train a HMM unsupervised.

We conclude that our word detector sometimes fails to detect phonemes with low energies. Another suggestion for improvement is to study the frequency content in low-energy noise compared to low-energy phonemes, for example by using another feature for word detection such as the *zero crossing rate*. This variable measures the rate by which the signal crosses zero, and has been proposed as a variable which facilitates separating voiced speech from unvoiced speech. In a previous study, unvoiced speech has been observed to be associated to having low energy and high zero crossing rates and voiced speech to having high energy and low zero crossing rate [4].

Finally, there are many questions regarding the MFCC feature vectors that require more exploration. The results indicate that these features are useful for classifying the word recordings from our subset of the TIDIGITS data.

7 Appendix

7.1 Details of the Forward algorithm

In this section we explain the Forward algorithm, which is a method for computing the probability of a sequence of emissions. We will use the following notation for sequences: $O_{1:\ell} = (O_1, \ldots, O_\ell)$. Let $\boldsymbol{O} = (O_1, \ldots, O_L)$ and $\alpha_\ell(i) = \mathbb{P}(O_{1:\ell} = o_{1:\ell}, Q_\ell = i \mid \lambda)$. The probability under the HMM with the parameter collection λ is denoted by $\mathbb{P}_\lambda$. By using the law of total probability, we have that

$$\mathbb{P}_\lambda(\boldsymbol{O} = \boldsymbol{o}) = \sum_i \mathbb{P}_\lambda(\boldsymbol{O} = \boldsymbol{o}, Q_L = i) = \sum_i \alpha_L(i).$$

For $\ell > 1$, the iterative step of the forward algorithm uses the following expression.

$$\alpha_\ell(j) = \sum_i \alpha_{\ell-1}(i) a_{i,j} b_j(o_\ell).$$

We will now prove that the above formula holds for all states j and $\ell > 1$. By the law of total probability,

$$\alpha_\ell(j) = \sum_i \mathbb{P}_\lambda(O_{1:\ell} = o_{1:\ell}, Q_{\ell-1} = i, Q_\ell = j). \tag{37}$$

To check if the terms of the last two sums equal eachother, the terms of the latter sum (37) will be factorized by conditioning.

$$\begin{aligned} &\mathbb{P}_\lambda(O_{1:\ell} = o_{1:\ell}, Q_{\ell-1} = i, Q_\ell = j) = \\ &\mathbb{P}_\lambda(O_{1:\ell} = o_{1:\ell} \mid Q_{\ell-1} = i, Q_\ell = j) \mathbb{P}_\lambda(Q_{\ell-1} = i, Q_\ell = j) \end{aligned} \tag{38}$$

We now check the two factors of the right-hand side separately. For the first factor of (38), we use conditional independence (8) to obtain

$$\begin{aligned} \mathbb{P}_\lambda(O_\ell = o_\ell \mid Q_\ell = j) \mathbb{P}_\lambda(O_{1:(\ell-1)} = o_{1:(\ell-1)} \mid Q_{\ell-1} = i) = \\ b_j(o_\ell) \mathbb{P}_\lambda(O_{1:(\ell-1)} = o_{1:(\ell-1)} \mid Q_{\ell-1} = i). \end{aligned}$$

For the second factor of (38), we use the Markov property.

$$\mathbb{P}_\lambda(Q_\ell = j \mid Q_{\ell-1} = i) \mathbb{P}_\lambda(Q_{\ell-1} = i) = a_{i,j} \mathbb{P}_\lambda(Q_{\ell-1} = i)$$

We combine the new expressions for the factors of (38) and obtain

$$b_j(o_\ell) \mathbb{P}_\lambda(O_{1:(\ell-1)} = o_{1:(\ell-1)}, Q_{\ell-1} = i) a_{i,j} = \alpha_{\ell-1}(i) a_{i,j} b_j(o_\ell).$$

7.2 The Viterbi algorithm

The algorithm begins by initializing the values for $\phi_\ell(i)$ in (39a), and then uses the recursive relation (39c) between $\phi_\ell(i)$ and $\phi_{\ell-1}(i)$ to progressively compute the maximal probabilities for hidden state sequences of various steps ℓ. The recursive relation is useful because it gives that the optimal sequence $q^*_{1:\ell}$ begins with a shorter optimal sequence $q^*_{1:(\ell-1)}$, and this is the used in the backtracking step. The log-probabilities of the optimal sequences are stored in an array of values $\{\psi_\ell(i)\}$. The procedure ends in the final state of the most probable sequence, and backtracks to the first state of that sequence.

