An Integrated Approach to Speech Recognition using Phrase-Based Units

Christopher James Watkins

A thesis submitted for the Degree of
Doctor of Philosophy

University of East Anglia
School of Computing Sciences

March, 2010

Abstract

In human-to-human dialogue, *formulaic sequences* are used to minimise the effort of both speech production and perception in the conversation. In production, the speaker apparently retrieves such sequences whole from memory, without the cognitive effort required for generation from a lexicon and grammar. In perception, context determines a set of similar phrases that the listener expects to hear, and this also reduces cognitive load.

This thesis describes techniques used to automatically acquire formulaic phrases from transcriptions of speech, which are then used to define variable-length units of speech and language. These are well suited for use in a template-based speech recogniser, which can easily adjust its modelling units for the examples that are found, with the aim of improving Automatic Speech Recognition (ASR) accuracy. Language modelling techniques are described, such as the *Word Phrase Link Bi-gram* (WPLB) language model, which combines words and phrases together, and the *Hybrid Syntactic Formulaic* (HSF), which clusters semantically similar phrases using syntax. The language models are then combined with speech, in both Hidden Markov Model and template-based speech recognisers. Techniques to reduce the complexity of the search space for the template-based recogniser are introduced, such as the hierarchical LDA filter.

As expected, the techniques gave significant gains when the language used was highly formulaic, and were less successful on a "standard" speech database which consisted of highly artificial utterances.

Acknowledgements

I would first of all like to thank Professor Stephen Cox, for first of all suggesting this work, and secondly for being a very optimistic and attentive supervisor. I would like to thank Dr. Ben Milner, my co-supervisor, for his help with flat-start labelling, and general availability for discussions about issues in my work. I would also like thank Professor Dirk Van Compernolle who agreed to be the external examiner for this thesis and Dr. Barry Theobald who was the internal examiner.

Thanks go out to my colleagues and friends that I have shared a lab with over the past four years; Jacob Newman, Nick Wilkinson, Qiang Huang, Ibrahim Almajai, Osama Dorgham, Ian Read, Alastair James, Mark Hadley, Jonathan Darch, Omar Caballero and Sarah Hilder.

Finally, a special thanks go to my fiancée Yuxuan Lan whom has kept me calm throughout these final few stressful months, cooking splendid cuisines for me, and supported me even though I was in the lab day and night for most of the past year!

Contents

List of Abbreviations

Abbreviation	Meaning
ASR	Automatic Speech Recognition
CD	Context Dependent
CI	Context Independent
DP	Dynamic Programming
DTW	Dynamic Time Warping
FNR	False Negative Rate
FPR	False Positive Rate
FSA	Finite State Automoton / Automata
FSN	Finite State Network
GMM	Gaussian Mixture Model
HMM	Hidden Markov Model
KNN	K Nearest Neighbour
LDA	Linear Discriminant Analysis
LM	Language Model
ML	Maximum Likelihood
NN	Nearest Neighbour
OOV	Out-of-vocabulary
PB	Phrase Bigram
PDF	Probability Density Function
POS	Parts of Speech
RBF	Radial Basis Function
RM	Resource Mangagement
SD	Speaker Dependent
SFSA	Stochasic Finite State Automoton / Automata
SI	Speaker Independent
SVM	Support Vector Machine
VNSA	Variable N-gram Stochastic Automata
VQ	Vector Quantisation
VTLN	Vocal Tract Length Normalisation
WB	Word Bigram
WER	Word Error Rate
WPB	Word Phrase Bigram
WPCB	Word Phrase Class Bigram
WPCLB	Word Phrase Class Link Bigram

Abbreviation	Meaning
WPLB	Word Phrase Link Bigram

List of Figures

List of Algorithms

List of Tables

Chapter 1

Introduction

1.1 Motivation and Aims

The data-driven, probabilistic-based paradigms that have been developed for ASR (notably the hidden Markov model (HMM) framework) were highly successful in advancing the technology during the 1980s and 1990s, but they contained simplifications and assumptions that are known to be untrue (for instance, the assumption that speech is produced by a first-order Markov process, or that language can be well modelled using n-grams of words). After years of intensive incremental development of the models, it seems that these modelling assumptions are now limiting the progress of ASR, and that these techniques may not be able to provide the leap needed to move ASR performance up to the level required for it to be usable in new applications [Moore, 2003].

One area in which the conventional approach to ASR may be questioned is in the use of statistical distributions. Clearly probability density functions (PDFs) constitute an essential element of hidden Markov modelling and provide a powerful method of generalising from seen to unseen data. However, the use of PDFs does represent a potential loss of information; the detail that is present in individual data samples is sacrificed in order to pool information in a controlled fashion.

In ASR, this realisation has led to a resurgence of interest in template-based recognition systems [De Wachter et al., 2003; Axelrod and Maison, 2004; Aradilla et al., 2005; Maier and Moore, 2005; Demange and Van Compernolle, 2009a].

Another practice that simplifies conventional ASR is the use of fixed levels of description, which enables a hierarchical and modular approach to recognition in which words can be easily constructed as sequences of phonemes and language models as n-grams of words. However, there is ample psycho-linguistic evidence that humans recognise and generate a great deal of language in ready-made chunks [Goldinger, 1996, 1998; Wray, 1999], which have been termed "formulaic sequences" by some linguists [Wray, 2002]. These sequences "appear to be pre-fabricated, that is stored and retrieved whole from memory rather than being subject to generation or analysis by the language grammar" [Wray and Perkins, 2000]. It has been argued that these phrases serve the important purpose of avoiding processing overload in both speaker and listener: the speaker retrieves them whole from memory, and the listener is more likely to understand a message if it is in a form that he/she has heard before. Analysis shows that much commonplace language is highly formulaic, and this is especially true in ASR applications where the application context is narrow and the discourse is goal-directed, factors that apply to most telephony ASR applications that provide information provision and booking services. Although many of these phrases have the status of "carrier" phrases, and as such have a low information content associated with them [Huang and Cox, 2006], recognition of them is essential for segmentation of the signal and extraction of the information content.

The template-based approach to recognition fits very well with the idea that the units used for recognition can be of different lengths: rather than insisting on modelling an utterance as a sequence of phonemes, we can adjust the lengths of our modelling units according to the examples we find. This has obvious benefits in capturing the acoustic/phonetic variation in commonly occurring fragments.

These ideas give the motivation for the work presented in this thesis, which will

attempt to define templates explicitly from commonly-occurring phrases found in transcriptions of speech which can then be used in a speech recogniser. This approach differs from that of De Wachter et al. [2007] where templates are defined at the *phone* level and then concatenated together based upon a set of costs. By explicitly defining the templates before the recognition, language modelling techniques can be investigated in an attempt to leverage prior information for the decoder, offering strong predictions based on certain perceptual contexts, as earlier described for human listeners in conversational dialogue. Thus, the aim of this thesis is to integrate commonly-occurring phrase-based units, modelled by language and speech, into a template-based speech recognition system.

1.2 Thesis Overview

This section will give an overview of each chapter presented in this thesis, starting with prerequisites in Chapter 2 and Chapter 3, followed by the three main chapters that define this work: Chapter 4 describes phrase-based language modelling, Chapter 5 describes bottom-up template selection, which is used to reduce the massive search space of the decoder in template-based recognition, and Chapter 6 describes the template-based decoder and recognition experiments which integrate the language modelling techniques and template selections together with the decoder. An overview of each chapter is now given.

Chapter 2 gives the required technical background for this thesis, including N-Gram language modelling, speech recognition techniques such as the Dynamic Time Warping (DTW) and Token Passing algorithms, integration of the language model into a speech recogniser, Vocal Tract Length Normalisation (VTLN), and Linear Discriminant Analysis (LDA) for a classifier.

Chapter 3 describes the datasets used in this thesis, including the feature vectors that are extracted, the HMM-based systems that are built for each dataset to act as baseline systems, and the number of templates that are contained in each

dataset.

Chapter 4 describes the methods used to acquire commonly-occurring phrases from transcriptions of speech and then integrate them into language models. A clustering algorithm is presented which groups phrases of a similar semantic function by using syntactic information. The individual phrases and classes of phrases are combined into Stochastic Finite State Automata (SFSA) representations of language models which can be later used to integrate the phrases with the models and templates of speech in the recogniser. The chapter ends with the baseline evaluation of the methods presented using a monophone HMM-based recogniser and then a discussion of these results.

Chapter 5 presents methods used to reduce the complexity of the template-based decoder, including an introduction to the Time Filter algorithm [De Wachter et al., 2003] which is a bottom-up acoustic pass over the utterance to find *approximately* matching templates. Extensions are made to the Time Filter algorithm, including a Vector Quantisation (VQ) method which is an approximate method to k-nearest neighbour (KNN) selection used in the Time Filter algorithm. A backward pass of the Time Filter is suggested, with details given of the implementation, and a Sigmoid-based distance normalisation function is described which is used to control the length of templates selected by the Time Filter for experimental purposes. The chapter also describes a hierarchical LDA classifier which is used to further filter the template candidates that are output from the Time Filter, with details of feature extraction and classifier decision points given. The chapter finishes by evaluating the performance of the described techniques for template selection, with tests such as how well the selected templates match the input utterances, and classification tests of the LDA filter.

Chapter 6 describes the template-recognition experiments, including the decoder architecture and how VTLN is applied to the templates. It also discusses the results presented, and gives conclusions.

Finally, Chapter 7 summarises the work and findings presented in this thesis,

integrating the conclusions of each chapter into a final conclusion which includes a discussion of the possible directions this work could take in the future.

Chapter 2

Technical Background

2.1 Introduction

Speech recognition is the task of converting an audio waveform into a string of words. The waveform is typically discretised into a sequence of vectors $\boldsymbol{X}$, known as *observations*, as part of a feature extraction process (more details are given in Section 3.4), giving the observation sequence

$$\boldsymbol{X}_1^T = \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_{T-1}, \mathbf{x}_T \tag{2.1}$$

The word string $\boldsymbol{w}$ that represents the observation sequence $\boldsymbol{X}$ is given by:

$$\boldsymbol{w}_1^m = w_1, w_2, w_3, \ldots, w_{m-1}, w_m \tag{2.2}$$

The general problem for finding the correct word string $\hat{\boldsymbol{w}}$ for the current observation sequence $\boldsymbol{X}$, which entails searching over all possible word strings in the language $\mathscr{L}$, can be formulated as finding $\hat{\boldsymbol{w}}$ such that:

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}\in\mathscr{L}}{\operatorname{argmax}} \Pr(\boldsymbol{w}|\boldsymbol{X}) \tag{2.3}$$

Applying Bayes' rule gives:

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}\in\mathscr{L}}{\operatorname{argmax}} \frac{\Pr(\boldsymbol{X}|\boldsymbol{w})\Pr(\boldsymbol{w})}{\Pr(\boldsymbol{X})} \tag{2.4}$$

The denominator in Equation (2.4) can be removed as it is constant over all possible word strings for the current observation sequence, giving:

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}\in\mathscr{L}}{\operatorname{argmax}} \Pr(\boldsymbol{X}|\boldsymbol{w})\Pr(\boldsymbol{w}) \tag{2.5}$$

Equation (2.5) is the driving force for the emergence of two distinct research fields within the area of automatic speech recognition (ASR): acoustic modelling and language modelling. $\Pr(\boldsymbol{w})$ represents the *prior* probability of observing a sequence of words $\boldsymbol{w}$ and is estimated using language modelling techniques, in particular the N-gram language model which will be described in Section 2.2. $\Pr(\boldsymbol{X}|\boldsymbol{w})$ represents the probability of the observation sequence $\boldsymbol{X}$ given a hypothesised word string and is estimated using acoustic matching and modelling techniques which are described further in Section 2.3 and Section 2.4. Section 2.5 will describe the techniques that are required to integrate language models into a speech recogniser. Section 2.6 will describe Vocal Tract Length Normalisation (VTLN) which is a method used to warp a speakers frequency scale to try and remove the variability introduced by different length vocal tracts. Finally, Section 2.7 will introduce and describe *Linear Discriminant Analysis* (LDA) which is used to form a classifier in Section 5.4.

2.2 N-Gram Language Modelling

The N-gram is motivated from the idea that the probability of a word w_i in a sequence of words w_1^m can be estimated based on the previous words in the sequence w_1^{i-1} using a relative frequency approach, such that:

$$\Pr\left(w_i|w_1^{i-1}\right) = \frac{C\left(w_1^i\right)}{C\left(w_1^{i-1}\right)} \tag{2.6}$$

where $C\left(\cdot\right)$ represents the count of the given word string which is attained from a collection of sentences known as the training text. The probability of a whole word sequence can then be calculated by the product of each word probability:

$$\Pr\left(w_1^m\right) = \prod_{i=1}^{m} \Pr\left(w_i|w_1^{i-1}\right) \tag{2.7}$$

Equation (2.6) is not tractable for real-world applications because as the word history becomes longer and longer it is less likely to have occurred in training data (and even on the web). As a solution to this problem, and the intuition for the N-gram probability, the *Markov* assumption can be made which says that the probability of a word can be calculated using the $N-1$ previous words instead of the whole history of the sequence, such that:

$$\Pr\left(w_i|w_1^{i-1}\right) \approx \Pr\left(w_i|w_{i-N+1}^{i-1}\right) \tag{2.8}$$

where

$$\Pr\left(w_i|w_{i-N+1}^{i-1}\right) = \frac{C\left(w_{i-N+1}^{i}\right)}{C\left(w_{i-N+1}^{i-1}\right)} \tag{2.9}$$

such that the approximated probability of a word sequence is given by:

$$\Pr\left(w_1^m\right) \approx \prod_{i=1}^{m} \Pr\left(w_i|w_{i-N+1}^{i-1}\right) \tag{2.10}$$

Although Equation (2.10) is given in general form for any N, the work reported in this thesis is only concerned with the case where $N = 2$, i.e. the *bigram*[1]:

$$\Pr\left(w_1^m\right) \approx \prod_{i=1}^{m} \Pr\left(w_i|w_{i-1}\right) \tag{2.11}$$

where

$$\Pr\left(w_i|w_{i-1}\right) = \frac{C\left(w_{i-1}w_i\right)}{C\left(w_{i-1}\right)} \tag{2.12}$$

2.2.1 Katz-Backoff

Even though the N-gram model reduces the data sparsity problem by, as discussed earlier, using a reduced history for an approximation of the probability of a word, there can still be cases when N-grams rarely or never occur in the training text but are observed in the test data. This is usually because of sampling issues when the vocabulary is large. If an N-gram in a test sentence has not been seen in the

[1]Word recognition experiments with the baseline HMM system were performed with the trigram language model and did not give improved performance over the bigram language model. It was concluded that this performance was due to the small size of the datasets used in this work.

training text then Equation (2.10) will evaluate to zero.

Katz backoff [Katz, 1987] offers a solution to this problem by applying *Good-Turing* smoothing [Good, 1953] (see also Lidstone [1920]; Witten and Bell [1991]; Kneser and Ney [1995] for alternative smoothing measures) to the counts of N-grams that appear less than or equal to k and re-distributing the leftover "probability mass" to the unseen N-grams using a recursive model that contains lower order N-gram distributions. The lower-order distributions (e.g. unigram and bigram if using a trigram model) allow a "backoff" procedure to take place if the current N-gram is not seen, thus reducing the context size of the current word by one and giving a $(N-1)$-gram, which makes it more likely to have been seen in the training data.

The Good-Turing discount method, shown in Equation (2.13), employs a *frequency of frequency* approach which estimates the probability of N-grams that occur r times based on the probability of N-grams that occur $r+1$ times:

$$r^* = (r+1)\,\frac{n_{r+1}}{n_r} \tag{2.13}$$

where r^* is the "smoothed" count, $r = C(w_{i-N+1}^{i})$, and n_r is the number of N-grams that occur r times in the training text. Good-Turing smoothing in the Katz backoff model is only applied to counts that occur fewer than or equal to k times (a good estimate for k is five but this may vary depending on the data), because it is assumed that N-grams that appear more than k times are reliably estimated. The discount ratio d_r, which is the ratio of the discounted counts r^* to the original counts r for N-grams appearing between 1 and k times (inclusive), and shown in Equation (2.15), can be used to estimate the discounted probability of the N-gram:

$$\Pr^*\left(w_i|w_{i-N+1}^{i-1}\right) = d_r \times \frac{C\left(w_{i-N+1}^{i}\right)}{C\left(w_{i-N+1}^{i-1}\right)} \tag{2.14}$$

where the discount ratio d_r (adjusted for k) is calculated as:

$$d_r = \begin{cases} \frac{\frac{r^*}{r} - \frac{(k+1)n_{k+1}}{n_1}}{1 - \frac{(k+1)n_{k+1}}{n_1}} & \text{for } 1 \leq r \leq k \\ 1 & \text{for } r > k \end{cases} \tag{2.15}$$

So far, Good-Turing smoothing has been described. The key part of Katz's work is in the backoff from higher order to lower order N-grams. Instead of re-distributing the leftover probability mass equally, a backoff weight α is used to transfer the mass to the lower order N-grams based on the $(N-1)$-gram context, such that:

$$\alpha\left(w_{i-N+1}^{i-1}\right) = \frac{1 - \sum_{w_i:C\left(w_{i-N+1}^{i}\right)>0} \Pr^*\left(w_i|w_{i-N+1}^{i-1}\right)}{1 - \sum_{w_i:C\left(w_{i-N+1}^{i}\right)>0} \Pr^*\left(w_i|w_{i-N+2}^{i-1}\right)} \tag{2.16}$$

It should be noted that all of the distributions (N-gram, (N-1)-gram, ..., bigram) are smoothed using Equation (2.14) except for the unigram distribution which is smoothed by:

$$\Pr^*(w_i) = d_r \times \frac{C\left(w_i\right)}{n} \tag{2.17}$$

where n is the total number of words in the training text (including repetitions). After smoothing, the leftover probability mass is passed to the next lower order distribution using the backoff weights of Equation (2.16). The leftover mass from

the unigram distribution is then reserved for out-of-vocabulary (OOV) words if they are permitted for the language model, else it is reabsorbed into the unigram probabilities.

Equation (2.18), in its recursive form, shows how the Katz backoff model can be used when estimating the probability of a test N-gram: if the N-gram is seen, then the smoothed probability $\Pr^*\left(w_i|w_{i-N+1}^{i-1}\right)$ is used, else a recursive backoff procedure is started which uses the backoff weights and lower order smoothed probabilities:

$$\begin{aligned}\Pr_{katz}\left(w_i|w_{i-N+1}^{i-1}\right) = \quad & \Pr^*\left(w_i|w_{i-N+1}^{i-1}\right) + \theta\left(\Pr^*\left(w_i|w_{i-N+1}^{i-1}\right)\right) \\ & \alpha\left(w_{i-N+1}^{i-1}\right)\Pr_{katz}\left(w_i|w_{i-N+2}^{i-1}\right)\end{aligned} \tag{2.18}$$

where

$$\theta\left(x\right) = \begin{cases} 1 & \text{if } x = 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.19}$$

such that the probability of a given word sequence w_1^m is calculated as

$$\Pr\left(w_1^m\right) \approx \prod_{i=1}^{m} \Pr_{katz}\left(w_i|w_{i-N+1}^{i-1}\right) \tag{2.20}$$

2.2.2 Representing Backoff LMs with Stochastic Automata

To use the backoff N-gram language model in a speech recogniser, Riccardi et al. [1996] introduced the *Variable N-Gram Stochastic Automaton* (VNSA) which allows an N-gram of any order (any N), with a backoff mechanism, to be input to a Viterbi-style decoder. This architecture is required for the *Token Passing* imple-

mentation of the Viterbi algorithm (see Section 2.3.4). As previously mentioned, the work presented in this thesis uses bigram backoff language models, and thus what follows is a simplified explanation of VNSA.

Figure 2.1 shows a real example of a section of a word bigram (WB) language model using Katz backoff with Good-Turing smoothing built from one of the datasets used in this work (see Section 3.2) represented in VNSA form. Each language model has a *start* and *end* state which represent the start and end of a sentence of text. Each word in the vocabulary of the training text is represented as a state — solid connections between word states represent bigram transitions, such that the connection between the state "can" and the state "i" represents the bigram "can i" which was seen in the training data; the log of the smoothed bigram probability is shown on the arc, defined by Equation (2.14) such that $\log\left(\Pr^*\left(\text{“can i”}\right)\right) = -0.94$.

Figure 2.1 also shows the backoff state[2], represented by empty brackets, i.e. "( )". All arcs coming into the backoff state represent the backoff weight defined in Equation (2.16), again in log form — for instance, the arc emanating from the "get" state and terminating in the backoff state is defined as $\log\left(\alpha\left(\text{“get”}\right)\right) = -2.16$. Arcs that are *leaving* the backoff state represent the smoothed unigram (log) probabilities defined in Equation (2.17) — for example, the arc leaving the backoff state and arriving at the "payment" state is defined as $\log\left(\Pr^*\left(\text{“payment”}\right)\right) = -4.62$.

Given the test utterance "can i get my payment address please", Figure 2.1 shows the relevant section of the bigram backoff language model with some alternative paths that are connected to the chosen local states. The total log probability for the utterance given the WB model of Figure 2.1 is shown in Equation (2.21). This is calculated by transitioning from the start state through each of the word states that match the input using the bigram connections (solid arcs) and

[2]A bigram model only contains one backoff state, but for $N > 2$ there is a backoff state for each context. For example, a state which represents the trigram context "i want" is connected to an associated backoff state that represents "want".

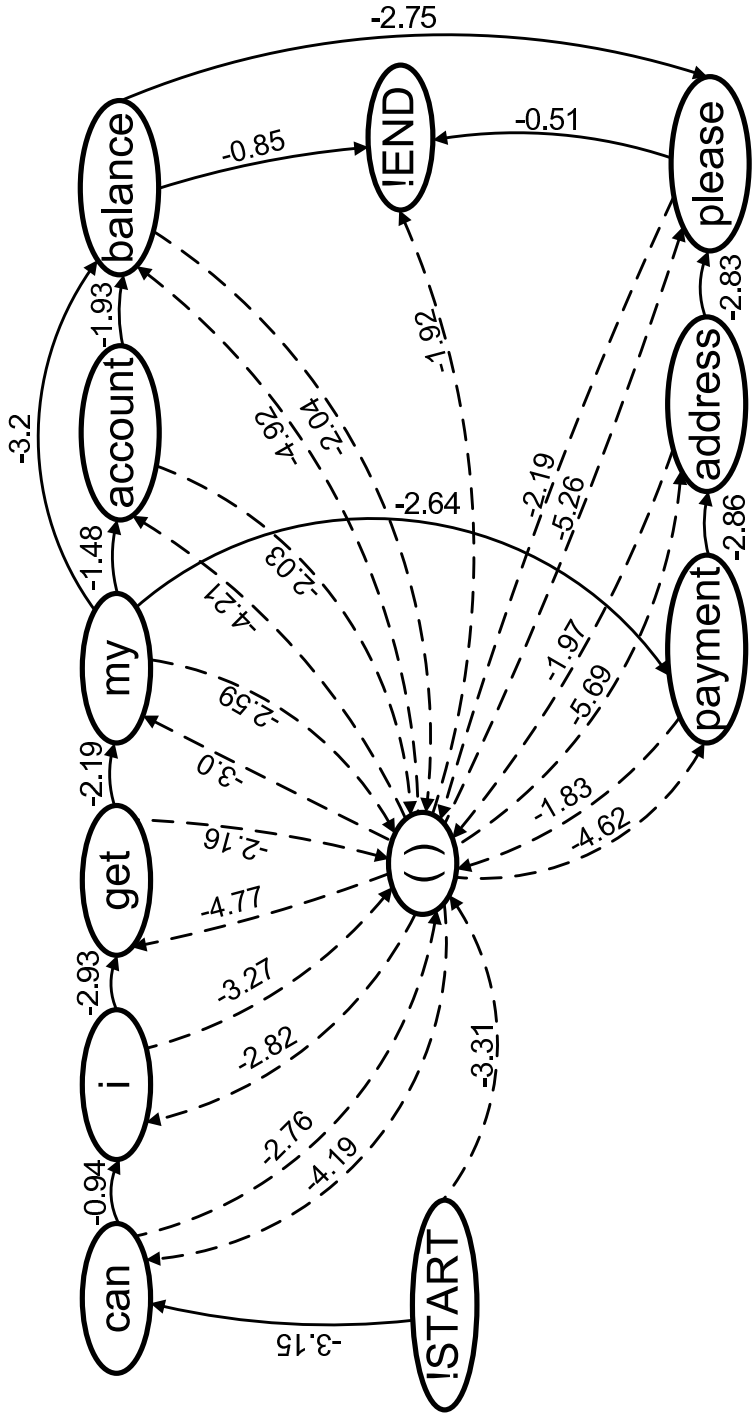


Figure 2.1: *Word-Bigram (WB) language model. Log-likelihoods are shown on the arcs between nodes, and the backoff state is represented by empty brackets "( )" with transitions to-and-from the backoff state shown in dashed lines.*

terminating in the end state.

$$\log(\Pr(\text{“can i get my payment address please”}|WB)) = \tag{2.21}$$
$$(-3.15) + (-0.94) + (-2.93) + (-2.19) + (-2.64) + (-2.86) +$$
$$(-2.83) + (-0.51) = -18.05$$

To take account of the log probabilities, Equation (2.20) is rewritten as:

$$\log\left(\Pr\left(w_1^m\right)\right) \approx \sum_{i=1}^{m} \log\left(\Pr_{katz}\left(w_i|w_{i-N+1}^{i-1}\right)\right) \tag{2.22}$$

Because the VNSA was designed to integrate a multiple-level N-gram language model with a speech recogniser, Equation (2.18) is not precisely how the language model is used to provide the prior probability of Equation (2.5). Equation (2.18) will only backoff to lower order N-gram if the current N-gram was not seen in the training data. The VNSA, which is used with Viterbi algorithm, uses a Maximum-Likelihood (ML) approach which means that all of the possible transitions are taken into account, i.e. the automaton is *non-deterministic*, with the best path chosen to be the one with the highest probability.

For example, following Figure 2.1, the bigram “get my” could be estimated by following the path from the “get” state to the “my” state, which has a log probability of -2.19; alternatively, the log probability could be the addition of the backoff weight and the unigram for “my”, i.e. moving from the “get” state to the backoff state, which gives the log of the backoff weight, and then following the path to the “my” state, which represents the unigram log probability of “my” — this gives a much lower log probability (-5.16), and so the bigram transition would be preferred by the Viterbi algorithm (Section 2.3.4).

2.2.3 Perplexity

The effectiveness of the methods described in Chapter 4 is evaluated using *perplexity* which is the standard metric for evaluating language models [Jurafsky and Martin, 2009]. The perplexity of a language or language model is the weighted average number of choices per word for some unseen text — the lower the perplexity is, the better the language model is. The perplexity of a language model $\mathscr{L}$ with respect to a dataset S where $|S|$ is the number of sentences in S is given by:

$$PP_{\mathscr{L}}(S) = \exp\left(-\frac{\log\left(\Pr\left(w_1^t\right)\right)}{t} \times \frac{1}{\log(2)}\right) \tag{2.23}$$

where

$$\log\left(\Pr\left(w_1^t\right)\right) = \sum_{i=1,m=|s_i|}^{|S|} \log\left(\Pr\left(w_1^m\right)\right) \tag{2.24}$$

and

$$t = \sum_{i=1}^{|S|} |s_i|, \tag{2.25}$$

where $|s_i|$ is the number of words in sentence s_i, and thus t is the total number of words in S, and $\log\left(\Pr\left(w_1^m\right)\right)$ is estimated from Equation (2.22).

2.3 Template-Based Recognition

Template-based speech recognition using *Dynamic Programming* (DP) was popular for ASR almost 40 years ago [Sakoe and Chiba, 1971], gaining strength from the late 1970s into the mid-1980's [Sakoe and Chiba, 1978; Sakoe, 1979; Rabiner and Shchmidt, 1980; Myers et al., 1980; Myers and Rabiner, 1981; Chamberlain and Bridle, 1983; Ney, 1984]. In the past few years, template-based ASR has seen a resurgence [De Wachter et al., 2003; Axelrod and Maison, 2004; Aradilla et al., 2005; De Wachter et al., 2007] in an attempt to offer improvements over the current HMM (Hidden Markov Model) approaches where progress appears to be stalling [Moore, 2003].

This section is structured as follows: Section 2.3.1 will give a formal definition of the *template*. Section 2.3.2 will describe between-frame distance measures that are required to match the stored templates to the input sequence. Section 2.3.3 describes the *Dynamic Time Warping* (DTW) algorithm which can be used to find the distance between a reference template and the input sequence for isolated speech recognition, with Section 2.3.4 introducing the *Token Passing* algorithm [Young et al., 1989] which can be used to find the *sequence* of templates that match the input for continuous speech recognition by introducing *between-template* distances to complement the *within-template* distances of the DTW.

2.3.1 Definition of a Template

A template in ASR is loosley defined as a sequence of acoustic feature vectors, or frames. In this thesis, we define a template as a sequence of frames from the training data, termed the *reference template*. The sequence of frames correspond to a defined unit of speech e.g. phone, syllable, word, or phrase. The frames that define the template appear consecutively in the training data, with each appearance of a given unit stored in a reference template database, i.e. there may be 200 examples of the word "i" and therefore the database will contain 200 "i"

templates. Figure 2.2 shows template definition at the phone, word, and phrase level.

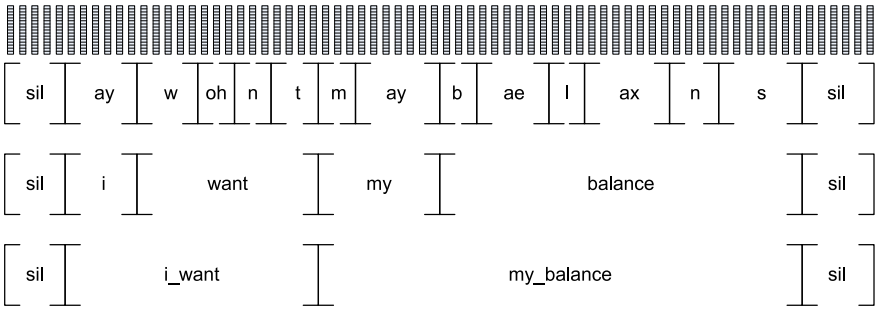


Figure 2.2: *Template definition. Templates at the phone, word, and phrase level for the same sequence of training frames.*

For the remainder of this thesis, a general reference template will be referred to as $\mathbf{Y}$, where the length of $\mathbf{Y}$, in terms of the number of frames that define it, is given as $|\mathbf{Y}|$, such that $\{\mathbf{y}_i \in \mathbf{Y} : 1 \leq i \leq |\mathbf{Y}|\}$. Additionally, *all* template-based experiments reported in this thesis use word and phrase-level templates, i.e. no sub-word templates are used.

2.3.2 Frame-Based Distance Measures

As mentioned previously, speech audio is converted into a stream of feature vectors, otherwise known as frames. Because the input is split into a sequence of frames, there needs to be some measure of similarity of the input frames to either stored models (HMMs in Section 2.4) or reference templates to perform the decoding.

A standard measure between two vectors is the Euclidean distance. The (squared) Euclidean distance between two vectors (frames) $\mathbf{x}$ and $\mathbf{y}$ of D dimensions is given as

$$d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{D} (x_i - y_i)^2 \tag{2.26}$$

which can be written in the general form

$$d(\mathbf{x}, \mathbf{y}) = (\mathbf{x} - \mathbf{y})^{\mathrm{T}} \Lambda (\mathbf{x} - \mathbf{y}) \tag{2.27}$$

where Λ is the identity matrix I. If $\Lambda = \sum^{-1}$, the inverse *diagonal covariance* matrix, then the *Mahalanobis* distance is obtained:

$$d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{D} \frac{(x_i - y_i)^2}{\sigma_i^2} \tag{2.28}$$

The Euclidean and Mahalanobis distance measures of Equations (2.26) and (2.28) are both examples of *global* distance measures where the space is transformed by a global transformation matrix Λ (which is the identity matrix I for the Euclidean distance and the inverse of the diagonal covariance matrix for the Mahalanobis distance). Bocchieri and Doddington [1986] showed that it is important to use a *locally* based distance measure in template based recognition. This distance can be estimated by defining an "average" template for each word, aligning every template example of that word to the average template, and hence estimating covariance information for each frame in each template.

HMM-based speech recognisers use local measures for each reference or *training* frame where each state in a HMM defines a probability density function (pdf) which gives a likelihood for a given input frame at that state (described further in Section 2.4). The multivariate Gaussian function is given by [Jurafsky and Martin, 2009]:

$$f(\mathbf{x}|\boldsymbol{\mu}, \sum) = \frac{1}{(2\pi)^{\frac{D}{2}} |\sum|^{\frac{1}{2}}} \exp\left(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^{\mathrm{T}} \sum^{-1} (\mathbf{x} - \boldsymbol{\mu})\right) \tag{2.29}$$

where $\mathbf{x}$ is the input frame, $\boldsymbol{\mu}$ is the mean vector at a given state, and $\sum$ is the

covariance matrix of a group of training frames at the given state ($\sum$ is often diagonal).

De Wachter et al. [2004] have shown that the Gaussian function (Equation (2.29)) can be equated to the distance function of Equation (2.27) by substituting the mean frame $\boldsymbol{\mu}$ for a single reference frame $\mathbf{y}$, taking the negative logarithm of the likelihood, and removing constants, such that $d(\mathbf{x}, \mathbf{y}) = -\log(f(\mathbf{x}|\mathbf{y}, \sum))$ is given as

$$d(\mathbf{x}, \mathbf{y}) = (\mathbf{x} - \mathbf{y})^{\mathrm{T}} \sum_c^{-1} (\mathbf{x} - \mathbf{y}) + \log(|\sum_c|) \tag{2.30}$$

where the reference frame $\mathbf{y}$ belongs to a class c from a set of M predefined classes, giving $\sum_c$ as the covariance matrix of class c. This is termed the *local Mahalanobis distance* and when $\sum$ is diagonal is defined as

$$d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{D} \left(\frac{x_i - y_i}{\sigma_{c,i}} \right)^2 + \log \left(\prod_{i=1}^{D} \sigma_{c,i}^2 \right) \tag{2.31}$$

where, $\mathbf{x}$ is the input frame, $\mathbf{y}$ is the reference frame for a class c, and $\sigma_{c,i}^2$ is the variance, for dimension i of D, of all reference frames contained in c.

Equations (2.30) and (2.31) both assume that all reference frames $\mathbf{y}$ in the reference database $\mathbf{R}$ can be grouped into M classes. The partitioning of the reference frames into M classes can be done using a HMM *state alignment* which outputs the state that each training frame is distributed to for pdf estimation — training frames in the same model (HMM) and state are grouped into the same class for covariance estimation.

De Wachter [2007] uses a context-independent phone HMM recogniser (i.e. monophones) to define the classes for the TIMIT dataset, which is small dataset of "phonetically-balanced sentences" [De Wachter, 2007], while a context-dependent

phone (HMM) recogniser (i.e. bi-phones, tri-phones) is used to define the classes for both the RM dataset (described in Section 3.3) and Wall Street Journal (WSJ) corpus, which is a set of read sentences from the Wall Street Journal newspaper. Both the RM and WSJ datasets are larger in size than TIMIT (in terms of the amount of training data). In all cases, the number of state classes is doubled by integrating *gender* information into the class distribution — each state class is partitioned into *male* and *female* classes. In this thesis, all classes of reference frames are defined using the state alignment from a context-independent mono-phone HMM recogniser and experiments are reported with and without the use of gender information. The number of classes without gender information is 135 for the Speaker-Dependent (SD) set (from a set of 44 phonemes + silence, described in Section 3.2) and 144 for the RM dataset (from a set of 47 phonemes + silence, described in Section 3.3) with the number of classes for the RM set increasing to 285 when splitting states with gender information (the silence states are not split, hence the number is not exactly doubled).

2.3.3 Dynamic Time Warp (DTW)

The *Dynamic Time Warping* (DTW) algorithm is a *dynamic programming* approach to matching two sequences of frames of (potentially) different lengths [Rabiner and Schafer, 1978]. Specifically, in *isolated* speech recognition, the task is to match the input sequence of frames $\mathbf{X}$ to all stored reference templates $\mathbf{Y} \in \mathbf{R}$, where $\mathbf{R}$ represents the reference template database, and find the closest matching reference template.

The DTW match is performed by "squashing" and "stretching" the reference template so that it is "time-aligned" with the input sequence (i.e. the template is *warped* to match the length of the input sequence) and then finding the distance between the two sequences using a between-frame distance such as the Euclidean distance (Equation (2.26)) or the local Mahalanobis distance measure (Equation (2.31)). Typically local constraints are enforced upon the DTW algorithm that

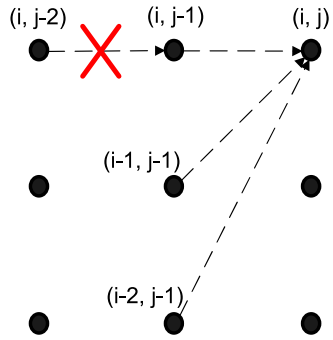


Figure 2.3: *Itakura constraints. A transition to point* (i, j) *is accepted if originating from points (i,j-1), (i-1,j-1), or (i-2,j-1). However, if transitioning from point (i,j-1), the previous transition* to *(i,j-1) can not originate from point (i,j-2), i.e. consecutive stalls are not allowed.*

limits the amount of "squashing" and "stretching" upon the template — a popular choice of constraints are the *Itakura* constraints [Itakura, 1975], as illustrated in Figure 2.3.

The Itakura constraints allow three kinds of movement in the warp: a *diagonal* move which means that there is no warp at that point in the template, a *stall* move which keeps the alignment at the same frame of the template (a "stretch" of the template), and a *skip* move which moves two frames through the template (a "squash" of the template). The *stall* constraint also has an additional constraint that does not allow two consecutive stalls in the warp. The DTW is formulated as a search for the best warping path through a matrix $\mathbf{D}$ of between-frame distances (template to input) that are appended with the relevant warping costs — at each point in the matrix (starting at $\mathbf{D}_{1,1}$ and ending at $\mathbf{D}_{|\mathbf{Y}|,|\mathbf{X}|}$, i.e. the start and end frames of $\mathbf{X}$ and $\mathbf{Y}$), the move (diagonal, stall, or skip) which gives the minimum distance to the input is chosen. The best warping path can be found by *backtracking* from the final element of the matrix $\mathbf{D}_{|\mathbf{Y}|,|\mathbf{X}|}$ by following the warping moves that are stored during the main iteration. Algorithm 2.1 details the steps of the DTW algorithm.

