CORE
Metadata, citation and similar papers at core.ac.uk
Provided by Research Commons@Waikato

http://waikato.researchgateway.ac.nz/

Research Commons at the University of Waikato

Copyright Statement:

The digital copy of this thesis is protected by the Copyright Act 1994 (New Zealand).

The thesis may be consulted by you, provided you comply with the provisions of the Act and the following conditions of use:

- Any use you make of these documents or images must be for research or private study purposes only, and you may not make them available to any other person.
- Authors control the copyright of their thesis. You will recognise the author's right to be identified as the author of the thesis, and due acknowledgement will be made to the author where appropriate.
- You will obtain the author's permission before publishing any material from the thesis.

Improving the performance of Hierarchical Hidden Markov Models on Information Extraction tasks

Lin-Yi Chou

A thesis submitted in fulfilment of the requirements for the degree of
PhD, The University of Waikato, 2006.

Abstract

This thesis presents novel methods for creating and improving hierarchical hidden Markov models. The work centers around transforming a traditional tree structured hierarchical hidden Markov model (HHMM) into an equivalent model that reuses repeated sub-trees. This process temporarily breaks the tree structure constraint in order to leverage the benefits of combining repeated sub-trees. These benefits include lowered cost of testing and an increased accuracy of the final model—thus providing the model with greater performance. The result is called a merged and simplified hierarchical hidden Markov model (MSHHMM).

The thesis goes on to detail four techniques for improving the performance of MSHHMMs when applied to information extraction tasks, in terms of accuracy and computational cost. Briefly, these techniques are: a new formula for calculating the approximate probability of previously unseen events; pattern generalisation to transform observations, thus increasing testing speed and prediction accuracy; restructuring states to focus on state transitions; and an automated flattening technique for reducing the complexity of HHMMs.

The basic model and four improvements are evaluated by applying them to the well-known information extraction tasks of Reference Tagging and Text Chunking. In both tasks, MSHHMMs show consistently good performance across varying sizes of training data. In the case of Reference Tagging, the accuracy of the MSHHMM is comparable to other methods. However, when the volume of training data is limited, MSHHMMs main-

tain high accuracy whereas other methods show a significant decrease. These accuracy gains were achieved without any significant increase in processing time. For the Text Chunking task the accuracy of the MSHHMM was again comparable to other methods. However, the other methods incurred much higher processing delays compared to the MSHHMM. The results of these practical experiments demonstrate the benefits of the new method—increased accuracy, lower computation costs, and better performance.

Acknowledgments

I would like to thank all the people in Digital Library Lab in the Department of Computer Science at the University of Waikato for being supportive in every way, socially and educationally.

Many thanks to my family for their love and support, especially my husband John Thompson who encouraged me throughout these years of study, and members of DL group, in particular Katherine Don, Michael Dewsnip and Chi-Yu Huang.

Most of all I would like to thank my supervisors Ian Witten, Tony Smith and Michael Mayo for their generous support and encouragement. I also would like to thank Alan Holt for his help and thesis writing expertise.

Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

> I find that a great part of the information I have was acquired by looking up something and finding something else on the way.
>
> Franklin P. Adams

This thesis focuses on improving the efficiency and accuracy of hierarchical hidden Markov models (HHMMs) on natural language processing and other text mining tasks. With the use of a hierarchical model structure, the thesis demonstrates that HHMMs closely reflect the underlying structure of the text being processed. In addition to extracting the meaning of individual events, this structure provides other benefits, such as the ability to merge repeated parts of the model and the ability to process sequences of observations that contain unseen events. HHMMs can be developed to expose the hidden structure in a wide range of real-world sequences, such as speech recognition [Chien, 1999], DNA sequences [Hu et al., 2000], handwriting [Fine et al., 1998], robot navigation [Theocharous et al., 2001] and video structure discovery [Xie et al., 2002; Bui et al., 2004].