Step 1: Initialization

$$\phi_\ell(i) = \log(\pi_i) + \log(b_i(o_1)) \text{ for } 1 \leq i \leq n. \tag{39a}$$

$$\psi_1(i) = 0 \text{ for } 1 \leq i \leq n. \tag{39b}$$

Step 2: Recursion

$$\phi_\ell(j) = \max_i \ (\phi_{\ell-1}(i) + \log(a_{i,j})) + \log\left(b_j(o_\ell)\right) \text{ for } 2 \leq \ell \leq L, 1 \leq j \leq n. \tag{39c}$$

$$\psi_\ell(j) = \operatorname*{argmax}_i \ (\phi_{\ell-1}(i) + \log(a_{i,j})) \text{ for } 2 \leq \ell \leq L, 1 \leq j \leq n. \tag{39d}$$

Step 3: Termination

$$\log P^* = \max_i \ (\phi_L(i)) \tag{39e}$$

$$q^*_L = \operatorname*{argmax}_i \ (\phi_L(i)) \tag{39f}$$

Step 4: Backtracking

$$q^*_\ell = \psi_{\ell+1}\left(q^*_{\ell+1}\right) \text{ for } \ell = L-1, L-2, \ldots, 1 \tag{39g}$$

7.3 EM-estimation

The estimation formulas presented in this section are those found in Rabiner's tutorial.

$$Q\left(\lambda \mid \lambda^{(t)}\right) = \sum_{\boldsymbol{q}} p\left(\boldsymbol{q} \mid \boldsymbol{o}, \lambda^{(t)}\right) \log p\left(\boldsymbol{o}, \boldsymbol{q} \mid \lambda\right) \tag{40a}$$

$$\lambda^{(t+1)} = \operatorname*{argmax}_\lambda Q(\lambda \mid \lambda^{(t)}) \tag{40b}$$

As stated in the main section, it holds that $\log p\left(\boldsymbol{o} \mid \lambda^{(t+1)}\right) \geq \log p\left(\boldsymbol{o} \mid \lambda^{(t)}\right)$, i.e. the new estimate has a higher likelihood. As the number of iterations grow, the estimates converge to an estimate with the highest likelihood.

7.3.1 Improved estimates

We begin by defining following quantity. In this section, we will explain why the the estimate at the next iteration $\lambda^{(t+1)}$ has a higher likelihood than $\lambda^{(t)}$. First, we define the function H.

$$H\left(\lambda \mid \lambda^{(t)}\right) = \mathbb{E}\left(\log p(\boldsymbol{Q} \mid \boldsymbol{O}, \lambda) \mid \boldsymbol{O} = \boldsymbol{o}, \lambda^{(t)}\right) \tag{41a}$$

By can factorize $p(\boldsymbol{o}, \boldsymbol{q} \mid \lambda)$ by conditioning, and obtain that $\log p(\boldsymbol{o}, \boldsymbol{q} \mid \lambda) = \log p(\boldsymbol{o} \mid \lambda) + \log p(\boldsymbol{q} \mid \boldsymbol{o}, \lambda)$. By combining this with the definition of the auxiliary function Q, we obtain the next result.

$$Q\left(\lambda^{(t+1)} \mid \lambda^{(t)}\right) = \log p\left(\boldsymbol{o} \mid \lambda^{(t+1)}\right) + H\left(\lambda^{(t+1)} \mid \lambda^{(t)}\right) \tag{42}$$