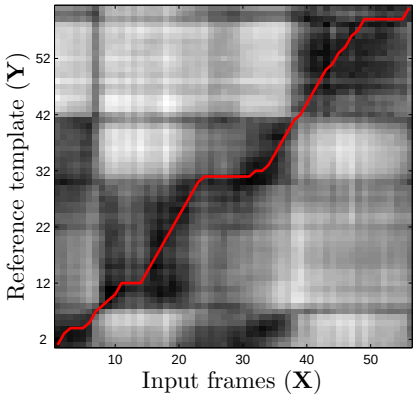
The results of matching a short input utterance "my balance" to a template "my balance", both spoken by the same speaker, are shown in Figure 2.4. Figures 2.4(a) and 2.4(b) both show the between-frame distance matrix for the reference

Algorithm 2.1 DTW algorithm with Itakura Constraints

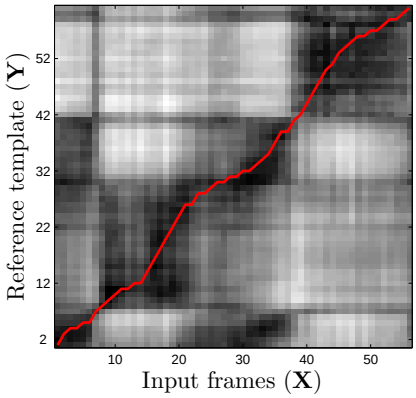
1: {Initialisation — construct pairwise distance matrix $\mathbf{M}$}
2: **for** $i = 1$ to $|\mathbf{Y}|$ **do**
3: **for** $j = 1$ to $|\mathbf{X}|$ **do**
4: $m_{i,j} = d(\mathbf{y}_i, \mathbf{x}_j)$
5: **end for**
6: **end for**
7: $\mathbf{D} = \mathbf{M}$
8:
9: {Calculate alignment cost matrix}
10: {D_Cost, H_Cost, and S_Cost are costs for diagonal, stall, and skip moves respectively.}
11: **for** $i = 1$ to $|\mathbf{Y}|$ **do**
12: **for** $j = 1$ to $|\mathbf{X}|$ **do**
13: **if** $\mathbf{D}_{i,j-1}$ not from stall **then**
14: $\mathbf{D}_{i,j} = \mathbf{D}_{i,j} + \min(\mathbf{D}_{i-1,j-1} + \text{D\_Cost}, \mathbf{D}_{i,j-1} + \text{H\_Cost}, \mathbf{D}_{i-2,j-1} + \text{S\_Cost})$
15: **else**
16: $\mathbf{D}_{i,j} = \mathbf{D}_{i,j} + \min(\mathbf{D}_{i-1,j-1} + \text{D\_Cost}, \mathbf{D}_{i-2,j-1} + \text{S\_Cost})$
17: **end if**
18: Store move to $\mathbf{D}_{i,j}$
19: **end for**
20: **end for**
21: Backtrack from $\mathbf{D}_{|\mathbf{Y}|,|\mathbf{X}|}$ to $\mathbf{D}_{1,1}$ using stored moves for best alignment.

template $\mathbf{Y}$, represented on the y-axis, and the input sequence $\mathbf{X}$, shown on the x-axis. The darker the shading is, the closer the frames that align in that matrix element are (in terms of Euclidean distance) — the lighter the area, the further the frames are from each other. Figure 2.4(b) shows the best warping path (red line) using the previously described Itakura constraints, whereas the best warping path in Figure 2.4(a) does not use the stall constraint, i.e. the difference between Figures 2.4(a) and 2.4(b) is that the DTW in Figure 2.4(a) allows consecutive stalls — this is illustrated by the more discrete nature of the best path compared to that of the full Itakura constraints in Figure 2.4(b). Figure 2.4(c) shows the best path set against the allowable paths when using Itakura constraints. It also displays, what is termed here, the *alignment cost* matrix $\mathbf{D}$ which is the between-frame distances of the template and test utterance, summed with the warping costs and local path distances (each point in the matrix represents the total distance of the

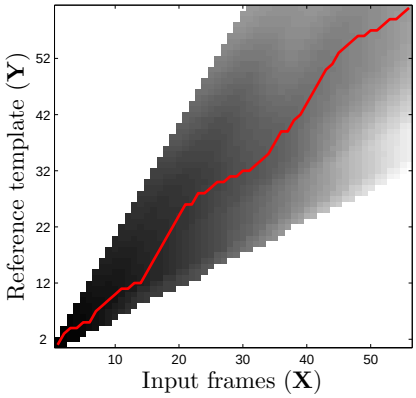
path from $\mathbf{D}_{1,1}$ to that point) — again darker entries represent smaller distances.



(a) *Between-frame distance matrix with no stall constraint.*



(b) *Between-frame distance matrix with full Itakura constraints.*



(c) *DTW alignment cost matrix showing allowable entries with full Itakura constraints.*

Figure 2.4: *DTW of the reference template* $\mathbf{Y}$ *to the input sequence* $\mathbf{X}$ — *both sequences are two instances of "my balance" uttered by the same speaker.*

2.3.4 Token Passing Algorithm

Section 2.3.3 was concerned with the DTW algorithm for *isolated* speech recognition, and thus each reference template is matched to the whole of the input utterance. This is useful for applications such as recognition of isolated digits, but the work presented in this thesis is concerned with *continuous* speech recognition where the best matching *sequence* of templates to the input is required.

The *Token Passing* algorithm formulates the DTW into an abstract process of passing tokens around a transition network [Young et al., 1989] which then simplifies the extension to continuous speech recognition. Each template in the reference database $\mathbf{R}$ is represented, as illustrated in Figure 2.5, as a sequence of connected states, where each state represents a *single* frame of the template, and connecting arcs between states represent the different costs associated with DTW. The Itakura constraints (Figure 2.3) are built into the network explicitly, with arcs representing the previously discussed diagonal, stall, and skip moves. The constraint on consecutive stalls cannot be built into the network, but this can be determined by examining the *path history* of a token which stores the states visited by that token.

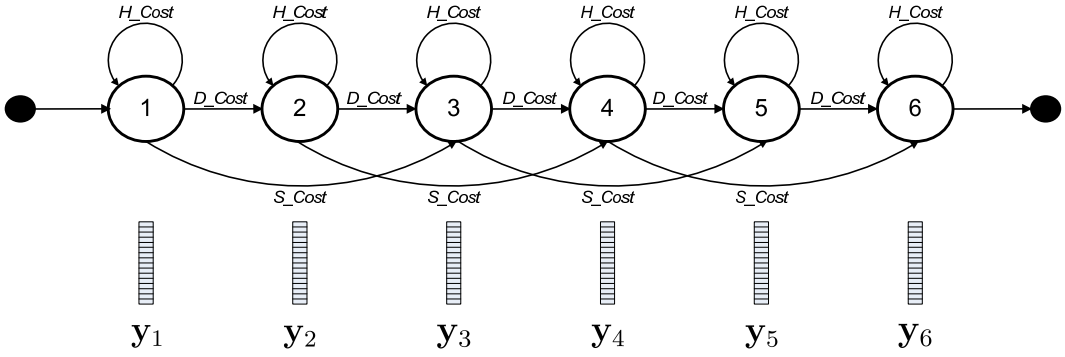


Figure 2.5: *Template formulated as a series of connected states with transition costs. Each state represents one frame* $\mathbf{y}_i$ *from template* $\mathbf{Y}$*, with an additional start and end state (all black) which are used to connected templates together in recognition.*

A simple model for continuous speech recognition is to allow all templates in $\mathbf{R}$ to follow any other template during the decoding process. This can be modelled using an *ergodic* network which is illustrated in Figure 2.6 — this simple example

only contains four templates in $\mathbf{R}$, with each state in the ergodic network an abstract representation of each template state model as defined in Figure 2.5. Each template is connected to another template via the start and end states with the arcs connecting them holding zero cost.

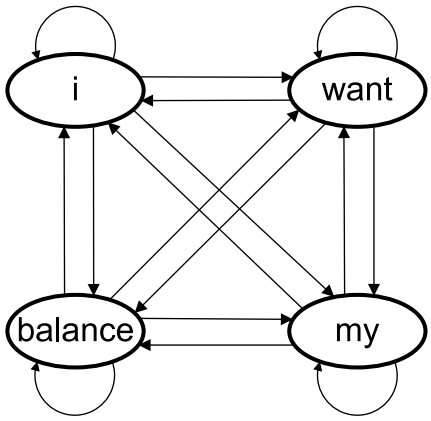


Figure 2.6: *A template ergodic network. A simple example of an ergodic network that allows any template to follow any other template for continuous speech recognition — this simple example contains just four templates. Each state is an abstract representation of a template which is actually represented as in Figure 2.5.*

Once the transition network (template state network + ergodic network) is defined, each first state of each template is *initialised* with a *token* object which is updated with the between-frame distance of the frame that the state represents (frame 1 in this case) and the current input frame $\mathbf{x}_t$ using a suitable distance measure (e.g. Equations (2.26) or (2.31)); an identifier representing the state is stored in the token's *path history*. The main token passing algorithm then begins, and is an iteration over the input frames $\mathbf{X}$, where $|\mathbf{X}| = T$, where a copy of each token is made and then passed to all connecting states, incrementing the current distance of the token with the transition cost along the arc that the token travelled, and the distance of the next state's frame to the current input frame; again the receiving state's identifier is stored in the path history. Figure 2.7 illustrates this.

It should be noted that each transition cost is now notated abstractly as a_{ij} where i represents the state that the token is sent from and j represents the state that receives the token, and that the distance between an input frame and a reference frame is notated as $b_j(\mathbf{o}_t)$ where j is the template state (and thus

represents frame $\mathbf{y}_j$) and $\mathbf{o}_t$ is the current input frame or *observation* ($b_j(\mathbf{o}_t)$ is equivalent to $d(\mathbf{x}_t, \mathbf{y}_j)$ as in Equations (2.26) and (2.31)). This change of notation makes the token passing algorithm more abstract and is required in Section 2.4 which shows how the token passing algorithm can also be applied to HMM-based speech recognition.

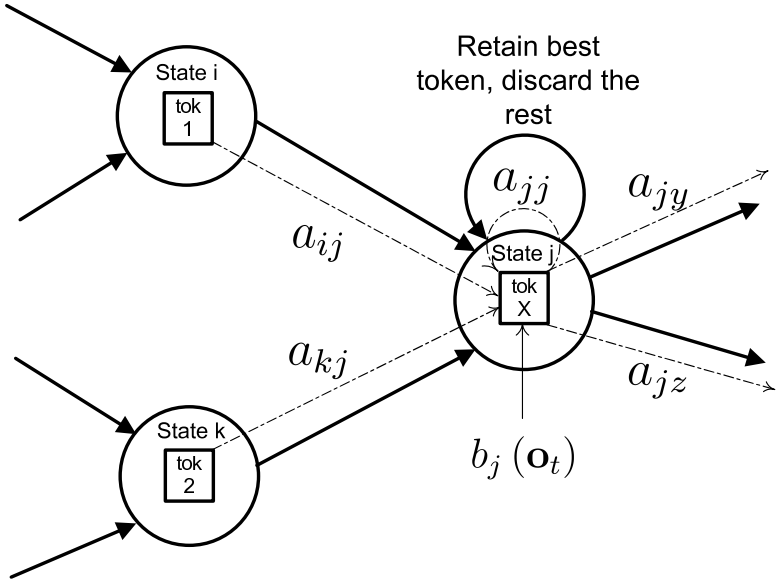


Figure 2.7: *Token Passing. State j receives tokens 1 and 2 from states i and k respectively, and updates each token with the respective transition cost a_{ij} and a_{kj}, as well as $b_j(\mathbf{o}_t)$ which is the distance between reference frame j and input frame t. The token with the lowest score at state j is retained with all remaining tokens discarded.*

Once all tokens have been copied to connecting states and updated with the relevant between-frame distances and transition costs, the original tokens that were copied are discarded. The next step is to, at every state, discard all but the best token, where the *best* token is defined as the one with the lowest distance. This step is equivalent to lines 14 and 16 in Algorithm 2.1, which selects the best path from the previous steps of the algorithm to the current point. The algorithm continues to iterate over the input frames with the best path chosen to be the token with the lowest distance from the remaining tokens after all input frames are processed. The best *template sequence*, which is the goal of the recognition process, can be found by *backtracking* through the path history of the best token. Algorithm 2.2 summarises the token passing process.

Algorithm 2.2 Token Passing Algorithm

1: {*Main iteration of token passing algorithm*}
2: **for** $t = 1$ to T **do**
3: **for all** states i **do**
4: Pass a copy of the token in i to all connecting states j.
5: Update the tokens distance by adding $a_{ij} + b_j(\mathbf{o}_t)$ to it.
6: Add i to the tokens path history.
7: **end for**
8: Discard the original tokens.
9: **for all** states i **do**
10: Search through the tokens at i and discard all except the one with the lowest distance.
11: **end for**
12: **end for**
13:
14: Find best token and backtrack through path history for the best warping path and template sequence.

2.4 HMM-Based Recognition

Hidden Markov Models (HMMs) have been the primary technique for speech recognition research, taking over from DTW template-recognition in the mid-to-late 1980's [Rabiner, 1989]. The HMM is a connected state model which is used to represent speech sounds, such as the phone, by training pdfs and transition probabilites from some training data. Unlike template-based recognition, there is usually only one model to represent a sound, with each example of that sound in the training data used to train the model parameters using the *forward-backward* algorithm [Baum, 1972]. Figure 2.8 shows a typical topology for a HMM phoneme model with three *emitting* states[3], each modelled with a Gaussian Mixture Model (GMM) which is described below.

A simple HMM may use just a single multivariate Gaussian function at each state, which was previously defined in Equation (2.29). During decoding, the pdfs at each state are used to calculate an *observation likelihood* $b_j(\mathbf{o}_t)$ for an input

[3]The start and end state are used to link different HMMs together during decoding and do not model any part of the speech sound

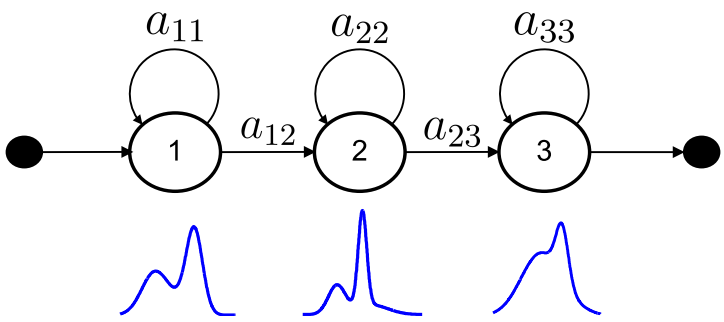


Figure 2.8: *HMM topology. The HMM has three emitting states (states 1,2,a and 3), where each state's output is modelled by a GMM. Transition probabilities are given on each arc between the states.*

frame $\mathbf{o}_t$ at a given HMM state j, which is re-formulated from Equation (2.29) as:

$$b_j(\mathbf{o}_t) = \frac{1}{(2\pi)^{\frac{D}{2}} |\sum_j|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{o}_t - \boldsymbol{\mu}_j)^{\mathrm{T}} \sum_j^{-1} (\mathbf{o}_t - \boldsymbol{\mu}_j)\right) \tag{2.32}$$

However, using a single Gaussian pdf at each state assumes that the distribution of the training data is *Normal* when this might not be the case. Often the observation likelihood is modelled by *mixing* more than one multivariate Gaussian together at each state to give a *Gaussian Mixture Model* (GMM), where the multivariate Gaussian pdfs are mixed by a weighted summation, given by

$$b_j(\mathbf{o}_t) = \sum_{m=1}^{M} w_m \frac{1}{(2\pi)^{\frac{D}{2}} |\sum_{jm}|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(\mathbf{o}_t - \boldsymbol{\mu}_{jm})^{\mathrm{T}} \sum_{jm}^{-1} (\mathbf{o}_t - \boldsymbol{\mu}_{jm})\right) \tag{2.33}$$

where there are M mixture components each with an associated weight w_m. The forward-backward algorithm can again be applied to train the HMMs using GMMs.

2.4.1 Decoding with Token Passing

Section 2.3.4 described how the token passing algorithm [Young et al., 1989] is used in the decoding process for template-based continuous speech recognition. The

templates were formulated as N-state transition models, where N is the number of frames in the given template, each with transition costs a_{ij} from state i to state j along each arc. The distance between the input frame and the reference frame was also reformulated as $b_j(\mathbf{o}_t)$ to map to the equivalent measure in the HMM systems, i.e. the pdf likelihood from Equations (2.32) and (2.33). In HMM-based systems, the transition costs a are actually probabilities of moving between states in the HMM, and are trained during the forward-backward algorithm, which also trains the observation likelihoods. Figure 2.7 is the common topology for the HMM and DTW-based template systems.

Once the transition probabilities and observation likelihoods are trained for the HMMs, the decoding process is exactly the same as with the DTW template-based decoder, i.e. an implementation of the token passing algorithm as defined by Algorithm 2.2, except that the distances are reformulated as log probabilities. This gives Algorithm 2.3. The "best" token is defined now by the token with the highest log probability.

Algorithm 2.3 Token passing with probabilities, not distances

1: **for** $t = 1$ to T **do**
2: **for all** states i **do**
3: Pass a copy of the token in i to all connecting states j.
4: Update the tokens logprob by adding $\log a_{ij} + \log b_j(\mathbf{o}_t)$ by it.
5: Add i to the tokens path history.
6: **end for**
7: Discard the original tokens.
8: **for all** states i **do**
9: Search through the tokens at i and discard all except the one with the *highest* logprob.
10: **end for**
11: **end for**
12:
13: Find best token and backtrack through path history for the best state sequence.

HTK (HMM toolkit) [Young et al., 2009] is a popular toolkit to build HMM-based speech recognisers and then to decode an input sequence using trained HMMs (using an implementation of the token passing algorithm). HTK was used

for all HMM-based speech recognition experiments reported in this thesis. This section has provided a very brief overview of HMM-based speech recognition; the interested reader should refer to Rabiner [1989] , for example, for a more in-depth walk-through of techniques used in speech recognition, or alternatively Gales and Young [2007] which offers a more up-to-date description.

2.5 Integrating the Language Model into the Recogniser

Sections 2.3 and 2.4 were concerned with the definition of the template and HMM, describing how both approaches to speech recognition can be performed when using an *ergodic* network, i.e. a network in which any template or HMM can follow any other template or HMM when decoding, including itself. Typically in speech recognition, as defined and described in Section 2.2, a language model is used to constrain the sequence of templates or HMMs that is recognised.

To integrate the language model with the templates or HMMs, the language model is viewed as being the top level of a hierarchy, with the templates or HMMs at the bottom. Figure 2.9 shows how this hierarchy works for a small section of a language model. The example uses phoneme HMMs, and thus the integration with the language model requires a *pronunciation dictionary* where the sequence of phonemes that define the pronunciation of each word is given. The templates used in this example are at the *word* level, and for a simple template-based system, every template representing a given word in the language model is loaded in parallel. A backoff mechanism is also included in the figure for completeness.

The template-based system requires a further adjustment; the distances that are output by the DTW need to be converted to log-likelihoods to fit correctly with the language model probabilities (the Figure shows the language model transitions as probabilities, but in reality these are converted to log probabilities). In Section 2.3.2 it was shown how the distance measure of Equation

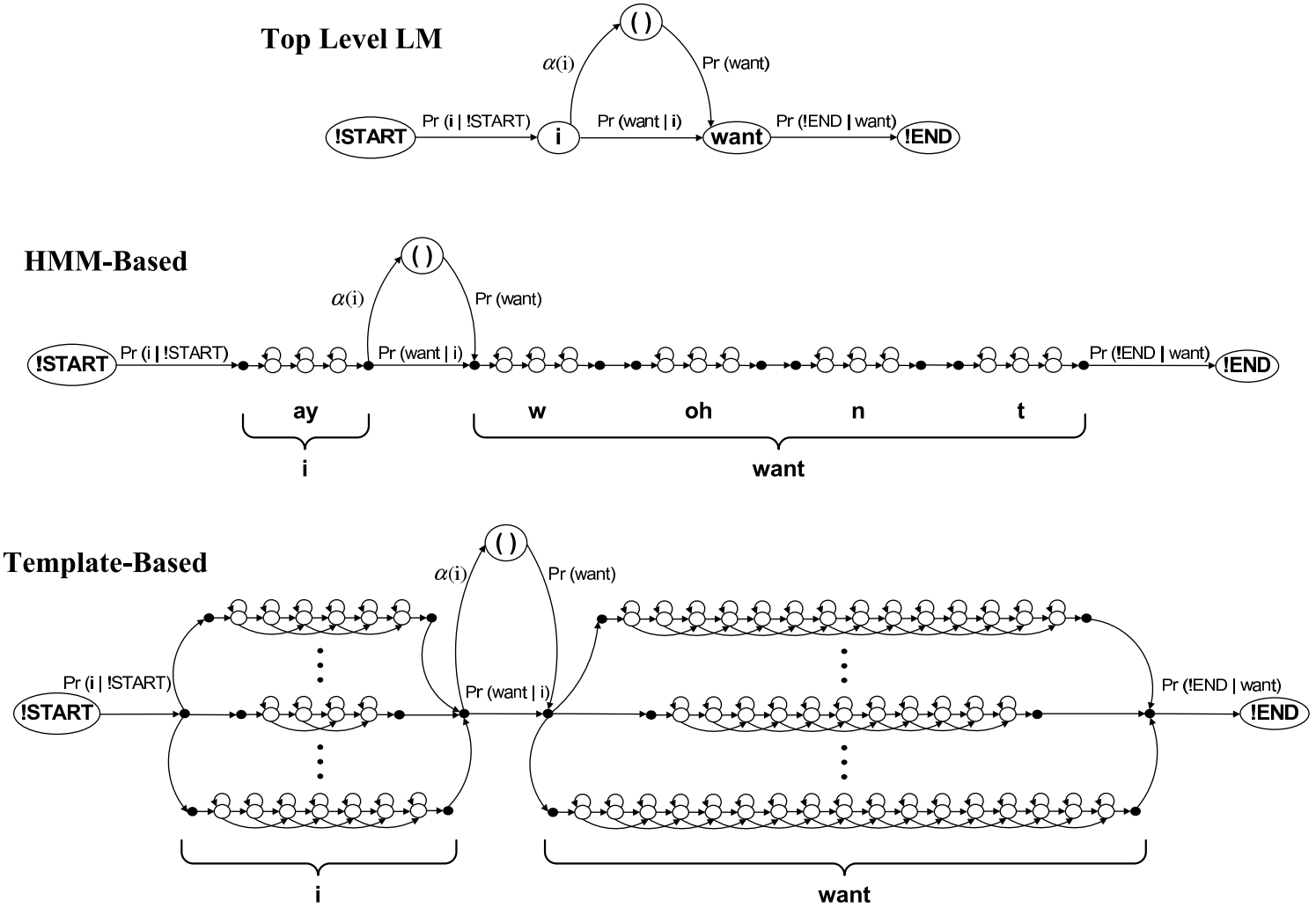


Figure 2.9: *The language model (LM) and HMM / Template Hierarchy. Shows the integration of the LM with the HMM-based system and the template-based system.*

(2.27) can be equated to the multivariate Gaussian function of (2.29), such that $d(\mathbf{x}, \mathbf{y}) = -\log(f(\mathbf{x}|\mathbf{y}, \sum))$. Thus, it can clearly be seen that by making the distance of Equation (2.31) negative an equivalent log-likelihood measure ll is found, such that

$$ll = -d(\mathbf{x}, \mathbf{y}) = -\left[\sum_{i=1}^{D}\left(\frac{x_i - y_i}{\sigma_{c,i}}\right)^2 + \log\left(\prod_{i=1}^{D}\sigma_{c,i}^2\right)\right] \tag{2.34}$$

Finally, there are two more methods that are used when integrating the language model with the speech "models": a *language model scaling factor* (LMSF) and a *word insertion penalty* (WIP) [Jurafsky and Martin, 2009]. The LMSF is used to balance the probabilities between the language model and the acoustic models by globally scaling all of the language model probabilities by the LMSF. The WIP is required after the use of the LMSF because by adjusting the language model probabilities, the natural "penalty" (e.g. word bigram probability) of transiting from one word to another is reduced, and thus it is likely that the decoder will prefer a higher number of shorter words (or templates). By adding in an extra penalty (WIP) for word transitions, the effect of the LMSF can be rebalanced.

Equation (2.35) gives an updated version of Equation (2.5) to include the LMSF and the WIP [Jurafsky and Martin, 2009]:

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}\in\mathscr{L}}{\operatorname{argmax}} \Pr(\boldsymbol{X}|\boldsymbol{w}) \Pr(\boldsymbol{w})^{LMSF}\ WIP^N \tag{2.35}$$

where N is the number of words in the word sequence $\boldsymbol{w}$. In reality, the final term in Equation (2.35), WIP^N, is integrated into the decoding networks (Figure 2.9) by taking the log of it and adding it to each arc of the LM level.

2.6 Vocal Tract Length Normalisation (VTLN)

In Chapter 6, the results of the methods described in Chapters 4 and 5 will be presented for template-based recognition, including experiments on the speaker-independent RM dataset (described in Section 3.3). A popular technique used for *speaker-independent* speech recognition is called *Vocal Tract Length Normalisation* which is used to compensate for the fact that different speakers have different length vocal tracts — men typically have longer vocal tracts than women and hence their formants are, on average, lower in frequency. The average length of the vocal tract[4] for a male is 16.9 cm, while the average length for a female is 14.1 cm [Stevens, 2000]. By minimising the effects of the vocal tract, the difference between two different speakers can be reduced.

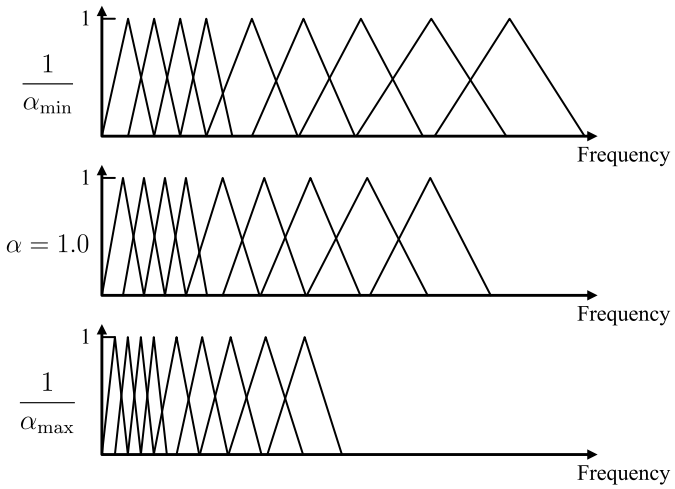


Figure 2.10: *The effect of VTLN on the Mel-Scale filterbank. The unwarped filterbank in the middle is expanded with a smaller warp factor $\alpha_{\min}$ (top) and compressed with a larger warp factor $\alpha_{\max}$ (bottom). The expanded filterbank (top) will typically represent a female's voice, while the compressed filterbank (bottom) will typically represent a male's voice.*

Experiments and results reported in Chapter 6 use a simple piecewise linear warping function that is applied in the frequency domain during filterbank analysis as part of the feature extraction process [Hain et al., 1999], with the implementation that is contained in HTK [Young et al., 2009] used for reported

[4]The average lengths for the vocal tract are given assuming that that the larynx and lips are in a *neutral* position.

experiments. Each *speaker* is normalised by "warping" the frequency axis in the filterbank analysis stage by the *inverse* of a *warping factor* α. Figure 2.10 shows how the Mel-Scale filterbank is compressed and expanded (warped) depending on the value of α. The top filterbank represents a low warp factor (and hence high inverse value) which typically represents a female's voice, while the bottom filterbank represents a high warp factor (and hence low inverse value) which typically represents a male's voice. The middle filterbank represents the original unwarped Mel-Scale filterbank.

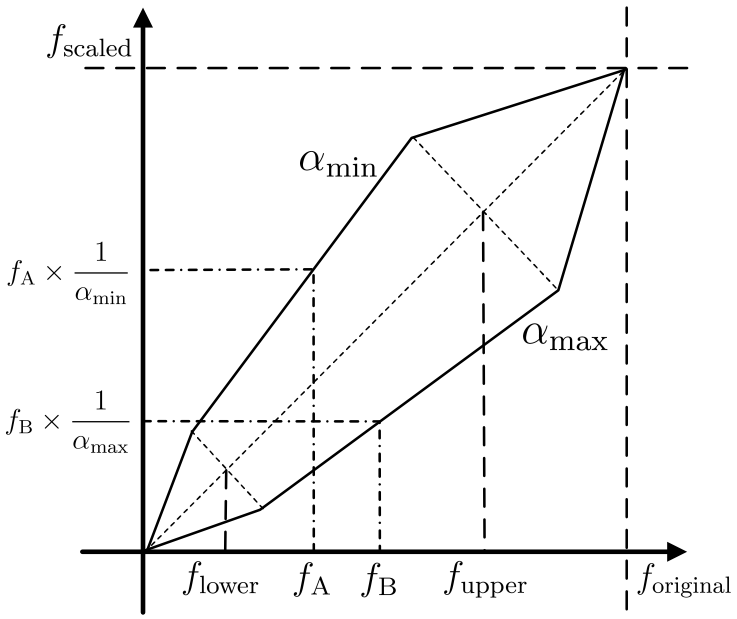


Figure 2.11: *Piecewise linear warping function. The original frequency axis* f_{original} *is scaled by the inverse of the warp factor* α. α_{min} *and* α_{max} *define the range of warp factors.* f_{upper} *and* f_{lower} *define the region of the frequency axis that is scaled. Points* f_{A} *and* f_{B} *are examples to show the effects of the warping function. This image is an adapted version of the one found in Hain et al. [1999].*

Figure 2.11 shows the warping function diagrammatically. The warping of the frequency axis is applied to the axis between two pre-defined points, f_{upper} and f_{lower} which are upper and lower cut-off frequencies that are used as controls to keep filters within the frequency range. Figure 2.11 shows two example frequencies on the original axis, f_{A} and f_{B} warped by two different warp factors (α_{min} and α_{max} respectively).

2.6.1 Finding Optimal Warp Factors

The warp factor for each speaker is estimated using the technique described in Lee and Rose [1998] for HMM-based recognition. The warp factor is chosen to be one of 13 factors in the evenly spaced range $0.88 \leq \alpha \leq 1.12$ which is chosen to "reflect the 25% range in vocal tract lengths found in adults" [Lee and Rose, 1998]. For the training data, the optimal warp factor is estimated for each speaker, with all warped training utterances used to train a final set of HMMs (described in Section 2.6.1.1), while for the recognition process, the optimal warp factor is estimated for each *utterance* (described in Section 2.6.1.2).

2.6.1.1 Training Procedure

To find the optimal warping factors for the training data, the training utterances for each speaker are split into two equal sets, T and A. T becomes an initial *training* set, while A becomes the *alignment* set. A set of HMMs λ (monophone HMMs with a single component Gaussian pdf at each state for the application in this thesis, which follows Lee and Rose [1998]) are built and trained for set T, and then used in a *forced-alignment*[5] of the utterances in set A for each warp factor, with the best warp factor for each speaker chosen to be the one which results in the highest probability from the forced-alignment, such that the best warping factor $\hat{\alpha}_i$ for speaker i is

$$\hat{\alpha}_i = \underset{\alpha}{\operatorname{argmax}} \Pr\left(\mathbf{X}_i^{\alpha} | \lambda_T, W_i\right) \tag{2.36}$$

where $\mathbf{X}_i^{\alpha}$ are the set of all utterances for speaker i in set A warped by α, λ_T are the set of HMMs trained on set T, and W_i are the word transcriptions for $\mathbf{X}_i^{\alpha}$. Sets T and A are then swapped and the training and forced-alignment process is

[5]Forced-Alignment is where the transcription of the input is already known, and so the process is to find the correct boundaries between the sequence of models (HMMs or templates).

repeated (with set A now the training set and set T the alignment set) and iterated until the warping factors converge, i.e. there is no *significant* change in the warping factors. At this stage, a final set of HMMs λ_F are trained on the optimally warped training utterances (using the optimal warping factors from Equation (2.36)).

2.6.1.2 Recognition Procedure

In the recognition process, i.e. with test data, each utterance is warped separately (as the speaker identity is not known) using a similar technique to that used in the training procedure described in the previous section. An initial transcription of the test utterance is retrieved by passing the *unwarped* utterance X_j through the recogniser using the normalised models λ_F. This *hypothesised* string W_j is then used as the transcription for a forced-alignment of X_j for the set of warping factors to find the best warping factor $\hat{\alpha}_j$ for utterance X_j, such that

$$\hat{\alpha}_j = \underset{\alpha}{\operatorname{argmax}} \Pr\left(\mathbf{X}_j^{\alpha} | \lambda_F, W_j\right) \tag{2.37}$$

choses the best warping of utterance X_j to be the one with the highest probability when aligned using the normalised HMMs λ_F. The warped utterance $X_j^{\hat{\alpha}}$ is then passed through the recogniser using the λ_F models to give the final hypothesised string $\hat{W}_j$ for utterance j.

2.7 Linear Discriminant Analysis (LDA)

Linear Discriminant Analysis (LDA) [Webb, 2002; Duda et al., 2001] is a method which is applied to n data samples belonging to K classes that linearly projects the samples into a $K-1$ dimensional space that best separates the data by maximising the ratio of the *between-class* scatter to the *within-class* scatter. Section 5.4 is concerned with data of only two classes, a special case known as the *two-class*

problem, where $K - 1 = 1$, so the data is projected onto a line (see Figure 2.12).

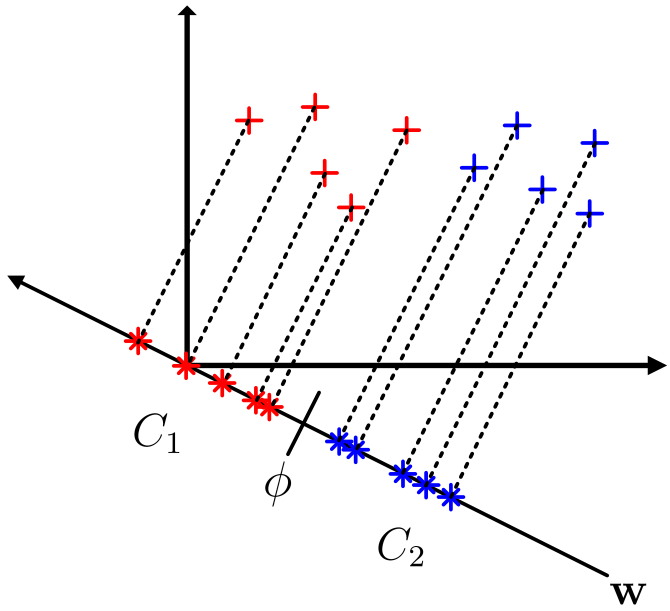


Figure 2.12: *LDA projection of data onto a 1D line defined by the vector* $\mathbf{w}$. *Samples to the left of (or above) a threshold* ϕ *are allocated to class* C_1 *and samples to the right of (or below)* ϕ *are allocated to class* C_2.

For the two-class problem, we seek to find the projection $\mathbf{w}$ which maximises

$$J(\mathbf{w}) = \frac{|\mathbf{w}^{\mathrm{T}}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)|^2}{\mathbf{w}^{\mathrm{T}}\mathbf{S}_W\mathbf{w}} \tag{2.38}$$

where $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$ are the sample mean for classes C_1 and C_2 , which contain n_1 and n_2 samples respectively. $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$ are defined as

$$\boldsymbol{\mu}_i = \frac{1}{n_i}\sum_{\mathbf{x}\in C_i}\mathbf{x} \tag{2.39}$$

and $\mathbf{S}_W$ is the within-class scatter matrix defined as

$$\mathbf{S}_W = \sum_{i=1}^{2} \sum_{\mathbf{x} \in C_i} (\mathbf{x} - \boldsymbol{\mu}_i) (\mathbf{x} - \boldsymbol{\mu}_i)^{\mathrm{T}} \tag{2.40}$$

Because it is the *direction* of $\mathbf{w}$ that is required and *not* the magnitude, the projection for the two-class problem can be simplified from a generalised eigenvalue problem (in the multi-class case) [Duda et al., 2001] to

$$\mathbf{w} = \mathbf{S}_W^{-1} (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \tag{2.41}$$

where the projected sample $\mathbf{y}$ of sample $\mathbf{x}$ is given by:

$$\mathbf{y} = \mathbf{w}^T \mathbf{x} \tag{2.42}$$

Once all of the samples have been projected onto the line, a simple classifier can be formed by selecting a *threshold* ϕ at some point on the projection line and allocating all projected samples that lie to one side of the threshold to one class, and all projected samples that lie to the other side of the threshold to the second class (again, refer to Figure 2.12). The choice of ϕ will be discussed further in Section 5.4.2.

Chapter 3

Dataset Description

3.1 Introduction

This chapter describes two datasets used for this work — one is a speaker-dependent set originally used for an experimental call-routing system (Section 3.2), and the other is the speaker-independent Resource Mangagement (RM) dataset (Section 3.3). A description of the feature extraction process (into MFCC vectors) is given in Section 3.4, with Section 3.5 describing the HMM baseline systems. The chapter ends with a comparison of the templates for each dataset in Section 3.6.

3.2 Speaker-Dependant Call-Routing Data

The first dataset used in this work consisted of transcriptions of telephone calls to an experimental call-routing system. Customers were invited to call up the system and to make the kind of enquiry they would normally make when talking to an operator. Only their initial query utterance was transcribed. The transcriptions were divided into a training-set of 4773 utterances and a testing set of 902. The training set vocabulary size is 1504 words, and the test-set size is 569 words after removal of any utterances that contained out-of-vocabulary (OOV) words.

The utterances themselves were of low quality because of factors such as restricted bandwidth, noise and distortions. Because the low accuracy obtainable from recognising this material could disguise the effects of the formulaic language that is being investigated here, a single speaker re-recorded the transcriptions using high quality recording equipment. All original disfluencies in the speech, such as pauses, repetitions, and grammatical errors, were retained in the recordings. The original dataset, although read in (American) English, contained speakers of foreign nationality and so there were observed grammatical errors such as "i wanna know if if *what i do if* my card has been stolen" — as stated before, these grammatical errors were retained as much as possible.

This data was chosen because it contains many commonly occurring phrases that act as whole utterances, such as "I would like my account balance" (54 instances) and "Can you give me my account balance" (31 instances). If minor variants of theses phrases are included (e.g. the addition of "please", substitution of "I'd" for "I would" etc.), then a few phrases account for a high percentage of the utterances.

	Training data	**Test data**
Total length	3.5 h	40 min
# utterances	4773	902
# words	51,083	8639
# unique words	1504	569

Table 3.1: *SD call-routing data information. Shows the length of the training and test data in hours and minutes, the total number of utterances, and the total number of words and unique words.*

Table 3.1 presents the information about the call-routing data such as number of hours of speech and vocabulary size.

3.3 Speaker-Independent Resource Management (RM) dataset

The Naval Resource Management (RM) task [Price et al., 1988] recorded a speaker independent dataset, consisiting of 109 training speakers for the training set and 40 different speakers for the evaluation set. The speakers have a wide range of U.S. dialects, with 78 male speakers and 31 female speakers in the training data.

The sentences that were spoken were generated artificially from a grammar, and consisted of a query task to a naval database, containing information about ships, and other related properties such as locations, propulsion types, and fuel sizes. Each speaker was a naive user of the system, i.e. they had no previous experience with the naval database.

	Training set	**feb89 (dev. set)**	**oct89/feb91/sep92 (test set)**
Total length	4.1 h	16 min	50 min
# utterances	4358	300	900
# words	39,051	2561	7727
# unique words	988	576	798

Table 3.2: *RM dataset information. Shows the length of the training, development, and test sets in hours and minutes, the total number of utterances, and the total number of words and unique words.*

Table 3.2 gives detailed information about the RM datasets used in experiments throughout this thesis. Like De Wachter [2007], the original evaluation set is partitioned so that a development set can be formed. The feb89 set becomes the development set, with the oct89/feb91/sep92 sets forming the test set.

3.4 Feature Extraction

Feature extraction is the process of converting the speech waveform into a sequence of parameter vectors by positioning windows over segments of the waveform, usually overlapping. Each segment defined by the window is then used to calculate

the feature vector, known as a frame. The features extracted for all datasets in this thesis are Mel-Frequency Cepstral Coefficients (MFCCs) [Davis and Mermelstein, 1980] and are calculated using HTK [Young et al., 2009]. The following are the brief steps involved in MFCC feature extraction:

1. **Preemphasis of the waveform**. The energy of higher frequencies is boosted by using a high-pass filter.

2. **Windowing**. Apply *Hamming* window to overlapping segments of the waveform.

3. **Apply the Discrete Fourier Transform (DFT)**. The DFT is applied to each windowed segment of the waveform to determine the energy at different spectral bands.

