HÁSKÓLI ÍSLANDS

HUGVÍSINDASVIÐ

Isolated Word Recognition for Icelandic

A Machine Learning Investigation into Word Modelling

Ritgerð til MA-prófs

Alex Murphy

Maí 2015

Háskóli Íslands

Hugvísindasvið

Máltækni

Isolated Word Recognition for Icelandic

A Machine Learning Investigation into Word Modelling

Ritgerð til MA-prófs í máltækni

Alex Murphy

Kt.: 240587-3949

Maí 2015

Isolated Word Recognition for Icelandic - A Machine Learning Investigation into Word Modelling

Ritgerð þessi er 30 eininga lokaverkefni til MA-prófs við Íslensku- og menningardeild, Hugvísindasviði, Háskóla Íslands.

Reykjavík, 2015.

Acknowledgments

I would like to thank all of teachers and faculty members at the University of Iceland and Reykjavík University who have taught me over the past two years. I arrived in Iceland as a linguist with no formal experience in mathematics, programming, statistics or computer science. The fact that I could take on such a project at this level so soon after being introduced to many of the fundamental concepts of each field is a testament to the quality of teaching at these universities. I would also especially like to thank Eiríkur Rögnvaldsson for his unwavering determination to battle for better language resources for Icelandic in an increasingly digital world, especially in the context of speech recognition, which inspired me to undertake this project.

I would also like to thank my parents, who have supported me and encouraged me in many different ways and I would not have been able to pursue this degree without their help. A final thank you goes to my fellow Máltækni students who have helped me through the many late-night study sessions over the last two years.

Ágrip

Þróun tækni gegnir sífellt mikilvægara hlutverki í lífi okkar allra og mikil aukning er á því að nota þessa tækni til þess að hafa samskipti við aðra. Miklar breytingar hafa orðið á því hvernig við notum tækni, en áður fyrr var handvirkur innsláttur algengari hjá notendum, en nú er mikið meira um að tækni sé stjórnað með töluðu máli. Þetta getur valdið vandræðum hjá málhöfum smærri tungumála, en þeir gætu átt í erfiðleikum með að þróa sams konar tækni í takt við þjóðir á heimsmælikvarða. Mikilvægt er að eiga kost á að nota sitt eigið tungumál við notkun nýrra tækja í stað þess að neyðast til að nota alþjóðlegt samskiptamál. Þessi ritgerð hefur það að markmiði að kanna hvort hægt sé að búa til einstaklingsbundið stakorðagreiningarkerfi sem er þjálfað á örfáum þjálfunardæmum. Athyglisvert er að benda á að mörg þeirra dæma eru beygingarmyndir tiltekins flettiorðs og eru þar af leiðandi af mjög svipaðri gerð. Gert er ráð fyrir því að einhvers konar talstjórnuð reiknivél muni nýta svona kerfi. Mismunandi aðferðir í vélanámi eru kannaðar í því skyni til að ákvarða hversu mikla nákvæmni þessar aðferðir bjóða upp á þegar orðaforðinn samanstendur af orðum sem eru mjög svipuð að gerð. Sýnt verður fram á að einfaldar framkvæmdir tiltölulega flókinna vélanámsreiknirita, eða hulinna Markov-líkana og gervitauganeta, geti spáð fyrir réttri flokkun nýrra orða upp að meira en 92% nákvæmni. Þessar niðurstöður sýna að frekari rannsókn hefur að geyma mjög áhugaverða framtíðarmöguleika og gæti hugsanlega verið undirstaðan að mörgum raddstýrðum notendaviðmótum sem eru nú þegar til staðar í mörgum tækjum og að notkun þess mun eflaust fjölga mjög mikið í framtíðinni.

Abstract

As technological advancements play an ever-increasingly important role in the daily lives of people and how they interact with the world via technology, the means by which interactions are taking place are gradually moving from a more manual system to a more vocal one. This poses a specific problem for languages with relatively few speakers, like Icelandic, who need to keep up with advancements elsewhere in the world so that speakers have the choice to be able to use their native language in new technologies and are not forced to adopt a *lingua franca* in place of their native language. This paper aims to explore how a speaker-dependent isolated word recognition system might be implemented with only limited training data, with specific consideration to the problem of phonetically similar vocabulary items that arise from Icelandic's case system. A theoretical goal of a speech-driven calculator is adopted and various machine learning techniques are investigated to determine what kind of accuracy can be expected when modelling a non-trivial amount of phonetically-similar words. It will be shown that simple implementations of relatively advanced machine learning algorithms, namely Hidden Markov Models and Artificial Neural Networks, can easily correctly predict previously-unseen word recordings at a level higher than 92% accuracy. These results show that further research has a lot of potential to expand the scope of such an isolated word recognition system and could provide the basis for a technology that could be used in a diverse range of automated voice interfaces which are available in many technologies in the modern world already and whose predominance in society will only become more prevalent in the future.

Table of Contents

Table of Figures

Introduction

In this project I will investigate various machine learning techniques to model recognition of a non-trivial amount of isolated words in Icelandic, based on a vocabulary set that would be required in order to implement a basic speech-driven calculator. I will consider all the major aspects that are involved in the task of isolated word recognition (IWR) from the initial processing of the input speech signal as it first enters the system right through to the feature extraction, modelling and classification stages. The nature of the results reported in earlier sections this paper are exploratory and involve trying various model parameters in order to empirically verify what works best with a simple classifier evaluated on the test set. Techniques to empirically validate the usefulness of the feature set will then be explored. The establishment of a reliable feature set on a simple classifier in order to determine a baseline for the results will then be applied to more advanced machine learning techniques in the later sections. Issues such as dimensionality reduction, multi-class discriminant analysis, vector quantization, clustering algorithms, hidden Markov models and artificial neural networks will be among the topics that will be considered in this project. The end goal of this paper is to demonstrate the level of accuracy an isolated word recognition system for Icelandic can achieve without extensive training data based on various classification and modelling techniques inherent in the chosen machine learning algorithms.

Layout

This paper is organised into 5 sections. Section 1 explains the context of the problem and provides a motivation for why such an investigation is needed, then outlines the rationale and exactly what will be investigated over the course of this paper. An account of the vocabulary is then given alongside potential problems that can be expected from such a data set. The section will close with a more detailed consideration of what needs to be accomplished in order to achieve accurate isolated word recognition.

Section 2 looks at background information required in order to conceptualise the structure of the problem and how solutions can be designed that build upon such an understanding of the problem specification. The issue of the nature of speech sounds, models of speech production and feature extraction will then be examined, culminating in a detailed analysis of the processing strategies required in order to bring the raw digital signal into a format from which it is possible to then extract acoustic features.

Section 3 moves beyond the background information required in order to conceptualise the problem, and having detailed the feature extraction process, will go on to look at a basic implementation of isolated word recognition using a simple and fairly crude technique in the first half of this section. The second section addresses the issue of dimensionality reduction, removing noisy features from the feature set that do not provide a lot of descriptive quality in the data set treated as a whole. This process will then be further refined in order to take into account a multi-class classification problem and the results of the algorithm applied to the simple modelling technique will then be described.

Section 4 then deals with two more advanced machine learning algorithms that can be used to tackle the problem at hand, continually comparing performance and implementational issues with the baseline results given in Section 3. This section is split up into two main sub-sections, each of which deals with a specific machine learning implementation, includes a results section and a brief discussion, addressing the pros and cons that arise in such implementations.

Section 5 provides a conclusion of the previous sections, addressing interesting observations that have arisen earlier in the project and examines which implementational approach is the most suitable to the given task at hand based on the experimental results. This section ends by addressing how such a system might be implemented in the real world and lists potential improvements and avenues for future work that could be undertaken to expand on the groundwork laid out in this paper.

1 Preliminaries

In this section I will explain the context of the problem and provide the rationale that motivates this project. I will then give an account of the vocabulary items that will comprise the classifiers' target outputs, after which I offer an explanation of the data acquisition process and implementational details such as equipment used and how the data is stored and accessed. I will then end this section with a discussion on the potential problems inherent in the data set that could have an impact on a classifier's ability to model phonetically-similar vocabulary items.

1.1 Project Motivation

Icelandic currently has a Large Vocabulary Continuous Speech Recognition (LVCSR) system accessible via and courtesy of Google. This LVCSR system is also available to all users of the Android mobile operating system (Guðnason et al., 2012). Developing such a system is a timely and costly process that relies on a lot of training data (often both written and spoken), learning algorithms and an expert implementation to achieve a high enough accuracy so that users of the language would take it seriously and it can be considered to have a viable user base. The Almannarómur project is currently working on developing an open source LVCSR system that utilises the training data that was recorded for Google's speech recognition system (ibid.)

The speech recogniser itself is completely owned by Google and is therefore out of the control of any Icelandic institution, which is problematic because it is foreseeable that a decision could be made to discontinue the speech recognition system at a moment's notice, given that the level of importance afforded to the continued use and success of the recogniser is not equally shared between Google and research and educational institutions in Iceland. Other problems inherent in Google's ownership of the speech recogniser are that usage costs could be introduced in the future and it is problematic to incorporate the recogniser into other technology without running it through a Google server (Rögnvaldsson, 2014) therefore requiring any device to be currently connected to the internet.

Many industries employ partially or fully automated voice user interfaces (VUIs) in their dealings with the public, a trend that is predicted to greatly expand in the future (Rehm et al., 2012). Such systems allow for human-computer interaction via speech recognition and the range of applicability can be seen in domains such as banking, supply chain and public transportation. Such technology is also being employed in car navigation systems and the use of spoken language as an alternative to touchscreen interfaces in smartphones are just a few examples of the other ways in which restricted speech recognition can and is being applied around the world today (ibid.)

Contrary to LVCSR systems, IWR systems only aim to model a subset of a language's vocabulary and the task of training such a system is a much simpler problem. For a VUI system to be implemented, only a limited set of words is required for simple applications. My goal in this project is to investigate various machine learning techniques that aim to model a non-trivial vocabulary set with very limited training data for each item in the vocabulary to be recognised. If it can be shown that reasonable success can be achieved in a small system, factoring in scalability conditions, I hope to be able to show that isolated word recognition for Icelandic has the potential to be used in appropriate situations where it would otherwise not be feasible to employ a LVCSR system.

Isolated word recognition in Icelandic poses a particular problem in that it is important to distinguish between case forms that are very similar and only differ in some instances by a single phone. This problem is usually aided by the use of language models in larger automatic speech recognition (ASR) systems which are not usually applicable in IWR contexts. Working with a vocabulary that focuses highly on number recognition means that this problem can be investigated from the perspective of acoustic feature analysis based solely on the information in the recorded signal.

If it is possible to extend the techniques applied in this project to a speaker-independent context that is capable of consistent reliability in various recording environments, then I hope that there is potential for a viable extension of this project that could be adopted in wider, real-world contexts where Icelandic would be the expected medium of communication.

1.2 Vocabulary Set

There are a total of 61 different vocabulary items considered in this project. The majority of words are numbers (53) in various genders and case forms. A minority (8) are not related to numbers but consist of words that are likely to be used in basic calculator commands, which is a number-centred real-world application that such an isolated word recognition task could be used for and hence prepositions, maths-specific words and other functional words were included as well. The numbers can be divided into two groups - cardinals and ordinals. The ordinal numbers are all in weak oblique form ranging from *fyrsta* (first) to *sjöunda* (seventh) with the exception of *öðru* (second) which only has a strong declension in comparison to the others which only exhibit a weak form. The cardinal numbers range from *núll* (zero) to *tuttugu* (twenty) in all word forms for different gender and case and *og* (and) was included in the non-number group to provide the combinatorial potential to express up to 29. With the addition of eight more multiples of 10 it would be possible to express all numbers between 0-100. The non-number words that are included are *deilt* (divided), *með* (with 'by'), *í* (into), *og* (and), *rót* (root), *veldi* (power), *plús* (plus) and *mínus* (minus). Plural-only numbers (i.e. *tvennir*) were not included in the vocabulary. The longer *-ur* endings were chosen for the dative plural of numbers greater than *einn* (one) where there is a common shorter ending, i.e. *tveim/tveimur* and *þrem/þremur*. This was due to the fact that the more acoustic content in the word means a greater chance for the classifier to distinguish it from other potentially similar words.

For a full implementation of a speech-driven calculator there would need to be many more words and full commands which would require more advanced grammatical structure processing. With the absence of any natural language processing and with a vocabulary size of 61 words, I felt this was a sufficiently non-trivial number with which to investigate the recognition rate with various machine learning techniques. The full vocabulary list is given below in Figure (1.1).

einn	tveir	þrjá	fimm	fjórtán	þriðja	plús
ein	tvær	þremur	sex	fimmtán	fjórða	mínus
eitt	tvö	þriggja	sjö	sextán	fimmta	rót
einan	tvo	fjórir	átta	sautján	sjötta	sinnum
eina	tveimur	fjórar	níu	átján	sjöunda	veldi
einum	tveggja	fjögur	tíu	nítján	og	í
einni	þrír	fjóra	ellefu	tuttugu	núll	
einu	þrjár	fjórum	tólf	fyrsta	deilt	
eins	þrjú	fjögurra	þrettán	öðru	með	

Figure 1.1. A list of all the vocabulary items in the data set

1.3 The Dataset

The data set used in this project consisted of 10 recordings for each word in the vocabulary. The total data set therefore contains 610 files. For each word in the vocabulary, 7 of these recordings were used for training and 3 for testing. One central folder contained all 610 files for evaluation and each word has its own subfolder in the same directory. Each of these subfolders contains 7 recordings of the same word (used for training) and therefore when all of the words are examined by the classifier in the folder that contains all 610 recordings, 30% of the whole data set (183 files) will not have been seen before. This assesses the classifier's ability to generalise to new data samples that it was not trained on.

The words in the vocabulary set can be viewed as consisting of many possible varying main segments, and depending on how many segments the signal is split up into, the classifier's ability to model the word is affected. The effects of varying the amount of segments of each word is tested throughout this project. This value will be referred to as the *word split value* throughout this paper. If there is a split value of 2, it means the word is treated as two separate parts and the feature vectors are averaged over all separate segments. With a word split value of 8, this means there is a more fine-grained average of the signal over 8 consecutive segments which will contain a lot more information about how the signal changes over time, but therefore will require more training data to work well.

The program code is written in Python. The environment in which the program was developed is Spyder 2.3.0. Each word is computationally represented as a class object and the class constructor takes the lemma of the word that that instance will represent. As each word instance is created the lemma information is used to search the training folder for a subfolder of the same name and then combines all the acoustic data as a class variable. This class variable is a list of matrices that contains all the extracted features for every frame of the recording.

In the reported results throughout this paper there will be an initial percentage which represents the classifier's accuracy to model the training data. Following this percentage there will be an accuracy result based only on the test set (the unseen data). This means that it is possible to see in discussions of the results how the classifier scored based solely on recordings it has not been trained on and therefore reflects its ability to generalise to new data samples, while the initial result is a reflection of how well the classifier is able to model the data set as a whole, i.e. to capture the data the classifier was trained on.

The recordings were made in Audacity using both an internal microphone and an external one over the course of the recordings. The sound files are all in Waveform Audio File Format (.wav) and were recorded at a sample rate of either 16 kHz or 44 kHz (Audacity's default). By allowing for variation in data acquisition in terms of recording equipment used and sampling rate, it is possible to achieve results that reflect different setup conditions. If good results can be achieved in the recognition process then this demonstrates that the results are not dependent upon a specific recording setup that has to be precise, i.e. a specific type of microphone or sampling rate that should be used and has to be consistent for all files. This is an advantage because it allows for an extraneous variable (i.e. reliability on a specific setup) to be taken out of the equation if a good accuracy can be achieved. By demonstrating that a recognition algorithm can handle a combination of varying setup parameters, this means that the classifier is more likely to be relatively robust to classifier-external conditions, which is desirable in any ASR context.

1.4 Potential Difficulties

The major problem faced with such a largely similar vocabulary set is that there are a lot of closely related words that need to be classified but that only differ minimally in acoustic information. This is due to related case forms in Icelandic and is a feature of the language that comes up everywhere. If any system faces the task of speech recognition for Icelandic then similar word forms will be an important issue that needs to be solved for any reasonable system to achieve any chance at being scalable. Larger speech recognition systems use a combination of a language model and acoustic model in the task of decoding feature vectors to strings of words and it is precisely the kind of training data that a language model is trained on that helps the system in such cases of homophonic ambiguity and similar word pronunciations.

Isolated word recognition does not typically require a language model because the nature of the task it is aimed at is not one of continuous speech recognition where such n-gram models play a useful role. For example, if acoustic information could not determine whether a sequence of feature vectors represented the word *hann (he; nominative)* or *hans (his; genitive)*, then the fact that the previous word was *til* (which takes the genitive case) would be the determining factor in constraining the probability calculation in favour of *hans* over *hann*. Any ASR system that does not use a language model is therefore faced with a problem of relying solely on the acoustic information to discern the correct word that was spoken.

This would not be much of a problem if all the words in the system's vocabulary were phonetically very different from one another. An investigation into words with differing phonetic quality would, however, not be realistic because a real-world application would never be that easy; the results would always be overly optimistic. Conversely, by picking a data set that is relatively difficult to model then aiming for the best accuracy would give a relative lower bound to the kind of accuracy that could be expected in real-world applications. With the addition of more diverse words on top of a system that could handle words that were phonetically similar, we could only expect that such similarity would be modelled to the same degree of accuracy as found in that system. It is not always the case that precise lemma or dictionary form identification is

required and if the end goal was just to identify a lemma then the performance of a system almost certainly has to be greater than a system that tries to discern each possible derived word form.

The calculator example that this project was framed around is precisely one type of environment where precise lemma identification is not necessary. It does not matter if a classification technique confuses *fjórir (four; nominative masculine)* or *fjórar (four; nominative feminine)* as long as the symbol passed to the calculator is '4'. That being said, the ability for a basic IWR system to accurately model such small differences in a system with a non-trivial vocabulary size is an interesting research question in itself and is worthy of investigation. For this reason, the attention to correct identification of phonetically similar word forms based solely on the acoustic information is an aspect of isolated word modelling that will be investigated throughout this project.

To illustrate the problem visually, a series of spectrograms are given in Figure (1.2). The first vocabulary item is for the word *nítján* (nineteen) and the following words are *fjórar* (four) and *fjórða* (fourth) respectively.

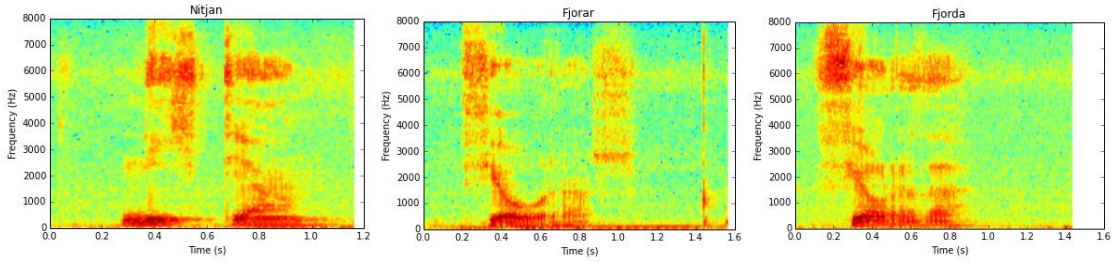

Figure 1.2. Spectrograms for the words (from left-to-right) *nítján*, *fjórar* and *fjórða*

The distinction between *nítján* on the one hand and *fjórar*/*fjórða* on the other is clear in the spectrograms given above. A clear formant shift is discernible for the final two spectrograms, corresponding to the diphthong [ou]. The higher energy due to the voiceless labiodental fricative is also visible at the start of both of these words. The differences between *fjórar* and *fjórða*, however, are much smaller and it is the goal of the classifier to find a way to model such fine-grained distinctions based solely on acoustic information. In this case the voiced dental fricative [ð] would be an important distinguishing factor between the latter two spectrograms and this can be seen in the lower formant bands at the end of the third spectrogram.

2 Feature Modelling

The structure of this section is twofold. I will first outline the nature of speech sounds and how they are produced by the speech organs in the human body. I will then detail the major steps involved in preparing a digital audio signal for a speech recognition task. The nature of how speech sounds are produced leads to some important assumptions in how acoustic features can be derived from the audio signal, therefore reviewing this process will illustrate the rationale behind the assumptions made later on during the feature extraction stage.

2.1 Nature of speech sounds

In order to produce a speech sound, air is first expelled from the lungs in a pulmonic egressive airstream initiated by the diaphragm, which then results in an increase of subglottic pressure moving towards the larynx, itself a composition of cartilage, muscles and ligaments that modulate the airflow via a membrane opening called the glottis. When the glottis lies open and there is an absence of subglottic pressure, air can pass through freely and no speech sounds are made, but when the vocal fold membrane becomes constricted, the subglottic pressure builds up underneath the vocal folds until enough pressure has amassed to temporarily push through. When this occurs, the subglottic pressure is released and the vocal folds contract back to their initial position and then pressure starts to build up underneath once again. This oscillatory pattern is an instance of the Bernoulli Effect (Rögnvaldsson, 2013) and the rate of vibration is reflected in the fundamental frequency of the voiced sound (Rogers, 2010). When there is a lot of subglottic pressure but the glottis lies further apart than in voiced speech, this creates a turbulent airflow that then flows through the vocal tract and is an instance of an unvoiced speech sound.

Therefore the presence of vocal-fold vibration implies that a voiced speech sound is being produced and a turbulent stream of air caused by a slightly constricted glottis indicates that an unvoiced speech sound is being produced. As the air travels through the vocal or nasal tract, the positioning of the speech organs such as the tongue, soft palate and lips all modulate this airstream as it exits the mouth and/or

nose. For voiced sounds, various higher harmonics dependent on the shape of the vocal tract are strengthened and others attenuated, resulting in the specific characteristics of a single speech sound, also known as the sound's pitch structure. Voiceless sounds typically do not exhibit any pitch structure (McLoughlin, 2009).

With an understanding of the difference between the origin of airflow coming the lungs and how that airflow is modulated in the resonance chamber of the nasal or vocal tract, the essential concepts of the Source-Filter Model can now be explored, which takes advantage of viewing the speech signal as such a dichotomous procedure.

2.1.1 Source-Filter Model of Speech Production

The process of speech production can be thought of in two distinct stages. As the air leaves the larynx it is then moulded by the vocal (and often nasal) cavities which depend on the shape of the speech organs at the moment in time the air passes through the open cavity. The effect that the vocal/nasal tract has on the air passing through it is that the precise synchronic placement of the speech organs, such as tongue position and height, creates a set of resonance frequencies that will be amplified and a set that will be attenuated, i.e. similar frequency components present in a signal passed through the same resonance cavity will be affected in a similar way.