1.1 Motivation

Automated text mining has increased in importance due to the rapid growth of online documents. The ability to obtain useful knowledge from unstructured text is now a key technique for handling and organising textual data. Each document is processed to find

entities and relationships that are likely to be meaningful within a particular context, and the extracted information can provide more distinct and specific data for text mining processes. The main types of tasks in text mining are *Information Extraction* (IE) [Eikvil, 1999] and *Information Retrieval* (IR) [Greengrass, 2001].

Information Extraction is the process of identifying information within natural language text based on predefined knowledge [Eikvil, 1999]. The knowledge can be predefined using training data to which structured information has previously been added, and is particularly useful for locating specific information from a natural language document. Information Retrieval is the process of determining which documents from a collection are related to a users query based on predefined categories [Greengrass, 2001]. The IR model is trained using a set of sample data, where each training document is hand labelled with its own categories.

The key difference between IE and IR tasks is that the former extracts relevant facts out of documents for users to analyse, whereas the latter returns relevant documents for users to analyse. A trained IE system can extract pre-specified types of entities and relationships from new texts and store this information into a structured database record. When the data has pre-specified entities it can also undergo automatic analysis, providing a means for further interpretation of these patterns.

This research is based on using several variations of hierarchical hidden Markov model (HHMM)—before and after the application of improvement techniques—to perform IE tasks, in particular:

- reference tagging
- text chunking.

The results from these tasks are evaluated against the equivalent hidden Markov model (HMM) and the regular hierarchical hidden Markov model (RHHMM) [Fine et al., 1998]. The performance of the model can be measured in terms of efficiency—volume of training

data needed and processing time—and also in terms of accuracy—stability and precision of results.

1.1.1 Reference Tagging

It is common for researchers to publish their papers and other research materials online. To locate these, and other similar materials, one generally uses a search engine like Google Scholar[1], or perhaps a more specialized engine like CiteSeer[2]. The drawback of these sources is that they often provide an excessive number of documents—indeed a searcher may spend a considerable amount of time filtering out documents that have no relevance. A searcher may eventually find the documents desired—often also discovering that the documents themselves cite, or refer to, further relevant documents. Caplan [2001] defines this reference or citation relationship as "the ability to go directly from a citation to the work cited, or to additional information about the cited work". Given this situation, it would seem beneficial to present as many forms of connection between similar materials as possible, both to aid in general reference searching—such as grouping research materials together by author's name—and to ensure greater relevance of the materials that are linked to. The greater relevance is due to the fact that the writers of the material will already have gone to some lengths to find only matching or closely related work. Moreover, these references (assuming they have reviewed before being published) have already been checked and moderated by several expert referees. Thus the growth in online information availability needs to be matched by systems that create these links between documents, preferably in an automated fashion. There are, however, difficulties in doing so. For instance, the links may appear in various different contexts, like citations in published research versus references from a bibliography.

Figure 1.1 [Slomin et al., 2002] is an example of a reference section as it appears in a scientific paper. The information allows the system to extract relevant *linking information* [Bergmark, 2000] within online documents. Such relationships will enable readers to gain information on related articles.

[1]Google Scholar/scholar.google.com

[2]CiteSeer.IST Scientific Literature Digital Library, http://citeseer.ist.psu.edu/

tently comparable or superior to the other algorithms. Specifically, while using the minimally pruned model, the micro-averaged precision and recall of the *DVMM* are 95% and 87%, respectively. This implies a break-even performance of at least 87% (probably higher). We therefore compared these results with the break-even performance reported by Dumais et al. [11] for the same task. In that work the authors compared five different classification algorithms: *FindSim* (a variant of Rocchio's method), *Naive Bayes*, *Bayes nets*, *Decision Trees* and *SVM*. The (weighted) averaged performance of the first four were 74.3%, 84.8% 86.2% and 88.6%, respectively. The *DVMM* is thus superior or comparable to all these four. The only algorithm which outperforms the *DVMM* was the *SVM* with averaged performance of 92%.

We see these results as especially encouraging, as all of the above algorithms were used with the *words* representation, while the *DVMM* was using the low level character representation.