4. **Mel-Scale Filterbank**. A bank of triangular filters (known as channels) are placed evenly over the frequency bands from the DFT and then warped by the Mel-Scale to provide the spectral magnitude within each channel. The Mel-scale stretches the channels at higher frequencies modelling the property of the human auditory system which is less sensitive at higher frequencies. 20 channels are used here.

5. **Log**. The log of each of the mel spectrum magnitudes is taken. This models human hearing which is less sensitive to small changes in amplitude at high amplitudes than at low amplitudes.

6. **Calculate Cepstral Coefficients**. Extract the first 12 cepstral coefficients from the log filterbank magnitudes using the Discrete Cosine Transform.

7. **Deltas and Energy**. Add the energy of each frame to the 12 cepstral features to give 13 feature dimensions and add velocity and acceleration features (the deltas) for each of the 13 defined features to give a final feature vector of 39 dimensions. The deltas model the change in features (over a window of 2 frames) over time.

On both datasets, a Hamming window with a width of 20 milliseconds was used while the frame-rate was set at 10 milliseconds resulting in overlapping windows.

3.5 HMM Baseline Recognisers

For both the SD call-routing and RM datasets, a set of 3-state (emitting) monophone HMMs with 20 Gaussian mixture components defined at each state were trained using HTK Young et al. [2009]. The number of mixture components was arrived at experimentally.

The SD call-routing dataset used a set of 44 phones with the silence model added. The RM dataset used a set of 47 phones with the silence and *short-pause* models added. Both datasets were labelled automatically using flat-start monophones from word transcriptions of each utterance.

3.6 Template Information

Table 3.3 shows the number of templates defined for both datasets used in this thesis. Although the number of phrase templates is dependent on the acquisition method to be described in Chapter 4, the final number used in the experiments of Chapter 5 and Chapter 6 are displayed here to present the full-picture to the reader. It should be noted that there are word templates that are also contained within a phrase template, i.e. both templates contain the same sequence of reference frames.

Dataset	**SD Call-Routing**	**RM**
Vocabulary Size	1505	991
Num. Word Templates	51,083	38,960
Num. Phrase Templates	12,899	11,354
Total Num. Templates + Silence	77,762	64,085

Table 3.3: *Dataset template information. Shows the number of word templates, phrases templates, and total number of templates when combined with silence templates for each dataset.*

The total number of silence templates in each dataset are added to the total number of word and phrase templates to give the total number of templates for each system. However, in the experiments reported in Chapter 5 and Chapter 6, a subset of the silence templates are chosen with an even distribution over all of the available lengths of the silence templates as it is not necessary to store thousands of examples of the silence template. In reality, there are about 2000 silence templates selected for each dataset.

Chapter 4

Phrase-Based Language Modelling

4.1 Introduction

As previously stated in Chapter 1, there is a significant amount of pyscho-linguistic evidence [Goldinger, 1998; Wray, 1999] that suggests that, in production of speech, humans use ready-made chunks termed, "formulaic sequences" [Wray, 2002], which "appear to be pre-fabricated, that is stored and retrieved whole from memory rather than being subject to generation or analysis by the language grammar" [Wray and Perkins, 2000]. These formulaic sequences in turn lead to the human listener being primed with a set of formulaic phrases, depending on the context of the conversation, which can then be used in an efficient perception mechanism [Pickering and Garrod, 2004]. Given that much of commonplace language is highly formulaic, this chapter will aim to acquire a lexicon of commonly-occuring phrases from transcriptions of speech, and then model the human listener by creating language models that enable the prediction of the given phrases for given contexts.

This Chapter is structured as follows: Section 4.2 gives a survey of the literature related to phrase integration into language modelling with various applica-

tions such as speech recognition, call-routing, or just language modelling purposes. Section 4.3 describes the *Multigram* segmentation model [Deligne and Bimbot, 1995] that is used for phrase acquisition in this study. Section 4.4 introduces a new clustering algorithm for the acquired phrases called *Hybrid Syntactic Formulaic* (HSF) clustering which adapts Nasr et al. [1999] to use syntactic information from *Parse Trees* [Charniak, 2000] to group frequently occuring phrases from the multigram segmentation. Section 4.5 describes methods for integrating phrases into the popular *N-gram* framework by formulating the problem as a language model topology issue and describes how the phrase classes resulting from the HSF clustering of Section 4.4 can be integrated into these language model topologies. Section 4.6 gives the results of these methods in terms of language model perplexity and a speech recognition word accuracy baseline measure using a HMM-based system. Finally, Section 4.7 summarises the methods and results described in the chapter.

4.2 Literature Survey

One method to model sequences of words, i.e. phrases, in a language model is to model the phrases as individual dictionary items in an N-gram model. The phrases are selected using a phrase acquisition algorithm, and then processed as a single word would be in N-gram probability estimation. Giachin first applies a word clustering algorithm[1], so that each word is assigned to a class, then proceeds to acquire phrases which are actually sequences of classes [Giachin, 1995]. Giachin provides details of an *optimal* procedure and *heuristic* procedure to identify phrases. The optimal procedure cyclically determines the pair of words[2] that when connected into a sequence of words results in the largest reduction of perplexity on the training text. This iteration repeats until the algorithm converges

[1]Giachin does not give details of the word clustering algorithm, but the interested reader should refer to Brown et al. [1992] as an example of word clustering.

[2]Once words are clustered into classes, the word is replaced by the label of the class that contains it in the training data, but word in this sense refers to an item in the dictionary.

to minimum perplexity. The heuristic procedure combines words into phrases by choosing the pair of words that have maximum mutual information. This algorithm continues to iterate as long as the perplexity decreases. The point at which the perplexity increases is chosen as the stopping point, but, in some cases the number of phrases chosen can be high, so a second stopping point is introduced, determined by the number of phrases. Both the optimal and heuristic algorithms give almost identical performance in terms of perplexity reduction, both achieving approximately a 20% reduction in test set perplexity relative to a baseline word bigram model. The best WER achieved using the phrase bigrams reduced the WER by 2% absolute over the word bigram baseline.

Nasr et al. [1999] describe a method that combines Stochastic Finite State Automata (SFSA) [Parekh and Honavar, 2000], which represent classes of phrases, and N-grams which model the global relationships between the local SFSA models. Phrases are acquired from the training data by partially parsing the data (annotated with Parts of Speech (POS) tags) with a greedy finite state parser. The partial nature of the parser means that whole sentences are not parsed, while the greedy nature of the parser means that the first rule that fits the data's structure is used to parse the data. The parser will only acquire phrases of recognised constituents (e.g. a noun phrase). Phrases are then grouped into classes based on their context, i.e. phrases of the same constituent appearing in the same left and right context are grouped together. At this point, classes are discarded if the number of phrase tokens[3] is less than a pre-defined threshold. A vector based (frequency of each phrase) class merging procedure is then applied, which iteratively merges the two closest classes until their distance is larger than a threshold. For each class, an SFSA is built. The original training text is rewritten with class labels in place of phrases, and an N-gram model is built. At the time of decoding, a phrase has a global probability, represented by the N-gram probability of the label of the class that contains it, and a local probability, which is the probability through

[3]The number of phrase *tokens* is the count of all phrases within the class including the number of occurrences of each phrase. Phrase *types* refers to the number of *different* phrases within a class.

the SFSA that represents the class that contains it. The two probabilities are combined to provide the overall probability of the phrase. It does not appear as though the SFSAs are actually used for anything other than structural purposes, i.e. the probabilities are in fact just relative frequency. Nasr et al. report an 8% reduction in test set perplexity on a french dataset of telephone communications (CNET's AGS corpus), but report a 14.2% reduction in perplexity on sentences containing eight or more words. No speech recognition experiments were reported.

Arai et al. [1999] describe a *grammar fragment acquisition* method which they use in an automatic call-routing application. Phrases are first constructed from the training data by counting all sequences of words up to three (this could be any number, but three is chosen for their experiments). If these phrases have a frequency greater than some threshold then the phrases become fragments. The fragments represent the phrase in an FSA (Finite State Automaton). Each fragment has a list of the preceding and succeeding phrases that surround it, and the number of times that those phrases appear with the fragment. Each fragment also contains a list of the call-types that it is used in — this is a semantic association and is relevant for call-routing purposes. To cluster fragments, the most frequent fragment that has not been clustered already is used as the reference, and the remaining fragments are sorted by distance from the reference fragment. Three distances are calculated (one for the preceding phrase distribution, one for the succeeding phrase distribution, and the final one for the call-type distribution) using the *Kullback-Leibler* distance measure, and the fragments that commonly occur in all three distance lists are considered a good match for the reference fragment and are clustered together with the reference. The clustering continues to iterate until all fragments have been clustered. Arai et al. also apply a generalisation method to the fragments which finds substrings within the current fragment that are also fragments which appear more frequently than the current fragment. If this is the case, the label of the more frequent fragment replaces the substring in the current fragment. This results in the modelling of unseen phrases from the training data.

Lin et al. [1997] describe a *key-phrase* spotting system that uses a combination of N-grams and finite state grammar (FSG) models. The FSG models are used to cover all of the key-phrases while the N-gram (specifically trigrams in experiments) model is used for non-key-phrases. The FSG is triggered when the decoder spots a word that starts the FSG — the FSG and N-gram model are then run side by side in parallel. The two models essentially compete, with the likelihood scores being compared. If the input can fully traverse the FSG and have a higher likelihood than the N-gram, then a key-phrase is judged to have been found. The FSG has an *initial boost* factor which determines how easy it is for the FSG to be entered, and a *selective penalty* factor which provides varying levels of punishment for the FSG depending on the depth of traversal through the FSG. These measures, which are chosen by hand, were introduced to help generalise the model, i.e. to minimise the effect of a change of application. On a task to detect and recognise 7-digit telephone numbers within sentences of read speech, the model achieves a 4.8 % word error rate which is a 69 % relative reduction to the baseline trigram model which achieved a 15.3% word error rate.

Continuing with this hierarchical type language model, Galescu and Allen [2000] describe an example of such a model that consists of a trigram layer at the top and an SFSA underneath to model sub-word units for numbers. This model, called the *hierarchical hybrid statistical language model* (HSLM), is an attempt to improve adaptation of language models. The idea is that the top trigram layer is adapted using current interpolation methods [Kneser and Ney, 1995], whereas the sublanguage models (SFSAs) are left unchanged, although the sublanguage models can be adapted with data from other sources, which does not affect the trigram layer. The experimental model uses only one sublanguage model, for decimal numbers, as mentioned earlier, which is trained using a regular expression to identify all decimal numbers in the training data. The basic unit of the sublanguge model is the grapheme, i.e. a character. The HSLM achieves small improvements over the baseline adapted model in terms of *adjusted perplexity*[4].

[4]Adjusted perplexity is a measure introduced to accurately compare perplexity between mod-

Solsona et al. [2002] propose a language model for a spoken dialogue system which combines a state-independent N-gram model with a state-specific FSG. The FSG is used to recognise commonly occuring phrases, and is run in a separate recogniser in parallel with the N-gram-led recogniser. Using an acoustic confidence measure (phone-based likelihood ratio) [Jiang et al., 2001], the "best" results from one of the recognisers is chosen as the final decoding, i.e. the only combination of the two recognisers (and thus language models) is the comparison of their results using the confidence measure.

Deligne and Sagisaka [2000] describe a class-based *n-multigram* which retrieves phrases of varying length (multigrams) and then estimates N-gram probabilities between them. The clustering and phrase acquisition is an iterated two-step procedure. Step one finds the maximum likelihood (ML) segmentation of the training data into phrases using N-gram estimations (for an initial segmentation, the relative frequency (unigram probability) is used). Step two finds the optimal membership of phrases into classes by moving each phrase from its current class to the remaining classes and calculating the likelihood after each move (initially the N most frequent phrases are added to their own class, and the remaining phrases are all placed into one class from which phrases can only be removed). The phrase exchange that results in the overall best likelihood is chosen for the current iteration. The new N-gram distribution is then calculated and the algorithm repeats until the likelihood has converged (i.e. no exchange of phrases results in an increase in likelihood) or until a predefined number of steps are complete. Deligne and Sagisaka construct a bi-multigram model (without classes, i.e. using just the ML segmentation of the training data) and a class-based bi-multigram model and interpolate the two models for speech recognition tests on the ATIS database[5]. The interpolated model achieves a 10 % relative reduction in the word error rate compared to that of the word trigram baseline.

els that contain a different lexicon. It adjusts the perplexity by a quantity based on the number of unknown words in the test set, and the number of their occurrences.

[5]Air Travel Information System (ATIS) database. ATIS contains utterances of customers speaking over the phone to make airline reservations.

Wray et al. [2004] describe a speech translation system for sign language in a post office which uses a speech recogniser to determine what the post office clerk has said (decoded into an N-Best list, from which the clerk chooses the closest match), and then translates the text into sign language via a computer generated avatar. The system uses an FSA to represent hand-picked phrases which are chosen as formulaic phrases [Wray, 2002] — the system attempts to model commonly occurring phrases within the context of the post office, and includes open slots in the phrases to allow for small variations in the wording [Cox, 2002].

Finally, it should be noted that all previously reported results in this literature survey are on academic systems, and by no means represent an upper bound on speech recognition performance. Nuance's speech recognition software, *Dragon NaturallySpeaking 10*, claims to achieve 99% accuracy while recognising upto 160 words per minute [Nuance, 2008]. The software allows users to dictate straight into a word processor document and also control the operating system with simple commands, such as "start menu", "send an email to", etc. While not specified, it is clear that the *command and control* interface uses pre-defined phrases, although it is not clear whether or not the phrases are defined soley at the grammar level or at the acoustic level also.

4.3 Phrase Acquisition using Multigrams

The segmentation of utterances from transcriptions of speech is one method of phrase acquisition (each segment being a group of one or more successive words from an utterance). Outlined below is a Maximum Likelihood (ML) approach to segmentation using variable-length word sequences which are known as *Multigrams* [Deligne and Bimbot, 1995, 1997b; Deligne and Sagisaka, 1998, 2000].

The multigram model is designed to retrieve *"sequential variable-length regularities within streams of observations"* [Deligne and Bimbot, 1997b]. This notion / model fits very well to the idea of formulaic phrases [Wray, 1999, 2002] in which

phrases of different lengths repeatedly occur in a collection of utterances. The multigram model is a general model that can be applied to any stream of observations — it has also been applied to vector quantised (VQ) audio speech data [Deligne and Bimbot, 1997a].

In this work, the goal is to segment text utterances into phrases. The observation stream is the current utterance, and the words within the utterance are the observation symbols. Deligne and Bimbot [1997b] describes the multigram model as a production model (see Figure 4.1) where some source emits a sequence of multigrams **Z** where each multigram is a variable-length sequence of observations. The observation stream **O** is segmented into one or more segments in **S**, and each segment equates to a multigram z_i in **Z**.

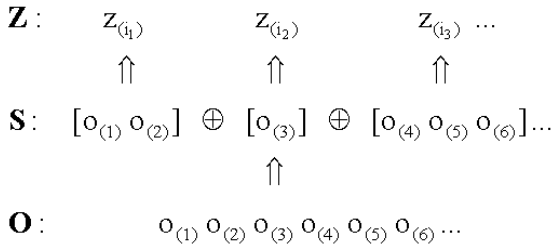


Figure 4.1: *The Multigram Production Model (reproduced from Deligne and Bimbot [1997b]). Each multigram z_i emits a sequence of observation symbols which when concatenated form the observation sequence* **O**. *The multigrams,* **Z**, *can be retrieved from* **O** *by finding the correct segmentation* **S** *of* **O**.

Before segmenting the utterances, several initialisation steps are performed. The first step is to construct an initial set of multigrams. This is done by constructing sequences of words (phrases) of length 1 to L at each word in each utterance of the training data. For example, the utterance "what is my account balance" contains the following initial multigrams for $L = 3$: *what, what is, what is my, is , is my, is my account, my, my account, my account balance, account, account balance, balance.* For each of these constructed phrases, a frequency count is made using a hash table. Algorithm 4.1 summarises the initial multigram construction.

After the initial multigrams have been constructed, a threshold, θ_1, is applied so that any multigrams appearing in the training text less than θ_1 times are

Algorithm 4.1 Constructing the initial set of multigrams.

```
1: {U is the set of utterances}
2: {|u_i|) is the number of words in utterance u_i}
3: for i = 1 to |U| do
4:    for j = 1 to |u_i| do
5:       curPhrase= ϵ
6:       for k = j to j + L do
7:          Add word k to curPhrase
8:          if curPhrase exists in hash then
9:             Increment the count C(curPhrase)
10:         else
11:            Add curPhrase to hash
12:         end if
13:      end for
14:   end for
15: end for
```

discarded, except for single word multigrams (e.g. *my, is, account* etc.) which are retained to ensure that an utterance can always be completely segmented. The unigram probability of each of the remaining multigrams is then estimated using the frequency counts:

$$\Pr(m_i) = \frac{C(m_i)}{\sum_{j \in M} C(m_j)} \tag{4.1}$$

where m_i is the current multigram, $C(m_i)$ gives the number of occurences of m_i, and M is the total number of *different* multigrams.

Following the unigram estimation, each utterance can then be segmented using the Viterbi algorithm. What follows is a description of the implementation of the segmentation using Hidden Markov Models (HMMs) — using HMMs is a convenient implementation, but not essential to the segmentation.

A HMM is constructed for each of the initial multigrams. In each model, there is a separate state for each word with transition probabilities between each word state being 1.0. Figure 4.2 shows the HMM that is constructed for the multigram

"i want my". Each state also has a set of observation probabilities (shown above the states), where each word in the vocabulary is given a probability of observing that word given the current state. In this work, the observation probabilities have a discrete distribution, such that the word that the current state represents has an observation probability of one and all other words in the vocabulary have zero probability.

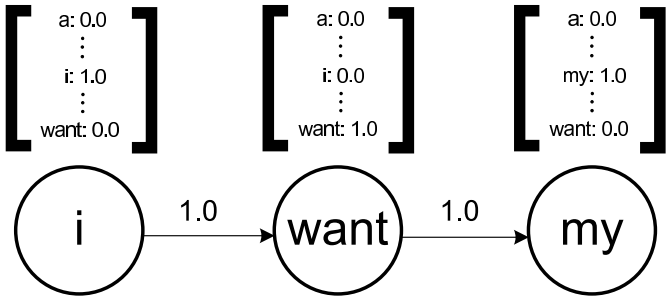


Figure 4.2: *A HMM for the multigram "i want my". Each state contains a set of discrete observation probabilities (shown above the states) where the word representing the state has a probability of* 1.0*, and all other words have a probability of* 0.0*.*

After the initialisation steps are complete, the iterative segmentation can begin. For each utterance that is to be segmented, a decoding network is generated (see Figure 4.3). This network, also known as an *Ergodic* network, allows any multigram HMM to be connected to any other multigram HMM within the context of the current utterance — the search space is pruned by only adding the multigram HMMs that can be applied to the utterance (using basic string comparison). For example, the decoding network in Figure 4.3 has been constrained for the utterance "i want my account balance". It shows the transition probabilities between states, where the first set of transitions leading from the start state[6] (state to far left filled in black) are the unigram probabilities as calculated in Equation (4.1), and the internal transition probabilities (between words) are 1.0 as previously mentioned. The end state, on the far right, is used to either loop back to the start, or to terminate the segmentation.

The decoding network is used to segment the utterance using the Viterbi search,

[6]This state is merely used a point to loop back to, and thus to connect multigrams. It has no observation probabilities, and all arcs into it have a probability of 1.0.

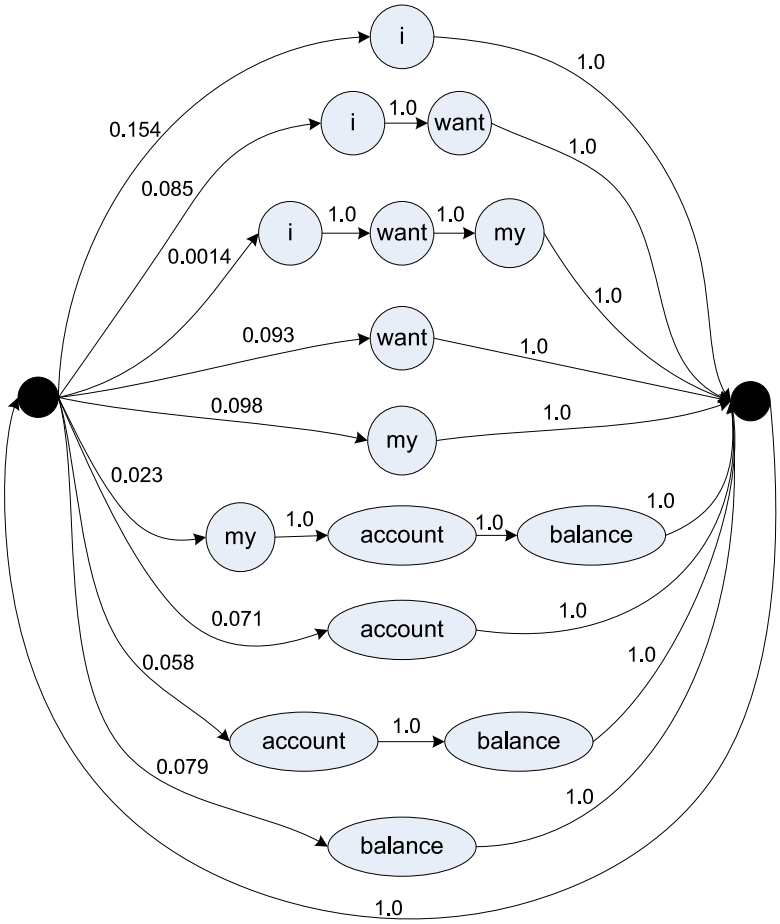


Figure 4.3: *A unigram decoding network for a restricted set of multigrams. The network is used to segment the utterance "i want my account balance". The solid black states are the start-state and end-state, with a loop-back connection from the end-state to the start-state. The initial arcs from the start-state represent the unigram probabilities of each multigram. Each sequence of word-states between the start-state and end-state represent seperate HMMs.*

implemented with the Token Passing algorithm (Section 2.3.4), which chooses the sequence of multigrams which gives the highest probability for the give utterance. The discrete observation probabilities ensure that sequence of multigrams matches the input utterance correctly (in terms of string matching). For the given network in Figure 4.3, the Viterbi algorithm would find the best (in terms of maximum likelihood (ML)) segmentation to be "[i want] [my account balance]" with a probability of 1.96×10^{-3}. Comparing that to the probability of the segmentation "[i] [want] [my] [account] [balance]", giving a probability of 7.87×10^{-6}, shows that even though the unigram probabilities of each multigram in the segmentation are

higher than the multigrams in the *best* segmentation, when combined in to a sequence of multigrams, the probability of the sequence will increase when fewer combinations of multigrams are used, and hence longer units will be chosen.

Algorithm 4.2 Segmentation algorithm for the training data

```
1: while Segmentation has not converged do
2:    for i = 1 to |U| do
3:       Create decoding network for u_i.
4:       Segment u_i using Viterbi with decoding network.
5:    end for
6:    Re-count multigrams in latest segmentation.
7:    Apply threshold, θ_2, to multigram counts.
8:    Re-estimate unigram probabilities of remaining multigrams.
9: end while
```

After each utterance has been segmented, the number of occurrences of each multigram appearing in the segmentation are found. A second threshold, θ_2, is applied to the counts of multigrams of two or more words (only), and the unigram probabilities are re-estimated as before. Any single word multigrams that are not contained in the segmentation (but are contained in the vocabulary of the training data) are re-introduced into the multigram vocabulary and are assigned a small probability (1×10^{-99}) to ensure that all utterances can be segmented fully.

The segmentation of the training utterances, followed by re-estimation of multigram probabilities, continues to iterate until a convergence of the segmentation is found: this is when the segmentation no longer changes, or when a pre-defined number of iterations for the algorithm is reached. Algorithm 4.2 summarises this process and Appendix A shows some samples of actual segmented utterances from the datasets used in this work (Chapter 3).

4.4 Phrase Clustering using a Hybrid Syntactic and Formulaic Approach

For human-to-human dialogue, there is pyscho-linguistic evidence that in a particular context in a dialogue, the human listener is primed with a set of *semantic* expectations [Pickering and Garrod, 2004], and these in turn may prime an appropriate set of formulaic phrases that the listener expects to hear.

Given this motivation, it is useful to try and define the semantic function of the commonly-occurring phrases acquired by the multigram segmentation: for instance, phrases such as "i'd like to", "could i please", and "may i" have the same semantic function of expressing a desire for some action. Interpreting the semantics of phrases is very difficult, due to factors such as ambiguous lexical items, competing anaphoric references[7][Nouwen, 2003], and ambiguous quantifier scopes [Jurafsky and Martin, 2009].

So, we rely on the fact that, in the case of applications where ASR is used for the provision of information and services over the telephone network (Section 3.2), phrases that have similar *semantics* often also have a similar *syntactic* function: for instance, the phrases previously defined all appear at the start of an utterance and will be followed by some form of a verb phrase in which the speaker defines his request.

As described in Section 4.2, Nasr et al. [1999] introduced a method for acquiring and grouping phrases using only the information from the parse tree. Phrases are extracted using a greedy parser, and then grouped into classes based on their constituent type, and the surrounding context. Every phrase that is acquired in this process is a legal constituent within the parse tree.

The remainder of this section describes a hybrid clustering method which adapts the work of Nasr et al. [1999] and uses the phrases acquired from the multi-

[7]An *anaphoric reference* is when a word or phrase refers to, or is related to other items in a given text. For example, in the sentence "Jim was bored, so he turned on the t.v.", the word "he" refers to "Jim".

gram segmentation, described in the previous section (Section 4.3), and groups together the phrases that have a similar syntactic use by extracting information from *parse trees*.

4.4.1 Clustering with Parse Trees

Figure 4.4 shows the *most probable* parse tree generated by the Charniak parser [Charniak, 2000] for the utterance "i'd like to get my balance". Each leaf node of the tree represents the words of the utterance, and each parent of the leaf nodes represent the *Parts of Speech* (POS) tags for the given words. All levels above that define different constituents, such as *noun phrase* (NP) or *verb phrase* (VP), which are defined by a combination of POS tags. Appendix B gives the complete list of POS tags and constituent definitions for the Penn Treebank which is used by the Charniak parser.

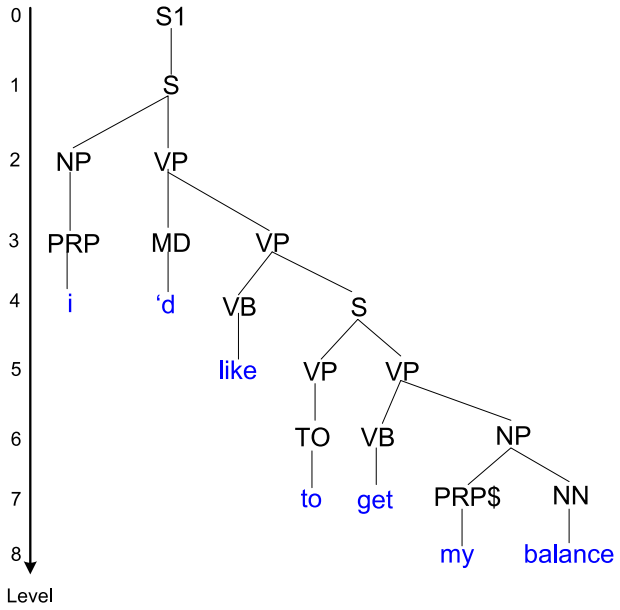


Figure 4.4: *A parse tree for the utterance "i'd like to get my balance".*

The clustering of phrases requires two inputs for each utterance: The corresponding parse tree and maximum-likelihood (ML) segmentation. For the sentence "i'd like to get my balance", a parse tree could look like that of Figure 4.4. The ML segmentation of that utterance might be [i'd like to get] [my balance].

The phrases generated by the segmentation are then searched for in each parse tree, and assigned a label. An example of the construction of this label is given here for the utterance shown in Figure 4.5:

1. *Find start and end word of the phrase and merge their POS tags*: e.g. for "i'd like to get", the POS for "i" is PRP, and the POS for "get" is VB. These are merged together to form "(PRP_VB)"[8].

2. *Append contextual information to each phrase label.* Take the initial label of the phrases (from (1) above) to the left and right of each phrase and append them to the current phrase label: e.g. the initial label of "i'd like to get" is updated with the left context "Null", and the right context "(PRP$_NN)", which represents "my balance", to produce the final phrase label "Null-(PRP_VB)-(PRP$_NN)". "Null" is used whenever there is no context, i.e. at the start and end of utterances.

This approach differs from that of Nasr et al. [1999] in that they attempt to *define* phrases using the parse tree. Such phrases will conform to the analysis of the parser, but may not be as frequently occurring as phrases found by the multigram segmentation. In fact, many frequently occurring phrases occur across the grammatical divisions defined by the parser and so would not be found using their approach. For example, in Figure 4.5, "my balance" actually matches the NP constituent perfectly, but "i'd like to get" crosses the division of NP and VP.

Once all phrases have been tagged with a label, the next step is to group all phrases that have identical labels, *excluding* the right context: for example, the phrase "my balance", as shown in Figure 4.5, would be grouped with phrases whose labels begin "(PRP_VB)-(PRP$_NN)". After all phrases are grouped in this manner the *initial classes* have been created. Our method, which applies tags based on POS tags, allows phrases to appear in more than one class because

[8]When the current multigram from the segmentation is a single word, the labels are treat slightly differently: the constituent type, e.g. NP or VP, is used as the label.

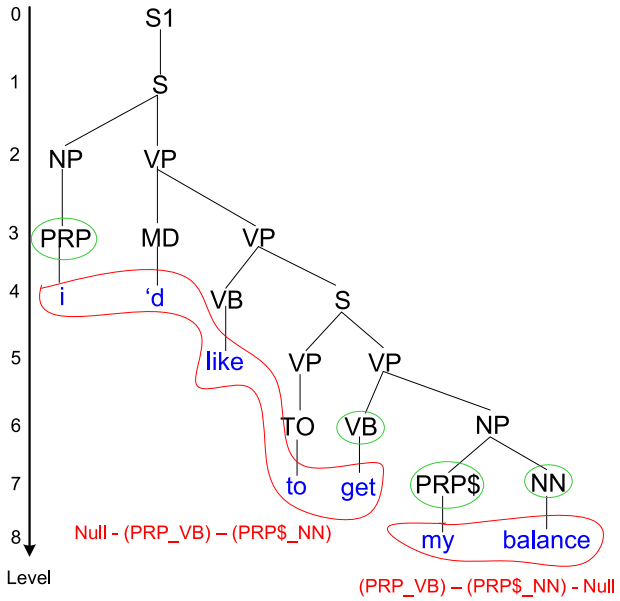


Figure 4.5: *Phrase labelling using parse trees.*

of contextual grouping (i.e. phrases that contain the same words, not the same phrase instance), and hence the fuzzy clustering produces a large number of initial classes (approximately 12000).

To reduce the number of phrase classes, a similar method to Nasr et al. [1999] is employed, where the classes are iteratively merged to a pre-defined number using a vector-based approach. The following section describes this merging process in more detail.

4.4.1.1 Class Merging

Given the initial classes, a class merging algorithm is applied which, with each iteration, merges the "closest" pair of classes, until a pre-defined number of classes is reached. This is achieved by using the cosine similarity measure: if V is the set of phrase types from the segmentation, then each class is represented as a vector of length $|V|$, where each element of the vector is the count of each phrase within the current class. All classes are then processed in a pairwise manner, with the most similar pair, in terms of cosine distance, being merged together. This

iterative process then continues until a pre-determined number of classes remain. The cosine similarity of two classes C_1 and C_2 is given by:

$$CosSim(C_1, C_2) = \frac{\mathbf{C}_1 \bullet \mathbf{C}_2}{|\mathbf{C}_1| \times |\mathbf{C}_2|} \tag{4.2}$$

where, $\mathbf{C}_1$ and $\mathbf{C}_2$ are the phrase vectors of classes C_1 and C_2 respectively, and $|\mathbf{C}_1|$ and $|\mathbf{C}_2|$ are the vector norms of classes C_1 and C_2 respectively.

The vector norms, $|\mathbf{C}_1|$ and $|\mathbf{C}_2|$, can be pre-calculated before the merging process begins, but for each pair of merged classes, the vector norm of the newly merged class needs to be recalculated for the new values. The vector norm for a class, C_x, is given by:

$$|\mathbf{C}_x| = \sqrt{\sum_{i=1}^{|V|} \mathbf{C}_x(i)^2} \tag{4.3}$$

where, $|V|$ is the size of the phrase vocabulary and thus the length of each vector, and $\mathbf{C}_x(i)$ gives the count of phrase i within the class C_x. Algorithm 4.3 summarises the class merging procedure.

Algorithm 4.3 Class merging process using cosine similarity

```
1: pre-calculate norms for all classes
2: while numOfClasses > finalNum do
3:    closestSim = −∞
4:    for i = 1 to numOfClasses do
5:       for j = i + 1 to numOfClasses do
6:          curSim = CosSim(C_i, C_j)
7:          if curSim ≥ closestSim then
8:             closestSim = curSim
9:             iBest = i
10:            jBest = j
11:         end if
12:      end for
13:   end for
14:   merge(C_iBest, C_jBest)
15:   recalculate norm for new class
16: end while
```

4.5 Integrating Phrases with N-Grams

Given that we have acquired a set of frequently occurring phrases, this section describes several methods that can be used to integrate the phrases into a language model. Section 4.5.1 describes different topologies for language models that can be used to integrate the phrases, while Section 4.5.2 describes the methods used to integrate classes of phrases into the language model.

4.5.1 Language Model Topologies

Given that the training text utterances are segmented using the ML multigram segmentation, a simple bigram language model can be built from that segmentation, i.e. counting bigrams of phrases (which also includes one word phrases). Figure 4.6 shows a section of that model, which we call the *Phrase-Bigram* (PB), that is relevant to decoding a test utterance "can i get my payment address please". Arcs with solid lines connecting two states represent a bigram seen in the training segmentation, while dashed-line arcs represent backoff transitions. Although the

figure shows two backoff states, the reader should be aware that there is in fact only one backoff state in the PB model — two backoff states are used to make the figure clearer. The reader should be aware that in a backoff language model, that *all* states contain a transition to the backoff state[9], and *all* states contain a transition *from* the backoff state[10], although Figure 4.6 only shows the backoff transitions that are required for the given test sentence.

For the PB language model shown in Figure 4.6, it should be clear that there is no direct path between word and phrase states to decode the given test utterance, i.e. without using backoff. This is because in the training segmentation, there is no context that exactly matches the test utterance — the closest matching context is "[can you give me my] [payment address please]". The bigrams "get my" and "[can i get] [my account balance]" offer part of the context required for the test utterance, but transitions through the backoff state are still required, and as mentioned in Section 2.2, the unigram probabilities are then used. The following group of equations show the log probabilities of the different decoding paths through the PB language model of Figure 4.6 for the given test utterance:

$$\log(\Pr(\text{"[can] [i] [get] [my] [payment] [address] [please]"}|PB)) = -45.12 \quad (4.4)$$

$$\log(\Pr(\text{"[can i get] [my] [payment] [address] [please]"}|PB)) = -32.19 \quad (4.5)$$

$$\log(\Pr(\text{"[can] [i] [get] [my] [payment address please]"}|PB)) = -34.53 \quad (4.6)$$

$$\log(\Pr(\text{"[can i get] [my] [payment address please]"}|PB)) = -21.60 \quad (4.7)$$

It is clear that if the training data does not contain the same or similar contexts to the test data then, if the PB language model is used, there will be a poor representation within the language model. Although the backoff allows any word in the vocabulary to be recognised, it offers a poorer predictor as it then leads to unigram probabilities guiding the search. Equation (4.4) summarises this by

[9]Except for the end state of the model (e.g. !END).
[10]Except for the start state of the model (e.g. !START).

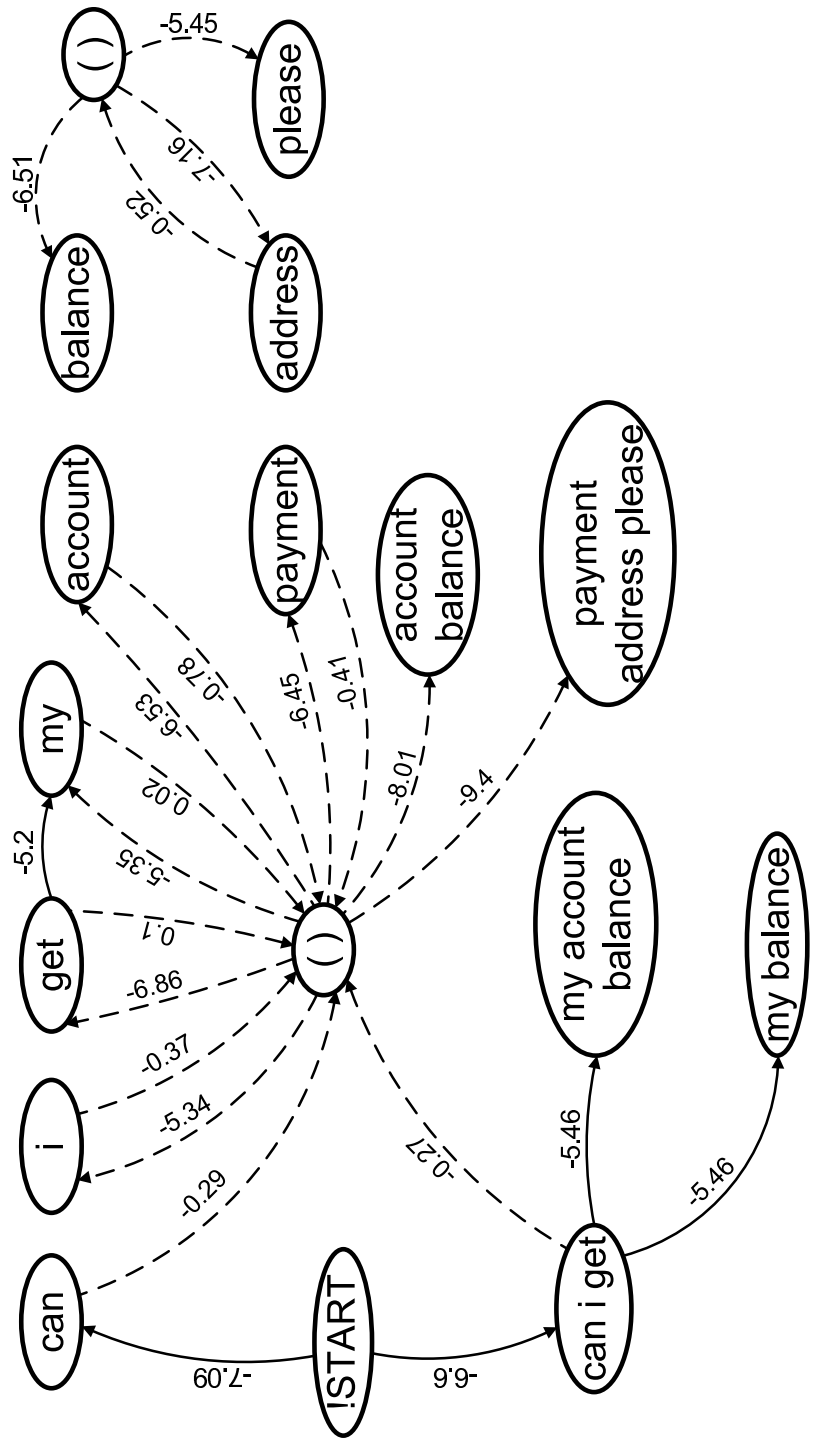


Figure 4.6: *Phrase-Bigram (PB) language model. All transitions required to parse the input utterance "can i get my payment address please" e.g. not all backoff transitions are shown. Log-likelihoods are shown on the arcs between nodes, and the backoff state, represented by empty brackets "( )", is shown twice for readability of the figure (it only occurs once in the actual model), with transitions to-and-from the backoff state shown in dashed lines — note that transitions to the backoff state represent backoff weights, hence why there are some arcs with values greater than zero.*

showing that the log probability of the sequence of all word states that match the test utterance is −45.12: this sequence consists of mainly backoff transitions, except for the transition from the start state to the state for "can", and the transition between the "get" and "my" states. The most probable sequence of states for the test utterance "can i get my payment address please" is to start in the "can i get" state, with a backoff to the "my" state, ending with another backoff to the "payment address please" state (Equation (4.7)). This leads to a much higher log probability compared to other sequences through the model (Equations (4.4) to (4.6)).