Many modelling techniques used in the field of speech recognition acknowledge this division between the pulmonic source of air pressure and the vocal/nasal tract filter. By envisaging the vocal/nasal tract as a linear filter to the pulmonic air source the production of speech sounds can be assumed to be functionally independent from the source airflow and therefore the speech sounds that are produced are shaped exclusively by the filter (vocal or nasal cavity). Modelling techniques that assume this distinction are representations of the Source-Filter Model of Speech Production.

Such an assumption of a linear filter is valid at frequencies below 4-5 kHz (Mariani, 2013), which proves to be a useful assumption since the range of typical speech sounds has been reported to be between 100 Hz and 4 kHz (McLoughlin, 2009) with a range of peak sensitivity around 3-4 kHz (Rossing, 2014). By assuming that speech sounds are independent of the source airstream and are instead a function of

the tract filter alone, a difficult problem is avoided in that different speakers with various tract lengths who exhibit different fundamental frequencies as a result, do not pose a problem to the recognition problem if sound identification becomes a function of the resonance effects of the vocal or nasal cavity.

The fundamental frequency, pulse rate and other information conveyed in the glottal source waveform contain a lot of speaker-specific information and would be essential in applications relating to speaker identification and recognition, but overall not in the task of identifying a specific speech sound (Mariani, 2013). Sounds can be produced in many different surroundings and can be spoken in a variety of different ways, for example with varying pitch contours or at different amplitudes that could be due to emotion, noisy surroundings or simply proximity to the recording device.

The underlying assumption in such an approach is that it is the vocal tract alone that moulds the source waveform and an ability to disregard extraneous information in the glottal source makes computation of characteristic features of specific sounds much easier. The concept of a *cepstrum*, which allows for the extraction of such features, will be discussed in more detail in Section 2.2.6.

2.2 Feature extraction

There are numerous pre-processing steps that are important to address relating to the capture and storage of audio data, such as enhancing the raw signal in anticipation of the feature extraction process further down the processing pipeline, among other steps. The main pre-processing steps are considered independently in the sub-sections that follow, after which a method for extracting reliable and stable speech features from a digital signal will be outlined.

2.2.1 Digital recording

Numerous aspects of digital recording and encoding need to be taken into consideration when acquiring data for a speech recognition task. Two of the major considerations are the storage format of the data and the sampling rate with which the recording was made. There are many audio storage formats available that are very popular today, such as the .mp3 extension format, but such storage formats are heavily compressed which means a lot of the relevant information has already been

removed and useful audio features are consequently destroyed in the process of compression (McLoughlin, 2009). For this reason, storage formats that do not compress the original representation of the signal are much more appropriate for speech recognition tasks. The storage format chosen for the dataset in this project was therefore Waveform Audio (.wav) files, a format which does not use any compression and therefore the entire recorded signal is available to the feature extraction process.

In order to capture a specific frequency component in a digital signal it is essential to have a sampling rate twice that of the highest frequency contained in the signal. This highest frequency present in a sampled signal is called the Nyquist frequency (McLoughlin, 2009) and any higher frequencies in the original analog signal cannot be captured and are lost due to a signal processing phenomenon called aliasing. The quantization and encoding step needs to take an amplitude measurement at the peak of each compression and rarefaction cycle in order to be able to detect it. An extreme version of the aliasing problem is if a sine wave of 5 Hz was sampled at 5 Hz, then the quantization/encoding step would always take the amplitude measurement when the sine wave is at a stable frequency and the digital signal would look stationary as a result and therefore the sine wave would be completely missed.

The range of frequencies that is typically found in human speech is between 100 Hz and 4 kHz (ibid.) and given that 8 kHz is a loose higher bound on this in more extreme instances, the sampling rate in the data set recordings was never lower than 16 kHz (the minimum sampling rate needed to guarantee that a 8 kHz frequency will be included in the signal). Telephone recordings usually use a sampling rate of 8 kHz (Mariani, 2013) and this isn't problematic for humans due to the psychoacoustic process of audio transduction which allows inference based on equally-spaced multiples of the fundamental frequency. Computers, however, do not possess such capability and therefore speech recognition on telephone recorded speech is likely to show differences than unaltered recordings from a microphone.

2.2.2 Pre-emphasis

An important consideration to bear in mind in any speech recognition application is the problem of spectral tilt. Vibration of the glottis in voiced speech sounds results in a

glottal pulse that reverberates through the vocal tract and expresses acoustic information relating to pitch. In such sounds, the energy levels at the lower frequencies are much greater than the energy present in the higher ones, unlike in many unvoiced speech sounds which are not associated with a particular fundamental frequency (McLoughlin, 2009). The effect this causes is a tilt in the proportion of energy present in a spectral waveform that favours the lower frequencies and can be problematic given that information on the energy that is present in higher formant ranges can often reveal useful information about the specific sound that is being produced.

A linear high-pass filter is applied to the digital signal that has the effect of attenuating the high-energy lower frequencies according to Equation (2.1).

$$y[n] = x[n] - alpha * x[n-1] \qquad 0.9 \leq alpha \leq 1.0 \tag{2.1}$$

In the feature extraction process used in this project, alpha is set to 0.97. A higher value of alpha indicates a looser relationship of energy values than a high value due to the fact that alpha scales a version of the previous signal value and subtracts it from the current index value at $x[n]$. The higher the scalar value is set to be, the larger the subtraction. Therefore the ongoing signal is more compacted together than would be done with a lower value of alpha. Figure (2.1) shows a frequency spectrum of the vowel [a] before (a) and after (b) pre-emphasis, from Jurafsky & Martin (2009).

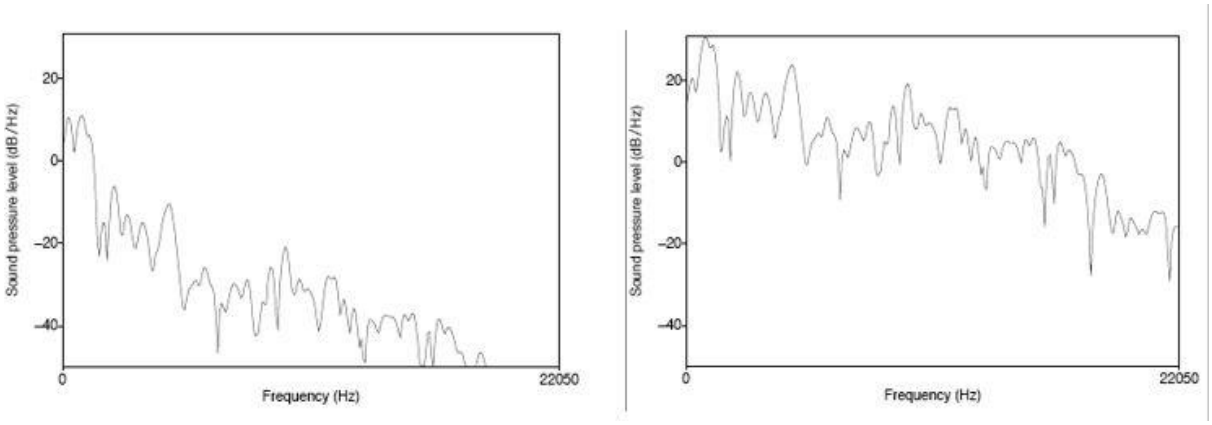

Figure 2.1 (a) Spectrogram of the vowel [a] before pre-emphasis; (b) after pre-emphasis.

2.2.3 Windowing

Windowing is an important consideration in the processing of audio data used for training acoustic models. The concept of windowing is that of segmenting a signal so that measurements can be taken at an instantaneous moment in time. The digital nature of modern signals allows for a snapshot of a signal to be taken between two short intervals of time and the result is stored for future processing later in the pipeline. A typical frame size in ASR tasks is between 10-20 milliseconds (Jurafsky & Martin, 2009) and this frame is then taken to represent the stationary characteristics of a signal at that moment in time. This assumption is not always accurate because features that occur within a single frame are cancelled out if the frame size is too large (ibid.) Frame sizes need to be frequent enough in order to capture a sufficient degree of signal fidelity, otherwise potentially significant information is lost (McLoughlin, 2009). Temporally smaller frame sizes come at a cost, however, in that it is desirable to have more information per frame in order for the features to be represented more accurately, and smaller frames naturally include less useful information.

A potential issue with splitting a signal up into equally proportioned frames is that features that occur at the extremities, for example at the end of one frame and the beginning of the following one, are lost if they are not captured within the same frame context. For this reason it is normal to consider a window length and step size where the window length is wider than the length of the frame. The signal representation is then based on a wider analysis window but is shifted along in accordance with the size of the step size. This ensures that windowing does not miss a significant amount of useful acoustic information.

When further processing such as the Discrete Fourier Transform (DFT) is applied to the window, the sharp discontinuities at the edges can cause problems in later stages of spectral analysis (Jurafsky & Martin, 2009). Not applying any adjustment to the signal window is known as taking a rectangular window of the signal (Waibel, 1988). More suitable techniques that reduce the extremities close to the edge of the signal window are therefore applied at the windowing stage of signal pre-processing. The most common choice in ASR research is the Hamming window (ibid.) The Hamming window applies Equation (2.2) to each window.

$$w[n] = \{\, 0.54 - 0.46\cos\frac{2\pi n}{L} \quad 0 \leq n \leq L-1 \,\} \tag{2.2}$$

The effect that this has on a windowed signal is that of reducing the extremities of the signal so as to avoid discontinuities which are problematic in later signal processing analyses. An example of the effect the Hamming window equation has on a windowed signal can be seen in Figure (2.2).

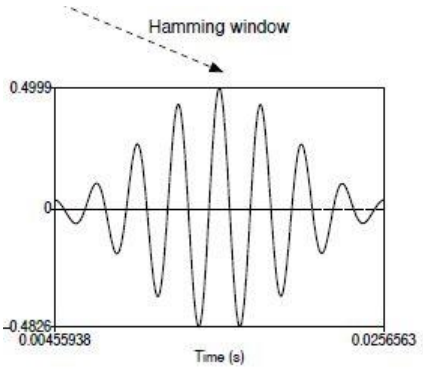

Figure 2.2. An illustration of the Hamming window. Taken from Jurafsky & Martin (2009)

In this project the window length is set to be 25ms with a step size of 10ms. This allows slightly more than double the step size to be included in the analysis window so that potentially significant features are represented in the same window, unaffected by the Hamming equation of bringing the extremities down to zero for purposes of signal continuity at the window edges. A typical undesirable consequence of the process of windowing is the amount by which the signal is extended, which results in more information to process and can become problematic in larger applications (McLoughlin, 2009). For IWR purposes, such extended signal information is minimal and it therefore allows for a comfortably wide window length in comparison to the frame step size, which then results in less chance of any feature loss at this stage of processing.

2.2.4 Discrete Fourier Transform

The information in the digital signal at this point is the quantized amplitude taken at every sample instance which can be plotted as a waveform with the x-axis representing time and the y-axis as the recorded amplitude. The waveform itself should provide all the necessary information in order to recognise a word but a lot of variability that is speaker-dependent accompanies the recorded speech waveform and

therefore needs to be separated from the underlying distinctive features that characterise each sound present in the recording by extracting a parametric representation of the signal (Shrawankar, 2013).

The Discrete Fourier Transform (DFT) is an algorithm that extracts the frequency information from the time-domain (waveform) and provides a more stable representation of the phonetic information in the signal. The DFT extracts the energy content of different equally-spaced frequency bands, the number of which is given by the reciprocal of the length of the analysis window (McLoughlin, 2009). This means that there is a greater frequency resolution as a result of the transform when more samples are included in the analysis window supplied to the algorithm.

This poses a potential problem because the DFT makes an assumption about each analysis window in that each window represents an infinitely long repetition of the same signal (Smith, 1997) and by increasing the number of samples in a given window, the ability to discern quick changes in the time-domain decreases markedly given the assumption that each window is not part of a temporally-changing sequence. The amount of positive frequency bins made available by the DFT is $\frac{N}{2} + 1$ where *N* represents the number of samples in each analysis window (McLoughlin, 2009).

This trade-off between time and frequency resolution is an unsolved problem in speech signal analysis but an analysis window of 20-30ms is considered optimal for robust feature extraction from a speech signal (Kim, 2010), where slightly longer analysis windows of up to 50ms are used to compensate for high levels of noise in a signal (ibid.) The length of the analysis window used in this project was therefore chosen to be 25ms. The frequency information returned from the DFT is the magnitude of each frequency bin and can be plotted as such in order to show a visual representation of the frequency content of a signal. This representation is one of a stationary signal where the x-axis shows the frequency content and not how the signal changes over time, as would be indicated in a waveform plot. Reorganising the layout of the plot and adding in another dimension, it is possible to plot the frequency against time and allow for a shaded representation of the magnitude, where darker colours indicate a higher magnitude of energy. When each window in the signal is

concatenated together it is possible to see how the frequency content of a signal changes over time. An example of this was given in Section 1.4 for three recordings of different words from the vocabulary.

The DFT is implemented via the Fast Fourier Transform (FFT) in modern speech signal processing applications because the time complexity of the DFT is an inefficient $O(N^2)$, while a substantially less complex implementation was discovered by Cooley and Tukey in 1965, that has a much more efficient time complexity of $O(N \log N)$ (Pollak, 2004). This discovery had profound implications for modern technology and in particular signal processing techniques. A requirement of the FFT is that the number of frequency bins be a power of 2 in order to work efficiently (Smith, 1997). Preliminary tests of the classifier indicated that an FFT size of 1028 performed slightly better than one of size 512 and therefore 1028 was implemented in this project. This shows that even with small recordings where time resolution is of arguably greater importance, good frequency resolution is also of fundamental importance to single-word modelling in an isolated word recognition task.

2.2.5 Mel filter bank

After the DFT has been applied to the sampled and windowed signal, the frequency information is stored in equidistant frequency bins, the size of which is a function of the samples included in the analysis window (McLoughlin, 2009). This means that lower frequency bands have the same ratios of energy as higher frequency bands and are treated proportionally. It turns out that the frequency information at the higher frequency ranges is a lot less informative than the lower range in a similar way to the human sound perceptual system (Jurafsky & Martin, 2009). By projecting the result of the DFT onto a scale that mirrors the basics of human sound perception, a more robust feature set can be derived from the signal (ibid.) and this is the rationale behind the mel scale.

Human sound perception is linear under 1 kHz but logarithmic at all frequencies above (McLoughlin, 2009). This means that the human ear hears equally-distant frequencies as being perceptually equally-distant as long as they are below 1 kHz but at a higher frequency range this perception becomes warped between pitch and

frequency and a much larger increase of frequency between two sounds is required to be able to discern two equidistant pitches. The mel-scale models this phenomenon and conversion formulas can be easily applied to a linear Hz signal to convert it to the mel scale and back again. A mel is a "unit of pitch defined so that pairs of sounds which are perceptually equidistant in pitch are separated by an equal number of mels" (Jurafsky & Martin, 2009:17).

With a DFT analysis on a windowed section of a speech signal, a triangular mel-filter bank can be applied which results in a conversion of the Hz frequency representation onto the mel-scale. A number of filters is specified upfront and these then split the frequency range into broader triangular filters logarithmically above 1 kHz that gradually increase in size, whereas filters under 1 kHz are all equally-spaced in order to reflect the linear-logarithmic threshold as evidenced in human sound perception (ibid.) An illustration of the triangular filter bank is given in Figure (2.3).

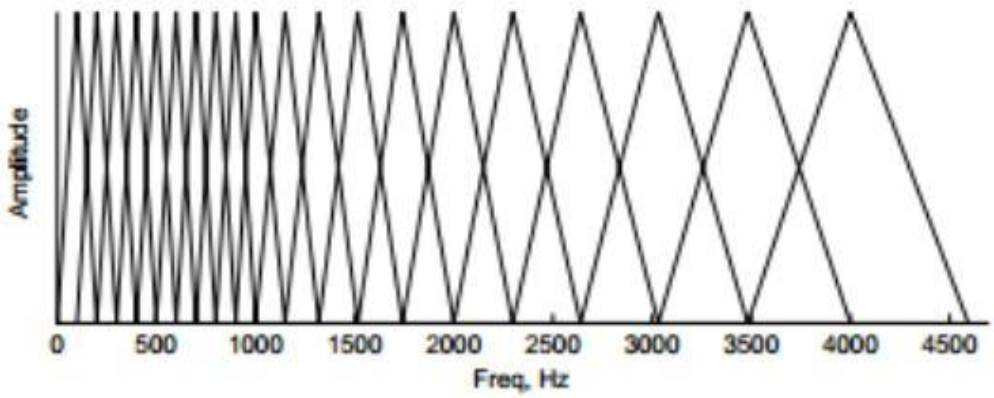

Figure 2.3. A representation of the triangular filter 4 as given in Davis and Mermelstein (1980).

A reasonable question to ask is why mirroring a facet of human sound perception should be an effective strategy for effectively modelling speech sounds in computers. A lot of research into feature extraction for speech recognition tasks has been conducted over the last 30 years and MFCC-approaches have gradually replaced previous methods such as the once dominant Linear-Predictive Coding (LPC) method (Skowronski & Harris, 2003). Methods such as LPC were considered adequate for vowel-recognition but not for consonant-recognition (Davis & Mermelstein, 1980); suppression of the potential variation of noise in the higher frequency bands is why the mel-scale approach outperforms linear methods such as LPC (ibid.) The higher

frequency bands contain less phone-identifying characteristics than the lower frequency bands, though some higher frequencies are also important, especially in detecting voiceless sounds that concentrate their frequency in the higher range.

Once the frequency content of the DFT is projected onto the mel-scale, the logarithm of each energy measurement is taken in each frequency bin. This is done in order to desensitise the energy coefficients from potential noise in the signal. Such noise could be derived from varying distances between the speaker and the recording device or from effects in the recording environment. Taking the logarithm of the values still retains the relative pattern of frequency alterations across the signals but wider ranges that are similar are mapped to a single value which helps to compensate for such potential variability, providing a more robust representation.

A high-pass filter is applied to the signal in order to make sure that all recordings are treated similarly in that when the mapping to the mel scale occurs, there is a higher bound on the rightmost triangular filter imposed by the filter. With a high-pass filter at 8 kHz, we enforce that all the filters scale in the same way across the frequency bands and are therefore comparable. This was done because the lowest sample rate in the recordings was 16 kHz, allowing for a maximum frequency of 8 kHz and therefore setting the high-pass filter to 8 kHz, this ensures the maximum representation while still allowing all recordings to be treated equally. Without taking this precaution, variability in the sampling rate of the recording would alter how the pre-defined number of filters are spaced out with the result that higher sampling rates have much broader filters than recordings with lower sampling rates. For example, with no high-pass filter set, the triangular filters could potentially extend up to around 22 kHz (half the value of the highest sampling rate used in the recordings).

If the value set on the high-pass filter were lower, there is a risk of losing speech intelligibility given the fact that frequencies under 8 kHz are the ones typically found in speech sounds. McLoughlin (2009) notes that if a high-pass filter at 2 kHz is applied to a signal at this stage, then only 70% of speech would be intelligible to a human, resulting in a likely worse recognition rate by a computer that does not explicitly possess the inference and gap-filling capability of the human sound processing system (ibid.)

2.2.6 Cepstrum

The cepstrum of a signal is defined as being the Inverse Discrete Fourier Transform (IDFT) of the log magnitude of the DFT of a signal (Jurafsky & Martin, 2009). Cepstral processing is a very useful technique in speech processing because it can be used as a way to separate the various components of a speech sound, namely the source and filter. In Section 2.1 and 2.2 the concept of treating speech sounds as a combination of source (glottal) and filter (tract) components was advocated and it was noted that this conceptualisation would become relevant at the feature extraction stage, and it is precisely in a discussion of the cepstrum that such a notion can be exploited. Cepstral computation is a technique in which such sound components can be efficiently separated. Depending on the task at hand it is possible to extract either the information on the source signal or the vocal/nasal tract filter.

Figure (2.4) shows three different signal representations that arise in computing the cepstrum. Figure (2.4a) shows a signal that is returned from a mel-scaled DFT. (2.4b) then shows the logarithm of the same signal with (2.4c) is then the inverse DFT of (2.4b), which treats the frequency component as if it were a waveform and has the property of separating out the source and filter information.

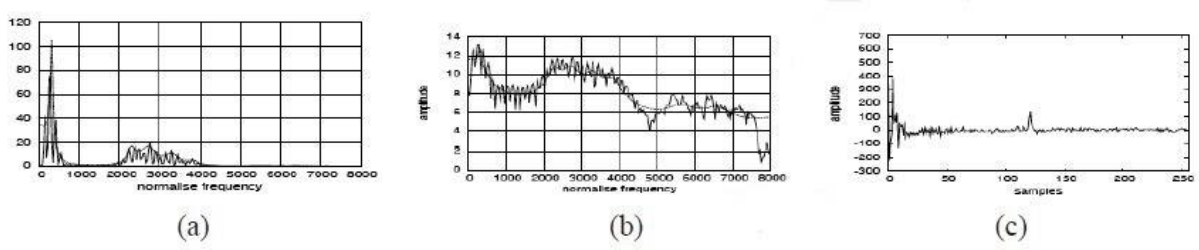

Figure 2.4. (a) DFT spectrum; (b) logarithm of (a); (c) inverse DFT of (b) (Jurafsky & Martin, 2009).

After taking the logarithm of the mel-scaled magnitude from the DFT in order to reduce variability in the higher frequencies, the representation of a signal is that of frequency vs amplitude (Figure (2.4b)). What is noticeable here is that there is a regular high-frequency component that pervades the signal. This repeating pattern is the fundamental frequency of the glottal source, which provides speaker-sensitive information and is generally not useful for phone identification. By treating the log magnitude spectrum as if it were a waveform, applying the inverse DFT to it results in a

further Fourier Transform that extracts the pervasive source signal and separates it from the lower frequency information, which shows the effect of the vocal/nasal tract. Figure (2.4b) contains a repetitive high-frequency component and Figure (2.4c) shows how this component is represented as a bump high up on the cepstrum after taking the inverse DFT.

The lower frequency range in Figure (2.4c) represents the information relating to the tract filter. If the task involved speaker identification or pitch analysis, one could run a low-pass filter and keep the higher content. If the purpose was phone identification then one could run a high-pass filter and only keep the lower frequency content of the cepstrum. After the source information has been removed, the cepstral coefficients can be extracted and used to represent that specific analysis window. An advantage of taking cepstral coefficients is they have the statistical property of being uncorrelated, which means complexity can be reduced in the representation of feature covariance matrices by just implementing a diagonal covariance matrix (Jurafsky & Martin, 2009) for different segments of the word split. This makes Gaussian probability density functions much simpler and also makes it possible to use simpler distance functions in clustering approaches by exploiting the requirement for uncorrelated features in Euclidean distance metrics, as will be discussed in Section 3.1.1.3 in reference to vector quantization approaches to word modelling.