We can now proceed by proving a lemma which will be used later. The claim of the lemma is the following. For all $\lambda^{(t)}, \lambda^{(t+1)}$ in the parameter space it holds that:

$$H\left(\lambda^{(t)} \mid \lambda^{(t)}\right) \geq H\left(\lambda^{(t+1)} \mid \lambda^{(t)}\right) \tag{43}$$

We now prove this lemma. First, assume that the sequence $\boldsymbol{Q}$ takes values in the sample space $\mathscr{Q}$ which does not depend on the parameterization λ. Assume that the $p(\boldsymbol{q} \mid \boldsymbol{o}, \lambda) > 0$ for all sequences $\boldsymbol{q}$ in the sample space and all parameters λ. The steps of the proof are given in the equations (44).

$$H\left(\lambda^{(t)} \mid \lambda^{(t)}\right) - H\left(\lambda^{(t+1)} \mid \lambda^{(t)}\right) = \tag{44a}$$

$$\mathbb{E}\left(\log p(\boldsymbol{Q} \mid \boldsymbol{O}, \lambda^{(t)}) \mid \boldsymbol{O} = \boldsymbol{o}, \lambda^{(t)}\right) - \mathbb{E}\left(\log p(\boldsymbol{Q} \mid \boldsymbol{O}, \lambda^{(t+1)}) \mid \boldsymbol{O} = \boldsymbol{o}, \lambda^{(t)}\right) \overset{\text{linearity of expectations}}{=} \tag{44b}$$

$$\mathbb{E}\left(-\log \frac{p\left(\boldsymbol{Q} \mid \boldsymbol{O}, \lambda^{(t+1)}\right)}{p\left(\boldsymbol{Q} \mid \boldsymbol{O}, \lambda^{(t)}\right)} \mid \boldsymbol{O} = \boldsymbol{o}, \lambda^{(t)}\right) \overset{\text{Jensen's ineq.}}{\geq} \tag{44c}$$

$$-\log \mathbb{E}\left(\frac{p\left(\boldsymbol{Q} \mid \boldsymbol{O}, \lambda^{(t+1)}\right)}{p\left(\boldsymbol{Q} \mid \boldsymbol{O}, \lambda^{(t)}\right)} \mid \boldsymbol{O} = \boldsymbol{o}, \lambda^{(t)}\right) = \tag{44d}$$

$$-\log \sum_{\boldsymbol{q}} p\left(\boldsymbol{q} \mid \boldsymbol{o}, \lambda^{(t)}\right) \cdot \left(\frac{p\left(\boldsymbol{q} \mid \boldsymbol{o}, \lambda^{(t+1)}\right)}{p\left(\boldsymbol{q} \mid \boldsymbol{o}, \lambda^{(t)}\right)}\right) = \tag{44e}$$

$$-\log \sum_{\boldsymbol{q}} p\left(\boldsymbol{q} \mid \boldsymbol{o}, \lambda^{(t+1)}\right) = -\log(1) = 0 \tag{44f}$$

We conclude this section by proving that the next estimate $\lambda^{(t+1)}$ has a higher likelihood than the previous one, $\lambda^{(t)}$.

$$\log p(\boldsymbol{o} \mid \lambda^{(t+1)}) - \log p(\boldsymbol{o} \mid \lambda^{(t)}) = \tag{45a}$$

$$\left(Q\left(\lambda^{(t+1)} \mid \lambda^{(t)}\right) - H\left(\lambda^{(t+1)} \mid \lambda^{(t)}\right)\right) - \left(Q\left(\lambda^{(t)} \mid \lambda^{(t)}\right) - H\left(\lambda^{(t)} \mid \lambda^{(t)}\right)\right) = \tag{45b}$$

$$\underbrace{\left(Q\left(\lambda^{(t+1)} \mid \lambda^{(t)}\right) - Q\left(\lambda^{(t)} \mid \lambda^{(t)}\right)\right)}_{\geq 0 \text{ by assumption}} + \underbrace{\left(H\left(\lambda^{(t)} \mid \lambda^{(t)}\right) - H\left(\lambda^{(t+1)} \mid \lambda^{(t)}\right)\right)}_{\geq 0 \text{ by lemma (43)}} \tag{45c}$$