If the phrases acquired from the multigram segmentation offer good coverage for the test data, then the PB language model could provide a good basis for a speech recogniser — the problem with the PB model is that it will start to fail as the test data becomes more dissimilar to the training data, for two reasons: the first reason is that the acquired phrases will be less likely to appear in the test data, and thus the backoff transitions to word states will be required, and the second reason is that if the backoff is used, it is not always the case that the required word will exist as a distinct multigram.

For example, the number of distinct words in the SD *training* data is 1504, but in the segmentation of the training data, there are only 1446 distinct words (i.e. words that are not contained within a phrase). This results in 24 words in the *test* data that are no longer represented by single words, i.e. there are effectively 24 out-of-vocabulary (OOV) words in the test data. This side effect means that, in some cases, it will be impossible to accurately recognise an utterance perfectly. To clarify, take the test utterance "i wanna add a one time buyer to my account". The word "buyer" is not contained as a single word in the training segmentation — it is contained within the training phrases "as an authorized buyer", "from my account as an authorized buyer", and "an authorized buyer". The available training phrases mean that it is impossible to recognise the given test utterance perfectly.

A simple solution to this problem is a language model that we term the *Word-Phrase Bigram* (WPB). This model combines the word bigram (WB) model (defined in Section 2.2) with the PB language model and is defined by the following *Set Theory* expression:

$$WPB = WB \cup (PB \setminus (WB \cap PB)) \tag{4.8}$$

where WB is the set of all word bigrams in the training text, and PB is the set of bigrams from the multigram segmentation.

Equation (4.8) essentially combines all of the bigram counts within the PB framework with those of the WB model *except* those that also occur in the WB language model, i.e. the counts of bigrams of words (or one-word phrases to fit with previous language) remain the same for the WPB model as they do for the WB model even if the bigrams were seen in the segmentation used to define the PB model. This step is necessary because if all counts of bigrams (from WB and PB) were just combined and then used to generate a language model, then some of the word bigrams would be biased as they would have essentially been counted twice.

Figure 4.7 shows, as with Figure 4.6 for the PB model, the section of the WPB language model required for the example test utterance: "can i get my payment address please". Each state, as before, contains both a transition to the backoff state and an incoming transition from the backoff state which represents the unigram probability of the current state. In Figure 4.7 though, not all of the backoff transitions are shown — only the transitions that are required to decode the test utterance are shown.

It is clear to see from Figure 4.7, in comparsion to Figure 4.6, that there are now bigram transitions between word states (because of the inclusion of WB counts) as well as the previously existing phrase bigrams from PB. The model can traverse

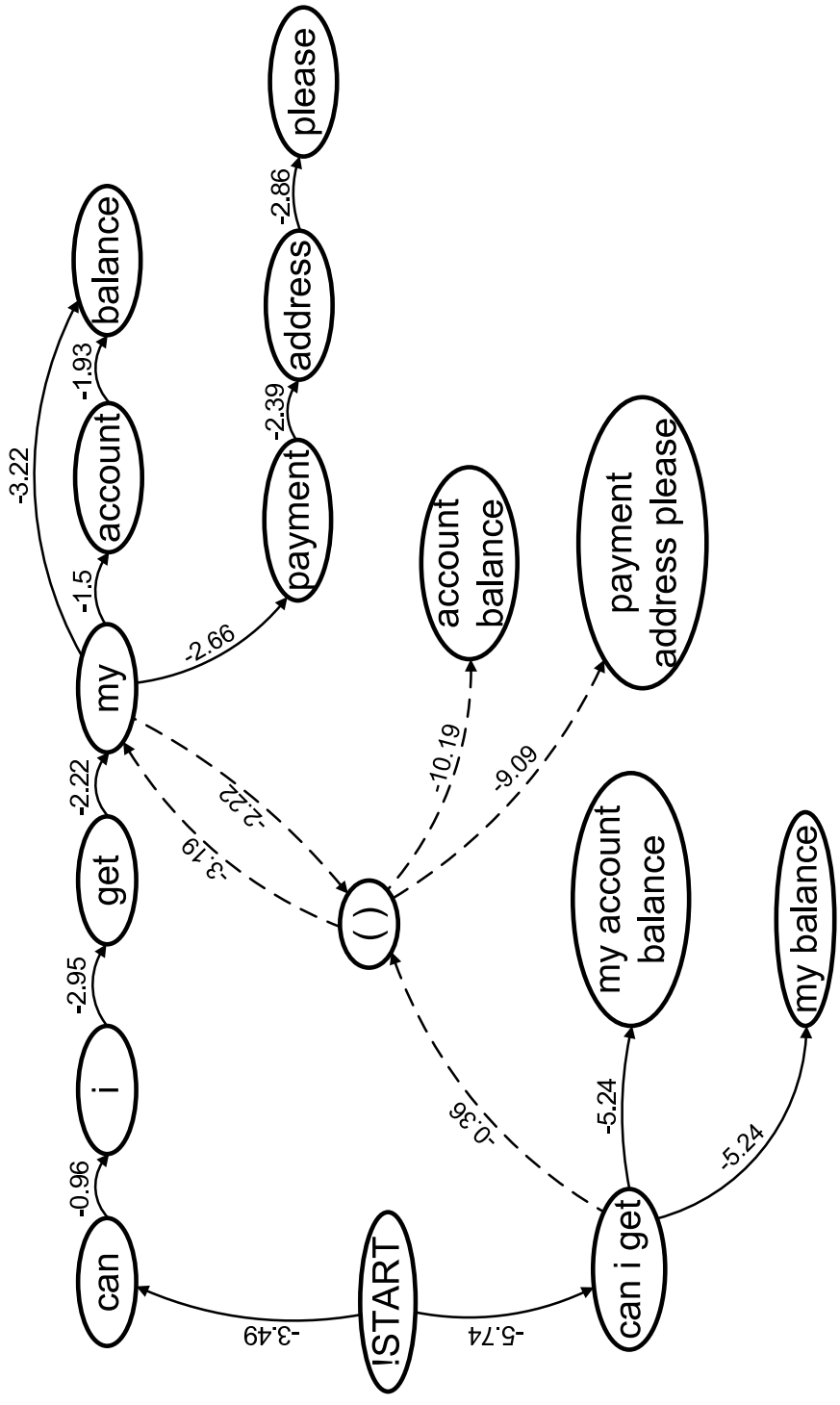


Figure 4.7: *WPB (word + phrase bigram) language model. All transitions required to parse the input utterance "can i get my payment address please" e.g. not all backoff transitions are shown. The backoff state is represented by empty brackets "( )", and log-likelihoods are shown on the arcs between nodes.*

between the word states and the phrase states via the backoff state where a direct bigram transition does not exist. As with Equations (4.4) to (4.7) for the PB model, the following equations show log probabilities of different paths through the WPB language model for the test utterance:

$$\log(\Pr(\text{“[can] [i] [get] [my] [payment] [address] [please]”}|WPB)) = -17.53 \quad (4.9)$$

$$\log(\Pr(\text{“[can i get] [my] [payment] [address] [please]”}|WPB)) = -17.20 \quad (4.10)$$

$$\log(\Pr(\text{“[can] [i] [get] [my] [payment address please]”}|WPB)) = -22.03 \quad (4.11)$$

$$\log(\Pr(\text{“[can i get] [my] [payment address please]”}|WPB)) = -21.70 \quad (4.12)$$

Clearly, as Equation (4.9) shows compared to Equation (4.4), the probability of an entire sequence of single word states given the WPB model for the test utterance is much higher than in the PB language model due to the inclusion of the word bigrams. Equations (4.10) and (4.11) also show a large increase in probability compared to Equations (4.5) and (4.6) respectively — each sequence essentially splits the test utterance in half, with one half of the parse using word states and the other a phrase state, with a backoff transition connecting the two “layers”. The sequence defined in Equation (4.12) actually has a slightly lower probability than the same sequence in the PB model (Equation (4.7)) which is due to the fact that the same backoff transitions are used and so the slight difference in probability is due to the change in probability mass after the inclusion of the word bigrams.

The WPB language model offers a more flexible and generalised topology than the PB model, but still, a large number of transitions between word and phrase states will take place via a backoff transition and thus the preferred route through the language model during recognition will be, generally, to remain within the word states or phrase states — with the word states becoming more likely as the the test data becomes more diverse in comparison to the training data.

To offer more generalisation to the language model and to reduce the number of backoff transitions used, we define the *Word Phrase Link Bigram* (WPLB) language model. The WPLB model extends the WPB language model by creating bigram connections between word states and phrase states (see Figure 4.8), where none exist already, using the training segmentation: for every occurrence of phrases of more than two words occurring next to each other in the segmentation (i.e. phrase bigrams), new counts are made for the end word of the left phrase and the whole of the right phrase, and then the whole left phrase with the start word of the right phrase. For example, Figure 4.7 shows that the "[can i get] [my account balance]" bigram is seen in the training data (and hence there is an arc connecting the two states). This bigram then leads to two further bigrams, as discussed, that were previously unseen — the end word of the left phrase "get" and the whole right phrase gives the bigram "[get] [my account balance]", and then the whole left phrase with the start word of the right phrase "my" gives the bigram "[can i get] [my]". These new connections can be seen in Figure 4.8.

The effect of adding these bigrams to the model can be seen in Equations (4.13) to (4.16). It can be seen that, in comparison to Equations (4.9) to (4.12) for the WPB model, the probability has now been increased for sequences that contain phrases, and in fact, all but one of the sequences containing phrases (Equation (4.15)) has a higher probability than the sequence containing purely words (Equation (4.13)) — this is because the first part of the sequence (Equation (4.15)) transits through the word states "[can] [i] [get]" where the phrase state "[can i get]" has a higher cumulative probability than the three word states due to it being a common phrase (especially at the beginning of an utterance).

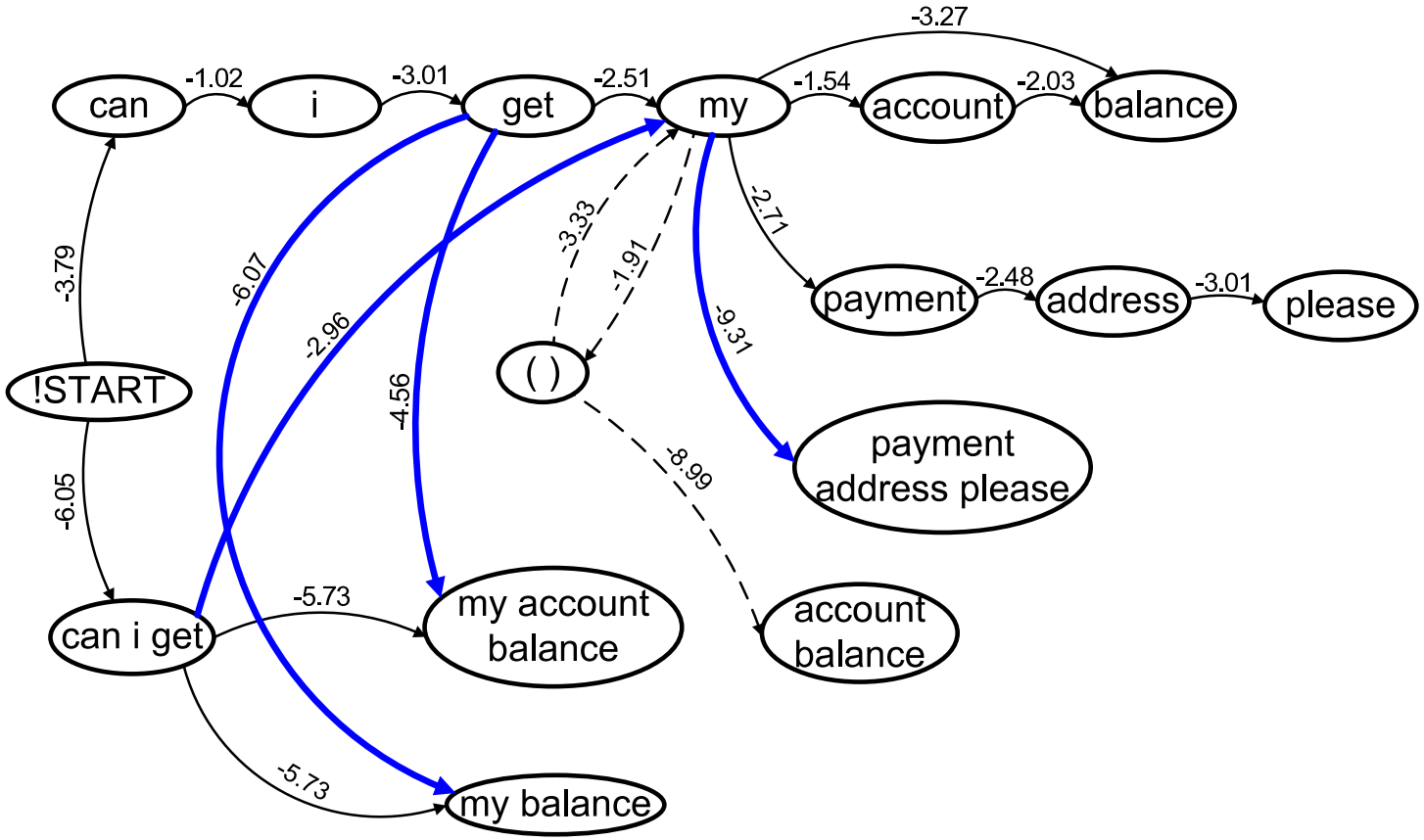


Figure 4.8: *WPLB (word phrase link bigram) language model (LM). All transitions required to parse the input utterance "can i get my payment address please" e.g. not all backoff transitions are shown. The backoff state is represented by empty brackets "( )", and log-likelihoods are shown on the arcs between nodes. Arcs in* bold blue *show the new transitions added to transform the WPB LM into the WPLB LM.*

$$\log(\Pr(\text{“[can] [i] [get] [my] [payment] [address] [please]”}|WPLB)) = -18.53 \tag{4.13}$$

$$\log(\Pr(\text{“[can i get] [my] [payment] [address] [please]”}|WPLB)) = -17.21 \tag{4.14}$$

$$\log(\Pr(\text{“[can] [i] [get] [my] [payment address please]”}|WPLB)) = -19.64 \tag{4.15}$$

$$\log(\Pr(\text{“[can i get] [my] [payment address please]”}|WPLB)) = -18.32 \tag{4.16}$$

4.5.2 Integrating phrase classes

Given that the phrases (including single words contained in the training segmentation) have been clustered using the method described in Section 4.4 (or any other method for that matter), the question is how can these clusters be used for language modelling?

A standard approach is to replace all phrases in the training segmentation with their respective class label [Nasr et al., 1999], and then estimate N-gram probabilities as before — all previously defined language models (WB, PB, WPB, and WPLB) can be adapted in this way. Figure 4.9 is an adapted version of Figure 4.8 (the WPLB model) that shows how phrases are replaced with their class label — this adapted model is termed the *Word-Phrase Class Link Bigram* (WPCLB)[11].

The states for concern in Figure 4.8 are all of the phrase states, i.e. “can i get”, “my account balance”, “my balance”, “account balance”, and “payment address please”. As shown in Figure 4.9, “can i get” is replaced by “[Class_499]”, “my account balance” and “my balance” are both contained in the same class and are replaced with “[Class_13]”, “account balance” is replaced by “[Class_1042]”, and finally “payment address please” is replaced by “[Class_1002]”. All of these classes, as well as some other examples, are shown in Appendix C.

[11]Similarly there is the *Phrase-Class Bigram* (PCB) and the *Word-Phrase Class Bigram* (WPCB).

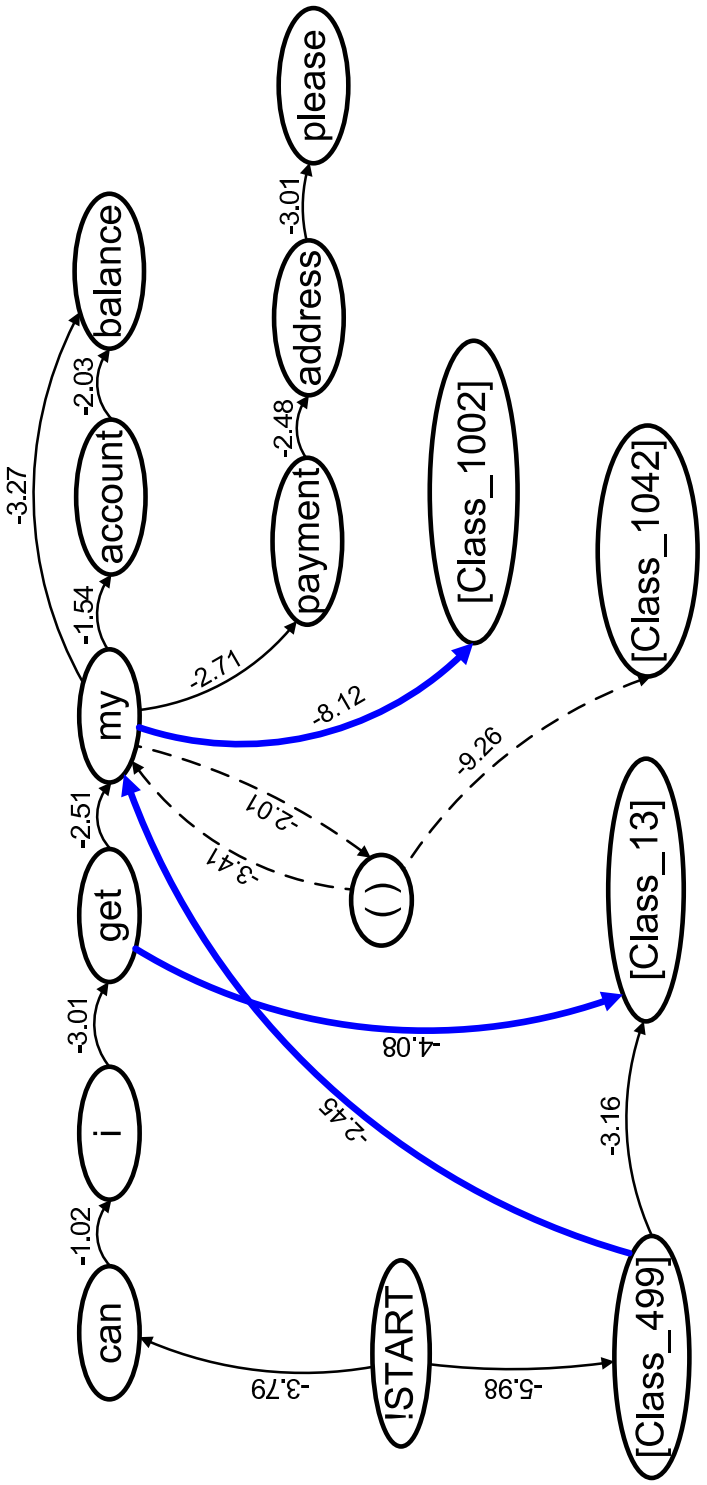


Figure 4.9: *WPCLB (Word-Phrase Class Link Bigram) language model (LM). Class labels replace phrases.*

The N-gram distribution of the newly introduced class labels defines a *global* distribution for *between-class* likelihoods. To complete the representation of the clusters, and to fully integrate the phrase clusters into the language model, a *local* distribution is required for the *within-class* likelihoods. As with Nasr et al. [1999] and Arai et al. [1999], a suitable representation of the clusters is the *Stochastic Finite State Automaton* (SFSA).

Figure 4.10 shows an actual class obtained from the HSF clustering method on the SD data with the phrase occurrences shown in brackets. An SFSA can be built for this class where each word in a phrase is represented by a separate state. The SFSAs are built using a stochastic *Grammatical Inference* technique [Parekh and Honavar, 2000] where each phrase is modelled as a separate string of word states which are then combined together in a *minimisation* process. The likelihoods are estimated by using every phrase in the class as a "positive example" [Parekh and Honavar, 2000] where each example is used to traverse the automaton which provides counts at each state — the likelihood on a given arc between two states is the ratio of the number of times each of the two states are visited during the traversal of the SFSA by each phrase token in the class. Figure 4.11 shows the SFSA for the class in Figure 4.10 (note that the probabilities given are *not* log probabilities).

can you get me (1)	could you give me (4)
can you give me (9)	could you please give me (3)
can you please give me (1)	could you please tell me (6)
can you please tell me (3)	could you tell me (9)
can you tell me (66)	should i (1)

Figure 4.10: *Class 174. Label after merging*: [(PRP_NNP), (PRP$_NN), null, NP, (AUX_VBN), (AUX_JJ), (VBG_IN), (WP_AUX), VP, SBAR, S, (TO_NN)] — **(MD_PRP)** — [(WRB_AUX), null, (IN_AUX), (WP_AUX), NP, (DT_NN), (DT_NNS), (WRB_RB), (WRB_JJ), (WP_JJ), (WRB_VBN), WHNP, (WRB_VBP), (WP_TO), (IN_MD), (DT_JJS), VP, (WRB_IN), (IN_PRP$), (WRB_NN), (PRP$_NN), WHADVP, (JJR_IN), (DT_CD), (PRP$_NNS), (PRP$_VB)]. *This class was generated when merging to 4000 classes using HSF clustering. The total number of phrases in the class is 103, with the count of each phrase shown in brackets.*

Using separate states in the SFSA for each word of a phrase means that the network can be easily integrated into a phone speech recogniser because the pro-

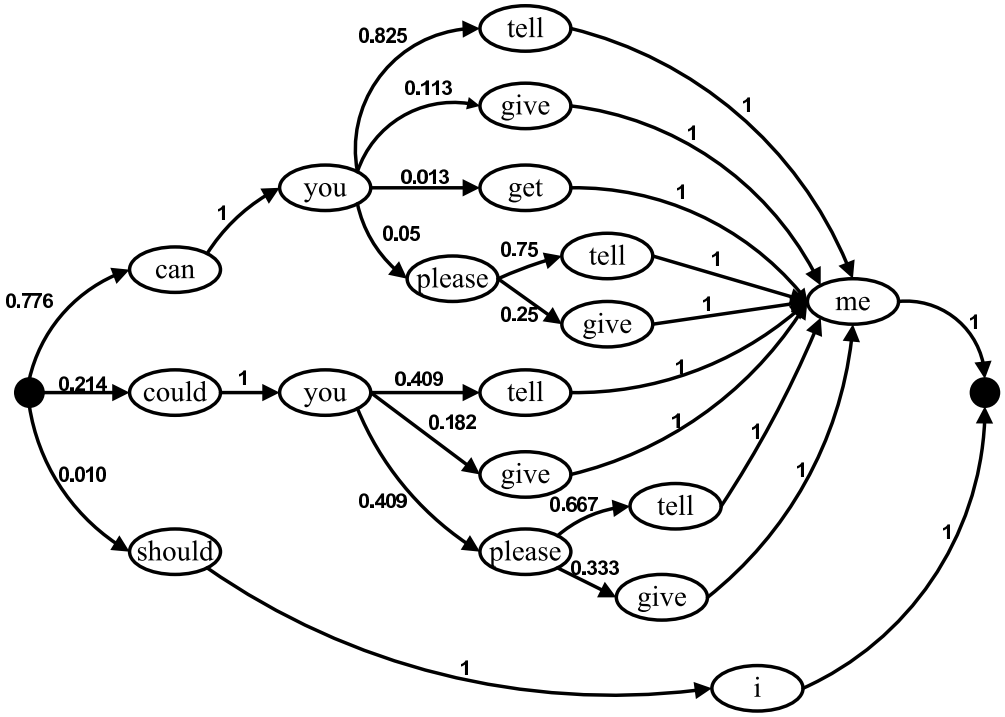


Figure 4.11: *The phrase class of Figure 4.10 represented as an SFSA. Note that the arcs hold* probabilities*, not* log *probabilities.*

nunciation dictionary will typically already contain the sequence of phonemes that represent a given word which means that SFSAs can fit into the hierarchy more easily. It also offers flexibility for modelling variations in phrases which are common in formulaic language [Wray, 2002]. However, example-based speech recognition requires that the SFSAs model the phrases as whole units, and thus the SFSA construction becomes a much simpler task — each phrase in the class is represented as a whole state and not divided into word states, while the probabilities of each phrase are estimated as their *relative frequency.*

Figure 4.12 shows the class of Figure 4.10 represented as an SFSA using a single state for each phrase. Integration of this representation into a phone speech recogniser now requires that the pronunciation dictionary contains every phrase that is used in the language model so that the correct sequence of phoneme models can be concatenated together to represent the phrase.

After the global and local probabilities are estimated for the phrases classes, the

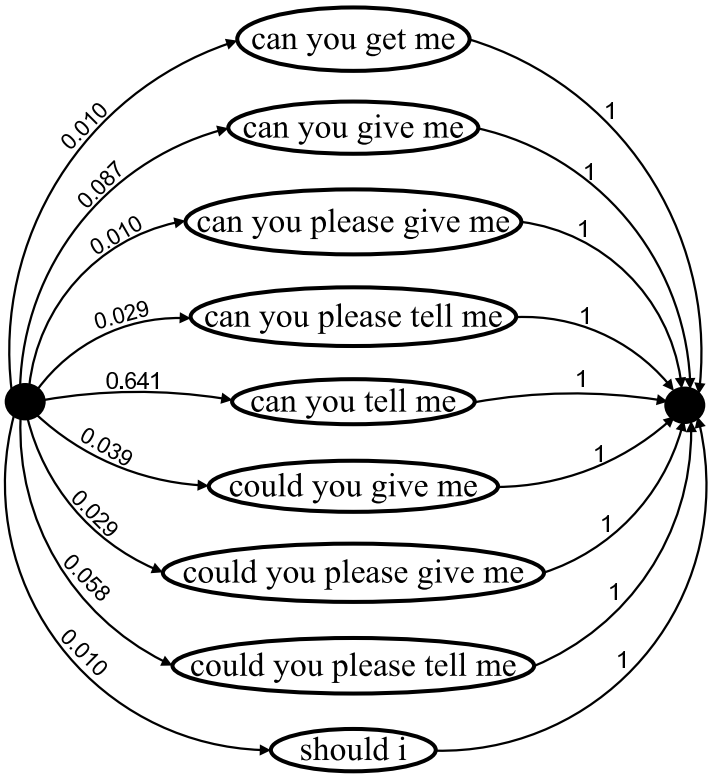


Figure 4.12: *The phrase class of Figure 4.10 represented as an SFSA with whole phrase states.*

remaining step is to "plug-in" the SFSAs for each class in place of the states that represented the classes. Figure 4.13 shows the result of integrating the relevant SFSAs into the WPCLB language model of Figure 4.9.

To summarise the integration of the SFSAs (local) into the N-gram framework (global), the following formal definition is given: the probability of a phrase w_{i-M+1}^{i} of length M words given the previous phrase w_{j-L+1}^{j} of length L words, where $w_{i-M+1}^{i} \in$ [Class_x] and $w_{j-L+1}^{j} \in$ [Class_y] is given by:

$$\Pr\left(w_{i-M+1}^{i} \middle| w_{j-L+1}^{j}\right) = \underbrace{\Pr\left([\text{Class\_x}] | [\text{Class\_y}]\right)}_{\text{global}} \underbrace{\Pr\left(w_{i-M+1}^{i} \middle| [\text{Class\_x}]\right)}_{\text{local}} \quad (4.17)$$

Equation (4.17) can be clarified with an example: given the previously defined WPCLB language model, in particular the paritions defined by Figure 4.9 and

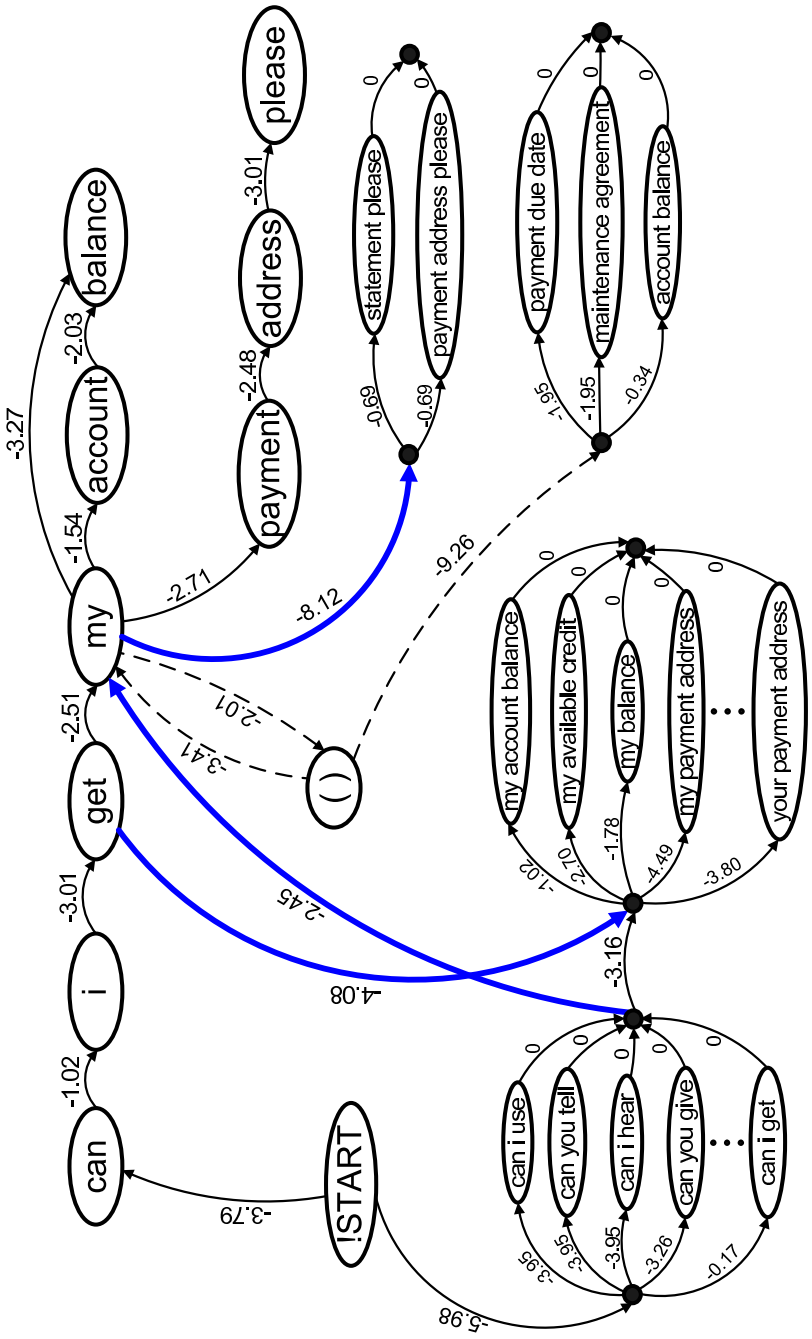


Figure 4.13: *WPCLB (Word-Phrase Class Link Bigram) language model (LM) with SFSAs to represent classes.*

Figure 4.13, the *log* probability of the phrase "my payment address" given the previous phrase "can i get" can be calculated as:

$$\begin{aligned}
&\log\left(\Pr\left(\text{“my payment address”}\middle|\text{“can i get”}\right)\right) = \\
&\log\left(\Pr\left([\text{Class_13}]|[\text{Class_499}]\right)\right) + \log\left(\Pr\left(\text{“my payment address”}|[\text{Class_13}]\right)\right) \\
&= -3.16 + (-4.49) \\
&= -7.65
\end{aligned} \tag{4.18}$$

Once all of the phrase classes are integrated into the language model, the token passing decoder can perform as normal because the structure of the language model is built into the VNSA, and so no adaptation of the decoder is required.

4.5.3 Adding a Bias to Phrase States

Because the work presented in this thesis is interested in the effects of using phrases with words in language modelling, compared to just words, a *phrase weight* is now defined which is a constant log-probability that is added to each arc that transitions *to* a phrase state.

The phrase weight is similar to the *word insertion penalty* (WIP) used in a speech recognition decoder (described in Section 2.5), except it is only applied to the phrase states, whereas the WIP is applied to all transitions in the language model between states (which can be words or phrases). The phrase weight can be viewed as a bias which either rewards or penalises the use of phrases depending on whether the weight is positive or negative respectively. Unlike with the WIP, the addition of the phrase weight is added to the language model *before* the decoding begins to avoid searching for phrase states each time the language model is loaded into the decoder.

4.6 Baseline Evaluation

This section will describe the evaluation of the techniques described in this chapter. The evaluation will use both language model *perplexity* (Section 2.2.3) and the percentage measure speech recognition *word accuracy*, which is defined by:

$$\text{word accuracy} = \frac{N - S - D - I}{N} \times 100 \tag{4.19}$$

where N is the total number of word labels in the transcription files, S is the number of substitutions, D is the number of deletions, and I is the number of insertions when aligning the hypothesised word strings from the recogniser to the actual word string transcriptions of each utterance using a dynamic programming (DP) string alignment method. For comparison of results to other system settings (i.e. different language models), the *Matched Pairs* statistical test [Gillick and Cox, 1989] is used to quantify whether or not the difference in the set of hypothesised strings of two systems is *statistically significant*.

The remainder of this section is as follows: Section 4.6.1 will describe the experiments and results on the SD call-routing data, while Section 4.6.2 will give the results for the experiments run on the SI RM dataset. Both datasets will be evaluated using HMM phone recognisers, previously described in Chapter 3, to provide a *baseline* measure for comparison to template-based recognition experiments that will be described in Chapter 6.

4.6.1 Speaker Dependent Results

Table 4.1 shows the word recognition accuracy for the SD call-routing data with the HMM-based recogniser for the previously described language models; WB, PB, WPB, WPLB, WPCB, and WPCLB. For the WPCB and WPCLB language models, which were created with the HSF clustering process (Section 4.4), the

optimal number of classes (in terms of word accuracy) is represented — for WPCB this is 2000 classes and for WPCLB it is 4000 classes. Table 4.1 also shows the average perplexity per word of each language model[12].

LM	PP	word accuracy (%)
WB	13.1	86.92
PB	n/a	86.79
WPB	11.56	87.52
WPLB	11.05	87.76
$WPCB_{2000}$	11.56	87.41
$WPCLB_{4000}$	11.35	87.79

Table 4.1: *SD perplexity (PP) and word accuracy on baseline HMM system for the WB, PB, WPB, WPLB, WPCB, and WPCLB language models where the class models reported are for the optimal number of classes, i.e. merged to 2000 and 4000 classes respectively. Each of the phrase-based language models uses a phrase weight of 0.5.*

Table 4.2 shows the Matched-Pairs tests on the recognition hypotheses for each language model. Each recognition system is compared to one another, and if one system is better than the other *statistically*, then the name of the system is entered into the table. For example, referring to Table 4.2, the WPB system is statistically better than the WB system, and so the WPB name is entered into row two, column four. When there is no statistical significance of the difference between two systems, then "same" is entered into the table. For example, the WB and PB systems are statistically equal (row two, column three). Taking a deeper look into Table 4.2 shows that the WPB, WPLB, and WPCLB systems are all judged to be significantly better than the WB baseline, but there is no statistical difference between each of the WPB, WPLB, WPCLB, and WPCB systems.

Figure 4.14 shows how the number of classes in the WPCB and WPCLB language models affect the word accuracy of each system, revealing a rise in the word accuracy until the optimal number of classes is found which is then followed by a rapid decline in the word accuracy when merging down to 500 classes from the

[12]The perplexity cannot be found for the PB model as it does not contain all of the vocabulary items that occur in the test data, i.e. the PB model does not use all the words of the training vocabulary; it combines them into phrases. Unless the test data contains exactly the same distribution of words and phrases to that of the PB model, then the perplexity cannot be found for the test data.

	WB	**PB**	**WPB**	**WPLB**	**WPCB**	**WPCLB**
WB		same	WPB	WPLB	same	WPCLB
PB			WPB	WPLB	WPCB	WPCLB
WPB				same	same	same
WPLB					same	same
WPCB						same
WPCLB						

Table 4.2: *Statistical significance tests on the SD test data. The Matched-Pairs test was used to determine if gains / losses in accuracy for different language models were statistically significant.*

initial number of classes which is 11,368 (right-to-left in the figure). As mentioned before, the optimal number of classes is 2000 for the WPCB model and 4000 for the WPCLB model. Figure 4.14 also shows the word accuracy when using the other language models as reported in Table 4.1.

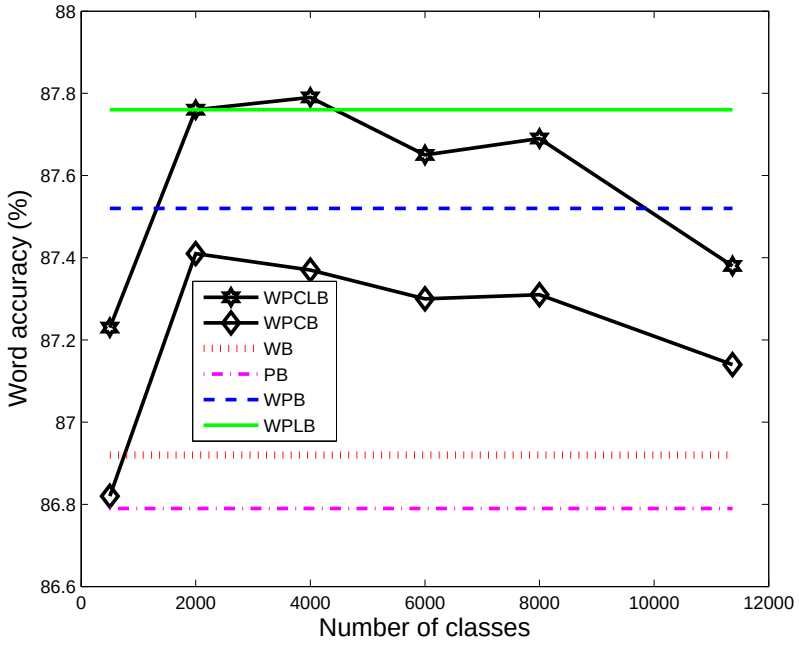


Figure 4.14: *Word accuracy on the SD test-set for HSF clustering with the WPCLB language model over varying numbers of phrase classes. WB, PB, WPB, and WPLB are shown for comparison.*

It is useful to visualise the perplexity of the language models in the same manner. Figure 4.15 shows how the perplexity *decreases* during the merging of classes to a local minimum (at 2000 classes for WPCB and WPCLB) followed by a

rise in perplexity. It is clear to see, by comparing Figure 4.14 and Figure 4.15, that the word accuracy is, to a certain extent, inversely proportional to the perplexity — although the minimum perplexity for WPCLB occurs at 2000 classes, while the maximum word accuracy occurs at 4000 classes.