The robustness of the mel-frequency cepstral coefficients lies in the ability for the source and filter to be separated, alongside mapping the potentially variable magnitude frequency values to their logarithms to ensure that only stable and phone-specific features are input to the cepstral calculation and therefore a reliable set of cepstral coefficients can be extracted. Once the glottal source information is filtered out of the signal, the first 22 cepstral coefficients will then be extracted. A typical speech recognition application would use fewer coefficients (Jurafsky & Martin, 2009) but given the lack of a language model and dependency on acoustic signal information alone, I decided to increase the amount of MFCCs to be extracted in order to increase the chance for nuanced word-specific information to be available to the classifier.

2.2.7 Delta features

With a feature set of 22 MFCCs it is possible to extract cepstral information from each analysis window in the classification task, but there is more signal information that can be provided that looks at the window's immediate context to see the rate of change between one window and another. A delta feature is the slope of a line connecting surrounding values and is useful to determine if there is a lot of dramatic change in features. For example, two analysis windows could have the same MFCC feature set but an additional set of features conditioned on the preceding windowed signal can also indicate whether the previous window or windows were relatively similar to the current frame, or if there was a quick change in energy to the following window.

The delta vector can be viewed as the regression coefficient of the previous *N* windows or as the first derivative (velocity) that describes the rate of change between feature values. A large delta value means that the corresponding feature in the previous window was markedly different and a small delta value conversely means a smaller transition. A classifier can use this information to potentially learn from the rates of changes what kind of environment the sound context describes, therefore utilising the wider acoustic context in order to classify a specific segment of a word.

It is also common to treat these delta values as an ever-changing feature representation of the signal and take the derivative of them, therefore representing a second derivative (acceleration) of the sequential MFCC deltas. These values are therefore called double-deltas and provide the classifier with information on the rate of change of the delta values, namely how fast the rate of change is of the delta values. By expanding the MFCCs with their first and second derivatives (delta and double deltas) this makes the feature set more robust to speaker and environment variability (Mariani, 2013). The frame context in which the deltas are calculated is variable depending on the application and larger speech recognition systems can use a frame context of up to 5 in order to calculate these delta values (Jurafsky & Martin, 2009). In this project, I used the *diff* function in NumPy in order to calculate the delta values, which only considers two adjacent frames. With only MFCC values, the feature set consists of 22 features; 44 with the delta values and 66 with deltas and double-deltas.

3 Baseline Results

In this first part of this section I will attempt to establish a baseline accuracy upon which the more advanced machine learning techniques can be based in Section 4. The rationale behind finding a baseline is to get a sense of what works at a rather crude level which can aid the choice of model parameters in more complex implementations, such as the number of features to use or how many segments to split the word into. The second part of this section will address the issue of dimensionality reduction and how it can be applied to reduce the complexity of a dataset while retaining maximum class-separability therefore reducing computational costs and in some classification tasks even projecting the dataset onto a new feature sub-space that also has the added benefit of being more linearly separable, which is a requirement for one of the more advanced machine learning classification techniques that will be discussed, namely Artificial Neural Networks. Both of these tasks are important steps in extracting design and feature set information which will be required later on and will both therefore be addressed in this section.

3.1 Finding a baseline

In attempting to classify individual words there are many possibilities for different model parameters that are both technique-specific (i.e. number of hidden layers in a neural network and specific probability density functions in Markov models) and also more generic parameters that apply across all implementations. Establishing a basic set of results for the generic case provides a reasonable heuristic so that later testing can be constrained by using baseline accuracies for generic implementational issues, i.e. how many segments to split a recording into or how many features to use in the feature set. A drawback with this approach is that the simpler classifier might be more sensitive to distinctions that more advanced techniques can handle better, but given the complexity of the testing space with regard to differing model parameters, it can still be informative to look at simpler methods first to see how they fare when given the same task.

3.1.1 Vector Quantization

The major architecture of a Hidden Markov Model (HMM) will be outlined in greater detail in Section 4.1.2 and therefore the details given in this section will be more specific to Vector Quantization (VQ) which is implemented via a basic HMM architecture. A VQ classifier is defined by three main components; a codebook, a clustering algorithm and distance metric (Jurafsky & Martin, 2009). Together, this provides a way to group features from various training data into discrete classes which together is known as a codebook (ibid). A prototypical representation of a codeword is derived for all items in the codebook and new samples are later compared to them according to a predefined distance metric. A larger codebook allows for more variability and therefore more chances to model various nuances in the data.

The underlying assumptions made in this technique about the data make it a useful implementation of a simple algorithm in order to map the acoustic data onto a restricted set of discrete numbers that is assigned to each word split. The sub-sections below will explore the role of each of the three main components of a VQ classifier in more detail.

3.1.1.1 Codebook

A codebook can be considered a function that maps instances of data, such as a vector of feature values, to a real number from a finite set. The advantage of codebooks is when one needs to work with discrete data that are difficult to model in a continuous way. A codebook is built up from training data via a clustering algorithm that finds *N* codewords in a data set. For each codeword in the codebook, all the instances of the training data that were grouped and assigned to a specific codeword are then averaged and the mean representation of that becomes a prototype for that specific codeword.

In speech recognition tasks that use a form of VQ, the codebook contains a discrete set of codewords, each of which represents a collection of feature vectors that are judged to be of the same class, according to a pre-defined distance metric. The more codewords that are in the codebook, the more chance for nuanced data to be modelled. The downside to this approach is that the larger the codebook size, the

more training data is required in order to successfully model the acoustic features of the word. This problem is known as the *curse of dimensionality* in the field of machine learning (Marsland, 2009), whereby the more complicated an implementation becomes, the more data is needed in order to effectively model the pattern under investigation. Given that the goal of this project is to see how well a classifier can perform on a restricted set of data, the size of the codebook cannot be considerably large and this is why the number of codewords in the codebook does not exceed 200 in the experimental results described later in this section.

Once the codebook has been built by the clustering algorithm, then when a new acoustic feature vector is considered, the distance, given by the distance metric, is calculated and then the codebook is searched in order to find the prototype codeword that has the smallest squared distance between itself and the feature vector that is being processed (Jurafsky & Martin, 2009). Once the closest codeword has been found, the new feature vector is replaced by a single number (the codeword) which is how continuous data can be modelled in a discrete way. Figure (3.1) shows a pictorial representation of how such a vector-to-number process can be conceptualised.

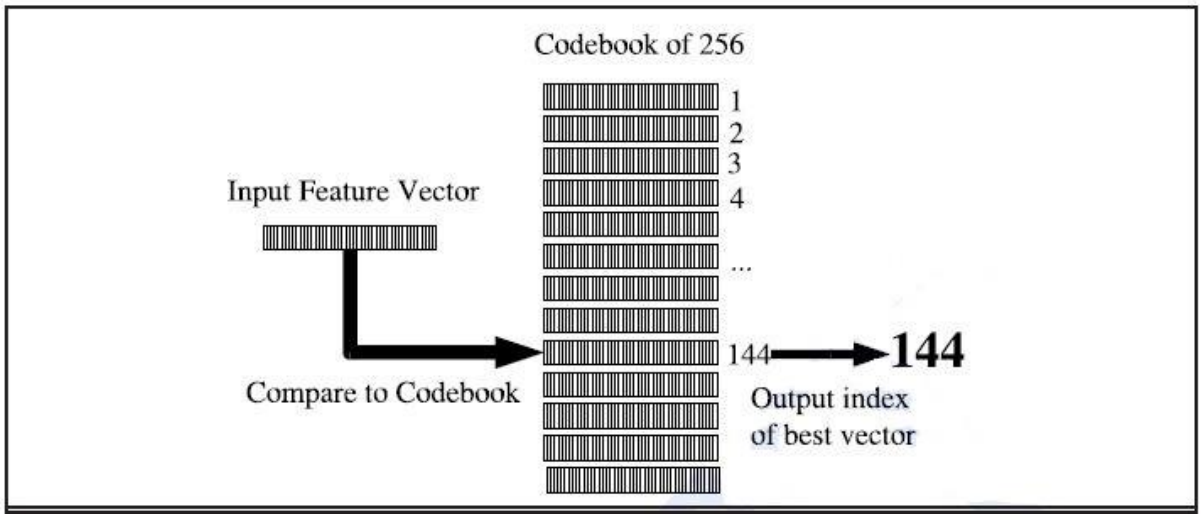

Figure 3.1. An illustration of a codebook, taken from Jurafsky & Martin (2009).

3.1.1.2 Clustering algorithm

The question that still needs to be answered now is how such a codebook can be constructed from the training data and then applied to the test data in order to assess its accuracy. For that, the K-means algorithm is used. The K-means algorithm is a

machine learning technique that is the hallmark of data clustering (Marsland, 2009). It is an exemplar of the unsupervised learning approach whereby a structure of the data is built up without any access to the correct class labels to learn from. The access to (or rather, use of) class labels is the fundamental distinction between supervised learning approaches and unsupervised ones (ibid.)

With the stipulation that the format of the data be numerical, the task of the K-means algorithm is to find a way to partition the data into *K* different clusters whose inter-point distances are small compared to the distances of data points outside of the cluster (Bishop, 2006). The algorithm views each data point as a point on a hyperplane with the same dimensionality as that of the feature set. The expectation is thus that each feature vector from the training data that belongs to a class is more similar to the other instances of data points in that same class than it is to points in different classes. If we did not have that expectation of separability then we would not expect that the clustering approach would be suitable in a classification problem of this kind.

The algorithm starts by randomly instantiating *K* different centroids in D-dimensional space, where *D* is the size of the feature set. If only MFCC values are used in this IWR task then each data point would be represented in 22-dimensional space; 44-dimensional space if the first derivatives are added and so on and so forth. All of the feature vectors derived from the training set are then assigned to the closest of the randomly instantiated centroids and then the average is taken of all the data points in that cluster and this becomes the new location of the centroid. This process continues and the centroid gradually shifts around the D-dimensional space until the distance of each centroid falls under a pre-set value which is usually very small or until complete convergence is reached. When this process has finished, each centroid represents the average of each cluster found and that location in D-dimensional space becomes that cluster's codeword (averaged feature vector) in the codebook.

There are a number of problems that must be considered when using clustering methods such as K-means, one of which is selecting the best value of *K*. There are various heuristics for selecting this value but each depends on the problem specification and one can never be sure that a value is optimal without experimenting with a range of different values in order to see which one performs the best on the

training and test set. Another problem is the random initialisation of centroids which changes every time new training data is added and the codebook is recomputed. This means that two separate instances of running the same code with the same data would likely result in a different accuracy, which is problematic if the goal is a reliable set of results. For this reason, trials involving K-means are typically run a few times and the accuracy is taken as an average of these results in order to avoid any particularly extreme or unlikely results that would be difficult to replicate in a future trial.

A final problem worth noting is the computational requirements of the algorithm which results in a fairly slow training phase when generating the codebook. This is because the distance metric which compares the similarity of data points is applied to every training data instance multiple times until the centroids converge to their final locations. Further considerations on the type of distance metric also vary in terms of complexity and will be discussed in the following in the following sub-section.

3.1.1.3 Distance metric

In order to determine the precise geometric distance between two vectors in multi-dimensional space, it is important to address the features of the data that might need to be taken into consideration which could increase the performance of the K-means algorithm. There are typically two main distance metrics that are commonly used with unsupervised clustering, Euclidean distance and Mahalanobis distance (Bishop, 2009).

The Euclidean distance between two vectors is measured according to the Pythagorean formula, $a^2 + b^2 = c^2$, which for vectors is equivalent to the square root of the inner product of the difference of the vectors, namely $\sqrt{(\boldsymbol{w^T} - \boldsymbol{y^T})\,(\boldsymbol{w} - \boldsymbol{y})}$ where $\boldsymbol{w}$ and $\boldsymbol{y}$ are vectors containing the acoustic features. This a relatively simple way of calculating how close two vectors are in higher dimensions, especially with highly optimised programming libraries for vectorised linear algebra operations such as MATLAB and especially NumPy in Python.

A drawback with the approach is that it assumes that all the values are scaled by their respective variances to have unit variance. Without implementing such a normalisation step then one feature which does not vary a lot across the training data might exhibit a small shift that is very significant in terms of that feature, but in a

distance calculation with respect to another feature that has a very high variance then this potentially important characteristic of the data would be insignificant and lost in the noise of other features.

A requirement of the Euclidean distance measure is that all data must have the same covariance matrix (ibid.), which is important on theoretical grounds but is a requirement that is often not followed in many applications but is still implemented because of the computational simplicity of the approach it affords while still proving to be useful (Theodorodis & Koutroumbas, 2009).

The Mahalanobis distance is an extension of the Euclidean distance measure whereby the equal covariance matrix requirement is relaxed and is explicitly factored into the distance equation. This allows for co-varying features to be taken into account when calculating distance in multi-dimensional space. The advantage of using the Mahalanobis distance is that the decision boundaries are elliptical and not circular as would be the case in the Euclidean distance measure (ibid.) Allowing for covariance information in the calculation is much more reliable because it has a better model of how the data are distributed. The downside is that is often significantly more computationally expensive to factor this information in an algorithm that calls upon the distance metric so frequently.

An ideal combination would be to have the computational simplicity of the Euclidean distance measure, with the ability to not lose potentially significant information that the Mahalanobis distance offers. Thankfully, we do get a good compromise in that a consequence of the cepstral analysis used to extract the acoustic features is such that the variance among these features tends to be uncorrelated (Jurafsky & Martin, 2009). This means that we can assume the diagonal covariance matrix required by the Euclidean distance measure because the co-variances of the features became uncorrelated via cepstral feature extraction. This means that a potentially critical loss of information is minimised and we can use the more computationally efficient Euclidean distance metric.

3.1.2 Implementation details

In order to calculate the probability of a word given a sequence of acoustic feature vectors, the Forward algorithm was used. This approach takes advantage of the fact that, unlike other HMM implementations, where the most likely sequence of states is calculated from a sequence of vectors via Viterbi decoding, because each word is represented by an individual model of a HMM then the Forward algorithm answers the question of how likely a HMM is given the observation of the features derived from the speech signal. The final probability is derived from a dynamic programming table that that is incrementally calculated according to the value of the word split (because the amount of time steps corresponds to how many segments the query signal has), which can also be thought of as a temporally-advancing division across which feature values are averaged in every word segment.

Given that this algorithm typically makes use of the product of the values in the previous word-split segment along with the transition probability of moving from a previous state to the current state in the calculation, the exact computation of the Forward probability in this implementation was slightly relaxed to only factor in the sum of the previous word-split segment because the transition probability is not relevant in an isolated-word recognition task, given that the vocabulary item is modelled as a whole HMM and not as a sequence of phones or sub-phones as would be the case in larger speech recognition applications (Jurafsky & Martin, 2009). The equation for the Forward algorithm for a certain cell in the probability table, namely at time-step ***t*** in state ***j***, is given below in Equation (3.1). The altered algorithm, not including the transition matrix value from state ***i*** to ***j***, a_{ij} is then given in Equation (3.2). The final element $b_j(o_t)$ represents the probability of state ***j*** generating the codeword found at time ***t*** in the observation sequence of acoustic vectors, namely ***o***.

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(o_t) \tag{3.1}$$

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) b_j(o_t) \tag{3.2}$$

3.1.3 Results

Figures (3.2), (3.3) and (3.4) given below show the various results obtained by running VQ on varying word splits and codebook sizes. The first graph shows the results using only MFCC features, the second graph shows the MFCC features with first derivatives and the third graph shows the results of MFCC features with both first and second derivatives (i.e. deltas and double-deltas) included. The continuous coloured lines represent overall accuracy of both the training and test set, followed by the dashed lines which represent the accuracy on the test set, i.e. the accuracy when applied to recordings that were not included in the training set. The non-dashed lines are to be understood as a statement of the classifier's ability to model the recordings it was trained on in addition to the test set, while the dashed lines represent an explicit statement on the classifier's ability to accurately generalise to unseen data. The results in the following graphs are an average over three separate classification runs in order to reduce the possibility that any one run might have provided especially optimal (or suboptimal) results given the random assignment of the initial centroids inherent in the K-means algorithm.

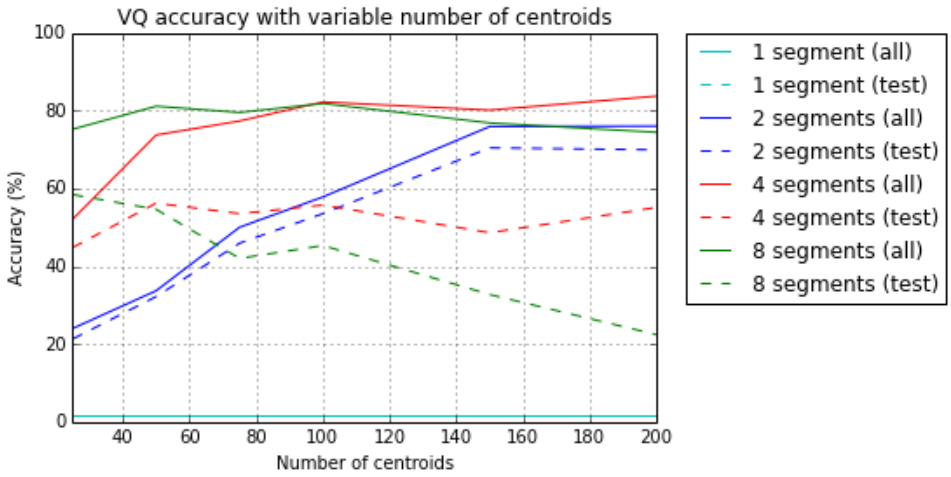

Figure 3.2. VQ accuracy using MFCCs only (22 features)

There is a noticeable pattern in the amount of features that is required in order to reach an accuracy rate of around 80%. The trial with the word split value of 1, i.e. the entire speech signal was treated as one continuous segment, didn't manage to model anything except the first default result, i.e. the same result is always returned from the classifier if there has been a failure to model the word features sufficiently, and 1.6% of the time this is actually the correct answer, which is shown above. As the

word split value slowly increases, the accuracy is higher at the lower end of the number of centroids. The highest accuracy when 40 centroids are used is with a word split value of 8 that results in just under 80% for the entire training and test set. When the word split value is 4, this value is 52.1%, a significant drop because there is less chance to model the nuances of each word with so few centroids. With an even coarser division of the word into only two segments, the accuracy drops again. The pattern holds for the test set explicitly, but an interesting observation is that as the word split value is smaller, there is a tighter relationship between the accuracy on the test set alone and on the combined set. The division between the green lines is much larger than the division between the red, and the same goes for the blue. The cyan lines lie completely on top of each other so this gives the appearance of only one line.

In contrast to the better performance as the word split value increases when the number of centroids used is fairly small, as this number increases, the reverse pattern of accuracy can be observed, namely that the larger the word split value, with 200 centroids, the lower the ability of VQ to correctly model the test set. With the exception of the word split value of 1, when 200 centroids are used, it is the binary split that performs the best, followed by a word split value of 4 and then 8. This initially points to the fact that it is better, at least for this feature set of MFCC values, to have a smaller word split value and a higher concentration of centroids to help model the vocabulary.

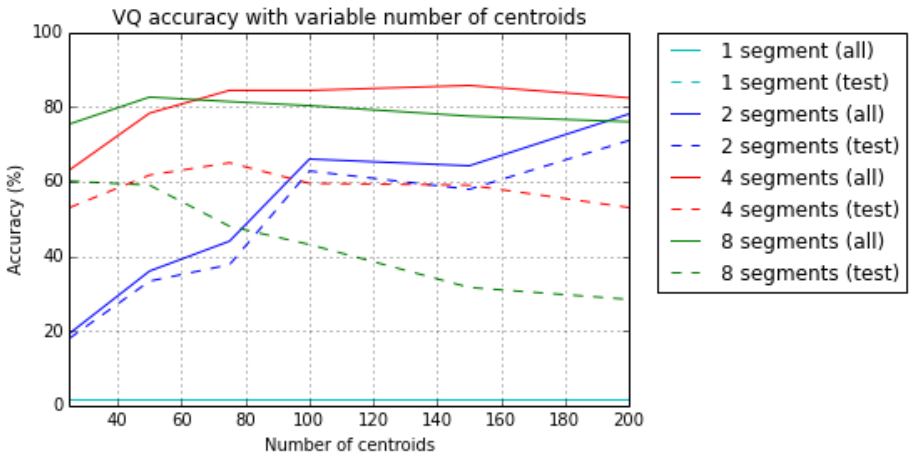

Figure 3.3. VQ accuracy using MFCCs and first derivatives (44 features)

The first noticeable point here is to observe that a higher accuracy can be achieved by extending the MFCC feature set with the deltas, specifically when the word is split into 4 segments and 150 centroids are used, an accuracy of 85.7% can be achieved on the combined set. There is a strong correlation between the combined set and the test set when the word split value is 2 and is in fact the only case that gradually achieves a better accuracy as the number of centroids increases. A similar pattern can be observed in Figure (3.3) as was observed in Figure (3.2), namely that as more centroids are used in the learning process, it is the smaller word split values that achieve the highest accuracy, still with the exception of no word split (i.e. treating the whole word as one segment).

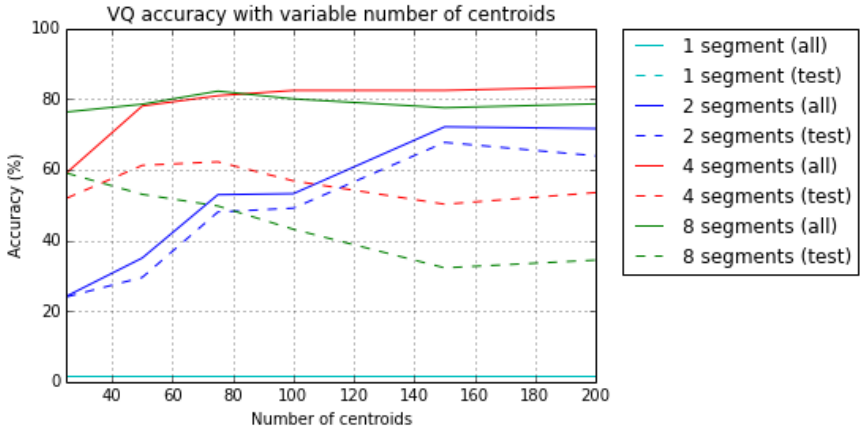

Figure 3.4. VQ accuracy using MFCCs and first/second derivatives (66 features)

Given the nature of double deltas, which are themselves fractionally smaller than the delta values themselves, it should come as no surprise that Figures (3.3) and (3.4) are rather alike, and this is the case as can be seen by clearly by the pattern exhibited by blue lines which represent a word split value of 2 and follow a very similar trajectory in both graphs. One noticeable difference that can be observed is the marginally poorer performance all round, suggesting that, at least with significantly fewer word segments, that these values are not useful. The blue dashed line representing 2 word segments gradually starts performing worse after 150 centroids are used, unlike in Figure (3.3).