7.3.2 Single sequence estimation

We begin by stating the maximization problem that is the M-step of the EM-algorithm, together with the constraints imposed on the parameter collection λ.

$$\max_{\lambda} Q\left(\lambda \mid \lambda^{(t)}\right) \tag{46a}$$

$$\text{s.t. } g(\lambda) \ = 1 - \sum_{i=1}^{n} \pi_i = 0 \tag{46b}$$

$$h_i(\lambda) = 1 - \sum_{j=1}^{n} a_{i,j} = 0 \quad \text{for } 1 \leq i \leq n \tag{46c}$$

$$k_i(\lambda) = 1 - \sum_{j=1}^{m} bi\,(j) = 0 \quad \text{for } 1 \leq i \leq n \tag{46d}$$

This problem is solved with the *Lagrange multiplier method*. This is achieved by by finding the coefficients u, v_i, w_i such that (47) is satisfied.

$$\nabla Q\left(\lambda \mid \lambda^{(t)}\right) + u\nabla g(\lambda) + \sum_{i=0}^{n} v_i \nabla h_i(\lambda) + \sum_{i=0}^{n} w_i \nabla k_i(\lambda) = \mathbf{0} \tag{47}$$

We now proceed by describing the solution to this optimization problem described by the equations (46). Assume that the emission sequence $\boldsymbol{o} = (o_1, \ldots, o_L)$ has been observed and we want to estimate the parameters of the HMM. We will denote the current estimate by λ. We begin by defining the following quantities.

$$\beta_\ell(i) = \mathbb{P}_\lambda(O_{(\ell+1):L} = o_{(\ell+1):L} \mid Q_\ell = i) \tag{48a}$$

$$\xi_\ell(i,j) = \mathbb{P}_\lambda(Q_\ell = i, Q_{\ell+1} = j \mid \boldsymbol{O} = \boldsymbol{o}) \tag{48b}$$

$$\gamma_\ell(i) = \mathbb{P}_\lambda(Q_\ell = i \mid \boldsymbol{O} = \boldsymbol{o}) \tag{48c}$$

The first quantity, $\beta_\ell(i)$, is known as the *backward probability*. We proceed by expressing these quantities in terms of our parameters and the forward- and backward probabilities.

$$\xi_\ell(i,j) = \frac{\alpha_\ell(i)a_{i,j}b_j(o_{\ell+1})\beta_{\ell+1}(j)}{\mathbb{P}_\lambda(\boldsymbol{O}=\boldsymbol{o})} \tag{49a}$$

$$\gamma_\ell(i) = \frac{\alpha_\ell(i)\beta_\ell(i)}{\mathbb{P}_\lambda(\boldsymbol{O}=\boldsymbol{o})} \tag{49b}$$

We are now ready to fully describe the computation of the parameter estimates for an emission sequence. We denote the current estimate by λ. For a left-right HMM, the parameter Π is not estimated since the process is assumed to start at the first state. For this reason, it will be omitted.

$$\widehat{a}_{i,j} = \frac{\sum_{\ell=1}^{L-1}\xi_\ell(i,j)}{\sum_{\ell=1}^{L-1}\gamma_\ell(i)} \tag{50a}$$

$$\widehat{b}_j(r) = \frac{\sum_{\ell=1}^{L}{}_{\text{s.t. } O_\ell=r}\gamma_\ell(j)}{\sum_{\ell=1}^{L}\gamma_\ell(j)} \tag{50b}$$

We note that the probability $\widehat{b}_j(r)$ equals the element $\widehat{b}_{j,r}$ of the estimated emission probability matrix $\widehat{B}$. The parameter collection at the next iteration step is then $\widehat{\lambda} = \left(\widehat{A}, \widehat{B}\right)$, whose elements are given by the expressions above.