6. Discussion and Future Work

The main contribution of this work is in describing a well defined framework for learning variable memory Markov models in the context of discriminative analysis. [14] The *DVMM* algorithm enables to extract features with variable length dependencies which are highly discriminative with respect to the statistical sources at hand. These features are kept while other, possibly numerous features common to all classes, are shed. They may also gain us additional insights into the nature of the given data.

The algorithm is efficient and could be applied to any kind of data (which exhibits the Markov property), as long as a reasonable definition of (or quantization to) a basic alphabet can be derived. The method is especially appealing where no natural definition of higher level features exists, and in classification tasks where the different categories share a lot of structure (which generative models will capture, in vain).

Several important directions are left for future work. On the empirical side, more extensive experiments are required. For the protein data, a thorough analysis of the top discriminating features and their possible biological function is appealing.

On the theoretical aspect, a formal analysis of the algorithm is missing. It may even be possible to extend the theoretical results presented in [16], in the context of discriminative *VMM* models.

[14] For a related approach to discrimination, using competitive learning of generative PST's see [6].

Acknowledgments

Useful discussion with Y. Bilu, R. Bachrach, E. Schneidman and E. Shamir is greatly appreciated. The authors would also like to thank A. Stolcke for help in using the *SRILM* toolkit.

References

[1] H. Almuallim and T.G. Dietterich. Learning with many irrelevant features. In Proc. 9th Nat. Conf. on AI (AAAI), 1991.

[2] T.K. Attwood et al. PRINTS and PRINTS-S shed light on protein ancestry. Nucleic Acids Res., Vol. 30, No. 1, 2002.

[3] L. R. Bahl, P. F. Brown, P. V. de Souza and R. L. Mercer. Maximum Mutual Information Estimation of Hidden Markov Model Parameters for Speech Recognition. in *Proc. of IEEE Int'l Conf. Acous., Speech & Sig. Proces. (ICASSP)*, 1986.

[4] R. Battiti. Using Mutual Information for Selecting Features in Supervised Neural Net Learning. *IEEE Transactions on Neural Networks*, 5(4):537–550, 1994.

[5] G. Bejerano and G. Yona. Variations on probabilistic suffix trees: statistical modeling and prediction of protein families. Bioinformatics, 17(1):23–43, 2001.

[6] G. Bejerano, Y. Seldin, H. Margalit, and N. Tishby. Markovian domain fingerprinting: statistical segmentation of protein sequences. Bioinformatics, 17(10):927-934, 2001.

[7] E. B. Baum and D. Haussler. What Size Net Gives Valid Generalization ? *Neural Computation*, 1(1):151–160, 1989.

[8] T.M. Cover and J.A. Thomas. Elements of Information Theory. John Wiley & Sons, New York, 1991.

[9] S. F. Chen and J. T. Goodman. An Empirical Study of Smoothing Techniques for Language Modeling. Technical Report: TR-10-98, Harvard University, 1998

[10] S. Della Pietra, V. Della Pietra, and J. Lafferty. Inducing Features of Random Fields. IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 19, No. 4, 1997.

[11] S. Dumais, J, Platt, D. Heckerman, and M. Sahami. Inductive learning algorithms and representations for text categorization. In *Proc. Int. Conf. on Inform. and Knowledge Manage.*, 1998.

[12] R. M. Goodman and P Smyth. Decision Tree Design from Communication Theory Stand Point. *IEEE Trans. on IT*, 34(5):979–994, 1988.

[13] K. Lang. Learning to filter netnews. In *Proc. of ICML, 1995.*

[14] A. K. McCallum. Reinforcement Learning with Selective Perception and Hidden States. Phd. Thesis, 1996.

[15] F. Pereira, Y. Singer and N. Tishby. Beyond Word $N-$grams. In Natural Language Processing Using Very Large Corpora. K. Church et al. (Eds.), Kluwer Academic Publishers.