3.1.4 Discussion

There are numerous interesting observations that can be seen from these results. Treating the word as one segment utterly fails to model the data in all cases. The

guesses all default to the first word in the vocabulary (*einn*) and therefore the classifier managed to classify the proportion of the data set where the correct word was actually *einn*, namely 1.6%. If there are too few codewords in the codebook, not enough nuanced characteristics from the feature set can help correctly model a specific word individually, while too many leaves the modelling process too weak to be able to generalise from similar (but not identical) pronunciations of the same word.

The highest accuracy on the test set was 71% with a word split of 4 using the MFCC values and the first derivatives with a codebook size of 200. The second highest was 70.4% with a word split value of 4 with a codebook size of 150 also using only the MFCC values. There appears to be a trade-off between the amount of features and word split value in that the fewer features used usually requires a higher word split value in order to achieve an accuracy in the range of 80% on the combined training and test set, while a lower word split value can achieve a similar accuracy if it has a codebook size of a sufficient value to model the acoustic features of the whole word. This is an interesting point because in more advanced HMM implementations, the ability to model nuanced data is greatly increased and if this can be achieved while splitting the word into only 2 or 4 segments (which here achieves the highest accuracies) then this is a useful result to bear in mind when testing accuracy at a later stage in a more advanced implementation.

Given the size of the vocabulary to be recognised, it does not appear that any combination of codebook sizes or word split values gives a result on the test set that exceeds 71%. This demonstrates a generalisable accuracy that is not sufficient for any real-world application and therefore it can be safely assumed that such an approach is not scalable to a larger problem set. This highlights the need for a better modelling technique and classification algorithm to tackle this problem. A potential problem with the feature set in this implementation is the interference of features that are not useful to the classifier. A way to increase the performance and potentially reduce the complexity is to remove all features that do not help identify the correct word being spoken. This feature processing technique falls under the field of dimensionality reduction and is the topic of the next sub-section.

3.2 Dimensionality Reduction

Having established a tentative estimate of the VQ approach to the task of correctly recognising all the words in the test set with varying amounts of features and codebook sizes, it is possible to use this information as a heuristic to constrain the varying parameters that will be applied later in more advanced techniques. The results from the previous section show that the best accuracy obtained was with a word split of 2 with a codebook size of 200. With an eye to scalability of the task, it makes sense that the word should not be split into too many segments which could lead to the inability to model characteristic portions of the speech signal; too few discrete codewords would also lead to a similar problem. If there were a lot of training data, then there would be more scope to allow for nuanced learning models but the task specification is to try to learn from minimal amounts of data, namely seven training files of the same spoken word.

In this section I will examine a method to reduce the dimensionality of the problem while focusing on rather small word split values. Then, I will re-apply this new reduced feature set onto the VQ approach in order to see how the classifier performs with fewer features. I will first describe Principal Component Analysis (PCA) which would treat the entire data set as a one whole and does not take into account the class labels. I will then look at an adaption of PCA to an algorithm that has the same goal of dimensionality reduction but does take into consideration class labels, namely Multiple Discriminant Analysis (MDA).

3.2.1 Principal Component Analysis (PCA)

The technical definition of PCA is that of finding an orthogonal projection of the data onto a lower dimensional subspace such that the maximum variance is retained in the data and this new feature space is then known as the principal subspace of the original data set (Bishop, 2009). This essentially means that it is possible to extract K components that constitute the feature set (with original dimensionality D) in order to represent the data in a lower-dimensional subspace, i.e. where $K < D$. Principal component analysis allows a data set to be represented in a simpler form if there is a

lot of redundancy in how data is stored, and for this reason it is a popular algorithm for data compression of photos, music and other common digital content.

If a data set is described by 50 features but only 5 provide the main characteristics that differentiate different data points, then it would be possible to represent this data set by only these 5 features while only suffering minimal loss in terms of descriptive quality of those other features. When this is possible, the space requirement is drastically reduced and is in general a much more efficient way to store digital media.

A consequence of this means that the computational complexity and storage requirements can be dramatically reduced whilst also removing a lot of data that would not be valuable in a classification task, therefore avoiding the curse of dimensionality problem when dealing with relatively restricted training data. This is a very useful technique to apply and is likely the reason it is such a popular algorithm and an essential pre-processing step in most large-scale classification tasks. The goal is to keep the top *K* components that maximise the variance in the data set, that is, components or features that help to distinguish between the entire set of different data samples. PCA when applied to classification tasks rather than data compression is an unsupervised procedure (Marsland, 2009). This method of dimensionality reduction relies on eigendecomposition of the covariance matrix of the data set, which also applies to MDA. The intuition of how PCA works is described in the following sub-section.

3.2.1.1. Intuition of PCA

The basic outline of the PCA algorithm involves concatenating the entire data set and removing all class labels (if any were present), calculating the mean of the D-dimensional matrix where *D* is the original size of the feature set. The covariance matrix of the entire data set is then calculated and from this it is possible to use the linear algebra technique of eigendecomposition, which represents the matrix in terms of eigenvectors and eigenvalues. For each dimension of the data in the original matrix there is a corresponding eigenvector and eigenvalue. The eigenvector represents a scaled value (unit length) of the data which has a direction associated with it and each

eigenvector has an associated eigenvalue which represents the corresponding magnitude of the vector (Bishop, 2009).

If an eigenvalue is relatively small, this means the corresponding eigenvector does not hold a lot of information about the distribution of the data and is thus it is not a desirable component in the feature set when the goal is using such a feature to help explain the distribution of data. A feature that has a relatively similar value in every data sample would therefore have a very small eigenvalue for its corresponding eigenvector. This means that the largest eigenvalues and their corresponding eigenvectors bear the most information about the distribution of the data and are therefore the feature components that are required for successful classification of new data instances.

By ordering the eigenvalues in decreasing value, coupled with their corresponding eigenvectors, picking the top *K* components, with $K < D$, it is possible to iteratively join the first *K*-largest eigenvectors in order to build a transformation matrix commonly known simply as w. The linear transform required to project the original data onto the new principal subspace is given by the equation $y = w^T x$. This equation states that the inner product of the transpose of w with the original data matrix x would return a projection of the original matrix onto the principal subspace with dimensionality *K*. If x consists of 100 data samples with 50 features, that is to say, the shape of x is (100, 50) and the new principal subspace should consist of $K = 25$ components, w^T would have the shape (50, 25). By the property of matrix multiplication this means that the y matrix would have a shape of (100, 25). Therefore, the original matrix of size (100, 50) can now be replaced with the result y, which has a shape of (100, 25), i.e. the dimensionality of the features (row-wise) has been reduced from 50 to 25.

PCA is useful in situations where one wants to compress data, where there is no need to take class labels into account. In cases where such distinctions are important, by using PCA, one runs the risk of removing important features that are essential for successful discriminative classification. This means that an adaption of the algorithm is required that achieves the same goal of dimensionality reduction, but takes into account the different classes and maximises the inter-class and intra-class

variance between data samples. An algorithm to accomplish this is outlined in the next section.

3.2.2 Multiple Discriminant Analysis (MDA)

The general PCA algorithm had already been worked out by the turn of the 20th century as can be seen in works such as Pearson (1901) but the problem still remained that it was not possible to reduce the dimensionality of a data set this way when it consisted of multiple classes. The underlying assumption that the data set was all of the same class did not hold in a lot of cases and although in some instances applying PCA can improve accuracy with a lower-dimensional feature set, there is no assurance that valuable components would not be removed as a result. This might happen where one feature is relatively stable across the whole class in a binary classification problem, but significantly different in all instances of the other class. PCA would analyse this feature as relatively unhelpful as it encompasses half of the data samples, but in fact, for that class, it is an extremely important characteristic and one that should be given more weight as an important component for class separability. This is the type of class-specific feature which MDA helps to retain in the lower-dimensional subspace in order to be more suitable for future classification (Barber, 2012).

The influential statistician Ronald Fisher developed an extension to PCA in 1936 in order to address the issue of binary classification problems, which goes under the name (Fisher's) Linear Discriminant Analysis (LDA). This idea was then later extended to multi-class problems by C. R. Rao in 1948, which is the method considered in this section (Raschka, 2014). Like PCA, MDA aims to find the eigenvalues and eigenvectors that favours the axes of variance that maximise the class separability of data. In the section that follows I will explain how MDA differs from PCA as explained in the previous section. I will then apply MDA to the data set and examine how this affects the results when applied to the VQ approach discussed in Section 3.1.1.

3.2.2.1 Intuition of Multiple Discriminant Analysis

There are a lot of similarities between MDA and PCA due to the fact that MDA is an adaptation of PCA when applied to multi-class data. The data set needs to be normalised so that the mean vectors of the data are all zero and each feature has unit

standard deviation. Both LDA and MDA have a formal requirement that the data under consideration is normally-distributed, with independent uncorrelated features and identical covariance matrices for each class (ibid.) This is quite a strict requirement of the data when these methods are used in classification tasks yet it has been noted that in many cases these requirements aren't adhered to and are often violated, yet still perform considerably well in practice (Tao Li et al., 2006).

The main difference between MDA and PCA is the fact that there needs to be a way to factor in the data from within each class and then the overall data set yet still maintain the class divisions. This is done by computing two scatter matrices, a within-class scatter matrix, Sw, and a between-class scatter matrix, Sb. Once the two scatter matrices have been computed, the generalised eigenvalue problem equation is applied to them, namely $Sw^{-1}Sb$. The inverse of the within-class scatter matrix is matrix-multiplied with the between-class scatter matrix and this gives the linear discriminants that provide the maximum variance whilst also mainlining maximum class separability of the data. The result is a set of eigenvectors and corresponding eigenvalues where the selection of the top K components and construction the w matrix is the same as in PCA.

3.2.2.2 Applying Multiple Discriminant Analysis

In order to see what effect MDA has on the data set, it is important to restrict the variable parameters of the problem, namely to decide in advance what the range of the word split value would be, how many features to use etc. The problem becomes too large in order to investigate MDA on every possible combination of these parameters, so the results of the VQ approach described in Section 3.1.3 will be used as a starting point with which the accuracy of a lower dimensionality feature set can be evaluated.

As pointed out in the results section of the VQ approach, the highest accuracy on the test set was with a word split value of 2, codebook size of 200 and a feature set consisting of 44 values (MFCCs plus first derivatives), closely followed by the second highest accuracy, with a word split value of 4, codebook size of 150 and a feature set consisting of 22 values (MFCCs only). This information would be used as a heuristic to reduce the possible values of word splits and codebook sizes in a more advanced

implementation of the classifier and the same reasoning applies in the task of dimensionality reduction. With a word split value of 2, values close to this will be tested but in a much more limited range around that number, namely 2, 4, 6 and 8. Averages of the features in each word split are used and therefore it is obvious that the double deltas will not supply useful information about the rate of change of the (single) deltas with so few word segments, therefore they will not be included here. It is less obvious how this would affect the delta values and for this reason they are included in the following tests.

Figure (3.5) below shows how dimensionality reduction affects the overall accuracy when using the 22 MFCC-value feature set. The results are based on a codebook size of 175, halfway between 150 and 200, which gave the two highest results on the test set in the previous section.

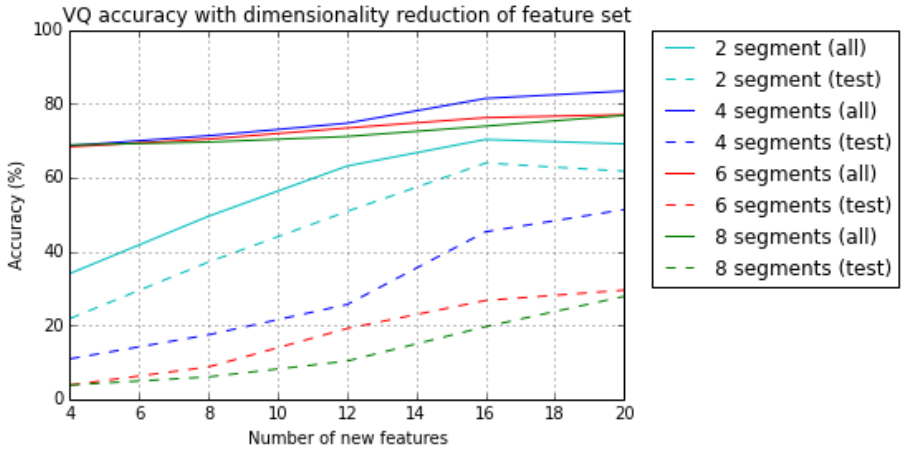

Figure 3.5. Effects of dimensionality reduction on MFCC feature set

What can be seen immediately is that there is great similarity on the combined set, consisting of both the training and the test data as one. When there is a binary split, with only 4 of the top features being used, this isn't modelled as well as the higher word split values, which are remarkably similar. A rather consistent pattern can be observed on the training data as well, the higher the word split value, the greater predictive capacity the algorithm has. When considering the test set only, the binary word split performs the best.

Figure (3.6) shows the same pattern of results but based on the larger feature set of MFCC values plus deltas.

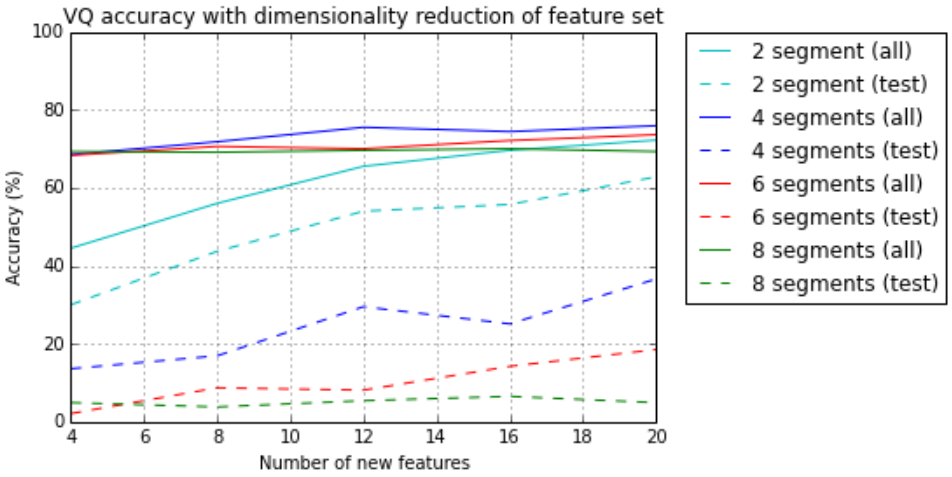

Figure 3.6. Effects of dimensionality reduction on MFCC plus delta feature set

In both graphs, the top 4, 8, 12, 16 and 20 features that are returned from the MDA algorithm were used as the feature set in a classification trial for the entire data set. As can be seen, the more features that are used, the more accurate the result is, yet as the word split value increases, it is less able to model the acoustic characteristics of each word. It should be expected that the vector quantization approach is not especially sensitive to all features on an individual level given, though there are more codewords for results in terms of absolute value, meaning there is more chance for nuances in the data to be captured. This means that with a better modelling technique, it should be possible to reduce the features from, for example, 22 in the case where all the MFCCs are used, and still obtain a better accuracy than is given via this VQ implementation if differences in the variances of the entire dimension of a specific feature can be taken into account.

If the codebook size were reduced significantly, thus allowing for a coarser ability to model the nuances in the data, then a higher accuracy should be attainable with significantly fewer features. A prediction can be made that if the only components that constitute up to 80-90% of the variance are used with a smaller codebook size, therefore with a cruder ability to model smaller differences in the recorded audio files, higher accuracies proportional to the reduced features should be expected. This approach is not one that can be expected to scale well in larger tasks and therefore explicitly testing dimensionality reduction on more crude distinctions does not aid in understanding the modelling potential of this approach. For this

reason, the results given above focus on a more promising setup of word split values and features that can be potentially adapted and scalable as more vocabulary items, and thus more acoustic diversity, is added to the training data from which word models are derived.

However, it is worth illustrating how different parameters can affect results and therefore underline the need to run baseline tests over many possible combinations of values so that initial results are not misinterpreted as overly optimal. This way it is possible to guard against the possibility of chancing upon a set of parameters, i.e. feature values and word split values that give good results but are too restrictive to scale up in larger applications. Figure (3.7) shows the percentage of variance that the top 5 components describes in the training set when using only MFCCs as features as input to the MDA algorithm.

Top 5 components	Variance (%)
1st component	70.5%
2nd component	15.3%
3rd component	4.3%
4th component	3.0%
5th component	1.7%

Figure 3.7. Distribution of variance in MFCC-only feature set.

The top 5 components account for 94.8% of the variance in the data. By using only these 5 features with a word split of 4 and a codebook size of 50, which received an average score of 73.7% on the entire training and test set and 56.2% on the test set alone (see Section 3.1.3), it should be possible that a substantial reduction in features would not have a correspondingly proportional reduction in accuracy on the test set. Applying this test, averaged over 3 runs to avoid any extreme results given the random centroid initialisation of the K-means algorithm in the codebook generation stage, results in an overall accuracy of 60.9% with an accuracy of 36.6% on the test set.

This shows that with only 22.7% of the original features, the accuracy only dropped by 12.8% on the combined data set and 19.6% on the test set. By showing that approximately 80% of the features can be discarded while only suffering a drop of under 20% in accuracy on the test set, this demonstrate precisely the attractive quality of dimensionality reduction. Such a drop in accuracy is not necessarily desirable, but

with cleverer modelling techniques that are more sensitive to features with high variance, a lot of noise can be removed from the feature set that does not aid in maximising class separability.

An interesting result is that with the deltas included in the feature set, there is an increase in accuracy when fewer features are used in the binary split, but the top 5 principal components only account for approximately 50% of the variance of the data in that instance. This likely means that the predominace of the first principal component for the MFCC-only feature set overshadowed the other features when calculating the Euclidean distance. The 5 principal components for MFCCs and deltas in the feature set are more evenly distributed,with no component anywhere near as dominant as in the MFCC feature set, therefore explaining how the results are more accurate on average when only considering the lower principal subspace used in the feature set. This is due to the fact that the selection of the relevant codeword is not as dependent on a single feature as it is in the MFCC case, resulting in a 10% increase in accuracy on the test set. As the word split value increases, the accuracy drops more dramatically than with just the MFCC features as would be expected since these principal components account for proportionally less of the variance than when just the MFCCs are used.

Given how the feature set consisting of MFCCs and deltas fared gradually worse with significant reduction in the dimensionality of the feature set, this confirms that it would not be feasible to expect that the double deltas would provide any clear value regarding maximum class-separability in the region of lower word split values.

In the following section, I will explore how more advanced machine learning techniques can model the training data and how dimensionality reduction compares in those cases when considering accuracy on the test set. Dimensionality reduction clearly has an important role to play, though its major benefits are not necessarily reflected in a VQ approach. The techniques used in the following section will be more responsive to the removal of unhelpful features and in those cases the true advantage of this approach can be appreciated.

4 Advanced Models

In this section I will address how more advanced machine learning algorithms can be applied to the word-modelling problem in an isolated word recognition task. I will first look at a hidden Markov model implementation, because it shares a lot of similarities with the vector quantization approach addressed in the last section and is a natural extension of that procedure. I will mention the improvements upon the vector quantization approach and how aspects of the algorithm can be altered to allow for a better word representation given acoustic features, followed by a brief illustration of how word modelling can be conceptualised in this approach before moving on to the results and discussion of the experiments. I will then move on to describe how artificial neural networks have been used to tackle pattern classification tasks, outline their structure and give a brief history to contextualise their use today, externally to and also within speech recognition tasks. This section ends with a description of the results and a discussion section to summarise the procedure.

4.1 Hidden Markov Model

Hidden Markov Models have been used extensively in speech recognition tasks since they were first brought to the attention of the wider scientific field by Rabiner's influential paper in 1989 (Mariani, 2013). There, the groundwork was set and effective algorithms for solving specific fundamental questions of the data were laid out in detail. The algorithms used today are largely the same as those described by Rabiner or simple extensions of them (Manning & Schütze, 1999). In this section I will give a few remarks about the advantages this implementation has over the vector quantization approach discussed in the previous section, the structure of standard HMM architecture, specific considerations on improving the algorithm and finally I will look at the accuracy attained by using this implementation in comparison to the established baseline outlined in the previous section.

4.1.1 Improvements upon Vector Quantization

There are numerous improvements that can be made in a fuller implementation of a hidden Markov model, which are not subject to the more severe restrictions that are

imposed on the problem specification by the VQ approach. One such restriction was that a codebook had to be used, which is essentially a function that maps a vector onto a discrete symbol from a finite set. This means that each dimension of that feature vector had a proportionally equal weight than if all feature dimensions were treated individually. This is a typical problem in discrete systems where separation boundaries have to be drawn and ultimately results in significant loss of potentially useful information.

A solution to this problem is to find a way to compare the similarity of two specific feature vectors that does not rely on reducing them to a single symbols, but rather on taking all comparable dimensions into account. In order to achieve this there must be a way to model the varying feature values in a continuous way, as opposed to a discrete way. In the discrete implementation in VQ, once the boundary between codewords was established, it was not possible to take any other symbols and compare how close they were to each other in multi-dimensional space. We could only say whether the feature vectors were mapped to the same symbol, so every other symbol was just as likely (or unlikely) to be related. An advantage of using continuous probability distributions also solves this problem whereby a distribution can return a result that states exactly how similar or non-similar two feature vectors are. This ability to continuously model the feature vectors, derived from the framed signal, is very important in how isolated word recognition can be used to model individual vocabulary items. By employing the Gaussian probability density function (PDF), such continuous modelling can be implemented effectively.

An issue in VQ was that the K-means algorithm had to randomly initialise a set of centroids of a predefined quantity, which then were redistributed until they had converged to specific clusters in the data. Having such a random process lie at the core of a classifier's ability to model a problem is not desirable if stable results are required. This also meant that multiple trials had to be run in order to take an average to avoid the potential pitfall of stumbling upon a particularly lucky (or unlucky) centroid distribution of the data. The HMM approach does not require such randomness and is therefore much more stable from the perspective of result replicability.

The use of a Gaussian PDF also means that temporal sequences can be modelled much more effectively than in a sequence of discrete symbols because each dimension of the data set is capable of providing more specific information based on the overall standard deviation of that feature dimension for each word split that is analysed. The ability to be able to pick up on patterns and relate gradual shifts in similar feature characteristics at specific points in the pronunciation of a word also allows for a much greater ability to model nuances in the data. This is an extremely important point in a task that relies on limited training data to successfully model words with such granularity.

With the benefit of a continuous probability distribution across the word splits used in all the individual word HMMs, it is possible to exploit this fact to constrain the likelihood algorithm that a specific sequence of acoustic feature vectors was generated by a specific HMM model in order to avoid allowing earlier segments to sway the probability of later segments that have repeated instances of similar phonetic quality. This will be clarified further in Section 4.1.3.