7.3.3 Multiple sequence estimation

Assume we have multiple emission sequences: $\boldsymbol{O}^{(1)}, \boldsymbol{O}^{(2)}, \ldots, \boldsymbol{O}^{(K)}$. We denote the number of elements in sequence $\boldsymbol{O}^{(k)}$ by L_k. The variables γ_ℓ, ξ_ℓ which previously corresponded to a single sequence, $\boldsymbol{o}$, will now be denoted by $\gamma_\ell^{(k)}$ and $\xi_\ell^{(k)}$ when they are evaluated at an emission sequence $\boldsymbol{o}^{(k)}$. The forward- and backward-variables will be denoted similarly. We get the following equalities.

$$\xi_\ell^{(k)}(i,j) = \frac{\alpha_\ell^{(k)}(i)a_{i,j}b_j(o_{\ell+1})\beta_{\ell+1}^{(k)}(j)}{\mathbb{P}_\lambda(\boldsymbol{O}^{(k)}=\boldsymbol{o}^{(k)})} \tag{51a}$$

$$\gamma_\ell^{(k)}(i) = \frac{\alpha_\ell^{(k)}(i)\beta_\ell^{(k)}(i)}{\mathbb{P}_\lambda(\boldsymbol{O}^{(k)}=\boldsymbol{o}^{(k)})} \tag{51b}$$

These expressions are used to compute the parameter estimation formulas below.

$$\widehat{a}_{i,j} = \frac{\sum_{k=1}^{K} \sum_{\ell=1}^{L_k-1} \xi_\ell^{(k)}(i,j)}{\sum_{k=1}^{K} \sum_{\ell=1}^{L_k-1} \gamma_\ell^{(k)}(i)} \tag{52a}$$

$$\widehat{b}_j(r) = \frac{\sum_{k=1}^{K} \sum_{\substack{\ell=1 \\ s.t.\ O_\ell^{(k)}=r}}^{L_k-1} \gamma_\ell^{(k)}(j)}{\sum_{k=1}^{K} \sum_{\ell=1}^{L_k-1} \gamma_\ell^{(k)}(j)} \tag{52b}$$

7.4 MFCC computation

In this section we will describe the feature extraction procedure with more details. The formulas come from Ganchev's description of the MFCC-computation from Auditory Toolbox by Slaney (1998), referred to as the MFCC-FB40 implementation [7]. The name stems from the filter bank having 40 filters, and has been adapted to signals with sample frequency 16000 Hz. If the sampling frequency of the signal is 8000 Hz, only the filters below 4000 Hz are kept.

Before the computation of the MFCC-coefficients begins, the amplitude sequence $\boldsymbol{X}$ of a recording is pre-processed by two techniques: *pre-emphasis* and *windowing*. The amplitude sequence is assumed to have been grouped in *frames*.

Ganchev notes that the pre-emphasis is applied in order to reduce the magnitude at lower-frequencies and increase the magnitude at higher frequencies in the recording. We denote the pre-emphasized signal by $\widetilde{X}_t$.

$$\widetilde{X}_t = X_t - aX_{t-1} \tag{53}$$

In the `mfcc`-function in Xin and Qi's MATLAB implementation, the pre-emphasis (53) is performed with $a = 0.97$.

The second step of the pre-processing is the *windowing*. In this step, the frames of the pre-emphasized amplitude sequence $\widetilde{\boldsymbol{X}}$ will be weighted by a windowing function in order to reduce distortion of the frequency content of the frame. The windowing function is known as the *Hamming window*, and is defined in equation (54). Let N be the number of points in a given frame of the pre-emphasized signal.

$$W(i) = \begin{cases} 0.54 - 0.46\cos\left(\frac{2\pi i}{N}\right), & \text{for } i = 0, 1, \ldots, N-1 \\ 0 & \text{for other } i \end{cases} \tag{54}$$

We can now proceed to describe the Discrete Fourier Transform (DFT) of a pre-processed frame. Let $s(n)$ denote the n:th point of the pre-processed frame, which contains N points. Let j denote the imaginary unit.

$$S(k) = \sum_{n=0}^{N-1} s(n) \cdot \exp\left(\frac{-j2\pi nk}{N}\right) \text{ for } k = 0, 1, \ldots, N-1 \tag{55}$$

The value $S(k)$ from equation (55) represent the Fourier coefficient of index k. By computing $|S(k)|$, we obtain the magnitude of the signal at the corresponding frequency f_k. The next step is to send the magnitudes to the filter bank. In order to give the complete formula for this, we will first define the filter bank.