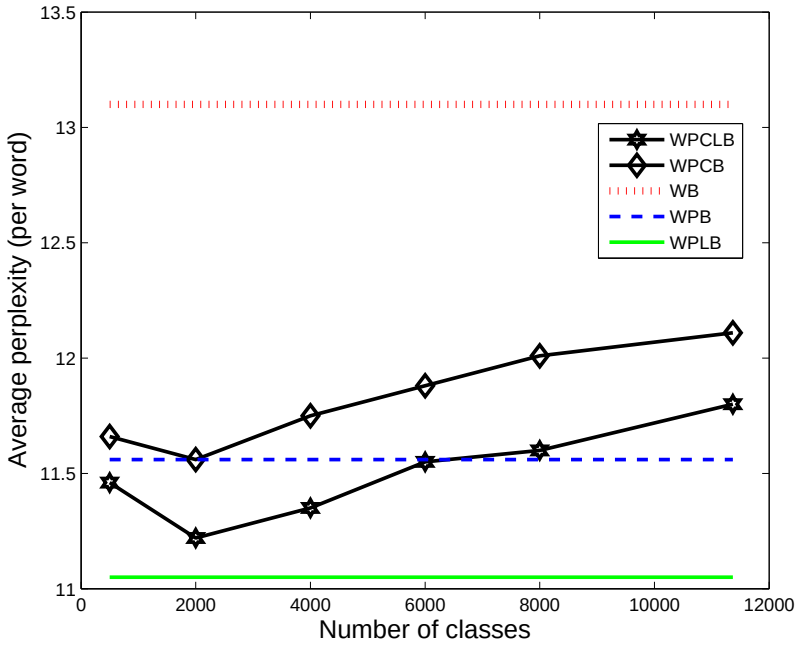
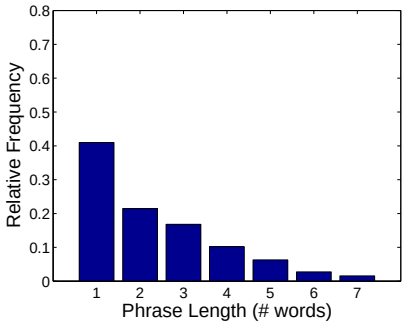


Figure 4.15: *Average perplexity per word on the SD test-set for HSF clustering with the WPCLB language model over varying numbers of phrase classes. WB, WPB, and WPLB are shown for comparison.*

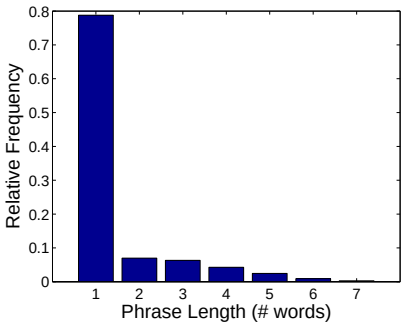
Figure 4.16 shows a collection of histograms showing the distribution of phrasal units on the training segmentation (Figure 4.16(a)) and used in recognition (on the test set) with the WPB and WPLB language models (Figure 4.16(b) and Figure 4.16(c) respectively). It is interesting to note, for both WPB and WPLB, that even though words (one-word phrase) account for only 40% of the phrases in the training data (segmentation), during recognition the language models guide the decoder to words over 70% of the time. This is likely to be because of a relatively poor generalisation of the training phrases to the test data, and so the words in the language models are used as a "backup" mechanism.

Comparing Figures 4.16(b) and 4.16(c) shows that there are some subtle dif-

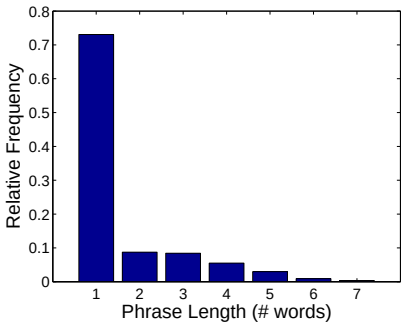
ferences between the distributions of the WPB and WPLB models. First of all, the percentage of words drops from approximately 79% to 73% respectively. This reduction of words in the WPLB recognition is due to an increase in the use of phrases, particularly two-word (64 more examples), three-word (85 more), and four-word phrases (48 more). The extra flexibility in the WPLB should allow better generalisation to the test data, as the decoder is given more choices for transiting between words and phrases which, as stated before, should mean that the decoder can use its "backup" option (i.e. the words) when a given sequence of the input is not well represented by the phrases, but it can then return to the phrase level when the input becomes a closer match to the training data. That said, it is important to stress, that although there is a big difference in perplexity of the WPB and WPLB models, this does not transfer to the word recognition accuracy which, as previously mentioned, sees no improvement *statistically*.



(a) *All training phrases.*



(b) *Used phrases in WPB recognition.*



(c) *Used phrases in WPLB recognition.*

Figure 4.16: *Histograms showing the number of words per phrase as a relative frequency for SD data.*

4.6.2 Speaker Independent Results

Table 4.3 shows the HMM baseline word accuracy results for the RM dataset along with perplexity scores for the WB, WPB, and WPLB language models[13]. Results are shown for the development set (dev set) and evaluation sets (oct89, feb91, and sep92), with (✓) and without (✗) Vocal Tract Length Normalisation (VTLN) applied to the audio. All recognition parameters such as language model scaling factor and word insertion penalty are optimised on the dev set and then applied to the evaluation sets.

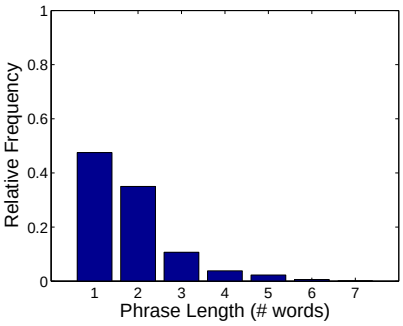
			Dev (%)	**Evaluation (%)**		
LM	**PP**	**VTLN**	**feb89**	**oct89**	**feb91**	**sep92**
WB	30.44	✗	91.33	89.94	92.11	86.32
		✓	91.76	89.34	91.95	86.32
WPB	27.7	✗	91.88	90.09	92.59	86.95
		✓	92.66	89.2	91.87	87.5
WPLB	27.43	✗	92.07	89.9	92.59	86.6
		✓	92.82	89.64	92.27	87.38

Table 4.3: *Word Accuracy on the RM dev set feb89 and three test sets oct89, feb91, and sep92 for WB, WPB, and WPLB language models with and without VTLN, using HMM-based recogniser. Language model perplexity (PP) is also shown. The WPB and WPLB based sytems use a phrase weight of 0.5 which was optimised on the feb89 development set.*

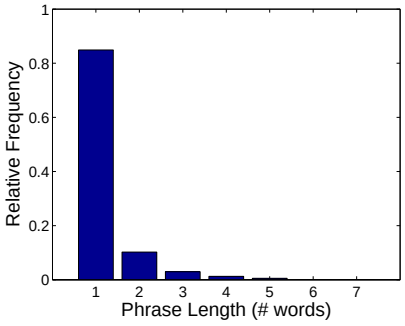
For the VTLN experiments, the HMMs are trained as described in Section 2.6.1.1, but for the recognition, there is a change to what was described in Section 2.6.1.2. Instead of optimally warping the input for each system (i.e. changing language models), the input was only warped for the WB baseline system and then stored. For the subsequent experiments with WPB and WPLB, the recognition was run on the WB warped input as a normal recognition pass (i.e. just one pass, and no warping). This approach was taken for consistency with the template-based recognition which will be presented in Chapter 6. For now, the reader is asked to accept this approach in the knowledge that justification will be given in Chapter 6.

[13]The PB language model was not used in the experiments as it was clear that it was not a good model for unseen data.

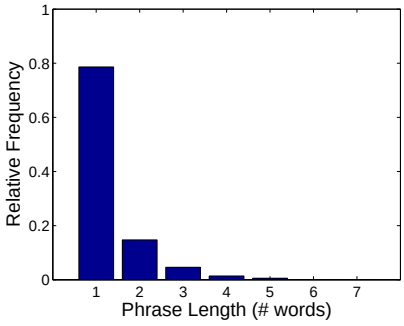
The Matched-Pairs test was used to compare the baseline recognition systems for each language model within each evaluation set, both with and without VTLN applied. All system comparisons were classified as statistically the same. To see why there is so much parity with the HMM results on RM, it is useful to analyse the phrase histograms, as with the SD data in Section 4.6.1.



(a) *All training phrases.*



(b) *Used phrases in WPB recognition.*



(c) *Used phrases in WPLB recognition.*

Figure 4.17: *Histograms showing the number of words per phrase as a relative frequency for the combined RM datasets oct89, feb91, and sep92.*

The first thing to note, from Figure 4.17(a), is that the distribution of the training phrases tapers away much quicker than the SD data from phrases of three or more words. There are more two-word phrases, but fewer three-word phrases. The fact that there are more shorter phrases (and more words) indicates that the RM data does not contain as much formulaic production as the SD call-routing data. It is clear from Figures 4.17(b) and 4.17(c) that, as with the SD data, the WPLB language model is constructed such that the best path will go through longer phrases than when compared to the WPB language model. However, both language models choose a higher number of words (85% for WPB, and 78% for

WPLB), and thus very few longer phrases of two or more words. This can be contrasted to the SD decoder which chooses words 79% and 73% of the time for WPB and WPLB respectively.

4.7 Conclusions

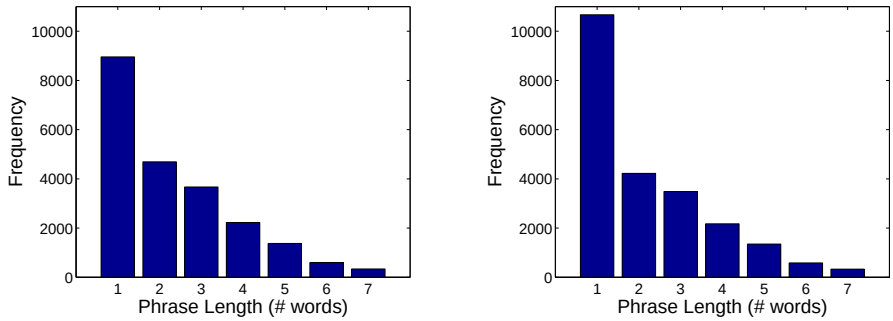
This Chapter attempted to, first of all, acquire frequently occurring phrases and then integrate them into an architecture suitable for speech recognition. The chapter began, in Section 4.2, by giving a survey of the literature related to phrase-based language modelling, both within speech recognition applications, and purely as language modelling techniques. Section 4.3 described how a multigram segmentation algorithm could be used to acquire phrases, purely from transcriptions of speech. Section 4.4 introduced a novel technique for clustering phrases, called the *Hybrid Syntactic Formulaic* (HSF) clustering algorithm, that used the acquired phrases from the multigram segmentation and combined them with parse-trees to assign syntax-based labels which could then be used to group other phrases. The clustering algorithm was complete with a class merging procedure which merged the "closest" pair of classes until a pre-defined number of classes was reached — a vector based approach with cosine similarity was used.

Section 4.5 was concerned with the integration of both the acquired phrases and the phrase classes, previously grouped by the HSF clustering algorithm, into the speech recognition architecture. It began by, in Section 4.5.1, formulating the language models within different topology schemes, describing first of all the *Phrase Bigram* (PB), followed by the *Word Phrase Bigram* (WPB), and finally the *Word Phrase Link Bigram* (WPLB). Starting from a basic bigram distribution of the phrases (PB), gradually more information was built into the language models, such as word bigram (WB) information, and then new bigram transitions between words and phrases within the language model, creating bigram transitions that were previously unseen. Section 4.5.2 showed how the phrase classes, represented

by Stochastic Finite State Automota (SFSA) could be integrated into the N-gram framework (bigram in this case), with the notion of global (between-class) and local (within-class) probabilities, and Section 4.5.3 described a *phrase weight* which is a log-probability that is added to all arcs (as a constant) in the language models that transition *into* phrase states and acts as a bias to influence how a decoder will use the phrases.

Finally, Section 4.6 gave the evaluation of the baseline models. The techniques described in the chapter were applied to the SD call-routing data and RM datasets (speaker independent), which were then used to run speech recognition experiments using HMM monophone recognisers with multiple mixture components. The recognition experiments were evaluated with word accuracy. Language model perplexity was also calculated for each of the language models previously described. For the SD data, both the WPB and WPLB language models gave statistically significant improvements over the WB baseline measure, as did the WPCLB for 4000 classes, although it was shown that the WPLB system gave no significant improvement over the WPB system. For the RM evaluation set, it was shown that the WPB and WPLB models gave no significant improvements over the WB model. A suggestion for why there were no significant improvements on the RM evaluation set was that the phrases available in the training data (from the multigram segmentation) were shorter than for the SD data, with a large number of words and two-word phrases. When the recognition outputs were analysed, it was clear that the decoder had a preference for even shorter units — over 80% of the recognised units were words when using the WPB and WPLB language models while that figure was less than 80% for the SD data.

This implies that the reason the phrases are not chosen as often as words is because the phrases do not generalise well to the unseen data. The methods described in this chapter keep the acquired phrases fixed, and do not allow for any small variations in the structure of the phrases which is likely to occur in the unseen data. To validate this claim, a HMM-based speech recognition experiment was run



(a) *Available training phrases (from segmentation).*

(b) *Used phrases in WPLB recognition on training data.*

Figure 4.18: *Histograms of chosen units by the HMM decoder on the SD training data compared to the available training phrases. Figure (a) actually just shows the histogram of the training text segmentation, it does not include all of the individual words from the original unsegmented text.*

on the SD call-routing *training* data, giving a word accuracy of 96.81%. However, the key information here is in the analysis of the units chosen by the recogniser, shown in Figure 4.18. Clearly the decoder chooses a much higher number of *longer* units than on the test data, resulting in a very similar distribution of unit length to the training segmentation which provides the phrases (the histogram does not contain the words of the training text that were combined with the phrases which are also available to the decoder, hence the number of one-word phrases should also be higher in Figure 4.18(a)). This is strong evidence of the chosen phrases not generalising well to the test data.

Chapter 5

Bottom-up Template Selection

5.1 Introduction

The datasets used in this research are moderate in size (refer to Chapter 3), yet still produce a large number of templates, with the RM and the SD (Speaker-Dependent) call-routing datasets producing approximately 52,700 and 64,860 templates respectively, for only word and silence templates. When the phrase templates are used, there is a typical increase of 11,350 and 12,900 templates respectively.

If the entire template database was to be used with a general token passing decoder (Section 2.3.4) using an *ergodic* network[1], then the branching factor at the end of each template would be in the order of 2.8 billion (for the smaller RM dataset), assuming that there is no pruning applied. Even with pruning, there is a huge overhead when passing tokens from template to template — a token that survives to the end of a template is then copied and passed to all other templates in the reference database. Language models can reduce the branching factor, although using backoff schemes means that the branching factor remains high (Section 2.2).

[1] An ergodic network is a network that allows any template to be followed by any other template, including itself.

To feasibly run a template-based decoder in reasonable time requires some kind of reduction in the template search space. An acoustic *look-ahead*, or pre-processing stage, has been popular in HMM systems, using simplified acoustic matching to prune the search space [Bahl et al., 1993; Ortmanns et al., 1997]. De Wachter et al. have applied this idea of an acoustic pre-decoding pass to a phone template-based recogniser, searching for templates that are an approximate match to the input [De Wachter et al., 2003].

The remainder of this chapter describes the template selection method of De Wachter et al. [2007] in detail, and gives the details of the extensions and alternatives to this method that have been developed in the work presented in this thesis. The chapter is structured as follows: Section 5.2 describes a Vector Quantisation (VQ) as an approximated k-nearest neighbours (KNNs) selection of reference frames to the input. Section 5.3 describes the Time Filter algorithm [De Wachter et al., 2003] as a method for selecting templates from evolving KNNs, while also introducing a backward pass of the algorithm in Section 5.3.1, with an alternative Sigmoid-based distance normalisation, which controls the strength of normalisation dependent on template length and a pre-defined weight, presented in Section 5.3.2. Section 5.4 describes a hierarchical LDA (Linear Discrimant Analysis) classifier which acts as a filter to the selected template candidates, describing methods used to extract suitable features for the LDA (Section 5.4.1) and the formulation of the hierarchical LDA into a decision tree (Section 5.4.2). Section 5.5 describes the evaluation of the methods described in this chapter, not with word recognition accuracy from a speech recogniser, but with methods such as how well the selected templates match the input utterances, and classification performance of the LDA filter. Finally Section 5.6 summarises the findings in this chapter and gives conclusions.

5.2 Vector Quantisation for K Nearest Neighbours selection

The bottom-up template selection begins with the selection of a set of k reference frames that are a close match to a given input frame, which are then used to find approximately matching templates (Section 5.3). This selection of frames is a K-Nearest Neighbour (KNN) problem, for which De Wachter et al. introduce an extended version of the Roadmap algorithm [Povey and Woodland, 1999], called the KNN Roadmap [De Wachter et al., 2004].

The KNN Roadmap, which is constructed in an offline training algorithm, is a graph, where each frame in the reference database is a node in the graph. Arcs between nodes are bi-directional, and connected using a *hill-climbing* search, with connections forming between nodes that are similar. The KNN Roadmap is actually built up of sub-graphs, where each state class, previously used for local distance measures (Section 2.3.2), is connected separately. We chose not to adopt this method for KNN selection, as it was reported that the KNN Roadmap "*only narrowly outperformed a simple brute force KNN calculation*" [De Wachter, 2007] for datasets of moderate size, such as those described in Chapter 3 (it is suggested that the true benefit of the KNN Roadmap will be for much larger datasets, with reference database sizes beyond 10,000,000 frames).

Our preferred approach is to use *Vector Quantisation* (VQ) implemented with *k-means clustering*[2], which aims to minimise the *sum-squared distance* within each cluster [Webb, 2002]. As with the KNN Roadmap, each state class is clustered separately, with the number of clusters dependent on the size of the class. The sum-squared distance for a cluster C_k is given by:

[2]It should be noted that k for k means clustering and k nearest neighbour are not related and are local to each algorithm or method.

$$SS_k = \sum_{\mathbf{y}_i \in C_k} |\mathbf{y}_i - \mu_k|^2 \tag{5.1}$$

where $\mathbf{y}_i$ is a frame within cluster C_k, and μ_k is the mean frame of C_k.

The technique uses the standard k-means clustering algorithm: within each class, k random frames are chosen to be initial cluster centres — k is proportional to the number of frames in each class. The remaining frames are assigned to the nearest cluster, using a direct comparison to the cluster centre. The mean frame is then calculated for each cluster and set to be the new centre. The algorithm then iterates, with frames being assigned to the nearest cluster, and new mean frames being calculated for the cluster centres until a convergence point is met, or if no convergence is reached, until a maximum number of iterations is reached. The algorithm converges when there is no change in the sum squared distance of each cluster, or when the mean change per cluster is lower than some pre-assigned threshold.

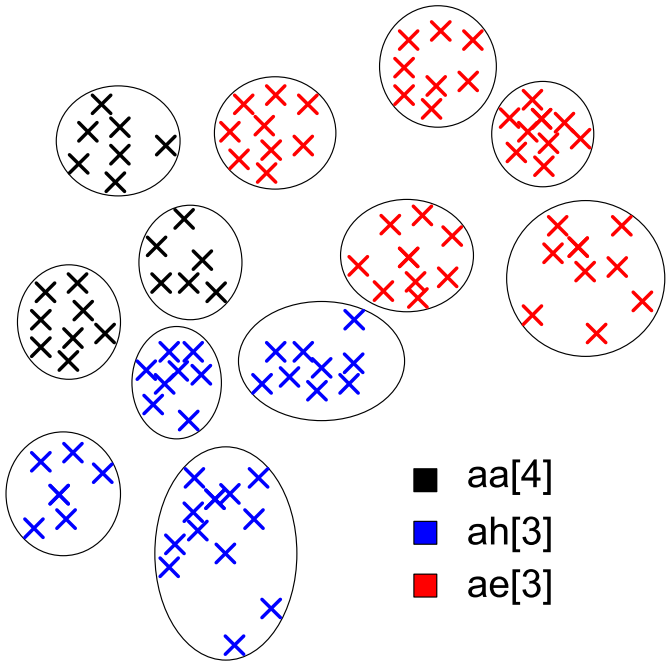


Figure 5.1: *Vector Quantisation of frames in 3 different classes; aa[4], ah[3], ae[3]*

In terms of VQ, a cluster mean is a *codeword* and the set of cluster means is the *codebook*. We formulate this problem as VQ, and not just k-means clustering, as the selection of KNNs to an input frame is determined by the closeness of the input frame to the cluster centres (codewords). At this stage of the processing the frames within the clusters are temporarily ignored, and effectively reduced to one codeword, i.e. the mean of their cluster.

To select the KNNs, the current input frame is compared to all codewords, over all classes, with the codewords being ranked in order of closeness to the current input frame. Each frame that was assigned to the codewords is then loaded into the KNN list, in order of codeword closeness, until K is reached or exceeded — every frame within the codewords is used, therefore K is an approximate target because reading the last codeword will often cause more frames than K to be loaded.

Figure 5.1 illustrates this process. It shows a subset of frames from the state classes aa[4], ah[3], and ae[3], and the clusters that contain them. It is important to note that the example gives idealised data, i.e. the frames are clearly separable. What Figure 5.1 shows is that it is a possibility that frames within one class may actually lie closer to frames, and thus clusters, from other classes. Using VQ for KNN generation is an effective measure for efficiency, but by grouping frames into potentially large clusters, it can only be used as an *approximate* KNN selection method because it is the cluster means that are used to select the KNNs.

5.3 Time Filter Algorithm

The aim of the time filter algorithm is to reduce the search space for the DTW decoder. It iterates over the input frames, looking for reference templates that are an approximate match to the input: When comparing the input to the templates, using a distance matrix approach (refer to Section 2.3.3), an approximate match is defined as one where the best path through the matrix is *approximately diagonal*

[De Wachter et al., 2003].

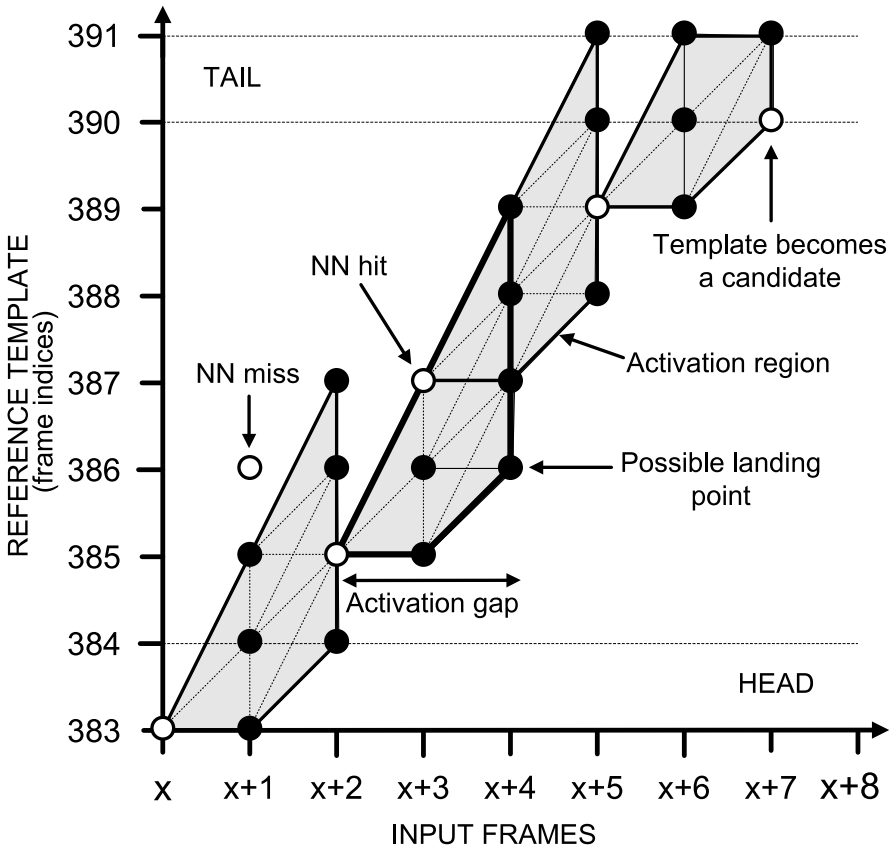


Figure 5.2: *The Time Filter Algorithm. White circles represent nearest neighbours (NNs) and black dots represent legal entry points for NNs within the reference template over time. Activations evolve over time from the head region to the tail region of the template.*

Both the time and memory consumption would be far too great if all templates were used in the time filter algorithm. Therefore, for each input frame, a list of k nearest neighbours (KNN) is generated (Section 5.2) and used to activate templates that contain those KNN frames. Figure 5.2 gives an example of the time filter in action, focused on one template. If one of the KNNs is contained at the start of a template, known as the *head* region, then an *activation* region is created. The head region is a pre-defined number of frames at the beginning of a template which can vary from template to template, usually increasing in size as the template length increases. The activation region spawns from the point at which the nearest neighbour (NN) is contained, also known as a NN hit, and is designed to be similar to the kind of region that would be defined by a DTW local

search, such as the Itakura Distance (Section 2.3.3).

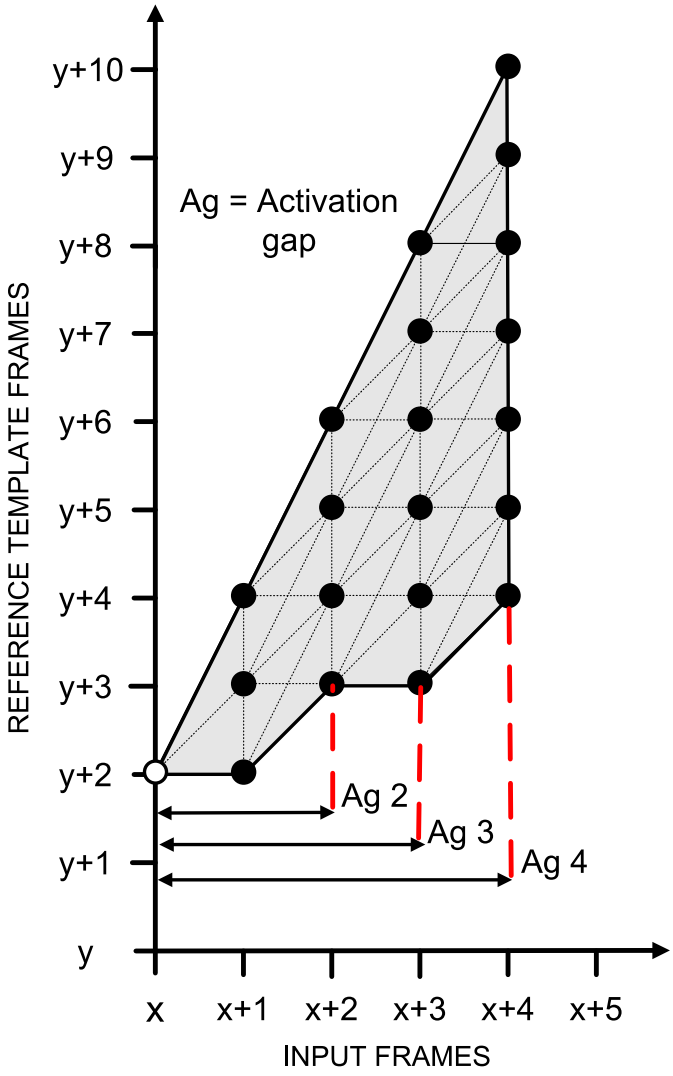


Figure 5.3: *Activation regions. The effect of increasing the activation gap upon the activation region (shaded area).*

The size of the activation region is determined by the *activation gap*, which is, again, a pre-defined number of frames. Figure 5.3 shows the effect of increasing the activation gap upon the activation region. Clearly, the larger the activation gap is, the larger the activation area, and thus the higher the chance that one of the KNNs will be contained within an activation region. In our experiments, the activation gap can vary in size for differing template lengths, or can remain fixed for all lengths. An activation region can only be created from within the head region, or from within another activation region; if one of the KNNs lands outside

of the head region and other existing activation regions then it is ignored, and the next NN is processed.

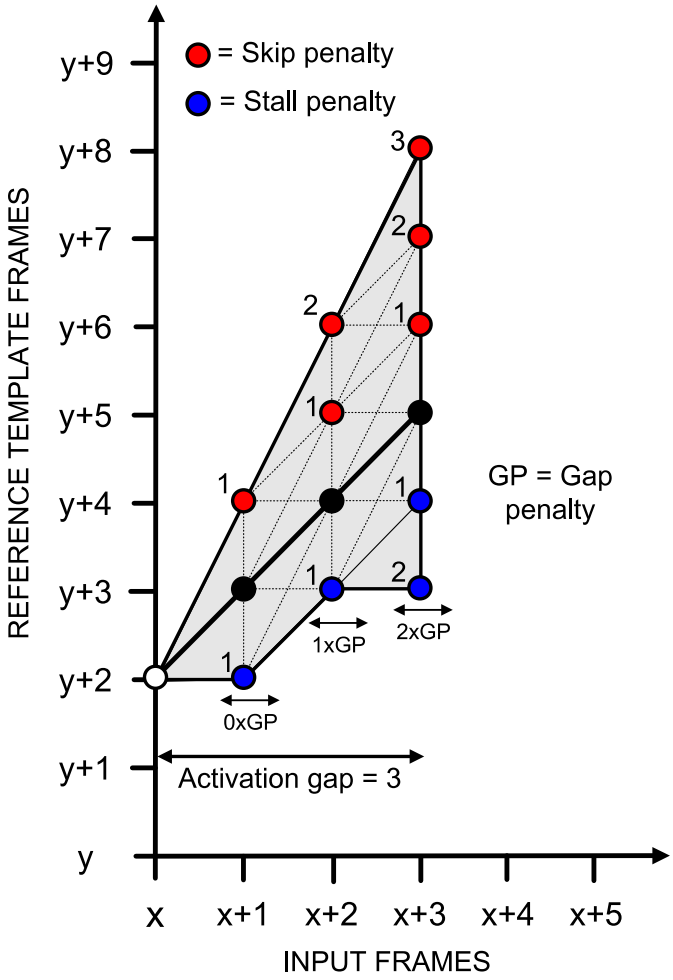


Figure 5.4: *Local penalties within activation regions. Stall and skip penalties are multiplied by a factor (next to entry points) dependent on how far from the diagonal of the region they are. A gap penalty (GP) is added to the score based on distance from the start point of the region to the NN hit.*

The generation of new activation regions from within existing activation regions from the head to tail regions of the template define an *activation path*. Each activation path originates from the activation regions that were created in the head of the template, creating new paths that branch off from existing paths in different directions, depending on the NN. Activation paths can also diverge back to the same point, and as with the Viterbi and the Token Passing algorithms (Section 2.3.4), when an NN hit occurs in more than one activation region, the associated paths of each region are effectively merged by retaining the activation region (and

its path) with the best score (lowest distance in our case) and destroying the remaining activation regions (and thus paths) that contained the NN.

Every time an NN hit occurs in the head of a template, the current input frame time is stored, and the distance between the current input frame and the NN is calculated and stored (using measures defined in Section 2.3.2). When an NN hit occurs in an existing activation region, the start time of the activation path is passed to the new activation region and stored. Again, the distance between the current input frame and NN is calculated, but this time additional penalties (which are closely related to DTW penalties (Section 2.3.3)) are added and the new distance is added to the existing distance for the current activation path.

Figure 5.4 shows the cases for applying penalties to the score of an activation path. For a given activation region, all of the possible NN points are shown, with the penalty that would be applied to them. Every point above the central diagonal of the activation region is classed a skip penalty, and everything underneath the diagonal is classed as a stall penalty. Each position has an associated factor with it, which is determined by the distance from the diagonal, and used to increase the penalty. There is also a *gap penalty* which is used to penalise gaps between NNs in the iteration of the input frames. The gap penalty, like the skip and stall penalties, has an associated factor which is determined from the distance of the NN to the origin point of the activation region in terms of number of input frames.

To clarify the penalty measures, and application of, consider an example where a NN hit occurs at $(x+2, y+3)$ in Figure 5.4. The penalty score will consist of a gap and stall penalty. In both cases the factor of the penalty is only one. Another example could see a NN hit at point $(x+3, y+8)$. This is at the far edge of the activation region, and thus receives a heavy penalty. A skip penalty, multiplied by the factor three, is added to the score along with a double gap penalty. A final example covers NN hits that occur on the diagonal of the activation region. Take point $(x+2, y+4)$ to be NN hit: there are no skip or stall penalties for this point, but there is still a gap penalty added (factor 1). It is important to pe-

nalise the NN hit on the diagonal, as it has not matched the input frame for frame.

Algorithm 5.1 The Time Filter Algorithm.

```
1: {X is the input frame sequence}
2: {NN(k) represents the kth nearest neighbour (NN)}
3: {templateLook(NN(k)) is a lookup for the template containing the kth NN}
4: for i = 1 to |X| do
5:    Get KNNs for X(i)
6:    for k = 1 to K do
7:       {Check NN(k) for hit}
8:       Y = templateLook(NN(k))
9:       if NN(k) in head of Y then
10:         Create activation region
11:         Update score
12:      else if NN(k) contained in activation region of Y then
13:         Update score
14:         if NN(k) in tail of Y then
15:            Select template Y as candidate
16:         else
17:            Create Activation
18:         end if
19:      end if
20:   end for
21: end for
```

The process of finding KNN hits inside activation regions continues until there is an NN hit that lands within the *tail* region of a template, within an existing activation region: the tail region, like the head region, is a pre-defined number of frames, typically dependent on the length of the template, and in this work, the tail is the same length as its corresponding head region. At this point, the current input frame number is stored as the end time, and the template in question is selected to become a candidate for the decoder (Chapter 6). For the example in Figure 5.2, the template is selected as a candidate with a start time at x and end time at $x + 7$. De Wachter et al. use both the start and end time to construct an *activation graph*, which connects templates by their start and end times and is used in a hierarchy of graphs within the decoder architecture [De Wachter, 2007]. Although we store both the start and end times for a selected template, only the

start time is currently used in the decoder (see Chapter 6).

Figure 5.2 shows an example of just one template within a subset of the input sequence. The Time Filter algorithm operates over all reference templates for each frame of the input. Using a simple lookup, the template that contains a NN (which is a frame within one of the reference templates) can be found quickly and the NN hits can be processed for each template. The KNN list can contain frames from many different templates, so it is important to note that at any one time (input frame) there are many activation regions within each template that contains hits from the KNNs. Algorithm 5.1 gives a summary of the basic Time Filter algorithm.

Once all input frames have been processed, there remains a list, for each input frame, that contains the selected templates for their given start time. At this point, the template lists for each time frame can be thresholded. In this work we use a Gaussian distribution of the activation path scores at each time frame, with the threshold set at a pre-defined number of standard deviations from the mean.

5.3.1 Adding a backward pass to time filter

The minimal template unit used in this work is the word-level template with phrase-level units of up to seven words as the maximum length; this means that the number of frames in each template, in most cases, is much larger than that of phone-length templates. This can cause a potential problem with the Time Filter algorithm. For a template to be selected as a candidate for the decoder, at the very least an NN has to occur within the head region of a template. If the head region is too small, then there is a strong possibility that a good template match is not selected — it may be that most of the template matches very well with a sequence of the input except the head region. If, for example 90% of a template is a match, then this template is a very good candidate for the decoder, but if there is no NN hit in the head region then the Time Filter would ingnore it; the closely matching portion of the reference template is never actually reached by the Time

Filter.

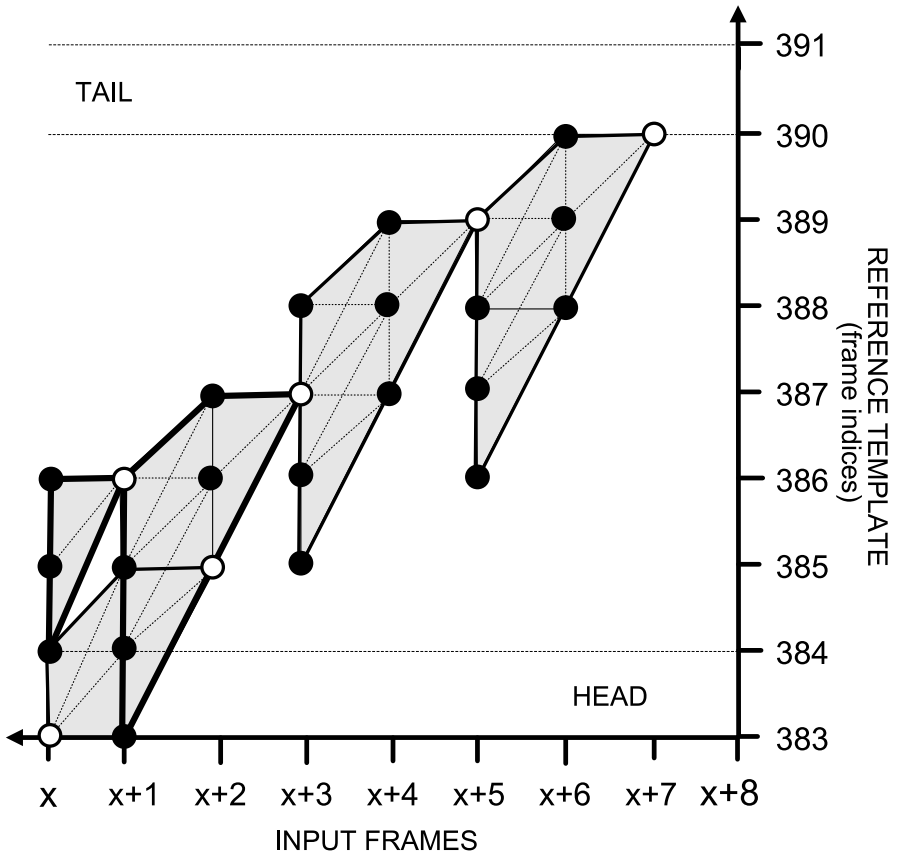


Figure 5.5: *The backward pass for Time Filter. Unlike the forward pass (see Figure 5.2), the input frames are processed in reverse order with activations initialised in the tail region and templates selected as candidates when a nearest neighbour (NN) hit occurs in the head region of the template within an existing activation. White circles represent NNs, while black circles represent legal landing points of NNs.*

To try and combat this side-effect of the Time Filter, we introduce a *backward* pass to the algorithm. The motivation is that if a template is a good match to the input, apart from the first few frames (which cover the head region), then doing a backward pass of the Time Filter algorithm will allow activation paths to cover a significant portion of the template before arriving at the sparser (in terms of NN hits) section of the template. By allowing the activation paths to evolve (as opposed to not creating any activation regions at all) from the tail, there will be an increased chance that one of the activation paths deviates enough to allow a NN hit in the head region of the template.

Figure 5.5 shows an example of an activation path evolving from the tail of the

template to the head, iterating backwards over the input frames. The activations follow the same constraints as the forward-pass, but effectively flipped upside down (visually), which leads to slightly different activation paths for the same set of KNNs. The implementation of the backward algorithm is an inversion of the forward pass: for an activation path to be created, a NN has to create a hit in the *tail*, and for a template to be selected as a candidate, a NN hit has to occur inside an existing activation region within the *head* region. The start time and end time of an activation path are inverted for the backward pass, with the time at which the template is selected being the start time within the head region, and the end time is the time at which the first activation region was created within the tail region for the selected activation path. Penalty scores are also inverted in the sense that NN hits, within an activation region, below the diagonal of the activation region, are penalised with the skip penalty (the stall penalty for the forward pass), and NN hits above the diagonal are penalised with the stall penalty (the skip penalty for the forward pass).

Once the forward and backward pass of the Time Filter are completed, the two sets of selected template candidates are merged by start time (more precisely, the time at which a NN hit occurs within the head region, i.e. when the activation is created for the forward pass, and when the activation path is terminated for the backward pass). For every time frame, the template candidates from the forward and backward pass are combined into one list, and where identical candidates exist (i.e. the same template with the same start times), the candidate with the highest score is discarded. Once both passes are merged, thresholding can be applied as before.