4.1.2 HMM Architecture

A Hidden Markov Model is defined by a finite set of states, a set of symbols, a transition matrix and an emission matrix, sometimes called an observation matrix. In discrete implementations, the set of symbols is finite (as in the vector quantization approach) but in continuous implementations the set of symbols is infinite (Jurafsky & Martin, 2009). HMMs represent a form of dynamic programming where probabilities are calculated in a tabular manner, called a trellis diagram (ibid.) For each observation sequence as input to the model, a table is created with the same amount of sequential time steps and a specialised algorithms calculate either the probability of a given HMM generating the entire sequence (the Forward algorithm) or how likely a sequence of states is, given the model (the Viterbi algorithm). When the model parameters need to be learned from scratch, the Baum-Welch algorithm, also called the Forward-Backward algorithm, is used to derive the transition and emission matrices.

The emission probability matrix (often abbreviated simply as the B matrix) has associated with it a probability function that is variable according to the specific task at

hand. For example, in discrete implementations the probability of a specific symbol can be worked out directly from the training data, i.e. how often a symbol was seen in a specific state, divided by all symbols seen in that state. In continuous implementations a probability function needs to be employed to estimate the probability based on the mean and standard deviation of a specific feature or vector of features. In order to estimate this, a Gaussian distribution is typically used. If there is only a single feature then the univariate Gaussian distribution is used. The multivariate Gaussian distribution is required for feature vectors with multiple values. The feature set used in this project is multi-dimensional and therefore requires the multivariate Gaussian distribution, which is defined in Equation (4.1):

$$p(\boldsymbol{x}) = \frac{1}{(2\pi)^{d/2}\,|\Sigma|^{1/2}}\, exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^t\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right] \tag{4.1}$$

A multivariate Gaussian distribution is defined by an average (mean) vector μ and a covariance matrix Σ. In order to employ the multivariate Gaussian distribution, these values must be computed at the training stage and then supplied to the probability function when calculating the likelihood that a specific state in a HMM produced the acoustic vector being queried.

4.1.3 Altering the Forward Algorithm

In the vector quantization approach discussed in Section 3.1.2, the Forward algorithm, which is used to calculate the probability of a sequence of acoustic feature vectors given a specific HMM, was altered because there was no transition information available given that the states of the model only represented separate portions of the same word and were not directly associated to shared linguistic elements such as phones or syllables. If there would be one HMM for the whole task and a specific path where states representing phones, sub-phones or syllables needed to be calculated, then the transition matrix would be very important, but given that in this implementation each vocabulary item is represented by its own HMM, the constraints that the transition matrix imposes on the final calculation do not serve to identify a specific word so were removed from the equation.

In the Forward algorithm, implemented in the vector quantization approach, the Forward probabilities were all summed from all previous time step in the trellis model (i.e. previous word segment for our purposes) via the altered algorithm given in Equation (3.2) repeated here as Equation (4.2).

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) b_j(o_t) \tag{4.2}$$

The undesirable feature of this algorithm is that if an acoustic feature vector (there mapped onto a specific codeword) matched a word segment anywhere in the total amount of word splits, this positive probability was used in accordance with the algorithm above. For example, if the final word segment of a new recording is being queried, it is not helpful in the classification task to count probability that recognised that the first word segment of a word that contained a similar set of features. To put this in a concrete example, if the codeword mapped the feature vectors for the phones [t] and [t^h] to a discrete symbol, i.e. to codeword 15 for illustrative purposes, and the query recording was of the word *fimmta* (fifth), when querying the HMM for the word *tveir* (two), the first word segment that contains [t^h] would also map to codeword 15, so this probability would positively count to the overall probability that the word was *tveir* even though the [t^h] occurred at the start of the word there, but it occurred almost at the end for the actual word, *fimmta*. It would be more sensible to devise a way to prevent this from happening.

A better way to model the sequential path through word segments would be to only count the probability of a specific feature symbol or feature vector *at that same point in the word*. This means that in the case described above, the occurrence of [t^h] would not receive any probability unless there was a corresponding [t^h] (or other phone that also mapped to the same codeword) at the same location in the word. This would reduce the amount of erroneous probability that is propagated through the trellis model during the calculation. The final probability would be taken as the value of the last state in the table and not the sum of the entire column as per the Forward algorithm used in the previous section.

The only cell of interest is then the one that has the probability at the last word segment. By incorporating the identity matrix as the transition matrix, the normal Forward algorithm would produce this constraining effect. The only difference would be that the final probability is taken from the bottom right cell of the probability table, since it can be assumed that this will be the only cell that has a non-zero probability. The maximum of each of the previous time steps is then multiplied together to give the correct sequence of probabilities for the entire duration of the recording. By taking the maximum, this then becomes more like the Viterbi algorithm than the Forward algorithm, but the identity matrix as the transition matrix still makes this implementation fairly unique. The probability of a given word-HMM generating a sequence of acoustic feature vectors (this time with a Gaussian probability function to calculate the observation probability) is given in Equation (4.3).

$$\alpha_t(j) = \max \alpha_{t-1}(i) a_{ij} b_j(o_t) \tag{4.3}$$

This equation states that the probability to be inserted into the trellis model at time ***t*** and at state ***j*** is the maximum probability from the previous cell, given as ***i***. Then the probability of moving from state ***i*** to ***j*** is factored in, alongside the observation probability of state ***j*** generating the observation symbol at time ***t***. Once the trellis has been filled, the probability of interest is then found in the final row and column.

4.1.4 Modelling Properties

It is instructive to see how the word modelling process is captured in a Gaussian implementation of a Hidden Markov Model. This illustrates the process behind which the probability derivation is calculated. This can be illustrated by plotting the Gaussians for various related sounds and comparing them against other words that contain a different sound at the same segment. Figure (4.1) shows a graph in which three words that start with the phones [tv] have their first word split (of four) plotted in red, against three other words that start with the phones [fj] plotted in green. In order to be able to plot this in two dimensions, only the first feature is used since this has the greatest inter-class discriminatory variance (see Section 3.2.2.2). The actual Gaussian distribution would be significantly more complicated and encompass the full feature

set, but to illustrate the process it is useful to show a simplified version of the way that different words with similar phonetic features are reasoned with in this model.

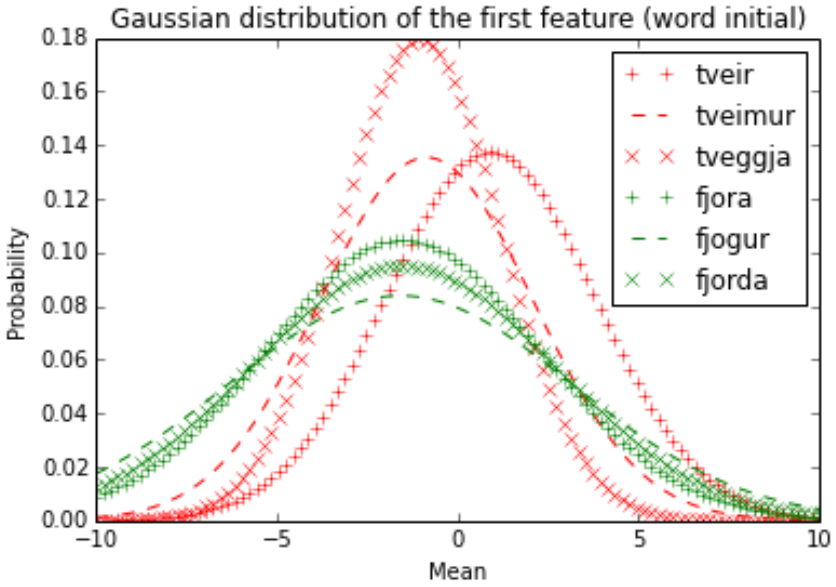

Figure 4.1. Univariate Gaussian distributions (first feature) for various recordings.

It is worth pointing out that while these words were chosen because of the phonetic sounds that they begin with, each word split does not necessarily correspond to a specific sequence of phones or specific speech sounds. The Gaussian distributions shown in Figure (4.4) very likely incorporate other data such as information in the sounds that immediately follow. It is for this reason it should not be expected that they should match perfectly because of the variable phonetic environment, but in the case of the words that start with [fj], they do closely match one another in this illustration.

What this illustration is intended to show is how similar words group together based on their phonetic quality and how systematically different phonetic sounds differentiate themselves with regard to the extracted features that characterise the speech signal, allowing the model to associate patterns with similar training data, providing an insight into how the HMM is implemented in the recognition task. Eigendecomposition was performed on the HMM word models in order to extract the first principal component, which entails a normalisation process, therefore the mean values are centred on zero in this illustration.

A noticeable point of observation is that all the [fj] words are relatively close together in this word split segment and they are all fairly similar in the phonetic quality of the initial segment. In the [tv] words, *tveimur* and *tveir* are relatively similar and are also followed by the same vowel, but there is a noticeable difference between these words and *tveggja*, which is the only word of this type in this chart that isn't followed by the diphthong [ei], showing that the initial segment likely does also incorporate features relating to the following vowel and this is likely the reason why this word stands out slightly among the others in Figure (4.1).

4.1.5 Results

The results of the HMM implementation are given in Figure (4.2) and Figure (4.3). Figure (4.2) shows the results for the MFCC feature set and MFCC feature set with deltas, across varying amount of word splits for the combined (training and test) data sets. This therefore models how well the HMM can model all aspects of the data it has access to, namely how it models the data it has been trained with alongside the test set. Figure (4.3) shows the same results but for the test set specifically, which is of more direct interest because it shows how well the model has learned to correctly identify novel data samples, showing its ability to generalise to new cases.

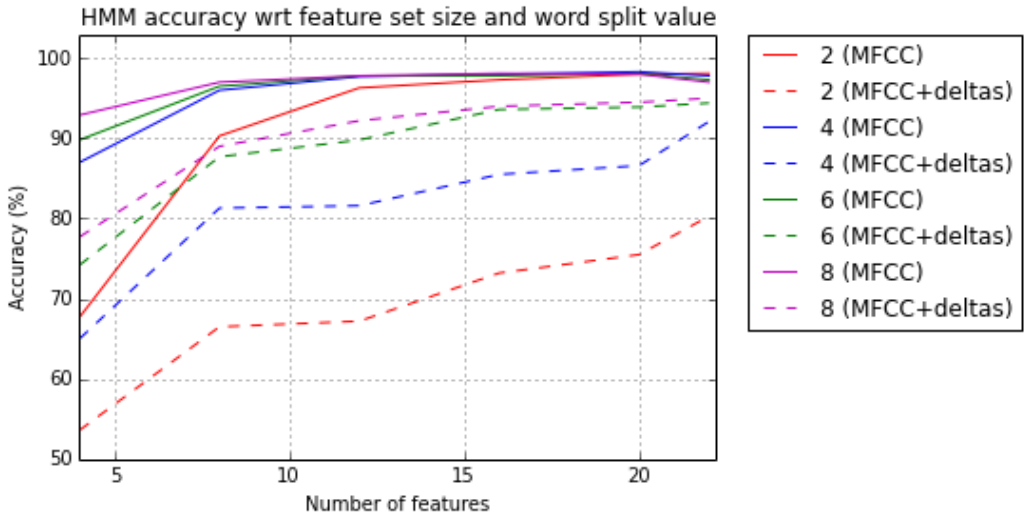

Figure 4.2. Accuracy on training set with varied word split value and feature set sizes

The pattern shown above reflects the fact that overall, the more features included in the feature set, the better the algorithm is at classifying the data samples correctly. However, these results are heavily biased towards data that have already been used in

the training process; a point which is important to bear in mind. As the word split value is increased, the better the model is at classification. When the word is split into 2 segments (shown in red), Figure (4.2) shows that many more features are required in comparison to the other values in order to achieve a similar level of accuracy, and on the MFCC set with deltas it also has the worst performance out of all other combinations of feature sets and word split values. The overall pattern is that the more segments the word is split into, the gradually better the algorithm performs in overall classification, particularly once the word is segmented into 4 or more parts. Figure (4.3) below shows the same setup as above but specifically in regard to the test set alone, which illustrates the algorithm's ability to generalise to new and unseen cases. This is the most interesting case from an experimental point of view as all learning mechanisms are expected to perform better on classification of the same data they were trained on.

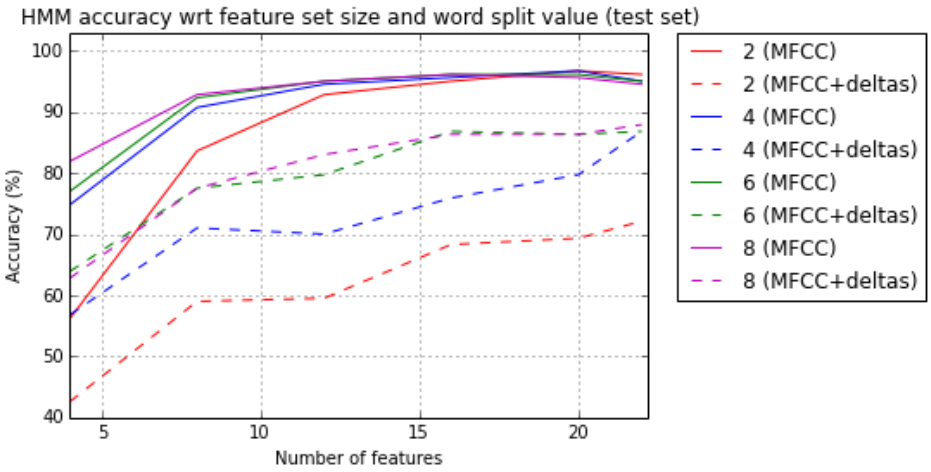

Figure 4.3. Accuracy on training set with varied word split value and feature set sizes

What is immediately clear is that there is a similar pattern that occurs regardless of the word split value, the amount of features used in the feature set follows a similar pattern. As in Figure (4.2) the larger the word split value, the better the accuracy, though this holds more for the MFCC feature set with deltas. With the MFCC feature set alone, the trajectories follow an almost uniform pattern when the top 12 features are included in the feature set. The highest accuracy achieved was with a feature set of 20 with the word split into 2 (red line) and 4 (blue line) after having removed two dimensions from the original feature set, showing how this process is

useful for removing noise and in effect allows a better representation of the data from which the classifier can build a better model. This can be seen in in Figure (4.3) where the accuracy on the MFCC set slowly starts to drop after the feature set contains more than 20 features. This pattern doesn't hold for MFCC and deltas, but the accuracy is markedly below that of the MFCC feature set alone.

In Figure (4.3) with a word split value of 4, the classifier scored 71% with 8 features but when a further 4 were added, the accuracy dropped to 70%, showing how it is not always the case that having more features available to learn from is beneficial to the classification process.

One result is missing from Figure (4.2) and that is how the HMM process dealt with the full 44 features in comparison to the VQ approach. Using this full feature set in a more advanced classification technique did not allow for a higher accuracy than half the feature set achieved, except with a word split value of 4 which scored only 3.2% higher than the corresponding value at half the feature set. All of the tested values for anything higher than 22 features in the MFCC and delta feature set never returned a higher classification accuracy than what had been retuned in Figure (4.3).

4.1.6 Discussion

The results show a dramatic improvement over what the VQ approach was able to successfully model. This is perhaps not surprising since this implementation of a HMM is more powerful and has the scope to be able to model more nuanced data as was described earlier. There are instances where fewer features leads to a better rate of classification accuracy, though the prominent pattern is also very visible that in many cases the more features, the better. Another interesting observation is that with regard to the amount of word splits, in the MFCC-only feature set, there is not much difference in the overall result. A much larger difference can be seen in the larger feature set that comprises the MFCCs and deltas. The difference between the lowest and highest accuracy for the MFCC feature set with 20 features is 0.3% on the overall data set and only 1.1% for the test set. This pattern can be seen elsewhere in the MFCC feature set and it is only when the number of features drops to 8 or fewer that a significantly wider gap in the lowest and the highest accuracy can be observed, i.e.

6.7% difference on the overall data set and 9.2% on the test set. This shows an ability to model varying word split values in a relatively stable way, which has the benefit of being applicable to both smaller vocabulary sizes but also scalable to situations where the word split value might need to be higher if longer compound words need to be recognised.

The minimal representation in the VQ approach, represented by only the top 4 principal components with a word split value of 2, gave an accuracy on the test set of 21.8% with the MFCC-only feature set and 30% with the MFCCs plus deltas. Considering that a totally random attempt to classify a word in the vocabulary is 1.6%, the VQ approach did show an ability to model at a much higher value than pure chance, but the MFCC-only feature set in the HMM implementation scored 56.2%, an even greater improvement upon the VQ approach, as is to be expected. Such a system is far too weak to ever be used in a real ASR implementation but as a research point, it is of interest to see how certain algorithms or modelling implementations can handle such extreme cases.

The highest accuracy scored in the VQ tests was 63.9% and the minimal representation in the HMM tests just mentioned was in fact the only score that scored lower than this. Every other score in Figure (4.3) performed better than the highest found in the entire VQ tests, demonstrating how careful consideration of the task at hand and implementational details allows for a much more efficient and successful classification algorithm.

The idea behind using delta features is such that the information can be captured as frames progress and for these additional features to be useful, the progression of analysis through the speech signal needs to be more detailed and not averaged over fairly low segments of a recording. An isolated word recognition task that had access to much more data could model the speech signal at a much higher granularity and that is where the true benefit of such features can be seen. With only seven recordings used as training data, the luxury of such a detailed analysis isn't present and is this is why averaging feature values over smaller segments was preferred over a more fine-grained and computationally expensive classification process. In other implementations considered in this project, it is likely that delta

values will be useful, for example when associated with specific artificial neurons, as will be addressed in the next section.

Hidden Markov models show a significant increase in the ability to model the single recordings and combined with a class-sensitive dimensionality reduction algorithm to further extract noisy features that don't help towards the recognition task, it can be shown that this methodology is very effective in predicting the correct class of 183 previously unseen recordings with 96.7% accuracy. It is likely that further improvements could be made with smarter feature extraction, more advanced noise reduction or even just different values of word splits and number of features used other than the ones tested here. The scope of the search space that considers all these possibilities is huge and therefore only a subset of those possibilities was explored in this section, guided by baseline scores of the VQ approach and some intuition of the problem that guided the implementation.

4.2 Artificial Neural Networks (ANNs)

In this sub-section I will begin by outlining what neural networks are and the fundamental concepts behind such a machine learning technique. I will then look at the history and development of this technique since its conception, after which I will move on to the modelling capabilities and typical problems ANNs have been used to solve. Having established the context behind what ANNs are and the rationale for their application in a diverse selection of problems, I will examine earlier attempts to model speech recognition problems with them and the accuracy that resulted. Finally, I will outline the intended implementation of the network and how it can be applied specifically to the Icelandic data set. This section will end with an account of the results and a brief discussion section.

4.2.1 Basic Outline of ANNs

A simple artificial neural network consists of a series of inputs that connect to a number of neurons, which could be only one in the simplest case, i.e. that of the perceptron (see 4.2.2). The inputs are all weighted and then summed up in the linked neuron, with the addition of a bias node in order to shift the decision boundary away from the origin, and then a specific function is applied to that summation of weighted

numerical input calculations in order to determine whether the neuron 'fires' or not. There are many different types of these activation functions used in different neural network implementations, but the most common one is the sigmoid function (Marsland, 2009) due to its lack of discontinuities which allows for the function to be differentiated at all points, an important factor required for the implementation of the learning mechanism known as gradient descent.

The weights that connect the inputs to the neuron(s) can be changed and in the learning stage of the neural network, these are changed iteratively. More advanced learning mechanisms were developed much later than Rosenblatt's perceptron that allowed for neural networks to learn non-linearly separable decision planes, namely via the Back Propagation Algorithm. Before this algorithm had been developed, neural network research came to a standstill after it was pointed out by Minksy & Papert that such models were unable to learn simple functions such as the XOR function (Graupe, 2007). The Back Propagation Algorithm (see Section 4.2.3) allows for the update of intermediate layers in the network, called hidden layers because the exact input that causes the output for later neurons in the network is not known due to the nonlinear activation function applied to the weighted input plus the bias node. This information is needed because, alongside a learning rate that guides the speed of learning, this is how error calculation is propagated through the network's weight values. A higher learning rate causes larger jumps but increases the speed of learning, yet too high a value can mean a level of precision that is desirable in many cases is also lost if weight values aren't allowed to shift more gradually.

A neural network model can have a various amount of output nodes and typically the output neuron that has the highest activation value is taken to be the correct class given the input, namely if a neural network model had five output neurons for a five-way classification problem, if the fifth node had the highest activation, this means that that the input has been assigned to the fifth class of the problem. In the simplest case, the firing or non-firing of the neuron signals the class distinction that is assigned to the input, either positive or negative in a binary classification problem.

A graphic illustration of a neural network is given in Figure (4.4). The small black dots represent the input information and the round circles are the neurons. Along the black arrows are lines which are represented as weights. The information feeds forward and is summed and passed to an activation function in the neurons and then the process is repeated at the output (rightmost) neuron, whose output represents the final classification decision.

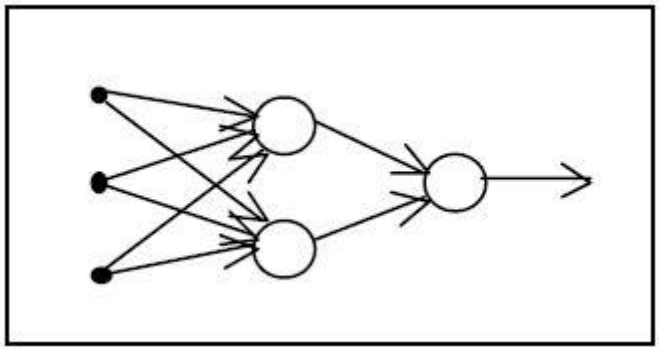

Figure 4.4. An example of a multilayer perceptron neural network (Samarasinghe, 2006)

More advanced neural network models are used in the real world today to tackle all sorts of various intricate problems. Recurrent neural networks include backward links between layers of neurons instead of a strict feed-forward neural network, which has been the most common way to conceptualise neural networks since their inception (Marsland, 2009). This allows for echoed data to take diverse paths through the network a potentially unlimited amount of times and is suited for many complex problems which are beyond the scope of the problem specification at hand and will not be discussed further. Instead, I will focus on the historical path that neural networks have taken and this will provide a context that supports a rationale for why this machine learning technique is appropriate for an isolated word recognition task.

4.2.2 Historical Perspective

The study of ANNs arose out of a desire to model learning in computers in a similar basic manner to the then-contemporaneous understanding of how learning takes place in the human brain (Mitchell, 1997). The human brain learns via a system of strengthened and weakened synaptic connections between neurons (Russell & Norvig,

2010) and via a chemically-driven process called long-term potentiation, neurons that are commonly activated at the same time form a closer bond and are therefore more likely to cause one another to fire in a dependent relationship in the future. This behaviour is summed up in the popular adage in neuroscience by Donald Hebb, “Cells that fire together, wire together,” and is eponymously known as Hebbian learning (Graupe, 2007). The notion of strengthening neuronal connections plays a critical role in how ANNs function, namely that learning via weight adaption means if one neuron is likely to fire, then it has a higher chance of causing adjacent neurons to fire via a higher weighted connection from one neuron to another.