The filter bank is composed of 40 filters. The first 13 of the filters are linearly spaced and the rest are uniformly spaced in the Mel-scale. Each of

these filters is triangularly-shaped. The points at the base are located in the frequency-axis as shown in figure 5. The points at the base of the triangle for filter i are: the center frequency f_{c_i} in the middle, the left boundary $f_{b_{i-1}}$ and the right boundary f_{b_i}. The boundary points are such that $f_{b_i} = f_{c_{i-1}}$, in other words: the left boundary frequency equals the center frequency of the filter to the left.

The center frequency of the final filter is pre-defined to be $f_{c_{40}} = 6400$ Hz. In order to properly define the base of the triangles, we define the following factor.

$$F_{\log} = \exp\left(\frac{\ln\left(f_{c_{40}}/100\right)}{27}\right) \tag{56}$$

We can now proceed by defining the center frequencies. Note that the indexes $i = 1, \ldots, 13$ correspond to the linearly-spaced filters.

$$f_{c_i} = \begin{cases} 133.33333 + 66.66667 \cdot i, & \text{for } \; i = 1, 2, \ldots, 13 \\ f_{c_{13}} \cdot F_{\log}^{i-13}, & \text{for } \; i \geq 14 \end{cases} \tag{57}$$

The weight function corresponding to filter i is defined by the function $H_i(k)$.

$$H_i(k) = \begin{cases} 0 & \text{for } k < f_{b_{i-1}} \\ \dfrac{2(k - f_{b_{i-1}})}{(f_{b_i} - f_{b_{i-1}})(f_{b_{i+1}} - f_{b_{i-1}})} & \text{for } f_{b_{i-1}} \leq k \leq f_{b_i} \\ \dfrac{2(f_{b_{i+1}} - k)}{(f_{b_{i+1}} - f_{b_i})(f_{b_{i+1}} - f_{b_{i-1}})} & \text{for } f_{b_i} \leq k \leq f_{b_{i+1}} \\ 0 & \text{for } k > f_{b_{i+1}} \end{cases} \tag{58}$$

The functions $H_i(k)$ are those shown in figure 5, for every filter $i = 1, 2, \ldots, 40$. We proceed by computing the weighted sum of the magnitudes with respect to each filter's weights. As stated before, the quantity f_k denotes the frequency which corresponds to the k:th coefficient of the DFT of a collection of N points.

$$A_i = \sum_{k=0}^{N-1} |S(k)| \cdot H_i(f_k) \tag{59}$$

Next, we perform the version II of the Discrete Cosine Transform (DCT-II). Let K denote the number of filters.

$$c_n = \sqrt{\frac{2}{K}} \sum_{i=0}^{K-1} \left(\log_{10} A_{i+1}\right) \cdot \cos\left(\frac{n(i+0.5)\pi}{K}\right) \; \text{ for } n = 1, 2, \ldots, R-1 \tag{60}$$

In the expression (60), the number $R \leq K$ denotes the number of unique coefficients which can be computed. According to Ganchev, the coefficients c_n for $n \geq K$ mirror the first K coefficients. For this reason, the number of unique coefficients is lower than K, and Ganchev explains that the exact number depends on the application. In our case, the `mfcc`-function returns the coefficients in (60) for $R = 13$.

7.5 Parameter estimation for complete data

Assume we have the sequences $\boldsymbol{q} = (q_1, \ldots, q_L)$ and $\boldsymbol{o} = (o_1, \ldots, o_L)$ and want to estimate the HMM parameter collection λ. Let the functions f, h_i, k_i and coefficients u, v_i, w_i be those that were defined in equation (29) of the main section. By the method of Lagrange multipliers, we need to solve

$$\nabla f(\lambda) + u\nabla g(\lambda) + \sum_{i=1}^{n} v_i \nabla h_i(\lambda) + \sum_{i=1}^{n} w_i \nabla k_i(\lambda) = \mathbf{0} \tag{61}$$