5.3.2 A length-based template score normalisation

As the time filter algorithm is based on DTW, a template's distance is based on the distance between input frames and template frames, as well as additional penalties for skips and stalls. This means that longer templates will accumulate

more penalties as the input is processed (unless they are perfect matches). To normalise a template's score, De Wachter et al. store the number of NN hits along the activation path of the selected template [De Wachter, 2007]. It is not explicitly stated how this is used, but the author of this thesis has made the assumption that the distance for the template's activation path is divided by the number of NN hits. So, the normalised distance, $D_{NN}(\mathbf{Y}, t)$, of a template $\mathbf{Y}$, using the number of NN hits for the templates activation path at start time t is given by:

$$D_{NN}(\mathbf{Y}, t) = \frac{D(\mathbf{Y}, t)}{C_{NN}(\mathbf{Y}, t)} \tag{5.2}$$

where, $\mathbf{Y}$ is the reference template, $D(\mathbf{Y}, t)$ is the pre-normalised acoustic distance of $\mathbf{Y}$ at start time t, and $C_{NN}(\mathbf{Y}, t)$ is the number of NN hits for the activation path of template $\mathbf{Y}$ at time t.

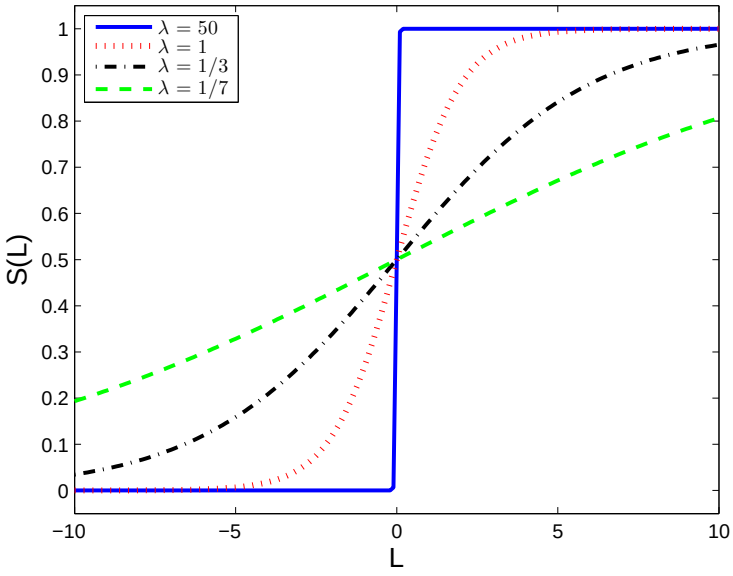


Figure 5.6: *General sigmoid function. By varying λ in Equation (5.3), the slope of the curve can be adjusted.*

In this work, we are interested in the effects of using phrases within speech recognition and how the different lengths of templates affect the recognition of an input sequence. To investigate these effects, we introduce a heuristic-based

normalised distance measure which uses a form of the *sigmoid function* [Mitchell, 1997], which is often used in *neural networks* as a non-linear transfer function that is used to isolate input pathways from neurons [Anderson, 1995]. Figure 5.6 shows an example of the general sigmoid function for $L \in [-10, 10]$ with varying weights λ, and is defined by Equation (5.3):

$$S(L) = \frac{1}{1 + e^{-\lambda L}} \tag{5.3}$$

By varying λ, the sigmoid can be controlled to be anywhere between a linear function ($\lambda \approx 1/7$ in Figure 5.6) and an "on-off" function ($\lambda \approx 50$ in Figure 5.6). The normalised distance measure, $D_\lambda(\mathbf{Y}, t)$, uses the sigmoid form of Equation (5.3) to incorporate the length of the template. The exponential term is no longer negative as we are seeking to *lower* the score of longer templates so that more phrase templates can be selected as candidates for the decoder, and hence the effects of using phrasal templates within the decoder can be more throughly investigated:

$$D_\lambda(\mathbf{Y}, t) = \frac{D(\mathbf{Y}, t)}{(1 + \lambda|\mathbf{Y}|)(1 + e^{\lambda|\mathbf{Y}|})} \tag{5.4}$$

where, $|\mathbf{Y}|$ represents the length of template $\mathbf{Y}$, λ is a user-defined weight, and $\mathbf{Y}$ and $D(\mathbf{Y}, t)$ are defined as before. The extra term, $(1 + \lambda|\mathbf{Y}|)$, is added to the denominator as a scalar to emphasise the length of the template and to introduce greater separation for varying values of λ. Figure 5.7 shows the effect of Equation (5.4) for varying lengths of templates. The linear normalisation method, where the template's score is divided by the template's length, is also shown for comparison. It is important to note that only the positive part of the sigmoid is shown in Figure 5.7, unlike in Figure 5.6, as there are no negative template lengths, i.e. we are only concerned with templates of length greater than one.

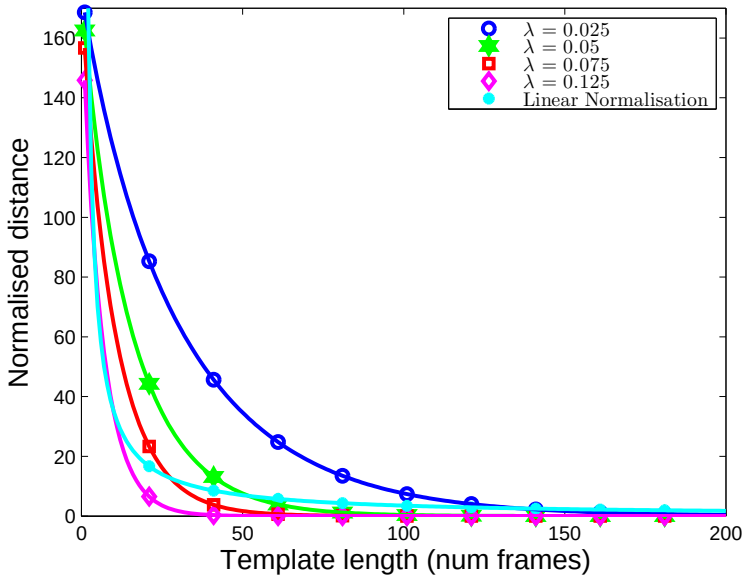


Figure 5.7: *Sigmoid-based normalised distance function for (in this case) a distance (unnormalised) of 350. The effect of varying λ upon the normalised distance (Equation (5.4)) is shown over different template lengths. The linear function is also shown (the distance is normalised by dividing by the number of frames in the template).*

5.4 Filtering Candidate Templates with Hierarchical LDA

The Time Filter algorithm produces a list of approximately matching template candidates. A side-effect of this is that, along with correctly matching templates, there are also incorrect templates: A "correct" template is defined as having a matching word string to the annotation that defines the current utterance and a start time that must be within a given number of frames from the start time in the annotation. Conversely, an "incorrect" template either has a correctly matching word string, but is outside of the frame window, or does not have a matching word string. Figure 5.8 shows a frame window of 5 frames centered around frame t.

Clearly, if the number of incorrect templates selected as candidates for the decoder was reduced, then the chance of increasing recognition accuracy is in-

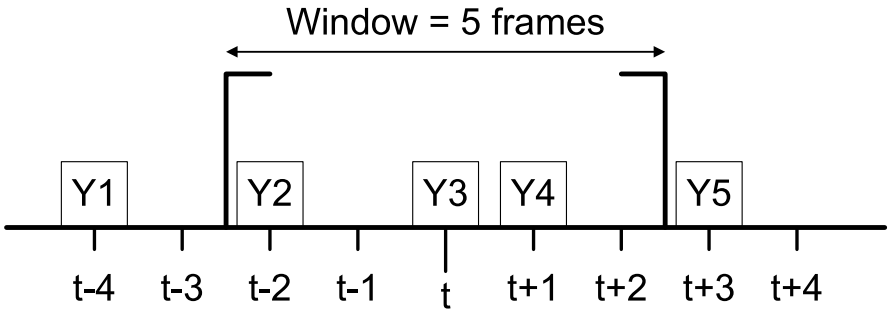


Figure 5.8: *Template classification. If the template is within the window (centered at frame t) and it's word label matches the annotation, then it is classed as correct. Templates $Y1$ and $Y5$ are classed as incorrect whether or not their word strings match the annotation.*

creased. This can be quantified by performing an *Oracle* test which removes all incorrect templates from the template candidates: For the Oracle test, templates were classed as incorrect only if their word strings did not match the transcription / annotation of the utterance, that is, no time information was used to eliminate templates. By removing incorrect templates, an increase in recognition accuracy of around 12% absolute is seen for the data used in this work[3].

To reduce the number of incorrect templates selected as candidates, some kind of classification method is required. Given a template candidate for a given utterance, the classifier needs to classify the template as correct or incorrect, with the incorrect templates being discarded. This problem is a binary problem, also known as a *two-class* problem (a template is either correct or incorrect) and is well-suited to the use of Linear Discriminant Analysis (LDA)[4] [Webb, 2002].

5.4.1 Extracting Features for LDA

To be able to use LDA on the selected template candidates, the first step is to extract a set of features from the selections. This requires some insight into what features may be useful for this task. An observation made on the template candidates from the training data is that when a template candidate is "correct",

[3]Chapter 6 will give more detail about the results and recognition experiments.
[4]Section 2.7 gives a more thorough technical background to LDA.

there are often many similar correct templates representing the same word string with the same start frame, or with a start frame that is within a few frames of that given template. This is less likely to occur for incorrect templates. To exploit this property, the first feature that can be extracted is simply the number of occurrences of a template, which is expressed as a probability.

A simple relative frequency probability could be defined for a template candidate by counting the total number of templates that represent the same word string and dividing by the total number of template candidates for the current utterance:

$$\Pr(\mathbf{Y}) = \frac{C(W_1^M)}{C(\forall_{W \in V})} \tag{5.5}$$

where, $\mathbf{Y}$ is the template candidate, W_1^M is the word string that $\mathbf{Y}$ represents, $\forall_{W \in V}$ represents all words W in the vocabulary V, and $C(\cdot)$ is the number of occurrences of template candidates that represent the given parameter. If we match word strings exactly, then the count for a template candidate can be underestimated because there may be many more template candidates that have the same first word but different subsequent words (if phrases). A first word match is important because a candidate's start time is determined by part of the first word, so lots of template candidates appearing with the same first word and similar start times gives good confidence of correctness, and hence should result in a higher probability for a given template candidate. Equation (5.5) is easily adapted to use only the first word for counts:

$$\Pr(\mathbf{Y}) = \frac{C(W_1)}{C(\forall_{W \in V})} \tag{5.6}$$

where $C(W_1)$ gives the number of template candidates that start with word W_1. As

mentioned before, observations of the training data showed that correct template candidates usually appear in groups of templates with matching or similar word strings within a few frames of each other. To extract this information, a window, which is $\pm\chi$ frames around the start frame of the current template candidate $\mathbf{Y}$ (and includes the start frame), can be incorporated into Equation (5.6), which leads to:

$$\Pr(\mathbf{Y}|t) = \frac{C(W_1, t, \chi)}{C(\forall_{W \in V}, t, \chi)} \tag{5.7}$$

where $C(W_1, t, \chi)$ gives the number of occurrences for all candidates with a first word W_1 and a start frame that is within $\pm\chi$ frames from the start frame t of $\mathbf{Y}$, and $C(\forall_{W \in V}, t, \chi)$ is the number of occurrences of all candidates within $\pm\chi$ frames of $\mathbf{Y}$ (including $\mathbf{Y}$).

A further improvement that can be made to Equation (5.7) is to smooth the counts within the window defined by χ. Using a Gaussian Radial Basis function (RBF) [Buhmann, 2003], the counts of template candidates can be reduced smoothly the further they are from the centre of the current window (see Figure 5.9). Smoothing with an RBF is useful in the situation where a template candidate is surrounded only by a small number of matching templates within a few frames of the central frame of the window, but with a larger number of matching templates at the outer regions of the window. In this situation, the current template candidate may be given a high probability (using Equation (5.7)) as all positions within the window are taken to be equal. By smoothing the counts, the further they are from the centre of the window, the lower is the probability of the template candidate (at the centre of the window). Conversely, a template candidate with the same number of matching candidates within the window will receive a higher probability if those matching templates are closer to the centre of the window. Equation (5.8) is the smoothed probability of a template candidate $\mathbf{Y}$ given the current time t:

$$\Pr(\mathbf{Y}|t) = \frac{C_{rbf}(W_1, t, \chi)}{C_{rbf}(\forall_{W \in V}, t, \chi)} \tag{5.8}$$

where, t, $\mathbf{Y}$, W_1, $\forall_{W \in V}$, and χ are defined as before, and $C_{rbf}(\cdot)$ is the weighted count function given by:

$$C_{rbf}(Z, t, \chi) = \sum_{i=-\chi}^{+\chi} C(Z, t+i)\varphi(i, \mu, \sigma) \tag{5.9}$$

where $C(Z, t+i)$ is the count of templates with a word string beginning with word Z and a start frame of $t+i$, and $\varphi(i, \mu, \sigma)$ gives the likelihood weight for a sample i from an un-normalised Gaussian distribution with a mean μ and standard deviation σ defined by:

$$\varphi(i, \mu, \sigma) = e^{-\frac{1}{2\sigma^2}(i-\mu)^2} \tag{5.10}$$

To clarify, at each frame t, the RBF function is centered at frame t, and the counts of templates that are within $\pm\chi$ frames of t are weighted by the likelihood from the RBF: templates that have a start time outside of the given window are given zero counts. In all cases, μ is set to zero as templates with the same start frame as the centre of the window have a distance of zero from the centre, and σ is pre-defined.

By using an un-normalised Gaussian RBF, the counts of templates that occur at frame t (the centre of the window) will remain the same because the RBF returns a likelihood of 1.0. Figure 5.9 shows an example of the RBF applied to template counts, where $\chi = 3$ and $\sigma = 1$. The original counts at each frame are shown in dark blue, with the resulting weighted counts shown in yellow. Again

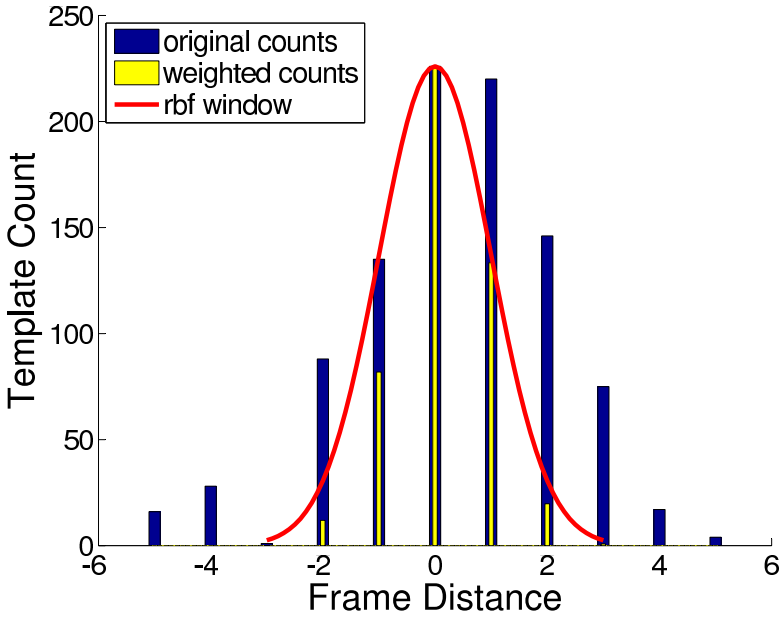


Figure 5.9: *Gaussian Radial Basis Function: A un-normalised Gaussian is used to weight counts of templates by distance from the centre frame. Here, the window is 7 frames, with ±3 frames from the centre frame, and the standard deviation σ of the Gaussian is* 1.

note that the counts at the centre of the window remain the same and counts outside of the window become zero.

Another property of the template selections on the training data is that when a template is correctly selected for a given time, then its score for that activation time is generally lower than an incorrect template as there is a closer match between the input and the template. To extract this information as a feature, the *z-score* for each template activation is calculated:

$$z(\mathbf{Y}, t) = \frac{D(\mathbf{Y}, t) - \mu_D}{\sigma_D} \tag{5.11}$$

where $D(\mathbf{Y}, t)$ is the score of template $\mathbf{Y}$ for a given activation starting at time t, and μ_D and σ_D are the mean and standard deviation, respectively, over all template scores within the selections for a given training utterance. The z-score of template activation scores gives the number of standard deviations from the

mean, thus providing a reasonable measure of how close a given template is to a sequence of the input *relative* to all other template selections. It should be noted that the distance measure $D(\mathbf{Y}, t)$ in Equation (5.11) can be replaced with the normalised distance measures of Equations (5.2) and (5.4).

A final observation of the template selections for the training data shows that even with normalisation methods for template scores (Section 5.3.2), there are examples of "incorrect" template selections with lower scores occurring when the template length is shorter. A shorter template candidate will almost always have a smaller score as there are less penalties to be added to the score, although normalisation should reduce the effect somewhat. To extract this information, a third feature, the template length $L(\mathbf{Y})$, is found for each template candidate.

Combining the three features discussed above provides the following feature vector $\mathbf{F}(\mathbf{Y}, t)$ which can be extracted for each template candidate selection $\mathbf{Y}$ at time t:

$$\mathbf{F}(\mathbf{Y}, t) = \begin{pmatrix} \Pr(\mathbf{Y}|t) \\ z(\mathbf{Y}, t) \\ L(\mathbf{Y}) \end{pmatrix} \tag{5.12}$$

5.4.2 LDA Decision Tree

Once the features for the template selections on the *training data* are extracted, LDA can be used to find the projection that best separates the two classes of data. Each template activation is labelled as correct or incorrect based on its word string and suggested start time as discussed at the beginning of Section 5.4 and shown in Figure 5.8. Using Fisher's Linear Discriminant (Section 2.7), each template selection (represented by the features described in Section 5.4.1) is projected onto a line (or into 1D space) which best maximises the ratio of between-class scatter to within-class scatter using the eigenvector $\mathbf{w}_a$.

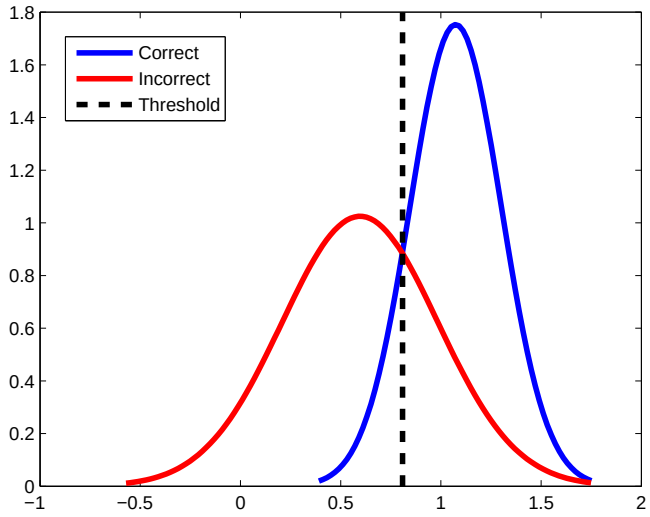


Figure 5.10: *Threshold selection: Distribution crossing point. Threshold chosen to be the point at which the correct (blue, right) and incorrect (red, left) distributions cross.*

After projecting the data, the next step is to find a threshold ϕ_a which will be used to classify samples as correct or incorrect depending which side of the threshold the samples lie on the line. A threshold can be placed anywhere along the projection line to control the acceptance / rejection of candidate templates: We choose to use the optimum decision point where the distributions of the correct and incorrect samples cross (see Figure 5.10 for an example). By assuming a Gaussian distribution, the threshold is determined by solving the following *quadratic* for x:

$$\left(-\frac{1}{2\sigma_1^2}+\frac{1}{2\sigma_2^2}\right)x^2+\left(\frac{\mu_1}{\sigma_1^2}-\frac{\mu_2}{\sigma_2^2}\right)x+\left(K-\frac{\mu_1^2}{2\sigma_1^2}+\frac{\mu_2^2}{2\sigma_2^2}\right)=0 \qquad (5.13)$$

where $K=\ln\left(\frac{1}{\sqrt{2\pi}\sigma_1}\right)-\ln\left(\frac{1}{\sqrt{2\pi}\sigma_2}\right)$, μ_1 and σ_1 are the mean and standard deviation for the incorrect distribution, and μ_2 and σ_2 are the mean and standard deviation for the correct distribution. As there are two roots when solving a quadratic, the root chosen to be the threshold will lie between μ_1 and μ_2: the means of both distributions. The derivation of Equation (5.13) can be found in Appendix D.

Figure 5.11 shows the histograms of the projected data used to generate Figure 5.10 — this shows that the assumption of a Gaussian distribution for the data is valid.

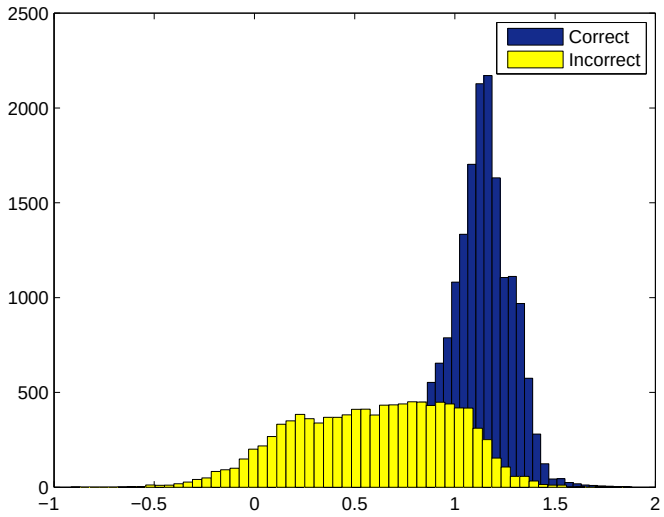


Figure 5.11: *Histograms of LDA projected data. Shows the correct (blue) and incorrect (yellow) samples after projection into 1D. The means and standard deviations of the sample sets were used to generate the Gaussian distributions in Figure 5.10.*

To apply the LDA classifier to an *evaluation set*, each selected template activation is represented by its extracted feature vector (Equation (5.12)) that is multiplied by the eigenvector $\mathbf{w}_a$ which projects the *training* template activations to the space that gives maximum separation between correct and incorrect classes. After the projection, each projected sample from the evaluation set is compared to the threshold ϕ_a, with samples above ϕ_a classified as correct, and samples below ϕ_a classified as incorrect.

After the projection of data to one-dimensional space there will often be an overlap of the class samples and the choice of threshold attempts to find the point where this overlap is minimised (Figure 5.10). Choosing the threshold to be the point where the two class distributions cross means that correct samples will be misclassified as incorrect, and likewise, incorrect samples misclassified as correct. By performing LDA once more on the samples that lie below threshold ϕ_a it may be possible to classify more samples accurately.

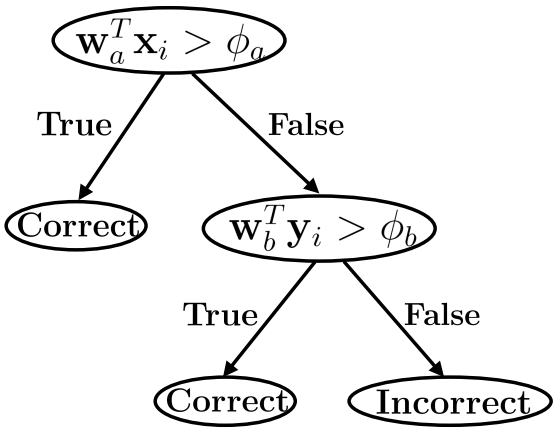


Figure 5.12: *The LDA decision tree. The LDA is first applied to all training samples,* $\mathbf{x}$*, then to all samples* $\mathbf{y}$*, where* $\{\mathbf{y} \subseteq \mathbf{x} : \mathbf{x}_i < \phi_a\}$*.* $\mathbf{w}_a$ *and* $\mathbf{w}_b$ *are the eigenvectors, with* ϕ_a *and* ϕ_b *the thresholds, for each LDA.*

Figure 5.12 shows a 2-level decision tree framework that can be used to implement the applications of multiple LDA to the data: The node at the top of the tree describes the first stage of the LDA mentioned before, which finds the best projection (i.e. eigenvector $\mathbf{w}_a$) for a template activation feature point $\{\mathbf{x}_i \in \mathbf{x}\}$, where $\mathbf{x}$ is all the template samples, and thresholds at the point at which the distributions of the correct and incorrect samples for the training data cross. If the sample values in the projected space are greater than threshold ϕ_a then they are classified as correct (left node on the middle level in Figure 5.12), otherwise they are initially pooled together in the incorrect class of samples $\mathbf{y}$, where $\{\mathbf{y} \subseteq \mathbf{x} : \mathbf{x}_i < \phi_a\}$.

The LDA is then applied to all samples in y (right node on middle level in Figure 5.12), giving the new projection $\mathbf{w}_b$, and the value of sample $\mathbf{y}_i$ after projection is compared to a second threshold ϕ_b which is calculated using the correct and incorrect distributions of the training data samples that lie below ϕ_a, i.e. all samples in $\mathbf{y}$. Samples projected above ϕ_b are now re-classified as correct, leaving the remaining samples to be classified as incorrect.

5.5 Evaluation

Ultimately the success of the example-based system is evaluated on word recognition accuracy, or word error rate (WER), but it is also useful to determine how well the Time Filter (Section 5.3), and LDA decision tree (Section 5.4) perform. Section 5.5.1 will give an analysis of the Time Filter output, looking at details such as how well the selected templates fit to the utterance transcription, and the average number of templates the Time Filter finds. Section 5.5.2 will analyse the effectiveness of applying LDA as a filter to the selected template candidates, focusing on the classification accuracy and the suitability of the selected features.

5.5.1 Template Selection

An obvious measurement for evaluating the output of the Time Filter, independent to the decoder (Chapter 6), is to see how well the template selections fit the annotation of the utterance: This includes both word string and time information, and is termed *template coverage*. For an utterance to have *full-coverage* by the template selections, all of the words within the annotation for that utterance must be represented within the given time window.

Determining whether a word template matches the utterance is trivial as it is a one-to-one mapping within the given time window ($\pm\chi$). However, phrases are matched to the utterance as whole tokens, i.e. all words within the phrase must be contained within the utterance, in the same sequence. Clearly the coverage estimate is not as accurate as it could be because of the way the phrase templates are handled. When calculating WER on a hypothesis from the decoder, the phrases are expanded into their word form, and words in the recognition output are aligned to the transcription of the utterance. This means that phrase templates that contain only *some* of the words within the utterance (e.g. "I'd like my account balance" matching to the utterance "I'd like my balance") could still lead to a decrease in WER (although insertions from incorrect words in the phrases

can also push the WER back up).

Another interesting evaluation measure is the ratio of correct template selections to incorrect template selections. This provides extra information about how successfully the Time Filter is selecting candidates and can also augment the recognition results later, giving insight into how the ratio of correct to incorrect selections affects the decoder, if it does at all.

Templates	Time Filter Method	Coverage (%)	C/I Ratio	Average Num. Templates
Words	Backwards	94.90	1.28	1986
	Forwards	94.87	1.86	1976
	Merged	94.96	1.81	2047
Words + Phrases	Backwards	96.6	1.28	2100
	Forwards	96.67	1.84	2092
	Merged	96.69	1.79	2167

Table 5.1: *Time Filter statistics for SD Test data. The Time Filter methods are the forwards and backwards passes of the Time Filter algorithm, while "merged" represents the the union of the two passes. C/I Ratio is the correct to incorrect template ratio.*

Table 5.1 shows some of the statistics, previously described, for the Time Filter algorithm on the SD test data[5]. It shows that when phrase *and* word templates are used the coverage rises by about 1.8% over the word-only templates, with an increase in the average number of templates selected per utterance. The performance of the backward and forward pass is given, as well as the merged results which is union of the two sets of template selections. It can be seen for word-only and word + phrase templates that the correct to incorrect ratio (C/I Ratio) is lower for the backwards Time Filter than the forward pass, although on average, the backward pass selects more templates.

For the word-only templates, the coverage is marginally better for the backwards pass, which implies, with the knowledge of the C/I Ratio and the average number of templates, that the backward pass finds *fewer* correct templates than the forward pass, and thus a *greater* number of incorrect templates. For the word + phrase templates, the backward pass coverage is slightly lower than that of the

[5]All experiments reported use the local Mahalanobis distance (Equation (2.31)) for calculating between-frame distances while the distance normalisation of Equation (5.2) is applied to a templates total distance.

forward pass, although the C/I Ratio and average number of templates follow a similar pattern to that of the word-only templates which again implies that fewer correct templates are found in the backward pass. When combining the template selections of the forward and backward passes together (the merged selections), in both the word-only and word + phrase templates selections, the coverage increases, although the C/I Ratio drops slightly which is due to the backward pass bringing a larger number of incorrect templates.

Templates	Time Filter Method	Coverage (%)	C/I Ratio	Average Num. Templates
Words	Backwards	93.17	0.33	1665
	Forwards	93.36	0.48	1656
	Merged	93.45	0.47	1726
Words + Phrases	Backwards	95.25	0.33	1775
	Forwards	95.47	0.48	1767
	Merged	95.51	0.48	1842

Table 5.2: *Time Filter statistics for the RM evaluation set, with VTLN applied. The Time Filter methods are the forwards and backwards passes of the Time Filter algorithm, while "merged" represents the the union of the two passes. C/I Ratio is the correct to incorrect template ratio.*

Table 5.2 shows the template selection statistics for the RM evaluation set (the combined oct89, feb91, and sep92) after Vocal Tract Length Normalisation (VTLN) is applied (see Section 2.6 for more information). Like with the SD data (Table 5.1), the word + phrase template selections increase the coverage percentage, this time by over 2%. However, the C/I Ratio for all selection methods is much lower on the RM data. This must be due to the inherent problem of speaker independence for template-based approaches, even after VTLN, which requires lots of examples to match the variation in speaker, especially if, like in this case, the smallest template unit used is the word. It is interesting to point out that for the word + phrase merged template selections, without VTLN (not shown in the table) the coverage drops to 95.29% while the C/I Ratio drops to 0.33.

5.5.1.1 Sigmoid-Based Distance Normalisation

In Section 5.3.2, a sigmoid-based distance normalisation function for template activations scores was introduced (defined in Equation (5.4)). In template selection experiments, the λ weight was varied from 0.025 in roughly even steps to 0.8 for the SD data to evaluate the effects of how the distance curve affects the template selection. Table 5.3 shows the results.

λ	**0.025**	**0.075**	**0.125**	**0.2**	**0.4**	**0.8**
Coverage (%)	95.76	95.86	95.88	95.88	95.9	95.9
C/I Ratio	1.79	1.84	1.85	1.86	1.88	1.89
Avg. Num. Templates	1981	1979	1979	1977	1977	1983

Table 5.3: *Time Filter statistics for the SD evaluation set when using Sigmoid-based distance normalistion.*

It is clear from Table 5.3 that, although there are slight improvements in the coverage percentage and C/I Ratio, the sigmoid-based normalisation is not very effective in terms of varying the template selection. The average phrase length in each utterance for the template selections also shows very little change. Clearly the Sigmoid distance normalisation did not work as hoped. It seems that although the normalisation function is a useful formulation, it matters where it is applied in the Time Filter algorithm.

Currently the normalisation is applied to the template activation score after the Time Filter completes, but before thresholding is applied at each frame. Actually, the normalisation should be applied at the selection of the k-nearest neighbours (KNN). By applying the normalisation after the nearest neighbours have been selected has a minimal effect because the available template selections have already been defined, and thus the length of the chosen templates can not be greatly influenced.

The problem that applying the Sigmoid normalisation to the KNN selection might introduce is that, when using word and phrases templates, a nearest neighbour (reference frame) that is contained in a phrase template will also be contained in a word template (both templates are defined over the same frame in a given

utterance). By reducing the distance of the nearest neighbour with the sigmoid normalisation, it helps that frame be selected as one of the KNN if it belongs to a longer template, but that also means that the equivalent word template will be initialised with an activation region from the nearest neighbour hit. Due to time constraints, this problem could not be addressed, and thus all experiments reported use the distance normalisation that divides by the number of nearest neighbour hits, previously defined in Equation (5.2).

5.5.2 LDA Filtering

Figures 5.13 – 5.17 show various plots for the features extracted from template selections for the SD training data. Figure 5.13 shows the features of Equation (5.12) in 3D space, with the correct samples in *blue* and incorrect samples in *red*. Because this is the training data, the class of each template feature vector is determined using the annotation of each utterance as explained at the beginning of Section 5.4, with both word string and time information used to classify: in the given figures, a frame window of 15 frames, i.e. $\chi = \pm 7$, was used to classify the samples as correct or incorrect.

Figure 5.14 shows just two of the features, $\Pr(\mathbf{Y}|t)$ and $z(\mathbf{Y}, t)$, which effectively gives an overhead view of Figure 5.13. Visually, it is clear from the figure that the occurrence based probability, $\Pr(\mathbf{Y}|t)$, provides more discrimination for the two classes (incorrect and correct) than the z-score, $z(\mathbf{Y}, t)$, and indeed the eigenvector, $\mathbf{w}_a = [0.99\ -0.14\ 0.01]$, confirms this as the corresponding dimension to the occurrence probability is 0.99, which means that the transformed samples will keep a *similar* spread to the occurrence dimension. Figure 5.15 shows the 1st level LDA projection of the template selections. It is clear that although there is large overlap between the two classes, there is separation at either end of the line (which is also visible in Figures 5.13 and 5.14).

Figures 5.13 – 5.17 represent a total of 20.8 million template selections (over all utterances in the training data), with 12 million labelled as correct, and 8.8 million

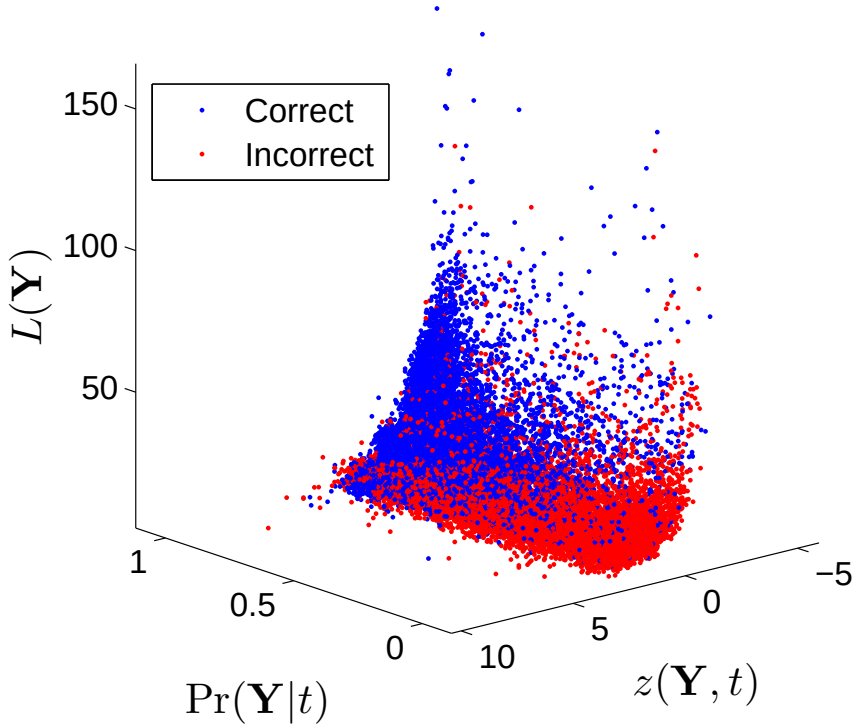


Figure 5.13: *Features for LDA: The features (defined by Equation* (5.12)*)for correct (blue) and incorrect (red) template candidates in 3D space. The data is the SD training data.*

labelled incorrect. After the first application of LDA, there are approximately 10.6 million correct and 2.1 million incorrect template selections that lie above ϕ_a. Applying the 2nd level LDA to the template selections that lie below ϕ_a results in 1.2 million correct and 2 million incorrect samples projected above threshold ϕ_b which means that about 11.7 million correct templates selections (out of 12 million) are classified by the LDA decision tree as correct, whereas 4 million incorrect template selections are classified correct (above the thresholds ϕ_a and ϕ_b), which is just under half of all incorrect template selections. This means that the ratio of correct to incorrect template candidates rises from approximately 1.36 before the LDA to 2.93 after the 2-level LDA is applied.

Figure 5.16 shows the distributions of the two classes within the first LDA projection space (for all template selections), while Figure 5.17 shows the distributions of the two classes within the second projection space (for all template

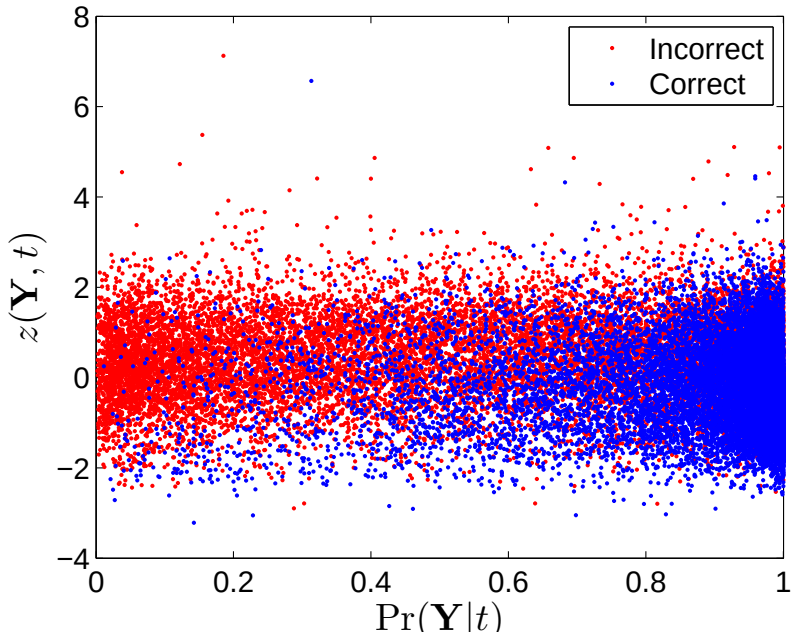


Figure 5.14: *Z-score ($z(\mathbf{Y}, t)$) plotted against the probability of template candidates ($P(\mathbf{Y}|t)$) for SD training data.*

selections that were projected below ϕ_a in the first LDA). The distributions in both figures confirm that there is an overlap of the classes, although the correct template selections have a lower variance from the mean compared to the incorrect template selections. The narrower variance (of the correct samples), in agreement with the plots in Figures 5.13 and 5.14, means that the features of Equation (5.12) provide a good extraction for *correct* selections, but do not necessarily represent the incorrect selections well: clearly a large number of incorrect samples lie at the opposite end of the sample space to the correct samples, but there are also a large number of incorrect samples that exhibit the same, or similar properties to the correct samples, hence the overlap of the two classes.