The first algorithmically described neural network was proposed by Franklin Rosenblatt in 1958 (Haykin, 2008), a psychologist who had taken Hebb’s principle of self-organised learning and found an implementation that expanded upon the pioneering work of McCulloch & Pits in the previous decade (ibid.) This new model was called a perceptron and consisted of a single neuron that could correctly classify two hypotheses provided they are linearly separable (Marsland, 2009). As mentioned in Section 4.2.1., a controversy emerged after a review by Minsky & Papert pointed out the inability of the then-current learning mechanism to model the XOR function, causing research into neural networks to come to a standstill until the discovery of the Back Propagation Algorithm in the 1980s, brought to the attention of the AI community by Rumelhart & McClelland at the Parallel Distributed Processing Research Group (Russell & Norvig, 2010).

The revival of neural network research is credited to the discovery of this algorithm (Marsland, 2009) and the mechanism with which it allows a neural network to learn is an intricate process that evaded many researchers who tried to solve the learning problem after Minksy & Papert’s destructive editorial. The following section looks in detail at the process and highlights some of the considerations that need to be taken into account when modelling networks to learn classification of acoustically-derived data.

4.2.3 Learning in ANNs

The discussion of learning in ANNs is split into two further sub-sections. The first will outline some general comments and comparisons with the method discussed in the previous section, addressing some important considerations that need to be taken into account in the modelling stage, linking to previously defined algorithms to help in the correct pre-processing of the training data. The second section will address the details of the learning algorithm itself, the Back Propagation Algorithm.

4.2.3.1 Overview

In the previous section, which concentrated on the HMM approach, the learning phase was focused on deriving the correct mean vectors and covariance matrices in order to evaluate new acoustic data via a Gaussian probability density function, namely the multivariate Gaussian, when a new recording was being queried. Learning in ANNs is a markedly different process, in which the adjustment of weights alone encompasses the entire learning phase. ANN learning is an iterative process that requires many training cycles, called epochs, in order to slowly change the values of the weights in the network based on training data that explicitly alerts the network as to whether the correct classification has been made or not, i.e. supervised learning. The training phase can be quite lengthy and is often considered as one of the unattractive features of ANNs, but after a model has been trained successfully, the efficiency of the classification phase more than makes up for the lengthy training phase (Marsland, 2009).

Given that many implementations where ANNs have been applied are used in real-time, the training vs testing trade-off is tolerated because a model can be pre-trained and once it has learned a model of the data, the quicker implementation is the more desirable feature. In terms of the problem at hand, in the previous section using HMMs, each of the 61 HMM word models is queried and the highest result is returned; in an ANN model, there would only be one network, with 61 output nodes that represent every possible word and the queried input would only be filtered through the system once, through a trivial amount of hidden layers (one in this case) in order to arrive at an output. The computations required are markedly reduced from modelling

individual words as HMMs and returning the highest scoring result. However, the interference that the training data can have is also potentially problematic.

If in the training data there exists a counter-intuitive pattern, namely one that alters a decision boundary in order to correctly classify a training instance, but another similar pattern belongs to a different class, this implies a shared region of the sample space that multiple classes occupy and this can be problematic and can prevent a neural network from learning to correctly classify training instances of that class. In order to aim to counteract such a problem, given that the problem specification at hand relies on correct classification of phonetically similar vocabulary items, a pre-processing step needs to be implemented in order to remove the noise in the data so that the decision boundaries that represent the different vocabulary items are as linearly separable as possible. By applying a form of dimensionality reduction that takes into account class labels, namely Multiple Discriminant Analysis, which was presented in Section 3.2.2., it is possible to prepare the data in order that only the components that describe the highest magnitude of the derived eigenvectors are present in the training data. This step is more of a necessity than it was with Hidden Markov Models, though applying it did see a dramatic increase in classification accuracy. The rationale for applying it as a pre-processing step to neural networks is more to allow better decision boundaries to be learned during the application of the Back Propagation Algorithm.

4.2.3.2 The Back Propagation Algorithm.

The algorithm itself is based on the principle that given a set of inputs and weighted connections, a linear combination of these is supplied to an activation function and this information progresses through the network. The most common activation function is the sigmoid function, defined in Equation (4.4), followed by a graphical representation:

$$a = g(h) = \frac{1}{1+\exp(-\beta h)} \tag{4.4}$$

A graphical representation of the sigmoid function is given in Figure (4.5).

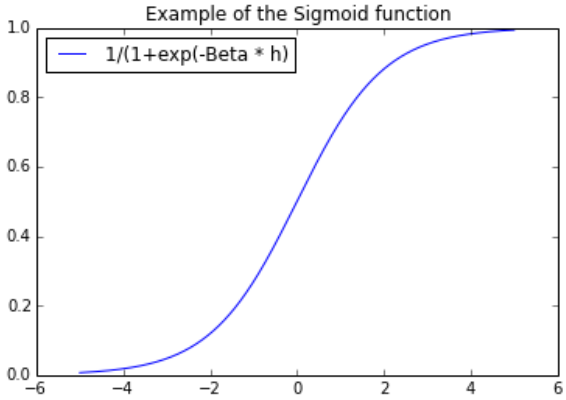

Figure 4.5. The sigmoid activation function.

The reason for using the sigmoid activation function, instead of a step function that makes a sharp decision based on the input value, is because the sigmoid function can be differentiated at all points along the line and given an error that is propagated backwards through a network, this means that the values can be calculated that would change the weight values so as to minimise the total error in the system.

In order to calculate a meaningful error at the output node, a specific error function needs to be defined. This can be done in many ways yet it still needs to apply the concept of blame to the input of the previous layer, i.e. if the correct answer is achieved, there should not be any change, but if an incorrect output is calculated at an output neuron, then that signals where the weight values need to be changed through a backwards passage through the network.

In calculating the error, it is important that opposing signs on error values do not cancel each other out. For example, if one neuron makes a mistake of a positive amount (because the sigmoid activation represents a sloped curve now) and another neuron generates an error of the same magnitude but with the opposite sign, then the error calculating at the output would cancel each other out. This means that two errors would actually be seen as the network behaving completely as desired, which is not desirable when trying to iteratively update the weight values in the network. In order to avoid this, a common error function is the sum-of-squares approach, which makes all errors have the same sign and avoids any chance that opposing values could cancel each other out. For reasons to do with ease of differentiability the function is often multiplied by a half. The sum-of-squares error function is given in Equation (4.5).

$$E(t,y) = \frac{1}{2} \sum_{k=1}^{n} (t_k - y_k)^2 \quad (4.5)$$

The main complexity of the Back Propagation Algorithm is that at the output, where the targets are known, the inputs are not accessible. Conversely, in the hidden layers when the input is known (i.e. in the first hidden layer) the targets aren't derivable because the information still needs to be fed forward afterwards.

The algorithm begins by feeding forward the input data through the network, with the input values to a neuron being multiplied by the weights of those connections, plus the bias node, which are then summed and evaluated according to the activation function, which in this case is the sigmoid function described above. Once the hidden layers have activation values this process continues through all the hidden layers until the following layer is the output layer, from which the error can be calculated directly with the respect to the target.

From the output layer, the sum-of-squares error is then calculated and the delta matrices containing the derivatives are then calculated. These matrices are then weighted by the learning parameter and then multiplied by the previous layer's activations, whether it be the activations in the hidden layer or the direct inputs from the input nodes. The weights are then updated by subtracting the value of the delta matrix for the corresponding weight values until the specified number of training epochs has been reached.

4.2.4 Modelling Properties of ANNs

Given that perceptrons were show to be inefficient before the development of the Back Propagation Algorithm, the tasks they were applied to were fairly exploratory and theoretical at the start, but once a way was found to propagate errors and make relevant weight adjustments through the entire network, researchers started putting ANNs to the test. I will first address the more technical aspects of decision boundary modelling in this section, concluding with some applications to real-world problems and the requirement in terms of hidden layers required for successful modelling.

4.2.4.1 Decision-Boundary Modelling

As was pointed out before, the issue of the types of decision boundaries that ANNs have been able to model has been a prominent feature in its history, with the first perceptron model being restricted to a linearly separable problem because the most complex decision boundary it could model was a straight line. Even though hidden layers were added into the basic perceptron model in later stages, the issue was then finding a way to train them adequately, with numerous attempts that were later unsuccessful. It was only when the Back Propagation Algorithm was devised that the modelling of the decision boundaries became a serious avenue of research in the academic study of neural networks.

Figure (4.6) shows a table taken from Lippmann (1989), who outlined in graphical form a basic example of the types of decision boundaries that could be modelled with ANNs depending on the amount of hidden layers.

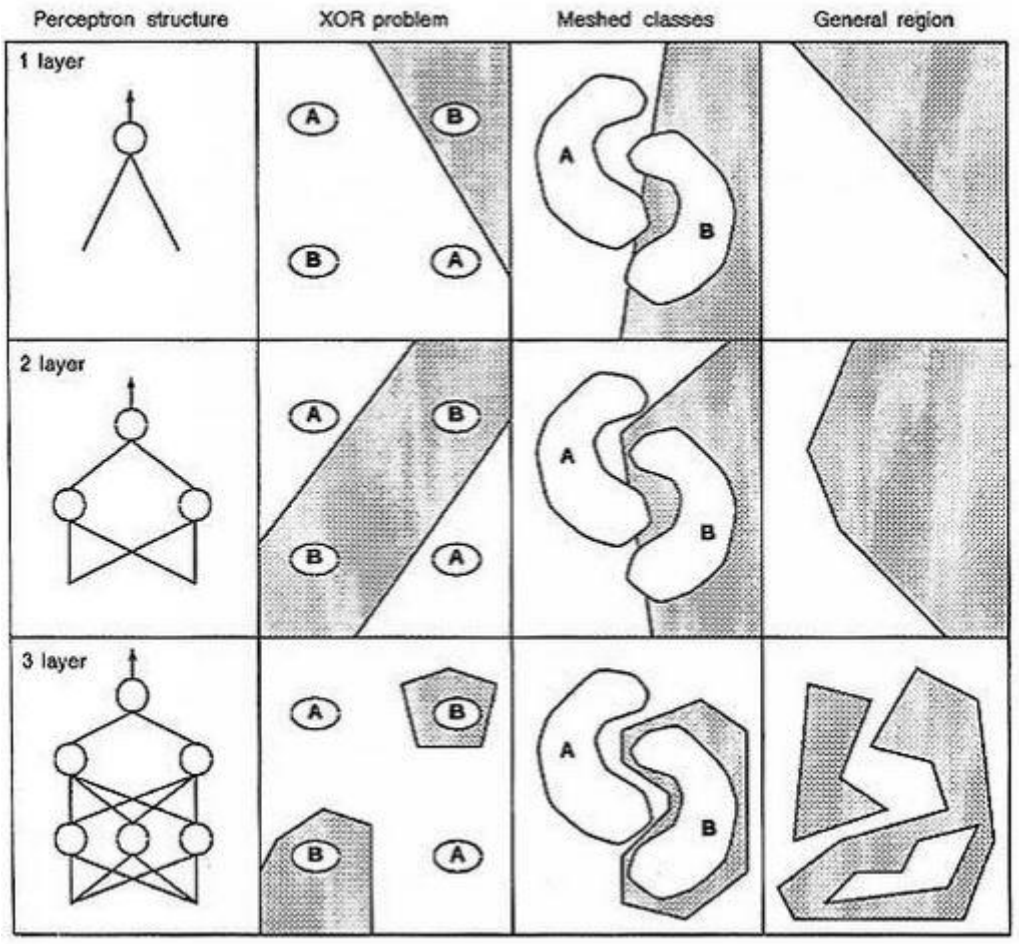

Figure 4.6. Decision boundaries modelled by ANNs with various hidden layers.

As indicated in the top example, which represents a one-layered perceptron, only a linear decision boundary is possible, representing the situation up until the discovery of back-propagation. The image shows the inability of the network to separate the two classes A and B.

With a hidden layer, the capabilities of the network to model more complicated decision boundaries becomes apparent, though it is still not perfect in the case of the meshed classes. With two hidden layers, the ability to model arbitrary classes that are separable can be seen. It is possible to include more than two hidden layers, though keeping track of the update stages is quite a tricky task, but one should never need more than two layers given the correct parameters and amount of hidden nodes in each layer (Marsland, 2009).

An unsolved issue regarding the amount of hidden nodes in a hidden layer is precisely the amount of hidden nodes to include given a specific data problem. There are a few loose heuristics in the literature but all come with the compulsory caveat that it is not a wise decision to not expand beyond such heuristics in order to see how the number of hidden nodes affects the accuracy of the problem.

4.2.4.2 Other Applications of ANNs

ANNs have been applied to various tasks and have attained remarkable success in their ability to capture and classify patterns based on only limited information. In order to give an impression of the type of tasks that ANNs excel at, it will be illustrative to consider two famous successful applications in this section. The first demonstrates an ANN's ability to learn facial characteristics from images and then extrapolate that data to perform generalisations with varying success. The other demonstrates how ANNs were used to program a vehicle to drive on roads.

The first task was that of facial recognition by Garrison Cottrell, who took 64 photos of different faces that demonstrated different facial expressions. In total there were 11 different individuals who made up the collection of 64 photos. 13 photos represented non-faces, i.e. a watch, bulletin board and other random objects. Cottrell used a neural network with only 3 layers (two hidden layers and an output layer) and the network was able to determine whether or not a photograph was of a face or not

with 100% accuracy on the entire test set (Cottrell & Fleming, 1990). Furthermore, the network learned to determine the correct gender of the person in unseen photographs with an accuracy of up to 81%. The features used in this were all determined algorithmically and not pre-set, showing an ability to perform unsupervised feature extraction in a meaningful way.

In the driving task, a camera input was programmed to extract features from a representation of the road in front of the vehicle and specific turns that the vehicle performed while at a specific computational configuration of a road in front were used to train an ANN. The computer built this information while observing a human driver and only required 5 minutes of observation time in order to learn well enough to steer on roads at speeds of 90 mph over a 70 mile distance on public roads (Mitchell, 1997).

The aim in this section has been to demonstrate the kinds of tasks which have been used in conjunction with neural networks, the minimal data they often need in order to establish a fairly complex model. While the learning mechanisms and properties of decision boundaries have been studied extensively, as was briefly outlined in the previous section, it is often not intuitive how a series of weight changes results in the representation of knowledge. ANNs are often described as 'black box' systems because unlike other machine learning algorithms that aim to provide insight, such as decision trees, weight values do not inherently correspond to any intuitive learned behaviour easily conceptualised by the human mind. In that sense they do exhibit a mysterious link to neurons in that how neurons in the brain represent knowledge via weighted synaptic connections is also a mystery to modern neuroscience. It will now be instructive to consider previous attempts to apply neural networks to the speech modelling problem, or to speech recognition in general.

4.2.5 Application to Speech Recognition

An early attempt at modelling isolated words with neural networks was by Richard Lippmann in 1989. The goal of his task was specifically to compare the results obtained with neural networks in comparison to previously dominant machine learning algorithms such as clustering methods like the K-means algorithm, K-nearest neighbours and Gaussian classifiers and found that a three-layered neural network

outperformed all other algorithms used in that task, though in some cases only marginally (Lippmann, 1989). In the isolated word recognition task, the speech signal was framed in a similar manner to the process described in Section 2. From there, the frame with the greatest magnitude of energy was taken, and then from that indexed point, a slightly earlier frame was also taken to provide additional information on the minimal movement that could be recorded between two points in a signal. The features were then extracted from the signal and then supplied to a neural network. The words from the vocabulary were only the first seven cardinal numbers of English, so there was also a 1 in 7 chance that the algorithm could pick the correct word by accident, but it did still show a slightly better performance in comparison to other modelling techniques in his experiments (ibid.)

In another application of neural networks to speech recognition, or rather vowel classification in single monosyllabic words, Lippmann extracted the first two formants of a signal and plotted them and in a similar way to the other task, measured the effectiveness of other machine learning techniques in order to see which ones performed best. An example of the decision boundaries learned during the process is given below in Figure (4.7).

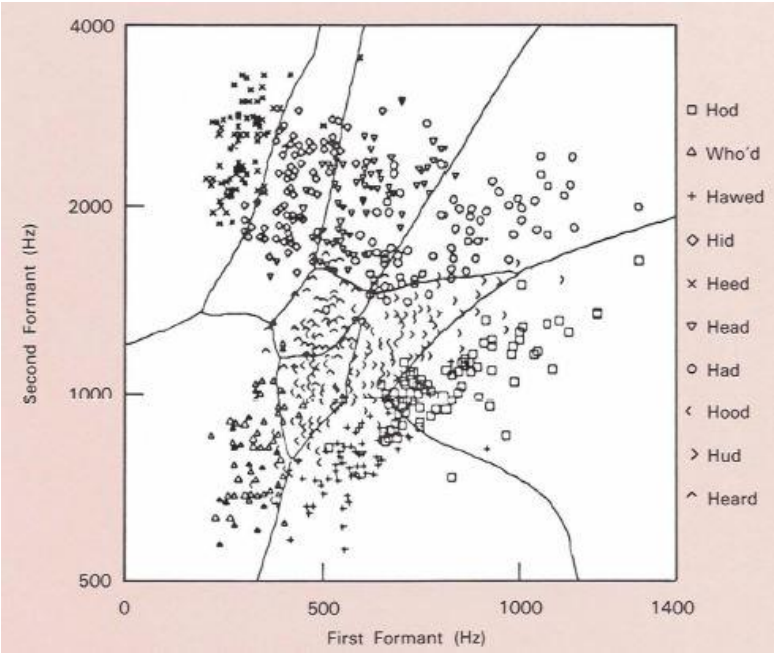

Figure 4.7. The decision boundaries learned in a vowel classification task (Lippmann, 1989).

In this experiment, the neural network came second in terms of performance. The only algorithm that performed (fractionally) better was K-nearest neighbours, which is hardly a mechanism with which any real learning takes place; it is rather a measured attempt at picking the closest 'neighbouring' point(s) and assuming the same majority classification label as its own.

What these experiments show is that, when the context is in relation to isolated words and the goal is to extract the spectral properties in order to determine an accurate classification of a new query instance, neural networks can actually perform quite well and there is a rationale for considering such a machine learning algorithm to tackle the isolated word recognition problem with such a tool.

4.2.6 Implementation

The neural network design is loosely based on programming code given in Marsland (2009). The network has one hidden layer, with an optional amount of inputs or outputs. The amount of these nodes is calculated from the training data that is supplied to the network upon initialisation. The input data is drawn from the same data as was used in the Hidden Markov Model tests, i.e. 61 vocabulary items all split according to the word split value given to the program, and then each feature dimension is averaged over the amount of segments the word is split into.

The feature set used was the MFCC set consisting of 22 features. The reason for using this is because the underlying mechanism of modelling adequate decision boundaries relies on using features that maximise the variance of the class-separated training data. As was pointed out in Section 3.2.2.2, the MFCC set had a very high coverage of the total variance of the training data within the first 5 features only, which increases the chance of removing any possible noise in the data that would prevent the weight change update rule from converging to a steady set of weights that would learn to correctly classify the training data, which would then give the best chance of having created a neural network that could generalise to the test set.

After generating the training data from all the vocabulary items, a new matrix was built so that the Back Propagation Algorithm could be trained effectively in what is known as batch update, which means the linear algebra modules in NumPy could

efficiently calculate the training set as one matrix rather than implementing an iterative loop over every training sample. The length of each row (data sample) was the word split value multiplied by the reduced feature dimensionality, so if a word was to be segmented into 4 sections, and the reduced feature set was set to be 10, then each row would contain 40 features, 10 from each of the word segments according to the word split value.

After a few test runs with various parameters, including the maximum number of training epochs, it appeared that after 2000 epochs, the results had virtually converged and didn't change a lot after this point so 2000 was set to be the maximum amount of cycles before the training process stopped. The other main parameters that could be varied in the network are the beta value inside the sigmoid function, which was set to be 1 in this setup, the momentum parameter, which was set at 0.9. These parameters were not altered in the following tests. The learning rate and the amount of hidden nodes in the hidden layer were altered and the effects they had on the learning process are described in the following section.

4.2.7 Results

There are many possible combinations of parameters such as a varying amount of features in the feature set, the number of hidden nodes in the hidden layer and the values relating to the learning parameter. The first results displayed below therefore represent a feature set consisting of 12 MFCC values, namely because this was the closest number to half of the entire feature set, namely 11 features, that has a corresponding result with the previous HMM scores, i.e. no feature set of precisely 11 features was tested in the previous section, so 12 was selected as an initial value to aid in comparing the two algorithms. The word split value was set to be 4, as this intuitively seems to allow a reasonable amount of variation within the speech signal to model the individual vocabulary items. The number of hidden nodes in the hidden layer was selected to be 25, a number picked at random for lack of any indication of a better value to pick. With these values selected, it was then possible to vary the learning parameter value in order to see how this affected the classification accuracy, which is given in Figure (4.8) below. The accuracy on the training set is given as a continuous (non-dashed) line and the corresponding colour's dashed line represents

the classification accuracy of the test set. Due to the random initialisation of weights, different runs can give significantly different results so therefore all results reported in this section are an average of two in order to reduce the likelihood that such a random initial setting of the weight parameters was specifically optimal (or suboptimal).

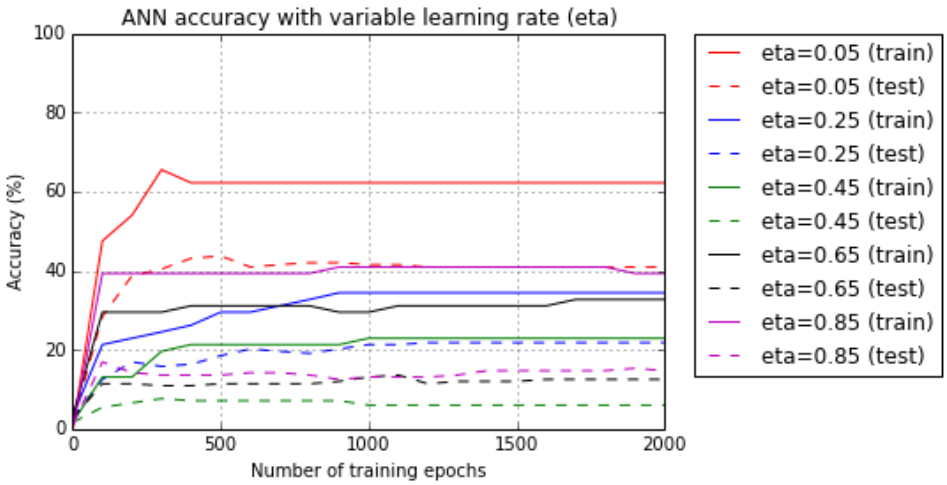

Figure 4.8. How changing the learning rate affects the network's classification accuracy.