We will use the following notation for the Kronecker delta indicator function.

$$\delta(x \mid \theta) = \begin{cases} 1 & \text{if } x = \theta \\ 0 & \text{else} \end{cases} \tag{62}$$

The solution to the problem is given by the expressions in (63), which have been described in lecture slides [2].

$$\widehat{\pi}_i = \delta(q_1 \mid i) \tag{63a}$$

$$\widehat{a}_{i,j} = \frac{\sum_{\ell=2}^{L} \delta(q_{\ell-1} \mid i)\delta(q_\ell \mid j)}{\sum_{j=1}^{n} \left(\sum_{\ell=2}^{L} \delta(q_{\ell-1} \mid i)\delta(q_\ell \mid j) \right)} \tag{63b}$$

$$\widehat{b}_i(j) = \frac{\sum_{\ell=1}^{L} \delta(q_\ell \mid i)\delta(o_\ell \mid j)}{\sum_{j=1}^{m} \left(\sum_{\ell=1}^{L} \delta(q_\ell \mid i)\delta(o_\ell \mid j) \right)} \tag{63c}$$

We begin by expressing the parameters in a form which is more suitable for differentiation.

$$\pi_{q_1} = \prod_{i=1}^{n} \pi_i^{\delta(q_1|i)} \tag{64a}$$

$$a_{q_{\ell-1},q_\ell} = \prod_{i=1}^{n} \prod_{j=1}^{n} a_{i,j}^{\delta(q_{\ell-1}|i)\delta(q_\ell|j)} \tag{64b}$$

$$b_{q_\ell}(o_\ell) = \prod_{i=1}^{n} \prod_{j=1}^{m} b_i(j)^{\delta(q_\ell|i)\delta(o_\ell|j)} \tag{64c}$$

The next step is to express the log-likelihood in terms of the expressions (64).

$$\log p(\boldsymbol{o}, \boldsymbol{q} \mid \lambda) = \log \pi_{q_1} + \log \prod_{\ell=2}^{L} a_{q_{\ell-1},q_\ell} + \log \prod_{\ell=1}^{L} b_{q_\ell}(o_\ell) = \tag{65a}$$

$$\log \pi_{q_1} + \sum_{\ell=2}^{L} \log a_{q_{\ell-1},q_\ell} + \sum_{\ell=1}^{L} \log b_{q_\ell}(o_\ell) = \tag{65b}$$

$$\sum_{i=1}^{n} \delta(q_1 \mid i) \log \pi_i + \sum_{\ell=2}^{L} \left(\sum_{i=1}^{n} \sum_{j=1}^{n} \delta(q_{\ell-1} \mid i) \delta(q_\ell \mid j) \log a_{i,j} \right) + \sum_{\ell=1}^{L} \left(\sum_{i=1}^{n} \sum_{j=1}^{m} \delta(q_\ell \mid i) \delta(o_\ell \mid j) \log b_i(j) \right) = \tag{65c}$$

$$\sum_{i=1}^{n} \delta(q_1 \mid i) \log \pi_i + \sum_{i=1}^{n} \sum_{j=1}^{n} \left(\sum_{\ell=2}^{L} \delta(q_{\ell-1} \mid i) \delta(q_\ell \mid j) \right) \log a_{i,j} + \sum_{i=1}^{n} \sum_{j=1}^{m} \left(\sum_{\ell=1}^{L} \delta(q_\ell \mid i) \delta(o_\ell \mid j) \right) \log b_i(j) \tag{65d}$$

Find the maximum likelihood estimator $\widehat{\lambda}$ by solving the optimization problem (29). By the method of Lagrange multipliers, we need to find the coefficients u, v_i, w_i that solve equation (66)

$$\nabla f(\lambda) + u \nabla g(\lambda) + \sum_{i=1}^{n} v_i \nabla h_i(\lambda) + \sum_{i=1}^{n} w_i \nabla k_i(\lambda) = \mathbf{0} \tag{66}$$