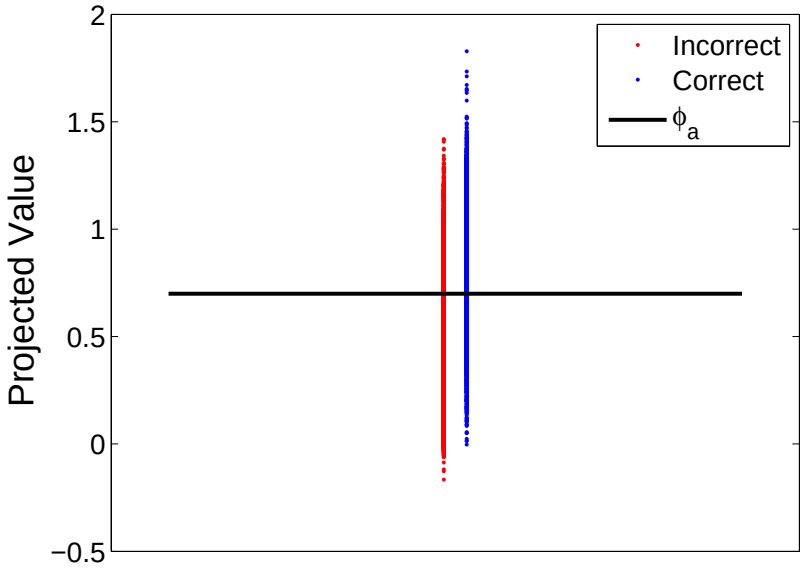


Figure 5.15: *Projection of template candidates into 1D for SD training data. Projection of the incorrect (red, left) and correct (blue, right) shown separately. ϕ_a is the threshold for classification.*

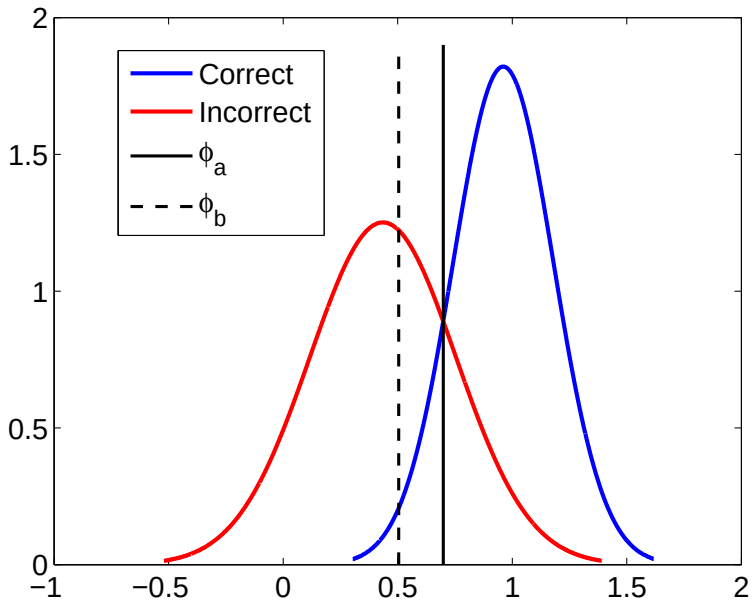


Figure 5.16: *Distributions for correct and incorrect template candidates, showing 1st level threshold ϕ_a with the 2nd level threshold, and ϕ_b which is calculated from the distributions shown in Figure 5.17.*

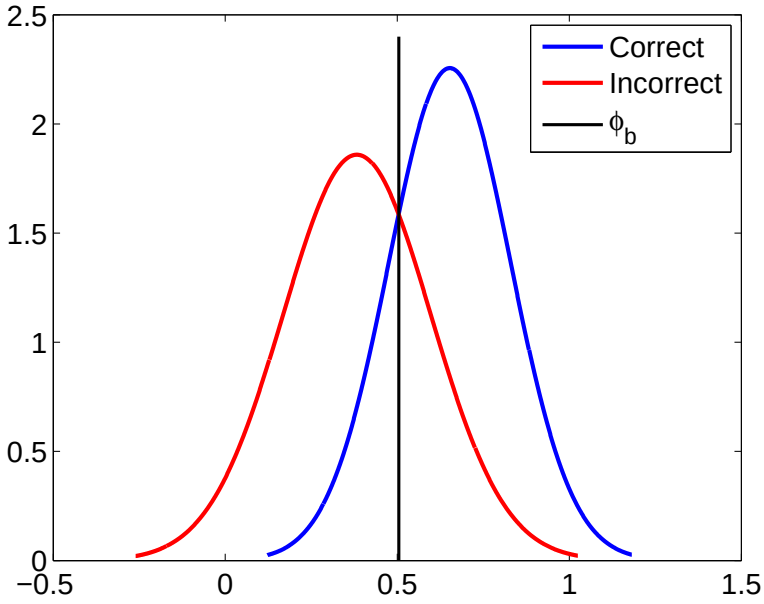


Figure 5.17: *Distributions of template candidates* $\mathbf{y}$ *where* $\{\mathbf{y} \subseteq \mathbf{x} : \mathbf{x}_i < \phi_a\}$ *with threshold* ϕ_b *shown.*

5.5.2.1 Classification Experiments

Tables 5.4 and 5.5 shows the results of applying the 2-level Hierarchical LDA classifier to the SD Test data and RM evaluation set respectively. For each dataset, a window of 15 frames is used to label a *training* template as correct or incorrect. For the evaluation data, an RBF window of 15 frames ($\chi = \pm 7$) is used, with a standard deviation $\sigma = 0.25$ for the SD data and $\sigma = 1$ for the RM data[6].

Templates	**Words**	**Words + Phrases**
Classification Acc. (%)	76.30	76.33
FNR (%)	1.91	1.88
FPR (%)	63.23	62.74
Average Num. Templates	1713	1804
Average Reduction (%)	16	17
C/I Ratio	2.81	2.80
Coverage (%)	89.70	93.77

Table 5.4: *LDA Filter classification results on the SD Test data. Shows the classification accuracy of the Hierarchical LDA classifier, False Negative Rate (FNR), False Positive Rate (FPR), average number of template selections per utterance, the average reduction in the number of templates compared to before filtering, the correct to incorrect template ratio (C/I Ratio) and the coverage of the template selections to the input utterances.*

Each table gives the evaluation of the classifier on word-only template selections *and* word + phrase template selections (Row one). Row two gives the classification accuracy of the classifier which is defined as the number of correctly classified templates divided by the total number of templates. Row three gives the False Negative Rate (FNR) which is a measure of how many times a template that should be classified as *correct* is misclassified as *incorrect*. Similarly, row four gives the False Positive Rate (FPR) which measures how many times an *incorrect* template is misclassified as *correct*. Rows five, seven, and eight were previously defined in Section 5.5.1, with row six defining the reduction in the average number of templates per utterance compared to before filtering (refer to Tables 5.1 and 5.2 for original values).

[6]For the SD data, these values were experimentally found on the test data, while for the RM data, the parameters were optimised on the feb89 development data

Templates	Words	Words + Phrases
Classification Acc. (%)	56.29	56.18
FNR (%)	3.22	3.30
FPR (%)	62.76	63.01
Average Num. Templates	1247	1317
Average Reduction (%)	28	29
C/I Ratio	0.72	0.72
Coverage (%)	85.93	91.68

Table 5.5: *LDA Filter classification results on the RM evaluation set using VTLN.*

It can be seen from Table 5.4 that the classifier achieves a high accuracy on the SD data (76%), with a low FNR but high FPR. As mentioned before, the features were selected to represent *correct* template selections, thus the low FNR, but the features don't directly model the *incorrect* template selections, hence the high FPR. This results in *most* of the correct templates being accepted, but with a large number of incorrect templates also being accepted. The average number of templates, C/I Ratio, and template coverage can all be directly compared to Table 5.1 to see the effects of filtering with the LDA classifer.

The coverage when using word-only templates and word + phrase templates both drop after filtering. It is clear that the correct templates that are misclassified, and hence filtered out of the template selections, directly affect the coverage percentage. However, even though the FNR on the word + phrase template selections is only marginally lower than that on the word-only templates, the difference in coverage percentage between word-only and word + phrase templates is more than double than before filtering. This same effect is seen on the RM data, in Table 5.5, where the difference on coverage between word-only and word + phrase templates compared to before filtering in Table 5.2 is almost trebled, even though the FNR is higher on the word + phrase templates than the word-only templates.

Even though the classifiers perform similarly on word-only and word + phrase templates in terms of classification accuracy, the difference in template coverage between the two sets of templates implies that template length has a significant effect upon the classifier. Figure 5.13 showed that there is a significant number of

correct templates in the training data that are longer than 40 frames, and although there are incorrect templates of this length, they are not as numerous, and it can be seen that the incorrect template lengths generally are below 40 frames. This means that, when word + phrase template selections are used in the Test sets, that the longer templates (i.e. phrases), if they have a high occurrence probability and lower z-score, are more likely to be classified as correct than shorter templates (i.e. words) with similar z-scores and occurrence probabilities.

5.6 Conclusions

This chapter introduced an approximate method of KNN selection using Vector Quantisation (VQ) (Section 5.2) for input into the Time Filter algorithm which was described in Section 5.3. The Time Filter was described in detail, giving information on how local activation regions are formed and how local costs are applied within each activation region. An additional *backward* pass of the Time Filter was suggested and described in Section 5.3.1. An alternative method to normalising the distance score of template activations was described in Section 5.3.2, using a Sigmoid-based method, where the normalisation strength was based upon the template length (in terms of number of frames) and a pre-defined weight so that the effect of using longer templates could be seen.

In Section 5.4, a hierarchical LDA filter was introduced which was used to further filter the list of template selections output from the Time Filter. To train the LDA classifier, three features were extracted from the template selections for the *training data*[7]. Based on observations, the first feature was an occurrence-based probability, which counted the number of template selections within a given window that all contained the same first word and divided that count by the total number of templates within that window. The window was applied over each template selection (which was at the centre of the window), and the counts were

[7]The Time Filter is run on the training utterances to get the template candidates for the training data.

then smoothed using a Gaussian RBF (Radial Basis Function) that was placed at the centre of the window. The RBF allowed a more accurate probability to be estimated by assigning lower probabilities to templates that were further away from high occurrence regions. The second feature was the z-score of the distances of each template selection, calculated over all template selections, observing the fact that, generally, correct templates have smaller distance scores. Finally the third feature was the length of the template selection, in terms of the number of frames it contains, observing the fact that the Time Filter introduced short incorrect templates.

The extracted features were then used to train the hierarchical LDA filter which was formulated as a two-level decision tree, with a projection vector and threshold at each level — thresholds, used to determine which class the template belongs to, were calculated as the intersection point of the distributions for the correct and incorrect classes of the projected training samples. For classification, the test data was first classified as correct or incorrect based on the top-level LDA classifier, with the template samples that were classed as incorrect then undergoing a second classification at the second level of the decision tree.

Section 5.5 provided evaluation of the methods described in the chapter. It was shown that the word + phrase templates result in a better coverage of the test utterances, with a 1.8% increase in word coverage on the SD test data, and over 2% for the RM evaluation set. The performance of the backward pass of the Time Filter was compared to the original forward pass of the algorithm and the merged set of template selections from both passes. It was shown that the backward pass typically resulted in a marginally worse template coverage of the input, although the C/I Ratio (Correct to Incorrect Ratio) was significantly lower for the backward pass, which also selects, on average, more templates per utterance. When merging the two sets of template selections (i.e. the union of the two sets), the template coverage was always better than the individual sets, although the C/I Ratio dropped slightly below that of the forward pass template selections. This

implies that the backward pass, although it has very similar template coverage, is actually finding less correct templates, and thus more incorrect templates than the forward pass, hence the slight drop in C/I Ratio on the merged set. However, because the template coverage improves after merging, the backward pass is clearly finding *unique* correct templates to the forward pass.

It was shown in Section 5.5.1.1 that the Sigmoid-based distance normalisation, which was introduced to control the length of the selected templates by the Time Filter, was not effective. It was shown that by applying the normalisation at the end of the Time Filter algorithm, but before thresholding is applied at each frame, has very little effect on the final set of template selections because it is the KNN selection that determines the available templates for selection, as well as the defined constraints and sizes of the activation regions created by each nearest neighbour hit. It was suggested that applying the normalisation method at the KNN selection stage would allow control over the template lengths, but because each phrase template shares its frames with word templates, there would be an added problem of normalising each word template with the same function as the phrase templates. Due to time constraints, the improvements to the Sigmoid distance normalisation were left to a future date.

Section 5.5.2 gave evaluation of the hierarchical LDA filter through classification experiments. It was shown that on both the SD call-routing and RM evaluation data, that the application of the LDA filter led to a decrease in template coverage, with a reduction in the average number of templates per utterance of 17% and 29% for the word + phrase templates on the SD and RM data respectively. The classifiers, on both datasets, gave a low False Negative Rate (FNR) and high False Positive Rate (FPR), although it is the FNR which directly affects the template coverage. This means that the majority of filtered templates are incorrect templates, but it is not yet clear how the loss of correct templates will effect the recognition word accuracy — this will be addressed in Chapter 6.

It was shown that when comparing the difference in template coverage on the

word-only templates to the word + phrase templates, before and after the LDA filtering, that the difference became greater after the filtering; in the case of the SD data it doubled, and for the RM data it was almost treble the pre-filtered difference. This was attributed to the template length feature which is extracted for the LDA. Because the classification performance was almost the same on the word-only templates *and* the word + phrase templates, the conclusion must be that the longer templates are not lost in the filtering process, but in fact it is the shorter correct templates that are filtered away — this is a side effect of the Time Filter introducing many incorrect templates that are short in length, thus training the LDA to prefer longer templates. Thus, one can conclude, that a longer template with a similar z-score and occurrence probability to a shorter template is less likely to be filtered by the classifier than the shorter template.

Chapter 6

Template-Recognition Experiments

6.1 Introduction

Section 2.3 introduced and described the techniques used in a basic Template-Recogniser where all reference templates are loaded into the decoding network, which as Section 2.5 showed, can be defined by the language model. Loading all reference templates into the decoder is impossible because of memory limitations, thus Chapter 5 described methods that can be applied (in a bottom-up manner) to reduce the number of templates loaded into the decoder for any one utterance.

This Chapter is structured as follows: Section 6.2 describes how the information from the bottom-up processing can be integrated into the template-decoder and will go into finer detail about sections of the decoder that were not covered in Section 2.3. Section 6.3 describes the method used for speaker normalisation of the templates using a VTLN approach. Section 6.4 provides a detailed results section for template-based recognition using phrase-based units, and shows how the results compare with the baseline results that were evaluated with a HMM-based monophone recogniser (Section 4.6). Finally, Section 6.5 gives conclusions

to the chapter.

6.2 Decoder Architecture — Extensions to the Template Decoder

6.2.1 Integrating Template Candidates

As described in Section 5.3, the Time Filter algorithm [De Wachter et al., 2003] outputs an *activation graph* which contains a list of candidate templates with a given start and end time for the given utterance. De Wachter [2007] uses the activation graph directly in the decoder, i.e. a token transitioning from the end of one template can only move to other templates that begin at the time (frame) that the current template ends in the activation graph. This approach, although efficient, can lead to "dead-ends" [De Wachter, 2007] in the graph, where there is no template to move to, i.e. there are no outgoing arcs at a given frame state. Figure 6.1 shows an example of a "dead-end" occurring in an activation graph.

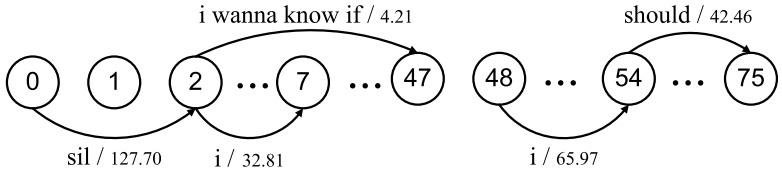


Figure 6.1: *Shows an example of an activation graph with a "dead-end" at frame 47.*

De Wachter et al. [2007] introduce "natural successors", which are the templates that follow a given template in the reference database (i.e. they are adjacent in the same reference utterance), into the activation graph to minimise the number of "dead-ends", although they can still occasionally occur. The solution that is applied in the experiments reported here is to simply use the start frame of the template *only*. By not specifying the end frame of a template candidate as a constraint, the *token passing* algorithm (with DTW) can determine the segmentation point. The token passing algorithm is implemented as described in Section 2.3.4

except for when passing a token *into* a template. At this point a "lookup" is performed to check that the given template can start at the current time frame — if the template's candidate start time matches the current time then the token is passed into the template, and the DTW can begin, otherwise no token is passed to the current template.

It should be pointed out that this does not fully remove the problem of "dead-ends": if the Itakura constraints are used, then the token passing algorithm will allow at most one stall in a given state, and thus removing the end constraint does not guarantee the dismissal of all "dead-ends". To fully remove the problem of "dead-ends", the Itakura constraints are eased on the stall transitions, i.e. consecutive stalls are accepted. The effects of this change upon the DTW was previously illustrated in Figure 2.4.

6.2.2 Token Merging

The process of selecting the best token at each state within the token passing algorithm is known as *token merging*[1]. It equates to choosing the best decoding path to the current state at the current time and is one of the key processes of the Viterbi algorithm. For the template decoder, at a certain state the collection of tokens at a given time will contain a high number of identical paths. This is because, even with a constrained network (using the activation graph), there are multiple templates representing the same word or phrase, and so the tokens arriving at the current state are likely to have travelled through these different template examples while containing the same "word" history.

Because the end-goal of the experiments presented here is to find the string of words for a given utterance, before the token merging is performed, tokens with an identical "word" history, i.e. tokens holding template sequences with identical label sequences, are combined together into one token with their total probabilities

[1]The tokens are not actually "merged", but it appears from outside view as if they do because multiple tokens are reduced to one remaining token (or N tokens in N-Best recognition).

added together. Templates representing phrases keep a whole label string which means that a token that has visited, for example, the word templates "i", "want", and "my" will not be combined with a token that has visited, for example, the phrase template "i_want_my". Because log-probabilities are used in the decoder, the sum of $\Pr(a) + \Pr(b)$ for two tokens a and b is calculated in log space as

$$\log(\Pr(a) + \Pr(b)) = \log \Pr(a) + \log(1 + \exp(\log \Pr(b) - \log \Pr(a))) \quad (6.1)$$

where $\log \Pr(a)$ and $\log \Pr(b)$ are the log-probabilities of token a and token b respectively. Equation (6.1) is used in all template-based recognition experiments reported later in Section 6.4 although it was found to give no statistically significant improvements over the traditional Viterbi recognition (no merging of identical hypotheses).

Equation (6.1) could be improved by integrating prior probabilities as a weighting mechanism — there are words and phrases that are represented by many templates in the reference database which could bias the token merging of Equation (6.1). The prior probabilities, estimated from word / phrase frequency (the inverse of), could be used essentially to normalise the probabilities when merging, although this is not implemented in the reported work.

6.3 VTLN for Templates

For the RM dataset, which is a speaker independent dataset containing both male and female speakers, Vocal Tract Length Normalisation (VTLN) was applied in order to reduce the variability introduced by the different lengths of vocal tracts in different speakers (on average, males have longer vocal tracts then women). Previously, the VTLN method for HMMs was described (Section 2.6), where a piecewise linear warping function [Hain et al., 1999] is used to warp the frequency

axis by the the inverse of a warp factor α which can be optimally found using the method described in Lee and Rose [1998].

Using VTLN on templates is not straight-forward. The number of reference templates is too large to choose the optimal warping factor by cycling through all of the possible warp factors[2]. For the experiments presented in this chapter, the application of VTLN to the reference templates is performed by using the HMM-based method (which was already performed for the baseline measure) to find the best warping factor for each speaker, which is then used to warp that speakers templates. For the recognition stage of VTLN, again the HMM-based approach was used: the Word-Bigram (WB) language model was used to find the best warping factor for each utterance in reference to the HMMs that were previously trained on the normalised speakers. Each warped utterance is then stored and can be used as input into the template-based Time Filter and decoder. Figure 6.2 shows the distribution of the *optimal* warp factors for each speaker in the training data (and thus reference template database). It shows the distribution of the male speaker warping factors and the female warping factors separately.

The application of speaker normalisation to template-based recognition was suggested in De Wachter et al. [2007] and recently Demange and Van Compernolle [2009b] have investigated VTLN for template-based recognition, using an on-line estimation of the warping factors on a sentence by sentence basis [Duchateau et al., 2006]. This method uses Gaussian mixture models (GMMs), trained on generic speech, to determine whether or not the input sentence is male or female. The warping factor is then based on how probable a speaker's sentence is to being male or female, and is also controlled with an extra parameter. Demange and Van Compernolle [2009b] was published after the work presented here was completed, and so represents a future improvement of the adopted HMM-based VTLN approach.

[2]There are thirteen warp factors spaced evenly in the range $0.88 \leq \alpha \leq 1.12$

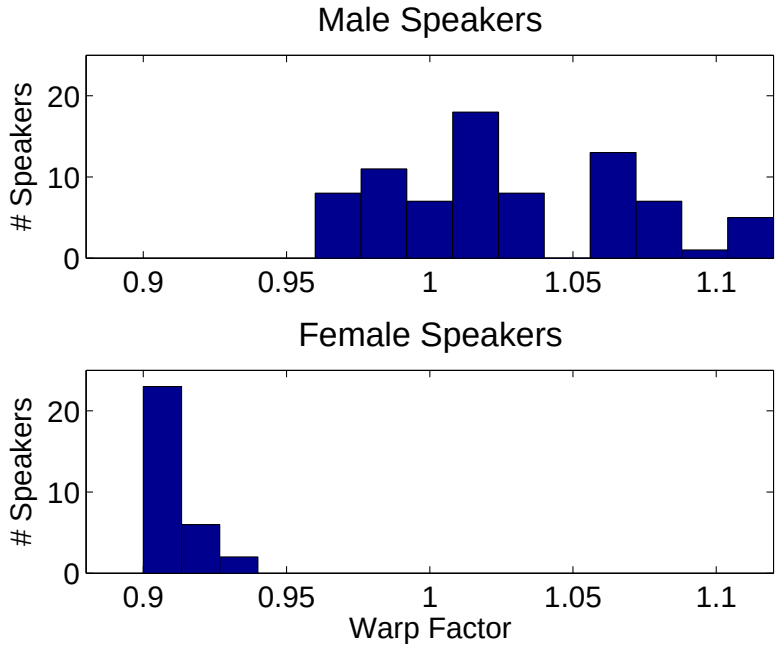


Figure 6.2: *The optimal warp factors selected for the RM training data. There are 78 male speakers and 31 female speakers in the training data.*

6.4 Recognition Experiments

6.4.1 Speaker Dependent Results

Table 6.1 shows the template recognition results for the SD test data as well as the HMM baseline results (previously presented in Section 4.6) for comparison. For the WB model, only word templates were loaded into the decoder (and Time Filter), while for the WPB and WPLB language models, word and phrase templates were used (the same set of activation graphs from the Time Filter are used by the decoder when using WPB or WPLB). Results are shown for all three language models with and without the LDA filtering — the LDA filter uses a window of 15 frames ($\chi = \pm 7$ around the centre frame) for the RBF which has a standard deviation of 0.25. For the Time Filter pass that was used to generate activation graphs for the decoder, the number of nearest neighbours was chosen to be 10,000.

Table 6.2 shows the *average* number of templates per utterance contained in the activation graphs for Time Filter passes with word-only templates, as well as

Templates	LM	Word Acc.(%)	HMM Word Acc. (%)
words	WB	83.88	86.92
words + phrases	WPB	85.6	87.52
words + phrases	WPLB	86.07	87.76
words + LDA	WB	80.96	n/a
words + phrases + LDA	WPB	83.85	n/a
words + phrases + LDA	WPLB	84.59	n/a

Table 6.1: *Word Accuracy on SD data for WB, WPB, and WPLB language models using template-based decoder using phrase weights of 20. Word accuracy is also shown after filtering the templates with the Hierarchical LDA method of Section 5.4. The results using HMMs (which use a phrase weight of 0.5), previously presented in Table 4.1, are also shown for comparison.*

word and phrases templates. The average number of templates remaining after the LDA filtering is applied is also shown.

Templates	Average Num. Templates	Relative Reduction
words	2047	n/a
words + phrases	2167	n/a
words + LDA	1713	16%
words + phrases + LDA	1804	17%

Table 6.2: *Shows the average number of templates per utterance for the SD Test data as candidates for the decoder from the Time Filter and the reduction in the number of templates when LDA filtering is applied.*

As with the results using the HMM-based recogniser, both WPB and WPLB achieve a higher performance than the WB baseline. WPLB, as with the HMMs, also performs better than WPB. It is clear to see from Table 6.1 that the LDA filtering has a negative effect upon the recognition accuracy, and is clearly a significant degradation in performance compared to the the pre-filtered templates. Table 6.3 shows the Matched-Pairs test for the pre-filtered template systems and baseline HMM systems.

First of all, Table 6.3 shows that WPB and WPLB are in fact significant improvements over the template WB system, and also that WPLB does indeed achieve significant improvements over the WPB system (which was not observed for the HMM-based equivalents). It can also be seen that the baseline HMM

	WB	**WPB**	**WPLB**	**WB**$_{\text{hmm}}$	**WPB**$_{\text{hmm}}$	**WPLB**$_{\text{hmm}}$
WB		WPB	WPLB	WB$_{\text{hmm}}$	WPB$_{\text{hmm}}$	WPLB$_{\text{hmm}}$
WPB			WPLB	WB$_{\text{hmm}}$	WPB$_{\text{hmm}}$	WPLB$_{\text{hmm}}$
WPLB				same	WPB$_{\text{hmm}}$	WPLB$_{\text{hmm}}$
WB$_{\text{hmm}}$					WPB$_{\text{hmm}}$	WPLB$_{\text{hmm}}$
WPB$_{\text{hmm}}$						same
WPLB$_{\text{hmm}}$						

Table 6.3: *Statistical significance tests on the SD test data. The Matched-Pairs test for hypotheses from the template-based systems and comparable HMM-based systems.*

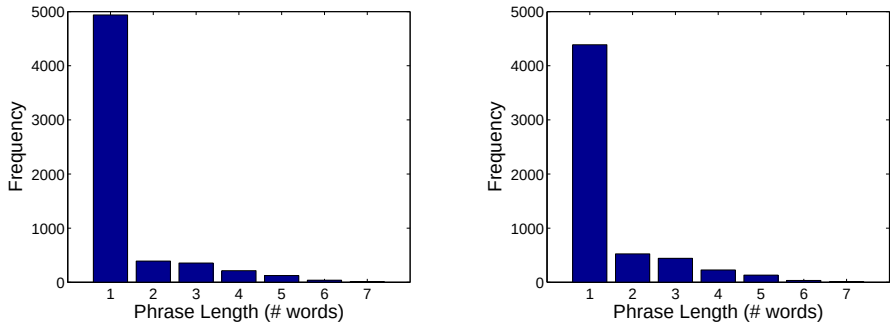
output performs competing systems that used the same language model) in the template systems, although the template-based WPLB is judged to give, statistically, the same level of performance as the HMM-based WB system (WB$_{\text{hmm}}$ in Table 6.3).

Figure 6.3 shows the histograms of the length of units (in terms of number of words) used by the template-based decoder when using WPB and WPLB language models[3]. As with analysis provided in Section 4.6 for language units selected by the HMM decoder, it is clear to see that the WPLB language model forces the template-based decoder to select fewer words, and thus longer phrases. The template-based decoder does select a higher number of words for WPB (81% of recognised units) and WPLB (76% of recognised units) than the HMM decoder (which uses 79% for WPB and 73% for WPLB), but of course the HMM decoder is concatenating phone models together, whereas for the template-based decoder each word and each phrase corresponds to a whole template example.

Returning to Table 6.1 and Table 6.2, the LDA filtering method is clearly degrading the performance of the recogniser, even though the number of templates is reduced. It seems likely that the templates that were filtered out were in fact "correct" templates, and thus there is a decrease in performance. As previously reported [Watkins and Cox, 2009], for a reduced-performance system[4] (79.41% word accuracy using an average 4773 templates (words + phrases) per utterance),

[3]The reader is reminded that the WPB and WPLB systems both have available exactly the same set of templates, i.e. they use the same activation graphs.

[4]At the time this was our best result for the SD data.



(a) *Used phrases in WPB recognition.* (b) *Used phrases in WPLB recognition.*

Figure 6.3: *Histograms of the length of phrase units chosen by the template-decoder on the SD data during recognition with WPB and WPLB language models.*

the LDA filter reduces the number of templates by 37% while increasing the word accuracy (80.07%). Because of the lower performance it is important not to over-emphasise the LDA filter performance, but it does show that the LDA filter can be effective if the Time Filter introduces a larger number of (incorrect) template candidates.

6.4.2 Speaker Independent Results

Table 6.4 shows the template recognition results for the RM evaluation sets (oct89, feb91, and sep92). The HMM baseline results were all better than the template-based results by at least 10% absolute (word accuracy) and are not included in the table for comparison, but the interested reader can refer back to Table 4.3 for the HMM-based word accuracy results. The same set of experiments as for the SD data were run, with the addition of VTLN. The LDA filtering experiment results are only reported for the normalised (VTLN) templates as these systems performed significantly better than the unnormalised templates. The LDA filter window used in the experiments was set to 15 frames again ($\chi = \pm 7$), but this time a standard deviation of one was used. The Time Filter used to generate the template candidates used 20,000 nearest neighbours. All of the parameters were optimised on the RM development set (feb89).

			Word Accuracy (%)		
Templates	**LM**	**VTLN**	**oct89**	**feb91**	**sep92**
words	WB	✗	75.78	76.85	71.24
words + phrases	WPB	✗	75.48	76.61	71.08
words + phrases	WPLB	✗	75.41	77.13	71.79
words	WB	✓	78.06	79.43	73.43
words + phrases	WPB	✓	78.69	79.79	75.69
words + phrases	WPLB	✓	78.54	80.15	75.65
words + LDA	WB	✓	74.14	76.13	69.75
words + phrases + LDA	WPB	✓	76.86	78.66	74.36
words + phrases + LDA	WPLB	✓	77.35	78.99	74.33

Table 6.4: *Word Accuracy on the three test sets oct89, feb91, and sep92 for WB, WPB, and WPLB language models, with and without VTLN. Word accuracy after applying Hierarchical LDA filtering to the templates is also shown for each LM with VTLN. The WPB and WPLB language models use a phrase weight of 15.*

Table 6.5 gives the average number of templates per utterance as delivered from the Time Filter for each of the systems reported in Table 6.4. The fourth column in Table 6.5 gives the reduction in the average number of templates per utterance for the word-only templates with VTLN and LDA filtering (row four), and the word+phrase templates with VTLN and LDA filtering (row five) relative to the normalised templates without LDA filtering.

Templates	**VTLN**	**Average Num. Templates**	**Relative Reduction**
words	✗	1756	n/a
	✓	1726	n/a
words + phrases	✗	1894	n/a
	✓	1842	n/a
words + lda	✓	1247	28%
words + phrases + lda	✓	1317	29%

Table 6.5: *Shows the average number of templates per utterance for the RM evaluation data (oct89, feb91, and sep92) as candidates for the decoder from the Time Filter and also the reduction (%) of the number of templates when using LDA filtering.*

It is clear from Table 6.4, that as with the HMM-based experiments, for the original templates (i.e. not normalised), the different language models (WB, WPB, and WPLB) have very little effect upon the final word accuracy. This is confirmed with the Matched-Pairs tests presented in Table 6.6 (oct89 and feb91) and Table

6.7 (sep92) which shows that the WB, WPB, and WPLB systems are all statistically equivalent.

	WB	**WPB**	**WPLB**	**WB**$_{\text{vtln}}$	**WPB**$_{\text{vtln}}$	**WPLB**$_{\text{vtln}}$
WB		same	same	WB$_{\text{vtln}}$	WPB$_{\text{vtln}}$	WPLB$_{\text{vtln}}$
WPB			same	WB$_{\text{vtln}}$	WPB$_{\text{vtln}}$	WPLB$_{\text{vtln}}$
WPLB				WB$_{\text{vtln}}$	WPB$_{\text{vtln}}$	WPLB$_{\text{vtln}}$
WB$_{\text{vtln}}$					same	same
WPB$_{\text{vtln}}$						same
WPLB$_{\text{vtln}}$						

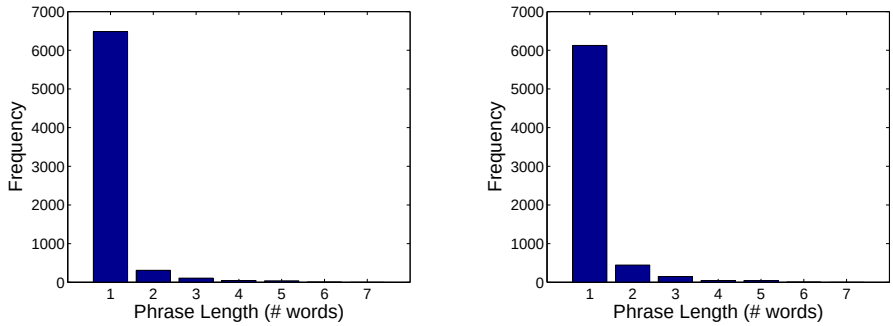
Table 6.6: *Matched-Pairs test on template-based systems using language models WB, WPB, and WPLB, with and without VTLN for the RM oct89 and feb91 evaluation sets.*

When using VTLN on the templates, there is a large increase in word accuracy, as can be seen in Table 6.4, in comparison to the same system without VTLN — there is an absolute improvement of at least 2% in each system. This is confirmed as significant by the Matched-Pairs test in tables 6.6 and 6.7. For the oct89 and feb91 evaluation sets (see Table 6.6), the VTLN systems again are statistically the same for changing language models (the same pattern as for the original templates). For the sep92 evaluation set, the VTLN WPB and VTLN WPLB systems both achieve a significant increase over the VTLN WB system (see Table 6.7).

	WB	**WPB**	**WPLB**	**WB**$_{\text{vtln}}$	**WPB**$_{\text{vtln}}$	**WPLB**$_{\text{vtln}}$
WB		same	same	WB$_{\text{vtln}}$	WPB$_{\text{vtln}}$	WPLB$_{\text{vtln}}$
WPB			same	same	WPB$_{\text{vtln}}$	WPLB$_{\text{vtln}}$
WPLB				same	WPB$_{\text{vtln}}$	WPLB$_{\text{vtln}}$
WB$_{\text{vtln}}$					WPB$_{\text{vtln}}$	WPLB$_{\text{vtln}}$
WPB$_{\text{vtln}}$						same
WPLB$_{\text{vtln}}$						

Table 6.7: *Matched-Pairs test on template-based systems using language models WB, WPB, and WPLB, with and without VTLN for the RM sep92 evaluation set.*

Figure 6.4 shows the familiar analysis of the length of unit used in recognition. As before, it can be seen that the WPLB model decreases the number of words chosen, which thus results in more longer phrases (typically of two or three words). The important thing to note is that, for the WPB system, 93% of the used units by the decoder are words, and for the WPLB system that figure becomes 90%. This contrasts to the units used by the HMM-based decoder which were 85% and 79% respectively.



(a) *Used phrases in WPB recognition.* (b) *Used phrases in WPLB recognition.*

Figure 6.4: *Histograms of the length of phrase units chosen by the template-decoder on the RM data during recognition with WPB and WPLB language models.*

Finally in this section, we return to the LDA filter results presented in Table 6.4. It is clear that, as with the SD data, the application of the LDA filter leads to a reduction in word accuracy, albeit with a large reduction in the number of template candidates for the decoder, as shown in Table 6.5. The degradation of performance on the RM data is not as great as on the SD data, and thus Matched-Pairs tests were performed to see if the LDA filtering led to a significant reduction in word accuracy. These are presented in Table 6.8.

Dataset	System 1	System 2	Significance
oct89	WB_{vtln}	$WB_{vtln+lda}$	WB_{vtln}
	WPB_{vtln}	$WPB_{vtln+lda}$	WPB_{vtln}
	$WPLB_{vtln}$	$WPLB_{vtln+lda}$	$WPLB_{vtln}$
feb91	WB_{vtln}	$WB_{vtln+lda}$	WB_{vtln}
	WPB_{vtln}	$WPB_{vtln+lda}$	same
	$WPLB_{vtln}$	$WPLB_{vtln+lda}$	same
sep92	WB_{vtln}	$WB_{vtln+lda}$	WB_{vtln}
	WPB_{vtln}	$WPB_{vtln+lda}$	same
	$WPLB_{vtln}$	$WPLB_{vtln+lda}$	same

Table 6.8: *Matched-Pairs test on RM evaluation sets for comparison of template-based VTLN systems with and without LDA filtering. Each system after LDA filtering (system 2) is compared to the same system before filtering (system 1).*

Table 6.8 compares each of the template-based systems (WB,WPB, and WPLB) with VTLN (e.g. WB_{vtln}) to the same systems after LDA filtering (e.g. $WB_{vtln+lda}$). For the oct89 evaluation set, it can be seen that the LDA filtering does result in a significant degradation in word accuracy for each language model. However, for the feb91 and sep92 sets, the WPB and WPLB systems do *not* give a significant

drop in accuracy after LDA filtering, even though the LDA filtering reduces the number of template candidates in the decoder for WPB and WPLB by 29% (Table 6.5).

6.5 Conclusions and Discussion

This chapter described the template-based decoder and the recognition experiments that were used to evaluate the template-based approach to continuous speech recognition. Section 6.2 described how the classic Token Passing decoder was adapted to deal with the bottom-up template selections from the Time Filter, including how the activation graph was used by the decoder. The token merging method within the token passing algorithm was also adapted so that tokens that had visited different template examples with the same word string were merged when the tokens arrived at the same state for a given time, with their probabilities summed together. Section 6.3 described how a HMM-based Vocal Tract Length Normalisation (VTLN) technique was used for the template-based approach to speaker normalisation, with the optimal warping of the reference speakers performed on a HMM system, and the optimal warping of the input utterances performed once using HMMs trained on normalised speech, with the normalised input utterances then used for all experiments.

The experiments, described in Section 6.4, were performed on the speaker dependent (SD) call routing data and the speaker independent RM data. For the SD data, it was shown that template-based results were approaching the HMM baseline results, in particular with the WPLB system which was judged to be statistically the same as the HMM WB system. The comparison of the template-based systems showed that the WPB and WPLB systems both achieved (statistically) significant improvements compared to word-only templates with the WB language model, while the increase in word accuracy from the WPB system to the WPLB system was also a statistical improvement.

For the RM dataset, the results of the template-based recognition were someway behind that of the HMM baseline results (an absolute difference in word accuracy of at least 10% after applying VTLN). For comparison of the template-based results, the WB, WPB, and WPLB systems (without VTLN) all gave statistically equivalent word accuracy — it was shown that for WPB and WPLB, over 90% of the recognised units were infact words which could be one reason for the parity. After applying VTLN to the reference templates and input utterances, large increases were found for each template-based system (at least 2% absolute compared to without VTLN), with the WPB and WPLB VTLN systems achieving statistically significant improvements over the WB VTLN system on the sep92 evaluation set only. The average number of reference speakers available to the decoder (i.e. contained in the activation graphs) per utterance rose from 103.8 to 108.4 when using VTLN (the total number of reference speakers available is 109) — while this seems to be a small increase, it is averaged over 900 utterances (oct89, feb91, and sep92 were combined togther). It was shown that using VTLN reduced the average number of templates per utterance by about 50 when using word and phrase templates with the Time Filter (Table 6.5). Recall from Section 5.3, that after the activation graph is formed by the Time Filter, a threshold is applied on the distance scores at each frame. Because the application of the VTLN reduces the average number of templates per utterance while increasing the word accuracy suggest that the VTLN reduces the distance of the "correct" templates[5] closer to the input, and thus the "incorrect" templates move further away, hence when thresholding is performed, the number of templates remaining decreases. Reducing the spectral variability with VTLN thus results in the templates that were similar to the input by spectral properties only being removed from the activation graph (after thresholding the distances).

Finally, it was shown that the LDA filter (described in Section 5.4) was generally unsuccessful in reducing the number of templates in the activation graph *and* increasing the word accuracy. On the SD call-routing data the average num-

[5]Correct in the sense that the word strings of the template match that of the input.

ber of templates per utterance were reduced by 17%, but this reduction led to statistically significant reductions in recognition performance. On the RM data, the average number of templates were reduced by 29% on the word + phrase templates with VTLN, with the WPB and WPLB systems achieving statistically equivalent performance to the unfiltered WB system (with VTLN) on the feb91 and sep92 datasets. It was shown that the LDA filter performs well when applied to an activation graph with a high number of incorrect templates — this result suggests that the features that were extracted for the LDA filter (Section 5.4.1) are suitable for identifying a large number of incorrect templates, but require a higher level of detail to make sure that correct templates are not removed from the activation graphs which is vital to maximising word recognition performance.