Given that the value 0.05 as the learning rate (eta) returned the best result, both on the training and test set compared to all other values tested, this was then used throughout the trials that follow. In order to see the effect of the number of hidden nodes on the classification accuracy, this was then examined. Figure (4.9) shows the results after 2000 training epochs.

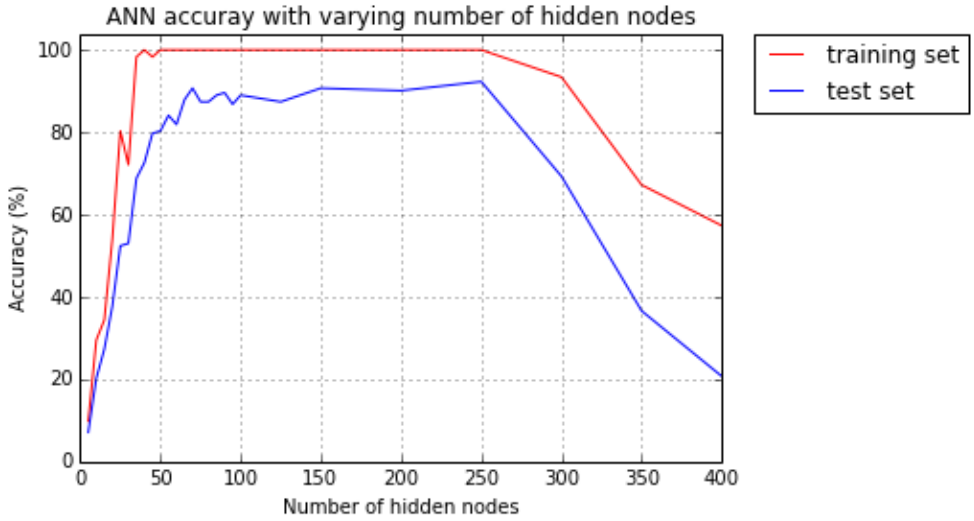

Figure 4.9. How changing the number of hidden nodes affects the network's classification accuracy.

These results indicate that the ANN performs better with a larger amount of hidden nodes, but after 250 the network is too broad and classification performance decreases sharply. With such results, it is also important to consider the effect that the word split value and size of the feature set has on the results. Selecting the amount of hidden nodes to be 250, which resulted in the highest accuracy on the test set (92.3%) in the previous graph, Figure (4.10) indicates how the varying amount of segments the word is split up into affects the accuracy of the overall classification, using the values of the results that performed best in the previous two graphs, namely 250 hidden nodes and a learning rate of 0.05.

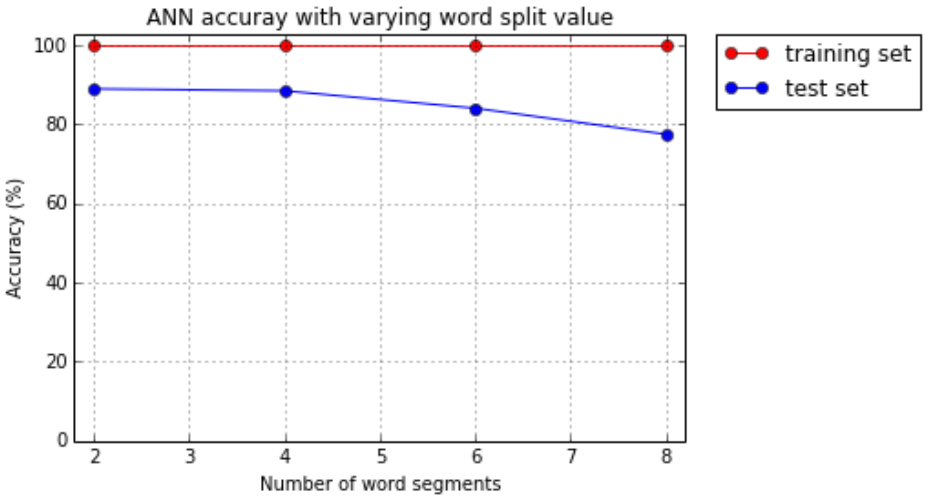

Figure 4.10. How changing the word split value affects classification accuracy.

Given that the best result was achieved with a word split value of either 2 or 4 in the test data, the last variable parameter that needed to be examined is that of the amount of features present in the feature set. Figure (4.11) displays how altering the amount of features present in the feature set affects the overall classification ability of the neural network.

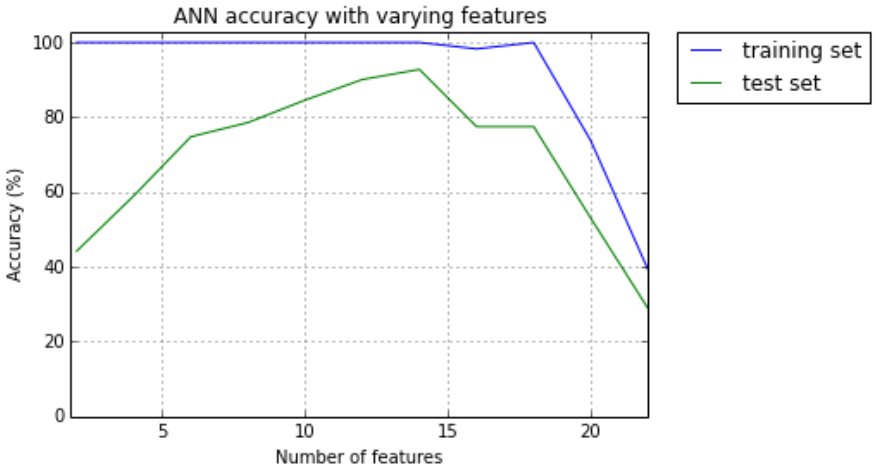

Figure 4.11. How changing the number of features affects the network's classification accuracy.

The best result on both the training and test sets occurs when a feature set of 14 is used. The combination of all the previous parameter values, namely 0.05 learning rate, 250 hidden nodes, word split value of 4 and 14 features in the feature set give the best result, namely 100% on the training set and 92.8% on the test set. A more in-depth interpretation of these results and of the network as a whole is given in the next section.

4.2.8 Discussion

As was briefly noted earlier, artificial neural networks are often described as a form of black box system (Marsland, 2009) due to the fact that it is not intuitive exactly how graded weight changes can represent knowledge of any substantial kind. In light of this observation, it is also possible to pair this with the observation of the numerous parameters of a network model, whose effects have been widely studied, but in terms of gaining a more in-depth understanding, do not help us gain a more intuitive understanding of how a neural network can describe what it has learned in any non-mathematical way.

There were many such parameters that were not altered during the experimental phase that the previous section describes. It is not yet known how the performance might have changed with a different β parameter in the sigmoid activation function or whether a totally different activation function, i.e. the hyperbolic tangent, might have performed better or worse for this problem. It could also be the

case that another hidden layer in the network might have been able to classify the entire test set correctly, but implementing such a network proved too complicated a setup to contend with, adding a lot more variable parameters to the problem. However, this is an avenue that could be promising in a future adaptation of this problem.

As shown in Figure (4.8), the learning rate performed significantly better with a smaller value. From this it can be inferred that a decision boundary requires slight adjustments that allows for more careful modelling compared to higher values that effect larger jumps of weight values that are potentially too coarse for the problem at hand. An expected result, knowing this, would be to see a steady drop in accuracy as the learning rate grew larger and larger, but it was interesting to see that this pattern does not hold in the graph. For example, when the learning rate was set to be 0.45, this performed significantly worse than a higher value, 0.85, which scored surprisingly high on the training set. The two lowest values of the learning rate do follow such a pattern but it is likely that interference from the random weight initialisation had a role to play in these results.

The number of hidden nodes appears to play a very important role in helping the network to correctly classify the test data. The accuracy plateaus at around 65 hidden nodes and stays relatively stable until 250 nodes. What is interesting is that once the number of hidden nodes increases beyond this number, there is a sharp decline in the accuracy. Too complicated a system for a given problem can also be as problematic as a system that is too simple in its design.

With regard to the word split value, I originally expected that an increase in the value would have led to a sharper decline as the word segments were split up into more than 4 separate parts. After this amount, there becomes less training data to consistently and adequately represent the acoustic characteristics of a specific word segment, yet this was never a problem for the training set, which was consistently modelled with 100% accuracy. The accuracy on the test set held at a fairly high level and only starts to drop off after a split into more than 6 word segments was made.

In Figure (4.11) of the previous section, the effect of the size of the feature set is considered. Having used the previous results to guide the next variable parameter, it was possible to achieve 92.8% on the test set. While it is always a consequence that a model trained on a specific set of data will perform better than compared to unseen data, it is still remarkable that with a feature set that comprised the top two features (i.e. those that maximised the class-separable variance) could still model the training set with 100% accuracy (as seen in the blue line). A small dip can be seen when 16 features were used and soon after that accuracy on both sets plummets. The slow climb of the test set is an expected consequence of the increasing use of more features to model each word in the vocabulary.

An important point to note is that, as was pointed out in Section 3 and in Section 4.3, dimensionality reduction is an important pre-processing step in order to remove noise in the data when using ANNs. Without applying such pre-processing to the data, the exact same setup that resulted in quite an impressive accuracy would have returned only 39.3% accuracy on the training set and 28.9% on the test set. By removing 8 of least useful features from this initial feature set, an accuracy of 100% and 92.8% on the training and test set respectively, can be achieved. Using only the top 2 features gives a better classification accuracy than utilising the 22 original MFCC features, as would have been the case without using MDA. This underlines the importance of applying a form of noise removal via dimensionality reduction in a machine learning algorithm like artificial neural networks. It further highlights the issue that what could seem like a poor result is only one different parameter value from returning a very impressive classification accuracy.

This underlines the point that it cannot be said that the parameter values used in this section were the most optimal. It is certainly conceivable, beyond the other possible changes that could be made with an extra hidden layer or a different activation function, perhaps a higher learning rate with more features and fewer hidden nodes would have resulted in an accuracy as high as was reported here, if not higher. However, in light of the HMM results, it appears to be a reasonable assumption that the parameter values here are close to optimal, given that the ANN modelled the training set perfectly, while the HMM model only reached 98.3% in this regard, though

it should be pointed out that in the HMM results, there was never a direct result for the training set alone, only a combination of the training and test sets as one. The discrepancy between the highest results recorded on the test sets in both models is only 3.9% (i.e. 96.7% in the HMM and 92.8% in the ANN). Given the markedly different ways both learning algorithms model the training data, this points to the fact that if these parameter settings were not optimal, they are likely very close to the optimal result that could be expected. A more detailed analysis would shed some extra light on this and is a possible avenue of future work.

5 Conclusion

The goal of this paper was to investigate the capacity of various machine learning algorithms to model a non-trivial amount of words from a vocabulary set, namely 61 items relating to a vocabulary that could be used for a speech-driven implementation of a basic calculator program. Inherent in such a vocabulary is the issue of many different words that are phonetically similar, which makes an isolated word recognition system considerably harder to model if no other linguistic clues are given which could help to constrain the selection of a particular word in a particular context.

In order to evaluate the more advanced machine learning algorithms, it was important to find a baseline result from a simpler and more coarse modelling technique so that more advanced implementations could be compared with something that is already understood to have certain limitations. Vector quantization was used in creating a baseline, due to the fact that many assumptions have to be made about the data that are not necessarily true in modelling isolated words via their acoustic features. After this, the issue of dimensionality reduction was performed on the initial baseline results in order to illustrate how the process works and to demonstrate the effects on the classification of data set. The baseline results performed fairly well when tested on a combination of the training and test set, but never achieved an accuracy above the mid-80% range. The performance on the test set was considerably worse, but did illustrate that some aspects of the words could be modelled and this was dependent on the feature set used in the problem among other considerations such as codebook size and the amount of segments the query signal was split up into, as was illustrated in the section on dimensionality reduction.

Hidden Markov models were then briefly outlined and a short theoretical extension was outlined from the vector quantization approach, namely in the ability to treat the input data in a continuous fashion (as opposed to the discrete manner that VQ mandates) that also took into account the variance across the dimensions of the feature set via the multivariate Gaussian probability density function. The calculation to derive the final probability was also extended in order to allow a more natural progression through the trellis model by incorporating the identity matrix.

Overall, the predictive capacity of this approach on the test set performed quite well, with the highest accuracy achieved being 96.7% on 183 recordings that were never before seen by the model. While this is not a perfect result, it is promising given that the algorithms dealt with in Section 4 are still fairly basic compared to state of the art adaptions used nowadays. There has been a lot of work in extending such machine learning concepts and this could in turn be applied to the isolated word recognition problem to likely perfect the classification process for this data set.

The next machine learning algorithm that was discussed was that of an artificial neural network, a technique that is not typically used in its simplest implementation to model time-varying data, but for small acoustic signals the modelling problem can be conceptualised simply as a division of sections and therefore it can be treated as a simple pattern recognition problem composed of multiple parts. After detailing the many variable parameters that can affect the performance of an ANN, it was shown that even a completely different modelling technique to HMMs, which relies on many conceptual foundations like linear separability in higher dimensions, this problem can be effectively modelled to a degree where over 92.8% accuracy can be scored on the test set.

It therefore seems as if the HMM approach is more apt at modelling such data, though it cannot be ruled out that a different set of parameter values in the ANN approach might have given even better results than the HMM implementation. If accuracy is key, then it appears that modelling with a Gaussian PDF should be preferred if a choice of exactly these two implementations was under consideration. It should also be noted, however, that the HMM approach implemented here treated every individual vocabulary item as a separate HMM and each time a new query was made to the model, this involved applying a calculation algorithm to each word in the vocabulary and returning the one with the highest score. As the number of vocabulary items increases, this cost of calculation is bound to become more and more noticeable and might not scale as well as the ANN approach.

In the ANN approach, though the final accuracy reported is slightly lower on the test set, the calculation cost is miniscule compared to how it is calculated for the HMM approach. In the ANN, there is one hidden layer and once the acoustic features are

extracted from the audio signal (a process which has to happen in both implementations), arriving at a prediction consists of only feeding forward the query through the network once, i.e. two matrix calculations with the weights and a selection of the highest value in the output layer. Given that an extra hidden layer can increase the modelling capabilities of the data by reducing the demands of a certain level of linear separability, the forward calculation through a neural network would only fractionally increase the time taken to query the network. If an extra hidden layer can increase the accuracy on the test set and all that is required is one extra linear algebra matrix multiplication, then the opportunity for scaling up the amount of words that the model can handle is much greater than that of the HMMs, whose calculation cost increases linearly in the amount of vocabulary items that need to be checked.

It is in this regard that I would propose that ANNs are likely to be the best avenue for future investigation, given that there are many theoretical reasons to expect that performance can be increased significantly with the addition of another hidden layer in the network, with the understanding that the cost to query a network does not increase linearly in the amount of vocabulary items and that training is often done offline so, although it is more computationally expensive to train, the performance cost would be worth it.

If the implementation should be able to handle the addition of a new word without a lengthy training process, then the HMM implementation would clearly be more advantageous given that the training recordings can be supplied, a HMM of the new word can be generated and if the means and variances of the processed query signal generated without changing anything in the entire system. This would not be possible in the ANN approach because the whole network would need to be trained to learn the correct weights to be able to identify the new word in the context of all the other words in the vocabulary.

If an implementation had variable needs in different situations then an ensemble approach could be implemented, where both classifiers vote on what they believe is the correct classification for a query sample and a result can be given if they both agree. If there is a discrepancy then the HMM result could be given, due to the fact that it performed better on the test set, but if quick classification was an issue, the

calculation could be performed by the ANN alone, yet if a new vocabulary item needed to be added but there wasn't enough time to retrain the ANN, then again the system could back off to the prediction of the HMM.

This paper aimed to investigate how fairly low-level implementations of simpler and more advanced machine learning algorithms could be used to model the recognition of isolated words in Icelandic. The particular set of vocabulary items is itself relatively homogenous given that many words are different gender or case forms of the same numerical lemma. At the outset, it was not known at all how these machine learning implementations would be able to handle such a dataset or if it would even be possible to model the training set at all given only 7 small recordings from which to learn a model of each word. Through careful consideration and understanding of the algorithms involved and the properties of the data set and the feature extraction process, reasonable hypotheses were tested and I believe it is clear that the approach was effective in determining what is possible given restrictions on the amount of training data. Beyond demonstrating that an accuracy of over 92% is possible given a vocabulary set of 61 words, I believe that a potential avenue of further development has been opened up not only in the development of a small isolated word recognition system, but in techniques implemented at a later date that could be used and compared with the results presented in this paper, i.e. more specialised learning algorithms that perform even better.

An interesting proposal for future work would be to examine how these algorithms perform on data from multiple speakers, examining to what extent the learned features were speaker-dependent. Interesting questions arise such as the issue of how much data is required to demonstrate a similar accuracy for a multi-speaker system, what the maximum amount of vocabulary items is before the learning mechanism start to saturate, and how changing the parameter values alongside a more varied data set handles these extra pressures. These are very exciting questions and ones I hope to be able to address in future work.

References

Barber, D. (2012). *Bayesian Reasoning and Machine Learning*. Cambridge: Cambridge University Press.

Bishop C. (2009). Pattern Recognition and Machine Learning. Berlin, Springer.

Cottrell, G., & Fleming, M. (1990). Face recognition using unsupervised feature extraction. Paris, France: Kluwer Academic Publishers, 322-325.

Davis, S. B., & Mermelstein, P. (1980). *Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences*. In IEEE Trans Acoust, Speech, Signal Processing 28(4), 357-66.

Graupe, D. (2007). *Principles of Artificial Neural Networks*. Chicago, CA: World Scientific Publishing.

Guðnason, J., Kjartansson, O., Jóhannsson, J., Carstendóttir, E., Vilhjálmsson, H. H., Loftsson, H., & Helgadóttir, S. (2012). ALMANNARÓMUR: AN OPEN ICELANDIC SPEECH CORPUS. In *Spoken Language Technologies for Under-Resourced Languages*.

Haykin, S. O. (2008). *Neural Networks and Learning Machines*. 3rd Edition. New Jersey, NJ: Prentice-Hall.

Jurafsky, D. & Martin, J. H. (2009). Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics. 2nd edition. New Jersey, NJ: Prentice-Hall.

Kim, C., (2010). *Signal Processing for Robust Speech Recognition Motivated by Auditory Processing*. PhD Thesis. Carnegie Mellon University.

Lippmann, R. P. (1989). Review of Neural Networks for Speech Recognition. In *Neural Computation*, 1-38.

McLoughlin, I. (2009). Applied Speech and Audio Processing. Cambridge: Cambridge University Press.

Manning, D., & Schütze, H. (1999). *Foundations of Statistical Natural Language Processing*. Cambridge, MA: MIT Press.

Mariani, J. (2013). *Spoken Language Processing*. Wiltshire: ISTE Ltd.

Marsland, S. (2009). *Machine Learning: An Algorithmic Perspective*. London: Chapman and Hall / CRC Press.

Mitchell, T. (1997). *Machine Learning*. London: McGraw Hill.

Pearson, K. (1901). On Lines and Planes of Closest Fit to Systems of Points in Space. *Philosophical Magazine Series* 6(2), 559-572.

Pollak, I. (2004). Digital Signal Processing with Applications - Class Notes [PDF document]. Retrieved from: https://engineering.purdue.edu/~ipollak/ee438/FALL04/notes/Chapter0.pdf

Raschka, S. (2014). Linear Discriminant Analysis bit by bit. Retrieved from: http://sebastianraschka.com/Articles/2014_python_lda.html

Rehm, G., & Uszkoreit, H. (2012). *The Icelandic Language in the Digital Age*. Berlin: Springer.

Rogers, H. (2000). *The Sounds of Language: An Introduction to Phonetics*. Harlow UK: Pearson Education Limited.

Rossing, T. (Ed.). (2014). *Springer Handbook of Acoustics*. New York, NY: Springer-Verlag.

Rögnvaldsson, E. (2013). Hljóðkerfi og Orðhlutaferfi Íslensku. Reykjavik.

Rögnvaldsson, E. (2014). Talgreining og Talgreinar [PowerPoint slides]. Retrieved from: https://ugla.hi.is/kv/index2.php?sid=219&adgerd=efnisatridi_skoda&kennsluvefur_kennsluvefsnumer=53614&kennsluvefur_efnisatridi=112

Russell, S., & Norvig. P. (2010). *Artificial Intelligence: A Modern Approach*. 3rd edition. New Jersey, NJ: Prentice-Hall.

Samarasinghe, S. (2006). *Neural networks for applied sciences and engineering*. Boston, MA: Auerbach Publications.

Shrawankar, U. & Thakare, V. M. (2013). *Techniques for feature extraction in speech recognition system: a comparative study.* In International Journal of Computer Applications in Engineering, Technology and Sciences, 412-18.

Skowronski, M. D. & Harris, J. G. (2003). Improving the Filter Bank of a Classic Speech Feature Extraction Algorithm. *Proceedings of the 2003 International Symposium on Circuits and Systems* (ISCAS) vol. 4, 281-284.

Smith, S. W. (1997). *The Scientist and Engineer's Guide to Digital Signal Processing*. California, CA: California Technical Pub.

Tao, L., Shenghuo, Z., & Mitsunori, O. (2006). Using discriminant analysis for multi-class classification: an experimental investigation. *Knowledge and Information Systems* 10(4), 453-472.

Theodoridis, S. & Koutroumbas, K. (2009). Pattern Recognition. 4th edition. California, CA: Elsevier.

Waibel, A. (1988). *Prosody and Speech Recognition*. London: Pitman.