The more explicit form of the equation (66) is found in equation (67)

$$\frac{\partial}{\partial \pi_i} f(\lambda) = \frac{1}{\pi_i} \delta(q_1 \mid i) \tag{67a}$$

$$\frac{\partial}{\partial a_{i,j}} f(\lambda) = \frac{1}{a_{i,j}} \left(\sum_{\ell=2}^{L} \delta(q_{\ell-1} \mid i) \delta(q_\ell \mid j) \right) \tag{67b}$$

$$\frac{\partial}{\partial b_i(j)} f(\lambda) = \frac{1}{b_i(j)} \left(\sum_{\ell=1}^{L} \delta(q_\ell \mid i) \delta(o_\ell \mid j) \right) \tag{67c}$$

We conclude that the following relations hold.

$$\frac{\partial}{\partial \pi_i} f(\lambda) = u \tag{68a}$$

$$\frac{\partial}{\partial a_{i,j}} f(\lambda) = v_i \tag{68b}$$

$$\frac{\partial}{\partial b_i(j)} f(\lambda) = w_i \tag{68c}$$

The parameters can be eliminated from the system of equations (68) in the

following manner.

$$\sum_{i=1}^{n} \delta(q_1 \mid i) = u \sum_{i=1}^{n} \pi_i \tag{69a}$$

$$\sum_{j=1}^{n} \left(\sum_{\ell=2}^{L} \delta(q_{\ell-1} \mid i)\delta(q_\ell \mid j) \right) = v_i \sum_{j=1}^{n} a_{i,j} \tag{69b}$$

$$\sum_{j=1}^{m} \left(\sum_{\ell=1}^{L} \delta(q_\ell \mid i)\delta(o_\ell \mid j) \right) = w_i \sum_{j=1}^{m} b_i(j) \tag{69c}$$

The solution (63) is then found by combining (67) and (69).

References

[1] Available in:
`https://www.math.uci.edu/~jxin/isolated_digits_ti_train_endpt.zip`, downloaded in 2019.

[2] *EM and HMM* lecture slides, available in:
`https://www.cs.cmu.edu/~epxing/Class/10701-08s/recitation/em-hmm.pdf`, Carnegie-Mellon University, downloaded in 2019.

[3] Agresti A. *Categorical data analysis*, 3rd ed., Wiley, 2013.

[4] Bachu R.G., Kopparthi S., Adapa B., Barkana B.D. *Separation of Voiced and Unvoiced using Zero crossing rate and Energy of the Speech Signal*, School of Engineering, University of Bridgeport, March 2008.

[5] Dempster A.P., Laird N.M. and Rubin D.B. *Maximum likelihood from incomplete data via the EM algorithm*, J. Roy. Stat. Soc., Vol 39, No. 1, pages 1-38, 1977.

[6] Forney G. D., *The Viterbi Algorithm*, Proceedings of the IEEE, vol. 61, No. 3, pages 268-278, March 1973.

[7] Ganchev T., *Contemporary Methods for Speech Parameterization*, Springer, 2011.

[8] Gold B., Morgan N., Ellis D., *Speech and Audio Signal Processing - Processing and Perception of Speech and Music*, 2nd ed., Wiley, 2011.

[9] Hastie T., Tibshirani R., Friedman J., *The Elements of Statistical Learning - Data Mining, Inference, and Prediction*, 2nd ed. corrected 12th printing, 2017.

[10] Marsland S. *Machine Learning - An Algorithmic Perspective*, CRC Press, 2009.

[11] Rabiner L. R., *A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition*, Proceedings of the IEEE, Vol. 77, No. 2, pages 257-286, February 1989.

[12] Rabiner L. R., Juang B. H., *Fundamentals of Speech Recognition*, Prentice Hall, 1993.

[13] Wilpon J. G., Rabiner L. R., Martin T., *An Improved Word-Detection Algorithm for Telephone-Quality Speech Incorporating Both Syntactic and Semantic Constraints*, AT & T Bell Laboratories Technical Journal Vol. 63, No. 3, pages 479-498, March 1984.

[14] Xin J., Qi Y., *Mathematical Modeling and Signal Processing in Speech and Hearing Sciences*, MS & A Volume 10, Springer, corrected publication April 2018.