Chapter 7

Discussion and Conclusions

7.1 Summary and Discussion

This thesis was motivated by the idea of improving ASR accuracy by exploiting *formulaic sequences* in language, which occur in human-to-human dialogue. To minimise the effort of both speech production and perception in the conversation, the human speaker will retrieve chunks of speech "whole from memory rather than being subject to generation or analysis by the language grammar" [Wray and Perkins, 2000]. This production in turn seems to prime the human listener with a set of semantic expectations [Pickering and Garrod, 2004] which then may prime an appropriate set of formulaic sequences that the listener expects to hear to enable quick and efficient perception.

The aim of this thesis was firstly to acquire formulaic sequences or *phrases* from transcriptions of speech and then to use them to define units of language and speech that have a variable length in a speech recogniser, with the aim of improving recognition accuracy. A summary of each chapter is given below, including methods and findings.

Chapter **2** gave the technical background required for the methods presented in this thesis. It gave a thorough background into N-gram language modelling,

including backoff models, representation within Stochastic Finite State Automata (SFSA), and the evaluation metric perplexity. It also described key techniques required in template-based recognition including frame-based distance measures, DTW (Dynamic Time Warping), and the token passing algorithm, which it was shown could be applied to HMM-based recognition also. It was also shown how to integrate language model SFSAs with HMMs and templates to construct the decoding network. A HMM-based method to perform Vocal Tract Length Normalisation (VTLN) was also described, and finally, Linear Discriminant Analysis (LDA) was described for constructing a simple binary classifier.

Chapter **3** described the data used to evaluate the methods in this thesis. A speaker dependent (SD) dataset, originally recorded by multiple speakers in a call-routing environment, was introduced, and was shown to contain a high frequency of commonly-occurring phrases. The Resource Management (RM) dataset, which contains multiple training and test speakers, was also detailed. The methods used for feature extraction and HMM training for each dataset were described, as well as information about the number of templates that both datasets contained.

Chapter **4** was concerned with acquiring commonly-occurring phrases from transcriptions of speech and investigating ways to model these in language models. A multigram segmentation framework [Deligne and Bimbot, 1995] was used to acquire the phrases, by finding the best segmentation of a given utterance. In an attempt to cluster phrases with a similar semantic function, and hence provide a set of "primed" phrases for a given context, a simplified syntactic approach was used, termed *Hybrid Syntactic Formulaic* (HSF) clustering, which combined the commonly-occurring phrases with the information from syntactic parse trees to assign phrases to classes based on their context. The acquired phrases and phrase classes were then integrated into the N-gram framework using different language model topologies, including the *Word Phrase Link Bigram* (WPLB) which combined word N-gram models with phrase N-gram models, linking the two with *new* unseen bigram transitions between the two model layers. The phrase classes were

integrated seamlessly into the WPLB model by representing each class as local models using SFSAs — this language model was termed the *Word Phrase Class Link Bigram* (WPCLB).

Using the baseline HMM-based monophone recogniser, it was shown that the WPLB model gave statistically significant improvements over the word bigram (WB) model, for the SD call-routing data, improving the word accuracy to 87.76% from 86.92% The WPLB model also gave a significant decrease in *perplexity*, from 13.1 to 11.05. The WPCLB model also gave a signficant increase over the WB model, achieving a word accuracy of 87.79% (not significantly better than WPLB) and a perplexity of 11.35. However, the same improvements in word accuracy of the recogniser were not observed on the RM dataset. In fact, there were no statistically significant improvements on any of the methods presented. An analysis of the (language) units chosen by the decoder showed that a high proportion of the units were in fact at the word level on both the SD and RM datasets, which implied that the chosen phrases did not generalise well to the unseen data, forcing the recogniser to fall-back onto the words. A recognition experiment on the call-routing *training data* showed that a much higher proportion of the units chosen by the decoder were phrases (greater than one word) which is strong evidence to suggest that the phrases do not generalise well to unseen data.

Chapter **5** described methods for the bottom-up selection of templates for the decoder. These were required to reduce the massive search space when using all templates defined by words and phrases acquired in Chapter 4. Because of memory limitations, it is impossible to use all templates on standard desktop computers. Extensions to the Time Filter algorithm [De Wachter et al., 2003] were defined including a new *backwards* pass of the algorithm and a distance normalisation method based on a Sigmoid function to control the length of templates selected. The normalisation method was shown to be ineffective, concluding that it should be applied at the selection of the KNNs. The backward pass was shown to find templates that the forward pass did not find, although this included a larger

number of incorrect templates than correct templates.

A second template filter, a hierarchical LDA filter, was introduced to further filter the templates after the Time Filter was applied, i.e. to filter the template selections. This was motivated by an *Oracle* test which showed that removing the incorrect template selections led to a large increase in speech recognition word accuracy. The LDA filter used features based on observations of the template selections, such as occurrence probability, to train two LDA classifiers which were placed in a decision tree, classifying a template as correct or incorrect depending on a threshold. Templates classed as incorrect were then classified again on the second level of the decision tree.

Evaluation of the hierarchical LDA filter was done with template coverage and classification accuracy. The LDA filter was found to remove a high number of incorrect template selections but also to remove some correct templates which led to a drop in template coverage accuracy to the input utterances. The classification performance of the LDA decision tree showed that indeed low False Negative Rates (FNR) were observed (1.8% on the SD data and 3.3% on RM), while False Positive Rates (FPR) were high (62.74% and 63.01% respectively). A key finding was that the *difference* in template coverage accuracy between word-only templates and word + phrases templates increased after the LDA filtering was applied (double the difference on the SD data and almost treble on the RM data). This was evidence that, generally, the longer templates that survived the Time Filter were correct selections, which influenced the LDA classification as template length was a feature used to train the LDA. It was suggested that length feature had a strong effect on filtering shorter correct templates.

Chapter **6** defined how the template-based decoder integrated the template selections from Chapter 5 and the phrase-based language models from Chapter 4 into a recognition system which was then used to perform recognition experiments. VTLN for templates in the RM dataset was described and the HMM-based method for selecting warping factors, described in Chapter 2, was used to normalise the

templates. Each experiment thus used the same templates and test utterances. It was shown that the template-based recogniser with the WPLB language model on the SD data, achieved a recognition accuracy of 86.07% which was statistically equivalent to the baseline HMM system using the WB model. It was also shown that the WPLB template-based system achieved a much higher word accuracy than the word-only template system using the WB model, which performed at 83.88% word accuracy.

For the experiments on the RM data, it was shown that, even after VTLN was applied to the templates and input utterances, the word accuracy for the template-decoder was much lower than the baseline HMM systems, with an absolute difference of at least 10%. It was shown that over 90% of the recognised units were words, and that, as with the HMM-based results, there were no statistical differences in performance when changing the language model (before VTLN). However, when VTLN was applied, the smallest improvement compared to template-based systems without VTLN was 2% absolute, with the WPLB template system achieving a significant increase over the WB template system (with VTLN) on the sep92 evaluation set. The WPLB template system achieved the highest overall word accuracy of 80.15% on the feb91 set, which was an improvement of 3.02% absolute over the same system without VTLN (77.13%). The largest improvement observed from using VTLN was about 4% absolute.

Experiments were also performed using the LDA filtered template selections, with the SD data showing significant degradation in word accuracy. However, on the RM feb91 and sep92 datasets, with a reduction of 29% in the average number of templates per utterance, the WPLB system with VTLN achieved statistically equivalent performance to the template WB system with VTLN and no LDA filtering.

7.2 Conclusion and Future Work

Selecting variable length units of speech and language for speech recognition, by segmenting transcriptions of speech shows promise for a template-based speaker dependent recognition system, but template-based performance falls some way short of a baseline phone HMM performance on speaker independent systems. It was shown that by using the word as the minimal unit and supplementing that with commonly-occurring phrase-based units, a recognition word accuracy of 86.07% could be achieved, which was shown to be statistically equivalent to the HMM-baseline results when using a simpler language model. However, experiments on a speaker independent system gave best results of 80.15% word accuracy after VTLN was applied which was over 12% (absolute) lower than the baseline HMM-system. It was shown that the poor generalisation of the phrases to unseen data was a major factor in poorer recognition peformace. The synthetic nature (sentences produced from a grammar) of the RM datset means that the data contains fewer formulaic phrases. This was observed in the segmentation of the transcriptions, which gave a higher proportion of shorter phrases than the segmentation of the highly formulaic call-routing dataset. This was a factor in the poorer performance on the RM dataset.

Efforts to improve the recognition accuracy were made with a hierarchical LDA filter which attempted to remove incorrect template candidates from the Time Filter selections by using a linear classifier to separate correct and incorrect templates based on features of occurrence, score, and length. This was shown to work well at removing a high number of incorrect templates, but also removed a (much) smaller number of correct templates, and this resulted in lower speech recognition performance.

Section 7.2.1 addresses the issues raised in this chapter, and indeed this thesis, with improvements that could be made in the future.

7.2.1 Future Work

Certainly the first issue that needs to be addressed is the minimum template unit length. It has been shown that by using phone templates and then concatenating these templates, with various associated concatenation costs, that high speaker independent performance can be achieved [De Wachter et al., 2007]. This is a commonly-known problem in template-based recognition where there usually are not enough examples of words, and certainly phrases, to accurately recognise speech from different speakers, and so reverting to phones provides a much higher number of template examples. Incorporating this idea [De Wachter et al., 2007] into the work presented here is a viable option: the segmentation of transcriptions of speech could be performed at the phone level, resulting in phones, words, and phrase units — words and phrases could be constructed from the phone sequence using a pronunciation dictionary. These units could then be used in the same way as described in this thesis, with all phones used as a backoff mechanism for unseen speech that does not match well to the reference speakers.

Another big problem in this work was shown to be the poor generalisation of the acquired phrases to unseen data. By using fixed phrases, any minor variants of the phrases in the test data will not be recognised at the phrase level, but at the word level. This leaves a large number of reference templates unused, which will become an even higher number the more the test data differs from the training data. One method to model the variations in phrases, would be to model things such as *open-class* items using slots in the phrases that allow different words to be inserted within a basic structure for a phrase. For example, a phrase structure such as "*X catch+TENSE Y red-handed*" [Wray, 1999] can allow phrases such as "he caught her red-handed", "Bob is going to catch Linda red-handed", and so on. Each individual phrase structure could be modelled as *fragments* [Arai et al., 1999] and integrated into an N-gram framework using SFSAs, as was shown for phrase classes in Chapter 4. Integrating clustering with the fragments can then result in modelling phrases that were unseen in the training data, thus addressing

the issue of generalisation to the test data.

Chapter 5 described a number of techniques that were tried to improve the performance of the Time Filter algorithm [De Wachter et al., 2003]. The new backward pass of the Time Filter algorithm essentially was a "mirror-image" of the forward pass, using the same constraints and costs as the forward pass. Different constraints on the backward pass could be investigated, investigating patterns in the speech that the forward pass misses. The backward pass could also be assigned less weight than the forward pass, employing it to select only the highest scoring templates. The hierarchical LDA filter showed some promise, but was found to be too aggressive. One of the reasons stated for this was the choice of features used in the LDA. While new features such as a prior probability based on a template's occurrence in the training data and a windowed z-score, or optimised features and threshold calculation on a development set could improve the LDA performance, attention would focus on whether a *non-linear* classifier, such as a Support Vector Machine (SVM) [Cristianini and Shawe-Taylor, 2000], could improve the separation of the correct and incorrect template classes.

Appendix A

Sample Output of Multigram Segmentation

A.1 Examples from SD Call-Routing Data

The following segmented utterances are actual examples chosen at random from the SD call-routing data. The maximum number of words per phrase L is set to 7, with the initial threshold θ_1 set to 3, and the iterating threshold θ_2 set to 1 (refer to Section 4.3 for more information on parameters):

<s> [i've] [to pay] [something] [i mean] [do i have] [something] [due] [in my account] </s>

<s> [i have a maintenance agreement] [so] [can you transfer me to some] [agent] </s>

<s> [my account balance] </s>

<s> [i need to know] [my account balance] </s>

<s> [i would like to] [talk to] [an agent] </s>

<s> [parts] [info] </s>

<s> [i'd like to add] [add somebody to my account] [so] [they] [can use my card] [too] </s>

<s> [i] [actually] [want] [to order a part] </s>

<s> [how] [i] [how do i] [add another person to my account] </s>

<s> [how many points do i have] [right now] </s>

<s> [yes] [i was wondering] [what] [your] [mailing address] [is] </s>

<s> [how can i get] [a premier card] </s>

<s> [what are the last few] [charges on my card] </s>

<s> [on] [what day] [did] [my check] [clear] </s>

<s> [how much do i owe] [now] </s>

<s> [where can i] [send my payment] </s>

<s> [has my] [check cleared] </s>

<s> [i want to] [raise my credit limit] </s>

<s> [how do i get] [another person] [to be able to use] [this account] </s>

<s> [how can i get] [more credit] </s>

<s> [where do i send my] [check] </s>

<s> [i] [bought something] [and i returned] [it and] [the] [the credit] [for it] [isn't] [showing] [up] [on my bill] </s>

<s> [i lost my] [credit] </s>

<s> [yeah] [you] [you have] [a] [sale] [oh] [coming up] [and i believe] [it's] [the end of] [october] [how long] [do you] [plan] [it to] [last] [i did] [i don't] [i don't know] [the] [date thank you] </s>

<s> [duplicate statement] [please] </s>

<s> [can you enroll me] [in] [rewards program] </s>

A.2 Examples from SI RM data

Sample segmented utterances from the RM data: $L = 7$, $\theta_1 = 3$, and $\theta_2 = 1$:

<s> [show] [only] [the] [visual] [sensor] [latitudes and longitudes] [available] [on] [hawkbill] </s>

<s> [what if] [plunger] [replaced] [the pollack] [in china sea] </s>

<s> [when is] [puffer] [arriving in port] </s>

<s> [what was] [ranger's] [readiness] [june] [fourteenth] </s>

<s> [give me] [sps-48 capable] [cruisers] [at sea] [today] </s>

<s> [list the] [carriers that] [were deployed on the] [eighth] [of september] </s>

<s> [what if] [queenfish] [increased] [its] [average cruising speed] [by one] [knot] </s>

<s> [get all] [c-codes for] [seawolf] </s>

<s> [find] [speeds for] [the ships] [that are in siberian sea] </s>

<s> [when's] [swordfish] [due in port] </s>

<s> [start editing] [position data for] [arkansas's] [track] </s>

<s> [who] [had the] [highest] [average] [c-rating] [during the last] [year] </s>

<s> [when is] [badger] [changing fleets] </s>

<s> [show] [areas] </s>

<s> [never mind] [the] [next] [chart] [display] </s>

<s> [give] [speeds] [of the] [c2] [submarines] </s>

<s> [review] [cheshire] [area alerts] </s>

<s> [what is] [bainbridge's] [propulsion] </s>

<s> [list] [pacflt] [ships that] [are c3 on equipment] </s>

<s> [get me] [lats and lons] [and] [speeds] [for the] [subs in] [gulf of tonkin] </s>

<s> [find] [latitudes] [and] [names of] [vessels that are in] [sea of japan] </s>

<s> [give me a list of] [longitudes] [of tracks that are in] [the formosa strait] </s>

<s> [get a list of] [positions for] [ships in gulf of california] [that went to] [c4] [nine] [january] </s>

<s> [show all] [locations of] [tracks for] [usn] [frigates] </s>

<s> [give] [latitudes and longitudes] [and] [names of any] [of eastpac's] [cruisers that were] [in the] [philippine sea] [on] [november] [twentieth] </s>

<s> [list the] [carrier's] [positions for] [april] </s>

<s> [what is the] [frigate's] [home port] </s>

Appendix B

Penn Treebank

This chapter contains the definitions of tags used in the Penn Treebank — the following tags and their respective definitions were all taken from Bies et al. [1995].

B.1 POS Tags

CC — Coordinating conjunction.

CD — Cardinal number.

DT — Determiner.

EX — Existential there.

FW — Foreign word.

IN — Preposition or subordinating conjunction.

JJ — Adjective.

JJR — Adjective, comparative.

JJS — Adjective, superlative.

LS — List item marker.

MD — Modal.

NN — Noun, singular or mass.

NNS — Noun, plural.

NNP — Proper noun, singular.

NNPS — Proper noun, plural.

PDT — Predeterminer.

POS — Possessive ending.

PRP — Personal pronoun.

PRP$ — Possessive pronoun (prolog version PRP-S).

RB — Adverb.

RBR — Adverb, comparative.

RBS — Adverb, superlative.

RP — Particle.

SYM — Symbol.

TO — to.

UH — Interjection.

VB — Verb, base form.

VBD — Verb, past tense.

VBG — Verb, gerund or present participle.

VBN — Verb, past participle.

VBP — Verb, non-3rd person singular present.

VBZ — Verb, 3rd person singular present.

WDT — Wh-determiner.

WP — Wh-pronoun.

WP$ — Possessive wh-pronoun (prolog version WP-S).

WRB — Wh-adverb.

B.2 Phrase-Level Tags

ADJP — Adjective Phrase.

ADVP — Adverb Phrase.

CONJP — Conjunction Phrase.

FRAG — Fragment.

INTJ — Interjection. Corresponds approximately to the part-of-speech tag UH.

LST — List marker. Includes surrounding punctuation.

NAC — Not a Constituent; used to show the scope of certain prenominal modifiers within an NP.

NP — Noun Phrase.

NX — Used within certain complex NPs to mark the head of the NP. Corresponds very roughly to N-bar level but used quite differently.

PP — Prepositional Phrase.

PRN — Parenthetical.

PRT — Particle. Category for words that should be tagged RP.

QP — Quantifier Phrase (i.e. complex measure/amount phrase); used within NP.

RRC — Reduced Relative Clause.

UCP — Unlike Coordinated Phrase.

VP — Vereb Phrase.

WHADJP — Wh-adjective Phrase. Adjectival phrase containing a wh-adverb, as in how hot.

WHAVP — Wh-adverb Phrase. Introduces a clause with an NP gap. May be null (containing the 0 complementizer) or lexical, containing a wh-adverb such as

how or why.

WHNP — Wh-noun Phrase. Introduces a clause with an NP gap. May be null (containing the 0 complementizer) or lexical, containing some wh-word, e.g. who, which book, whose daughter, none of which, or how many leopards.

WHPP — Wh-prepositional Phrase. Prepositional phrase containing a wh-noun phrase (such as of which or by whose authority) that either introduces a PP gap or is contained by a WHNP.

X — Unknown, uncertain, or unbracketable. X is often used for bracketing typos and in bracketing the...the-constructions.

B.3 Clause-Level Tags

S — simple declarative clause, i.e. one that is not introduced by a (possible empty) subordinating conjunction or a wh-word and that does not exhibit subject-verb inversion.

SBAR — Clause introduced by a (possibly empty) subordinating conjunction.

SBARQ — Direct question introduced by a wh-word or a wh-phrase. Indirect questions and relative clauses should be bracketed as SBAR, not SBARQ.

SINV — Inverted declarative sentence, i.e. one in which the subject follows the tensed verb or modal. SQ - Inverted yes/no question, or main clause of a wh-question, following the wh-phrase in SBARQ.

Appendix C

Sample Output for HSF Clustering of Phrases

Below are examples of phrase classes from the Hybrid Syntactic Formulaic (HSF) clustering algorithm presented in Section 4.4.

my account balance (32)	my current balance (2)
my account information (3)	my last transaction (1)
my account number (3)	my line of credit (2)
my address (1)	my minimum payment (1)
my available credit (6)	my next payment (1)
my balance (15)	my payment address (1)
my bill (1)	my payment date (3)
my billing date (1)	my payment due date (2)
my card (2)	my payment mailing address (1)
my check (1)	my rewards status (6)
my credit limit (1)	your payment address (2)
my current account balance (1)	

Figure C.1: *Class 13*: [(PRP_VB), (PRP_VBP), SINV, (VB_PRP), (WRB_RP), (NNS_IN), (TO_RP), (VBP_IN), (WRB_VBG), (UH_VBP), (PRP_VBG), (AUX_AUX), (WRB_VB), (PRP_AUX), (MD_VB), (PRP_WP)] — **(PRP$_NN)** — [null,(VB_PRP), (PRP_VB), VP, (MD_AUX)]. *Total number of phrases = 89.*

i'd like to (44)	i wanted to (3)
i'm calling to (1)	i wanted to be able to (1)
i'm trying to (3)	i want to (26)
i need the address to (2)	i would like to (25)
i need to (34)	

Figure C.2: *Class 14*: [null, (PRP_NN)] — **(PRP_TO)** — [(VB_TO), (VB_NN), (VB_PRP), (VB_AUX)]. *Total number of phrases = 139.*

add an additional user to my card(2)	get a new credit card (1)
add another person to my account (2)	get another copy of my statement (2)
add another user to my account (1)	get another one (2)
add my wife to my account (2)	get a replacement card (4)
add my wife to my card (1)	get a replacement credit card (1)
add somebody to my account (2)	increase my credit limit (2)
add someone to my account (1)	increase my credit line (1)
add someone to my card (1)	increase my line of credit (1)
cancel my credit card (1)	know where i can send my payment (2)
change my address (1)	make a payment (3)
change my due date (2)	pay my bill (1)
change my line of credit (1)	raise my credit limit (3)
change my payment address (2)	receive a duplicate statement (1)
change my payment date (1)	replace my credit card (2)
check my account balance (2)	report a lost card (5)
close my account (2)	request a credit limit increase (2)
enroll in the rewards program (5)	schedule a maintenance appointment (1)
get a copy of my statement (2)	schedule a repair (1)
get a higher credit limit (1)	speak to an agent (1)
get a line of credit increase (2)	speak with someone (1)

Figure C.3: *Class 52*: [(PRP_TO)] — **(VB_NN)** — [null]. *Total number of phrases = 71.*

can i get (44)	can you give (2)
can i hear (1)	can you replace (1)
can i use (1)	can you tell (1)
can you explain (1)	could you send (1)

Figure C.4: *Class 499*: [null, INTJ, ADJP, NP, WHNP] — **(MD_VB)** — [(DT_NN), (NN_IN), (DT_VB), (PRP$_VBN), (PRP$_VB), (DT_IN), (VB_NN), (NN_NN), (NNS_VBN), (DT_UH), (DT_PRP$), (JJR_NN), (IN_NNS), (PRP$_NN), (DT_NNS)]. *Total number of phrases = 52.*

payment address please (1)
statement please(1)

Figure C.5: *Class 1002*: [(MD_PRP$)] — **(NN_VB)** — [null]. *Total number of phrases = 2.*

account balance (5)
maintenance agreement (1)
payment due date (1)

Figure C.6: *Class 1042*: [INTJ, SQ, (MD_VB), null] — **(NN_NN)** — [null, VP]. *Total number of phrases = 7.*

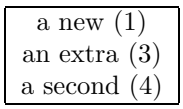


Figure C.7: *Class 1090*: [(PRP_VB), (MD_VB)] — **(DT_JJ)** — [(NN_NN)]. *Total number of phrases = 8.*

Appendix D

Gaussian Intersection: Derivation

Given two Gaussian distributions that intersect with one another (see Figure 5.10 for an example), the goal is to find the sample value x where the distributions overlap. The likelihood of a point y at a certain position on the surface of a Gaussian distribution can be given by:

$$y = \frac{1}{(2\pi\sigma^2)^{\frac{1}{2}}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2} \tag{D.1}$$

where, x is the value of the sample in the distribution that gives y, mu is the mean, and σ^2 is the variance.

The point at which two Gaussian distributions cross can be defined by:

$$\frac{1}{\sqrt{2\pi}\sigma_1} e^{-\frac{1}{2\sigma_1^2}(x-\mu_1)^2} = \frac{1}{\sqrt{2\pi}\sigma_2} e^{-\frac{1}{2\sigma_2^2}(x-\mu_2)^2} \tag{D.2}$$

Take natural log of both sides of Equation (D.2) to remove e:

$$\ln\left(\frac{1}{\sqrt{2\pi}\sigma_1}\right) - \left(\frac{1}{2\sigma_1^2}(x-\mu_1)^2\right) = \ln\left(\frac{1}{\sqrt{2\pi}\sigma_2}\right) - \left(\frac{1}{2\sigma_2^2}(x-\mu_2)^2\right) \quad \text{(D.3)}$$

Rearrange Equation (D.3) to equal zero:

$$\ln\left(\frac{1}{\sqrt{2\pi}\sigma_1}\right) - \ln\left(\frac{1}{\sqrt{2\pi}\sigma_2}\right) - \left(\frac{1}{2\sigma_1^2}(x-\mu_1)^2\right) - \left(\frac{1}{2\sigma_2^2}(x-\mu_2)^2\right) = 0 \quad \text{(D.4)}$$

Replacing $\ln\left(\frac{1}{\sqrt{2\pi}\sigma_1}\right) - \ln\left(\frac{1}{\sqrt{2\pi}\sigma_2}\right)$ with K and expanding gives:

$$K - \left(\frac{1}{2\sigma_1^2}(x^2 - 2\mu_1 x + \mu_1^2)\right) + \left(\frac{1}{2\sigma_2^2}(x^2 - 2\mu_2 x + \mu_2^2)\right) = 0 \quad \text{(D.5)}$$

Multiplying within two main brackets:

$$K - \left(\frac{x^2}{2\sigma_1^2} - \frac{2\mu_1 x}{2\sigma_1^2} + \frac{\mu_1^2}{2\sigma_1^2}\right) + \left(\frac{x^2}{2\sigma_2^2} - \frac{2\mu_2 x}{2\sigma_2^2} + \frac{\mu_2^2}{2\sigma_2^2}\right) = 0 \quad \text{(D.6)}$$

Expand and group terms to give:

$$K + \left(-\frac{1}{2\sigma_1^2} + \frac{1}{2\sigma_2^2}\right)x^2 + \left(\frac{\mu_1}{\sigma_1^2} - \frac{\mu_2}{\sigma_2^2}\right)x - \frac{\mu_1^2}{2\sigma_1^2} + \frac{\mu_2^2}{2\sigma_2^2} = 0 \quad \text{(D.7)}$$

Equation (D.7) can be written in quadratic form ($ax^2 + bx + c = 0$):

$$\underbrace{\left(-\frac{1}{2\sigma_1^2} + \frac{1}{2\sigma_2^2}\right)}_{a}x^2 + \underbrace{\left(\frac{\mu_1}{\sigma_1^2} - \frac{\mu_2}{\sigma_2^2}\right)}_{b}x + \underbrace{\left(K - \frac{\mu_1^2}{2\sigma_1^2} + \frac{\mu_2^2}{2\sigma_2^2}\right)}_{c} = 0 \quad \text{(D.8)}$$

where the two roots x_1 and x_2 can be found by substituting Equation (D.8) into the quadratic formula and solving:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{D.9}$$

The final choice for x is taken to be the one which lies between μ_1 and μ_2, the means of the two distributions.

Bibliography

Anderson, J. A. (1995). *An Introduction to Neural Networks*. MIT-Press.

Aradilla, G., Vepa, J., and Bourlard, H. (2005). Improving Speech Recognition Using a Data-Driven Approach. In *Proceedings of Interpseech*, pages 3333–3336.

Arai, K., Wright, J., Riccardi, G., and Gorin, A. (1999). Grammar Fragment acquisition using syntactic and semantic clustering. *Speech Communication*, 27:43–62.

Axelrod, S. and Maison, B. (2004). Combination of Hidden Markov Models with Dynamic Time Warping for Speech Recognition. In *Proceedings of IEEE ICASSP*, pages 173–176.

Bahl, L. R., De Gennaro, S. V., Gopalakrishnan, P. S., and Mercer, R. L. (1993). A Fast Approximate Acoustic Match for Large Vocabulary Speech Recognition. *IEEE Transactions on Speech and Audio Processing*, 1(1):59–67.

Baum, L. E. (1972). An Inequality and Associated Maximization Technique in Statistical Estimation for Probabilistic Functions of Markov Processes. In Shisha, O., editor, *Inequalities III: Proceedings of the 3rd Symposium on Inequalities*, pages 1–8. Academic Press, University of California.

Bies, A., Ferguson, M., Katz, K., and MacIntrye, R. (1995). Bracketing Guidelines for Treebank II Style Penn Treebank Project. Technical report, University of Pennsylvania. ftp://ftp.cis.upenn.edu/pub/treebank/doc/manual/root.ps.gz, 12th October 2009.

Bocchieri, E. L. and Doddington, G. R. (1986). Frame-Specific Statistical Features for Speaker Independent Speech Recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 34:755–764.

Brown, P., deSouza, P., Mercer, R., Pietra, V. D., and Lai, J. (1992). Class-Based n-gram Models of Natural Language. *Computational Linguistics*, 18(4):467–479.

Buhmann, M. D. (2003). *Radial Basis Functions: Theory and Implementations*. Cambridge University Press, Cambridge.

Chamberlain, R. M. and Bridle, J. S. (1983). Zip: A Dynamic Programming Algorithm for Time-Aligning Two Indefinitely Long Utterances. In *Proceedings of ICASSP*, pages 816–819.

Charniak, E. (2000). A Maximum-Entropy-Inspired Parser. In *First Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pages 132–139, Seattle, Washington.

Cox, S. (2002). Speech and Language Processing for a Constrained Speech Translation System. In *Proceedings of ICSLP*, pages 1149–1152.

Cristianini, N. and Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines and other Kernel-based Learning Methods*. Cambridge University Press.

Davis, S. B. and Mermelstein, P. (1980). Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences. *IEEE Transactions on Acoustic Speech and Signal Processing*, 28:357–366.

De Wachter, M. (2007). *Example Based Continuous Speech Recognition*. PhD thesis, Katholieke Universiteit Leuven, Belgium.

De Wachter, M., Demuynck, K., Van Compernolle, D., and Wambacq, P. (2003). Data Driven Example Based Continuous Speech Recognition. In *Proceedings of Eurospeech*, pages 1133–1136.

De Wachter, M., Demuynck, K., Wambacq, P., and Van Compernolle, D. (2004). A Locally Weighted Distance Measure for Example Based Speech Recognition. In *Proceedings of ICASSP*, pages 181–184.

De Wachter, M., Matton, M., Demuynck, K., Wambacq, P., Cools, R., and Van Compernolle, D. (2007). Template-Based Continuous Speech Recognition. *IEEE Transactions on Audio, Speech and Language Processing*, 15(4):1377–1390.

Deligne, S. and Bimbot, F. (1995). Language Modelling by Variable Length Sequences: Theoretical Formulation and Evaluation of Multigrams. In *Proceedings of ICASSP*, pages 169–172.

Deligne, S. and Bimbot, F. (1997a). Inference of Variable-Length Acoustic Units for Continuous Speech Recognition. In *Proceedings of ICASSP*, pages 1731–1734.

Deligne, S. and Bimbot, F. (1997b). Inference of Variable-Length Linguistic and Acoustic Units by Multigrams. *Speech Communication*, 23:223–241.

Deligne, S. and Sagisaka, Y. (1998). Learning a Syntagmatic and Paradigmatic Structure from Language Data with a Bi-Multigram Model. In *Proceedings of COLING-ACL*, pages 300–306.

Deligne, S. and Sagisaka, Y. (2000). Statistical Language Modelling with a Class-Based N-Multigram Model. *Computer Speech and Language*, 14:261–279.

Demange, S. and Van Compernolle, D. (2009a). HEAR: An Hybrid Episodic-Abstract Speech Recogniser. In *Proceedings of Interpseech*, pages 3067–3070.

Demange, S. and Van Compernolle, D. (2009b). Speaker Normalization for Tempalte Based Speech Recognition. In *Proceedings of Interpseech*, pages 560–563.

Duchateau, J., Wigham, M., Demuynck, K., and Van hamme, H. (2006). A Flexible Recogniser Architecture in a Reading Tutor for Children. In *Proceedings of ITRW on Speech Recognition and Intrinsic Variation*, pages 59–64, Toulouse, France.

Duda, R. O., Hart, P. E., and Stork, D. G. (2001). *Pattern Classification*. John Wiley & Sons, Inc., 2nd edition.

Gales, M. and Young, S. (2007). The Application of Hidden Markov Models in Speech Recognition. *Foundations and Trends in Signal Processing*, 1(3):195–304.

Galescu, L. and Allen, J. (2000). Hierarchical Statistical Language Models: Experiments on In-Domain Adaptation. In *Proceedings of ICSLP*, pages 186–189.

Giachin, E. P. (1995). Phrase Bigrams for Continuous Speech Recognition. In *Proceedings of ICASSP*, pages 225–228.

Gillick, L. and Cox, S. J. (1989). Some Statistical Issues in the Comparison of Speech Recognition Algorithms. In *Proceedings of ICASSP*, volume 1, pages 532–535.

Goldinger, S. D. (1996). Words and Voices: Episodic Traces in Spoken Word Identification and Recognition Memory. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 22(5):1166–1183.

Goldinger, S. D. (1998). Echoes of Echoes? An Episodic Theory of Lexical Access. *Psychological Review*, 105(2):251–279.

Good, I. J. (1953). The Population Frequencies of Species and the Estimation of Population Parameters. *Biometrika*, 40(3 and 4):237–264.

Hain, T., Woodland, P. C., Niesler, T. R., and Whittaker, E. W. D. (1999). The 1998 HTK System for Transcription of Conversational Telephone Speech. In *Proceedings of ICASSP*, pages 57–60.

Huang, Q. and Cox, S. J. (2006). Task independent call routing. *Speech Communication*, 48(3–4):374–389.

Itakura, F. (1975). Minimum Prediction Residual Principle Applied to Speech Recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 23:57–72.

Jiang, H., Soong, F., and Lee, C.-H. (2001). A Data Selection Strategy for Utterance Verification in Continuous Speech Recognition. In *Proceedings of Eurospeech*, pages 2573–2576.

Jurafsky, D. and Martin, J. (2009). *Speech and Language Processing*. Prentice-Hall, New Jersey, 2nd edition.

Katz, S. M. (1987). Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recogniser. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-35(3):400–401.

Kneser, R. and Ney, H. (1995). Improved Backing-Off for M-gram Language Modelling. In *Proceedings of ICASSP*, volume 1, pages 181–184.

Lee, L. and Rose, R. (1998). A Frequency Warping Approach to Speaker Normalization. *IEEE Transactions on Speech and Audio Processing*, 6(1):49–60.

Lidstone, G. J. (1920). Note on General Case of the Bayes-Laplace Formula for Inductive or a Posteriori Probabilities. *Transactions of the Faculty of Actuaries*, 8:182–192.

Lin, Q., Lubensky, D., Picheny, M., and Srinivasa Rao, P. (1997). Key-Phrase Spotting using an Integrated Language Model of N-Grams and Finite-State Grammar. In *Proceedings of Eurospeech*, pages 255–258.

Maier, V. and Moore, R. K. (2005). An Investigation into a Simulation of Episodic Memory for Automatic Speech Recognition. In *Proceedings of Interspeech*, pages 1245–1248.

Mitchell, T. (1997). *Machine Learning*, chapter 4. WCB-McGraw-Hill.

Moore, R. K. (2003). A Comparison of the Data Requirements of Automatic Speech Recognition Systems and Human Listeners. In *Proceedings of Eurospeech*, pages 2581–2584.

Myers, C. and Rabiner, L. R. (1981). A Level Building Dynamic Time Warping Algorithm for Connected Word Recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 29(2):284–297.

Myers, C., Rabiner, L. R., and Rosenberg, A. E. (1980). Performance Tradeoffs in Dynamic Time Warping Algorithms for Isolated Word Recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(6):623–635.

Nasr, A., Estève, Y., Béchet, F., Spriet, T., and de Mori, R. (1999). A Language Model Combining N-Grams and Stochastic Finite State Automata. In *Proceedings of Eurospeech*, volume 5, pages 2175–2178.

Ney, H. (1984). The Use of a One-Stage Dynamic Programming Algorithm for Connected Word Recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 32(2):263–271.

Nouwen, R. (2003). Complement Anaphora and Interpretation. *J Semantics*, 20(1):73–113.

Nuance (2008). Dragon NaturallySpeaking 10. Press Release, Burlington, Mass. http://www.nuance.com/news/pressreleases/2008/20080807_dns10.asp.

Ortmanns, S., Eiden, A., Ney, H., and Coenen, N. (1997). Look-Ahead Techniques for Fast Beam Search. In *Proceedings of ICASSP*, volume 3, pages 1783–1786.

Parekh, R. and Honavar, V. (2000). Grammar Inference, Automata Induction, and Language Acquisition. In Dale, Moisle, and Somers, editors, *Handbook of Natural Language Processing*. Marcel Dekker.

Pickering, M. J. and Garrod, S. (2004). Toward a Mechanistic Psychology of Dialogue. *Behavioral and Brain Sciences*, 27:169–226.

Povey, D. and Woodland, P. (1999). Frame Discrimination Training of HMMs for Large Vocabulary Speech Recognition. In *Proceedings of ICASSP*, volume 1, pages 333–336.

Price, P., Fisher, W. M., Bernstein, J., and Pallet, D. S. (1988). The DARPA 1000-Word Resource Management Database for Continuous Speech Recognition. In *Proceedings of ICASSP*, pages 651–654.

Rabiner, L. and Schafer, R. (1978). *Digital Processing of Speech Signals*. Prentice-Hall, New Jersey.

Rabiner, L. R. (1989). A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77(1):257–286.

Rabiner, L. R. and Shchmidt, C. E. (1980). Application of Dynamic Time Warping to Connected Digit Recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(4):377–388.

Riccardi, G., Pieraccini, R., and Bocchieri, E. (1996). Stochastic Automata for Language Modelling. *Computer Speech and Language*, 10:265–293.

Sakoe, H. (1979). Two–Level DP–Matching—A Dynamic Programming–Based Pattern Matching Algorithm for Connected Word Recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 27(6):588–595.

Sakoe, H. and Chiba, S. (1971). A Dynamic Programming Approach to Continuous Speech Recognition. In *Proceedings of International Congress on Acoustics*, paper 20 C–13.

Sakoe, H. and Chiba, S. (1978). Dynamic Programming Algorithm Optimisation for Spoken Word Recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49.

Solsona, R. A., Fosler-Lussier, E., Kuo, H.-K. J., Potamianos, A., and Zitouni, I. (2002). Adaptive Language Models for Spoken Dialogue Systems. In *Proceedings of ICASSP*, pages 37–40.

Stevens, K. N. (2000). *Acoustic Phonetics*. MIT-Press.

Watkins, C. J. and Cox, S. J. (2009). Example-Based Speech Recognition using Formulaic Phrases. In *Proceedings of Interspeech*, pages 3043–3046.

Webb, A. (2002). *Statistical Pattern Recognition*. John Wiley & Sons, Ltd., 2nd edition.

Witten, I. H. and Bell, T. C. (1991). The Zero Frequency Problem: Estimating the Probabilities of Novel Events in Adaptive Text Compression. *IEEE Transactions on Information Theory*, 37(4):1085–1093.

Wray, A. (1999). Formulaic language in learners and native speakers. *Language Teaching*, 32(1):213–231.

Wray, A. (2002). *Formulaic Language and the Lexicon*. Cambridge University Press.

Wray, A., Cox, S., Lincoln, M., and Tryggvason, J. (2004). A formulaic approach to translation at the post office: reading the signs. *Language & Communication*, 24:59–75.

Wray, A. and Perkins, M. (2000). The Functions of Formulaic Language: an Integrated Model. *Language and Communication*, 20:1–28.

Young, S., Evermann, G., Gales, M., Hain, T., Kershaw, D., Liu, X., Moore, G., Odell, J., Ollason, D., Povey, D., Valtchev, V., and Woodland, P. (2009). *The HTK Book (for HTK version 3.4)*. Cambridge University, Cambridge.

Young, S. J., Russell, N. H., and Thornton, J. H. S. (1989). Token Passing: A Simple Conceptual Model for Connected Speech Recognition Systems. Technical Report CUED/F-INFENG/TR38, Cambridge University Engineering Dept.