Appendix - Code Extracts

A selection of the main programming code follows in this appendix. Some code was reused in different files and duplicates have been removed. The entire collection of files is available upon request.

```
def getFeatures(filename, word, featureset, path=None):
    """ Load the audio signal from a given audio file and extract features """
    if path is None:
        rate, sig = wav.read(training_folder + word + "\\" + filename)
    else:
        rate, sig = wav.read(path)

    if featureset is 22:
        # MFCCs only
        mfccs = mfcc(sig, rate, nfilt=52, numcep=22,nfft=1024,highfreq=8000)
        final = np.matrix(mfccs)

    elif featureset is 44:
        # MFCCs and first derivatives
        mfccs = mfcc(sig, rate, nfilt=52, numcep=44,nfft=1024,highfreq=8000)
        mfcc_deltas = np.diff(mfccs,axis=0)
        final = np.concatenate((mfccs[0:-1,:],mfcc_deltas),axis=1)
        final = np.matrix(final)

    else:
        # MFCCs and first/second derivatives
        mfccs = mfcc(sig, rate, nfilt=52, numcep=22,nfft=1024,highfreq=8000)
        mfcc_deltas = np.diff(mfccs,axis=0)
        mfcc_double_deltas = np.diff(mfcc_deltas, axis=0)
        final = np.concatenate((mfccs[0:-1,:],mfcc_deltas),axis=1)
        final = np.concatenate((final[0:-1,:], mfcc_double_deltas),axis=1)
        final = np.matrix(final)

    return final


def getTransformMatrix(vectorDimension, splitValue, newVectorDimension):

    vocab = ["einn","ein","eitt","einan",  \
            "eina","einum","einni","einu","eins","einnar", \
            "tveir","tvaer","tvoh","tvo","tveimur","tveggja", \
            "thrir","thrjar","thrju","thrja","thremur","thriggja", \
            "fjorir","fjorar","fjogur","fjora","fjorum","fjogurra", \
            "fimm","sex","sjo","atta","niu","tiu","ellefu","tolf","threttan", \
            "fjortan","fimmtan","sextan","sautjan","atjan","nitjan","tuttugu", \
            "fyrsta","odru","thridja","fjorda","fimmta","sjotta","sjounda", \
            "og","null","deilt","med","plus","minus","rot","sinnum","veldi","i"]

    # Build X and Y
```

```
print len(vocab)
X = np.zeros(vectorDimension) # zero row so vstack will work in all cases
Y = []
for i in range(0,len(vocab)): # 0-5
   files = os.listdir("D:\\uni\\lokaverkefni\\" + vocab[i])
   for f in files:
      features = getFeatures("","",path="D:\\uni\\lokaverkefni\\" + vocab[i] + \
        "\\" +f,featureset=vectorDimension)
      X = np.vstack((X,np.mean(features,axis=0)))
      Y.append(vocab[i])
X = X[1:,:] # take out initial zero row

le = preprocessing.LabelEncoder()
lab_enc = le.fit(Y)
Y = lab_enc.transform(Y)
#"""
means = np.mean(X,axis=0)
stds = np.std(X,axis=0)
normX = (X - means) / stds # get normalised results (z-scores)
#"""
#normX = X

mean_vectors = []
for cl in range(0,len(vocab)): #0-5
   mean_vectors.append(np.mean(normX[Y==cl],axis=0))
   #print('Mean vector for %s : %s\n' %(le.inverse_transform(cl),mean_vectors[cl]))

S_W = np.zeros((vectorDimension,vectorDimension))
for cl,mv in zip(range(0,len(vocab)),mean_vectors):
   class_sc_mat = np.zeros((vectorDimension,vectorDimension))
   for row in X[Y==cl]:
      row, mv = row.reshape(vectorDimension,1), mv.reshape(vectorDimension,1)
      class_sc_mat += (row-mv).dot((row-mv).T)
   S_W += class_sc_mat

overall_mean = np.mean(mean_vectors,axis=0)

S_B = np.zeros((vectorDimension,vectorDimension))
for i,mean_vec in enumerate(mean_vectors):
   n = X[Y==i,:].shape[0]
   assert n == 7, "Amount of files found per class not equal to 7"
   mean_vec = mean_vec.reshape(vectorDimension,1)
   overall_mean = overall_mean.reshape(vectorDimension,1)
   S_B += n * (mean_vec - overall_mean).dot((mean_vec - overall_mean).T)
print('Between-class scatter matrix:')
#print S_B

eig_vals, eig_vecs = np.linalg.eig(np.linalg.inv(S_W).dot(S_B))
for i in range(len(eig_vals)):
   eigvec_sc = eig_vecs[:,i].reshape(vectorDimension,1)
   #print('\nEigenvector: ' + str(i))
   #print eigvec_sc.real
   print 'Eigenvalue: '  + str(i) + " = " + str(eig_vals[i].real)
```

```
    eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in range(len(eig_vals))]
    eig_pairs = sorted(eig_pairs, key=lambda k: k[0], reverse=True)
    eigv_sum = sum(eig_vals)
    print "Eigenvalues in decreasing order:\n"
    for i,pair in enumerate(eig_pairs):
        print(str(i) + " : " + str(pair[0]) + " (" + str((pair[0]/eigv_sum).real)+ "%)")

    W = np.hstack((eig_pairs[0][1].reshape(vectorDimension,1), \
eig_pairs[1][1].reshape(vectorDimension,1))) #2

    counter = 2
    while (counter < newVectorDimension):
        W = np.hstack((W,eig_pairs[counter][1].reshape(vectorDimension,1)))
        counter += 1

    return W


class word():
    def __init__(self, lemma, SV, numCentroids):
        self.lemma = lemma
        self.numStates = SV
        self.b = np.zeros((self.numStates, numCentroids))
        self.holdData = []
        self.train()

    # Looks for directory in CURRENT_DIRECTORY that matches the word's lemma
    # and looks for the training files in there, extracts features and then
    # updates the emission probability matrix
    def train(self):
        # Make sure folder with training files exists
        if os.path.isdir(training_folder + self.lemma) is False:
            print "No training recordings found for the word: " + self.lemma
            return
        # Get list of training files
        trainingFiles = os.listdir(training_folder + self.lemma)
        # Loop through files, building whole matrices for states and compiling
        # all of the right feature vectors into a matrix that represents all the
        # training data
        self.holdData = [] # To hold the data across the files
        firstFile = True # New matrices for states need to be added to holdData when first file is seen
        for trainingFile in trainingFiles:
            currentFeatures = getFeatures(trainingFile, self.lemma)
            # Find the range to divide up the data (according to the given Split Value)
            shiftDelta = np.shape(currentFeatures)[0] / self.numStates
            counter = 0
            low = 0
            high = shiftDelta
            while counter < self.numStates: # Loop until all acoustic vectors seen
                if firstFile: # Special case for the first file because vstack won't work otherwise
                    self.holdData.append(currentFeatures[low:high,:])
                else:
```

```
                self.holdData[counter] = np.vstack([self.holdData[counter], \
            currentFeatures[low:high,:]])
              # Update variables
              low += shiftDelta
              high += shiftDelta
              counter += 1
          firstFile = False
          # end while
        #end for

    def forwardAlg(self, obs):
        """ Returns the (altered) Forward probability of the acoustic observation sequence"""
        T = np.shape(obs)[0]
        alpha = np.zeros((self.numStates,T))
        alpha[:,0] = (1/float(self.numStates))
        # t = timestep, s = state
        # i.e. alpha[t = 5, s = 6] is the sixth state calculation at timestep 5
        for t in range(1,T):
            for s in range(self.numStates):
                assert sum(self.b[s,:]) != 0, "B matrix sum is zero"

                part1 = (self.b[s,int(obs[t][0])] / sum(self.b[s,:])) # probability dist of extracted codeword
                #print (self.b[s,obs[t][0]] / sum(self.b[s,:]))
                #print self.b[s,int(obs[t][0])]
                #part1 = 1
                part2 = sum(alpha[:,t-1]) # sum of previous time-step
                alpha[s,t] = part1 * part2 # update the alpha table

            #end for
        #end for
        return alpha[self.numStates-1,T-1]

    def eig_transform(self,matrix_transform):
        #print "matrix_transform: " + str(np.shape(matrix_transform.T))
        #print "self.holdData[0]: " + str(np.shape(self.holdData[0].T))
        self.eig_data = []
        self.eig_mean = []
        self.eig_covariances = []
        for state in range(self.numStates):
            result = matrix_transform.T.dot(self.holdData[state].T).T
            self.eig_data.append(result)
            self.eig_mean.append(np.mean(self.eig_data[state],axis=0))
            cov = np.cov(np.transpose(self.eig_data[state]))
            self.eig_covariances.append(cov)

def generateCodebook(SV, numCentroids, wordHMMs, newVectorDimension):
    """ Generates a codebook with numCentroids codewords """
    all_data = np.zeros((1,newVectorDimension))
    for word in wordHMMs:
        for item in word.eig_data:
            all_data = np.vstack((all_data,np.mean(item,axis=0)))
    codebook = kmeans(all_data[1:,:], numCentroids) # Remove first row of zeros
    return codebook[0]
```

```
    def trainEmissionMatrix(wordHMMs, codebook, SV, numCentroids, W):
    """ Trains the emission (B) matrix with the appropriate codeword in the correct split value """
    for word in wordHMMs:
        assert np.shape(word.b) == (SV,numCentroids), "Split/Dimension parameter mismatch"
        lemma = word.lemma
        for f in os.listdir(training_folder+lemma):
            features = getFeatures("","",path=training_folder+lemma+"\\"+f)
            features = W.T.dot(features.T).T
            low = 0;
            for i in range(SV):
                high = np.shape(features)[0] / SV + low
                codeword = vq(np.mean(features[low:high,:], axis=0), codebook)[0]
                word.b[i,codeword] += 1
                #print "Word: " + word.lemma + " Segment: " + str(i) + " Codebword: " + str(codeword)
                low = high
    return wordHMMs


def classifyAll(SV, numCentroids, newVectorDimension):

    print "Loading wordHMMs"
    wordHMMs = load_words(SV, numCentroids)
    print "Generating W"
    W = getTransformMatrix(VD, SV, newVectorDimension)
    print "Updating wordHMMs"
    for word in wordHMMs:
        word.eig_transform(W)

    print "Generating codebook"
    codebook = generateCodebook(SV, numCentroids, wordHMMs, newVectorDimension)
    print "Training Emmision Matrix"
    wordHMMs = trainEmissionMatrix(wordHMMs, codebook, SV, numCentroids, W)

    test_files = os.listdir(training_folder+test_files_foldername)
    correct = 0
    withheld_correct = 0
    for f in test_files:
        query = getFeatures("","",path=training_folder+test_files_foldername+"\\"+f)
        query = W.T.dot(query.T).T
        new_query = np.zeros((SV,1))
        low = 0;
        for i in range(SV):
            high = np.shape(query)[0] / SV + low
            new_query[i,0] = vq(np.mean(query[low:high,:], axis=0), codebook)[0]
            low = high
        result = [w.forwardAlg(new_query) for w in wordHMMs]
        result = np.argmax(result)
        #print "file: " + f + " prediction = " + wordHMMs[result].lemma
        if wordHMMs[result].lemma == f.split('_')[0]:
            correct += 1
            if f in withheld:
                withheld_correct += 1
```

```
    print "(Split: " + str(SV) +  ") (Features: " + str(VD) + ") (Centroids: " + str(numCentroids) + ")"
    print "Total accuracy: " + str((float(correct) / float(len(wordHMMs) * 10)) *  100.0) + "%"
    print "Withheld accuracy: " + str((float(withheld_correct) / float(len(withheld))) * 100.0) + "%"


def eig_prob(obs, means, cov, num_components):
    # Does anything change with 1.0 and 1.0 / 2.0 in part 1?
    part1 = 1 / ((2 * np.pi)**(num_components/2) * np.linalg.det(cov)**(1/2))
    part2 = (-1.0/2.0) * ((obs-means).dot(np.linalg.inv(cov))).dot((obs-means).T)
    return float(part1 * np.exp(part2))

def eig_classify(wordHMMs, filename, num_components):
    query = getFeatures("","",path="D:\\uni\\lokaverkefni\\testings\\" + filename)
    #print "Eigen_classify ... query shape : " + str(np.shape(query))
    new_query = np.zeros((SV,NEW_VD))
    query = W.T.dot(query.T).T
    #print "Eigen_classify ... new query shape : " + str(np.shape(query))
    low = 0;
    for i in range(SV):
        high = np.shape(query)[0] / SV + low
        new_query[i,:] = np.mean(query[low:high,:], axis=0)
        low = high
    # end for
    #print new_query
    results = []
    for hmm in wordHMMs:
        #print "HMM ... " + hmm.lemma
        result = 0.000000000000001
        for state in range(SV):
            probability = eig_prob(new_query[state], hmm.eig_mean[state], \
              hmm.eig_covariances[state], NEW_VD)
            #print "Probability: " + str(probability)
            if probability == 0.0:
                #print "Prob was zero"
                result = result
            else:
                result += np.log(probability)
            #print " Split " + str(state) + " : " + str(probability) + " (total: " + str(result) + ")"
        if result == 1E-15:
            results.append(None)
        else:
            results.append(result)
    arg_max = np.argmax(results)
    #max_result = max(results)
    return arg_max

def eig_tryAll(wordHMMs, num_components):
    files = os.listdir("D:\\uni\\lokaverkefni\\testings")
    accuracy = 0
    withheld_accuracy = 0
    for f in files:
        arg_max = eig_classify(wordHMMs, f, NEW_VD)
```

```
        print "Prediction: " + wordHMMs[arg_max].lemma + " (file: " + str(f) + ")"
        if wordHMMs[arg_max].lemma == f.split('_')[0]:
            accuracy += 1
            if f in eig_withheld:
                withheld_accuracy += 1
    print "Total accuracy: " + str((float(accuracy) / float(len(wordHMMs) * 10)) * 100.0) + "%"
    print "Withheld acc: " + str((float(withheld_accuracy) / float(len(eig_withheld))) * 100.0) + "%"

def multivariateGaussian(obs, means, cov, num_components):
    part1 = 1 / ((2 * np.pi)**(num_components/2) * np.linalg.det(cov)**(1/2))
    part2 = (-1.0/2.0) * ((obs-means).dot(np.linalg.inv(cov))).dot((obs-means).T)
    return float(part1 * np.exp(part2))

def classify(wordHMMs, filename, W_mat, SV, NEW_VD, GUI=False):
    #####
    if GUI is False:
        query = getFeatures("","",path="D:\\uni\\lokaverkefni\\test\\" + filename,featureset=VD)
    #print "OPENING: " + "D:\\uni\\lokaverkefni\\"+filename
    else:
        query = getFeatures("","",path=filename,featureset=VD)

    query = W_mat.T.dot(query.T).T
    new_query = np.zeros((SV,NEW_VD))
    #print "NEW QUERY SHAPE: " + str(np.shape(new_query))
    #print "QUERY SHAPE:" + str(np.shape(query))
    low = 0;
    for i in range(SV):
        high = np.shape(query)[0] / SV + low
        new_query[i,:] = np.mean(query[low:high,:], axis=0)
        low = high
    # end for
    results = []
    for hmm in wordHMMs:
        result = 0.000000000000001
        for state in range(SV):
            probability = multivariateGaussian( new_query[state],
                                    hmm.eig_mean[state],
                                    hmm.eig_covariances[state],
                                    NEW_VD)
            #print "Probability: " + str(probability)
            if probability == 0.0:
                #print "Prob was zero"
                result = result
            else:
                result += np.log(probability)
            #print " Split " + str(state) + " : " + str(probability) + " (total: " + str(result) + ")"
        if result == 1E-15:
            results.append(None)
        else:
            results.append(result)
    return results
    def classifyAll():
    """ HMM implementation """
```

```
    wordHMMs = load_all_words()
    W = getTransformMatrix(VD, SV, NEW_VD)
    for word in wordHMMs:
        word.eig_transform(W)

    files = os.listdir("D:\\uni\\lokaverkefni\\test_files")
    accuracy = 0
    withheld_accuracy = 0
    for f in files:
        prediction = wordHMMs[np.argmax(classify(wordHMMs, f, W))].lemma
        print "Prediction: " + prediction + " (file: " + str(f) + ")"
        if prediction == f.split('_')[0]:
            accuracy += 1
            if f in withheld:
                withheld_accuracy += 1
    print "Split Value: " + str(SV) + " Vector Dimensions : " + str(NEW_VD)
    print "Total accuracy: " + str((float(accuracy) / float(len(wordHMMs) * 10)) * 100.0) + "%"
    print "Withheld accuracy: " + str((float(withheld_accuracy) / float(len(withheld))) * 100.0) + "%"

class mlp:
    def __init__(self,inputs,targets,nhidden,words,beta=1,momentum=0.9,outtype='logistic'):
        self.nin = np.shape(inputs)[1]
        self.nout = np.shape(targets)[1]
        self.ndata = np.shape(inputs)[0]
        self.nhidden = nhidden
        self.words = words
        self.beta = beta
        self.momentum = momentum
        self.outtype = outtype

        # Initialise network
        self.weights1 = (np.random.rand(self.nin+1,self.nhidden)-0.5)*2/np.sqrt(self.nin)
        self.weights2 = (np.random.rand(self.nhidden+1,self.nout)-0.5)*2/np.sqrt(self.nhidden)

    def mlptrain(self,inputs,targets,eta,niterations,test_inputs=None,test_targets=None):
        """ Train the thing """
        # Add the inputs that match the bias node
        inputs = np.concatenate((inputs,-np.ones((self.ndata,1))),axis=1)
        updatew1 = np.zeros((np.shape(self.weights1)))
        updatew2 = np.zeros((np.shape(self.weights2)))

        for n in range(niterations):

            self.outputs = self.mlpfwd(inputs)

            error = 0.5*np.sum((self.outputs-targets)**2)
            if (np.mod(n,100)==0):
                print "Iteration: ",n, " Error: ",error
                copyOutputs = np.copy(self.outputs)
                counter = 0.0
                for i in range(len(copyOutputs)):
                    copyOutputs[i,:] = np.where(copyOutputs[i,:] == np.amax(copyOutputs[i,:]),1,0)
                    if np.argmax(copyOutputs[i,:]) == np.argmax(targets[i,:]):
```

```
                #print self.words[np.argmax(copyOutput[i,:])].lemma + " + is " + \
            str(np.argmax(targets[i,:]))
                counter += 1.0
            # end for
            if test_inputs != None:
              testOutputs = self.mlpfwd(test_inputs,testing=True)
              counter2 = 0.0
              for i in range(len(testOutputs)):
                testOutputs[i,:] = np.where(testOutputs[i,:] == np.amax(testOutputs[i,:]),1,0)
                if np.argmax(testOutputs[i,:]) == np.argmax(test_targets[i,:]):
                  counter2 += 1.0
            training_accuracy = ((float(counter) / float(self.ndata)) * 100.00)
            test_accuracy =  ((float(counter2) / float(np.shape(test_inputs)[0])) * 100.00)
            print "Training = " + str(training_accuracy)[:4] + "%, Test = " + str(test_accuracy)[:4] + "%"

          # Different types of output neurons
          if self.outtype == 'linear':
            deltao = (self.outputs-targets)/self.ndata

          temp1_1 = np.matrix(self.hidden*self.beta) # ok
          temp1_2 = np.matrix(1.0-self.hidden)
          temp1_3 = np.multiply(temp1_1,temp1_2)

          temp2 = (np.dot(deltao,np.transpose(self.weights2))) # ok
          deltah = np.multiply(temp1_3,temp2)

          updatew1 = eta*(np.dot(np.transpose(inputs),deltah[:,:-1])) + self.momentum*updatew1
          updatew2 = eta*(np.dot(np.transpose(self.hidden),deltao)) + self.momentum*updatew2
          self.weights1 -= updatew1
          self.weights2 -= updatew2

    def mlpfwd(self,inputs,testing=False):
        """ Run the network forward """

        if testing == True:
          inputs = np.concatenate((inputs,-np.ones((np.shape(inputs)[0],1))),axis=1)
          self.hidden_test = np.dot(inputs,self.weights1);
          self.hidden_test = 1.0/(1.0+np.exp(-self.beta*self.hidden_test))
          self.hidden_test = np.concatenate((self.hidden_test,-
np.ones((np.shape(inputs)[0],1))),axis=1)

          outputs = np.dot(self.hidden_test,self.weights2);

          # Different types of output neurons
          if self.outtype == 'linear':
            return outputs
          else:
            print "error"
        else:
          self.hidden = np.dot(inputs,self.weights1);
          self.hidden = 1.0/(1.0+np.exp(-self.beta*self.hidden))
          self.hidden = np.concatenate((self.hidden,-np.ones((np.shape(inputs)[0],1))),axis=1)
```

```
        outputs = np.dot(self.hidden,self.weights2);

        # Different types of output neurons
        if self.outtype == 'linear':
          return outputs
        else:
           print "error"

WSV = 4      # Word split value
VD = 22        # Vector dimension
NEW_VD = 22     # New vector dimension

# Load all the words into a list
words = load_all_words(WSV, VD)

# Get dimensionality reduction matrix and apply it
W = getTransformMatrix(VD, WSV, NEW_VD)
for word in words:
   word.eig_transform(W)

# To be able to get a lemma from an index
index_and_word = [(i,words[i].lemma) for i in range(len(words))]

def wordToIndex(queryWord):
   result = [index for index,word in index_and_word if word == queryWord]
   return result[0]

inputMatrix = np.zeros((len(words),(WSV * NEW_VD)))
targetMatrix = np.matrix(np.eye(len(words)))

# Go through every word
for word_index in range(len(words)):
   # Temp row to build
   temp = np.zeros((1,NEW_VD))
   # Loop through every word split
   for i in range(WSV):
      # First case is overwrite and not concatenate
      if i == 0:
         temp[:NEW_VD] = words[word_index].eig_mean[0]
      else:
         temp = np.concatenate((temp, words[word_index].eig_mean[i]) ,axis=1)
   # end for
   #print "inputMatrix[0,:] = " + str(np.shape(inputMatrix[0,:]))
   #print "temp = " + str(np.shape(temp))
   assert np.shape(temp) == (1,(WSV * NEW_VD)), "inputMatrix and temp not same size"
   # Make assignment
   inputMatrix[word_index,:] = temp
   # Move to next word_index
# end for
inputMatrix = np.matrix(inputMatrix)

# Generate test data
testInput = np.zeros((len(withheld()), WSV * NEW_VD))
```

```
for index in range(len(withheld())):
   new_query = np.zeros((WSV,NEW_VD))
   query = getFeatures("","",path="D:\\uni\\lokaverkefni\\test\\" + \
         withheld()[index],featureset=VD)
   query = W.T.dot(query.T).T # Reduce dimensions
   low = 0;
   for i in range(WSV):
      high = np.shape(query)[0] / WSV + low
      new_query[i,:] = np.mean(query[low:high,:], axis=0)
      low = high
   # end for
   temp = np.zeros((NEW_VD))
   for i in range(WSV):
      if i == 0:
         temp[:NEW_VD] = new_query[i,:]
      else:
         store = new_query[i,:]
         temp = np.hstack((temp,store))
   #end for
   testInput[index,:] = temp

testTarget = np.zeros((len(withheld()),len(words)))
for wordIndex in range(len(withheld())):
   currentFilename = withheld()[wordIndex]
   currentWord = currentFilename.split("_")[0]
   testTarget[wordIndex][wordToIndex(currentWord)] = 1
# end for

# initialise ANN
ann = mlp(inputMatrix,targetMatrix,250,words,outtype='linear')
# train the network

print "WSV: " + str(WSV)
print "NEW_VD: " + str(NEW_VD)
print "HIDDEN: 25"

ann.mlptrain(inputMatrix,targetMatrix,0.05,2000, test_inputs=testInput, test_targets=testTarget)
```