[16] D. Ron, Y. Singer, and N. Tishby. The Power of Amnesia: Learning Probabilistic Automata with Variable Memory Length. Machine Learning, 25:237–262 (1997)

[17] A. Stolcke. Entropy-based Pruning of Backoff Language Models. In *Proc. of DARPA Broadcast News Transcription and Understanding Workshop* pp. 270-274, 1998.

[18] P. C. Woodland and D. Povey. Large Scale Discriminative Training for Speech Recognition. in *Proc. of Int. Workshop on Automatic Speech Recognition (ASR)*, 2000.

[19] Y. Yang A study on thresholding strategies for text categorization. Proc. of the 24th Int. ACM SIGIR, 2001.

Figure 1.1: Example of a reference section

1.1.2 Text Chunking

Text chunking [Abney, 1991] is an important component of information extraction, as well as many other natural language processing systems. It involves grouping information into chunks, and is often a preliminary step for full parsing. For example, consider a sentence

from a CoNLL-2000[3] corpus:

> He reckons the current account deficit will narrow to only $1.8 billion in September.

The output from the chunking process would be:

> (NP He) (VP reckons) (NP the current account deficit) (VP will narrow) (PP to) (NP only $1.8 billion) (PP in) (NP September) (O .)

The sentence has been divided into part-of-speech chunks of different types, and the chunk information is identified by symbols of: *NP* (noun phrase), *VP* (verb phrase), *PP* (prepositional phrase) and *O* (null complementiser).

The chunking process includes identifying the non-recursive portions of phrases, such as noun phrases or verb phrases. It can also be useful for other purposes, including term indexing. For example, noun phrase chunking has proven to be favourable for web searching, where the topics or the focus of a text document can be represented by a list of noun phrases [Chen and Chen, 1994]. Collecting the noun phrases in a text document can provide a better understanding of the text. There is growing interest in Question Answering (QA) systems [Saggion et al., 2004], which analyse a question to locate noun phrases and main verbs, and return documents that match these features. START [Katz, 1997] is one of the first QA systems to use natural language processing (NLP) to match questions, where documents are segmented into chunks and phrases that are then used to describe the content of the data.

The outcome of applying the techniques investigated in this research to the reference tagging and text chunking tasks are explored in Chapter 5. The performance of several models, namely the HMM, RHHMM (regular HHMM by Fine [Fine et al., 1998]), SHHMM (simplified HHMM) and the MSHHMM (merged and simplified HHMM) are measured during this exploration.

[3] A workshop in conjunction with ICGI-2000 and LLL-2000 at the Instituto Superior Técnico in Lisbon, Portugal on September 13 and 14, 2000, *http://www.cnts.ua.ac.be/conll2000/*

1.2 Research Overview

This section introduces the main concepts of HMMs and HHMMs in a non-mathematical way, and discusses existing research related to the application of these methods to information extraction tasks. A formal discussion of these two models is given in Chapters 3 and 4 respectively.

A text document can be structured in many different ways. For instance, it can be divided into sections. The specific structure addressed by this thesis occurs in documents with hierarchical nature, such as those found in natural language documents. A good example, pertinent to the reference linking mentioned above, is the reference section at the end of papers or journal articles. The task of finding structure in this section requires the application of a hierarchical model to extract some information, where the higher levels of hierarchy contain sectioning information, and the lower levels express the model of section content. The models covered in this thesis focus on identifying the syntactic structure of the reference section, often referred to as the grammar, rather than attempting to gain some understanding of the semantics of the words involved.

A hierarchical model would seem well suited to natural language tasks because it identifies and makes use of repeated structure in different parts of the grammar, and in doing so, should outperform a standard non-hierarchical model. Similar to a tree structure, the hierarchical model consists of parent states (non-leaf nodes) and child states (leaf nodes). In order to realise the model's structural potential, it is likely to be beneficial to model the grammar in a hierarchical manner.

HMMs are often used when predicting the most likely state sequence for a given sequence of observations, where the states may contain unseen events. The Markov model provides transition information between states. When the sequence contains unseen events, the Markov model can predict these events by knowing the state before and after them.

An HHMM is a special kind of HMM that is designed to model tasks hierarchically. Any

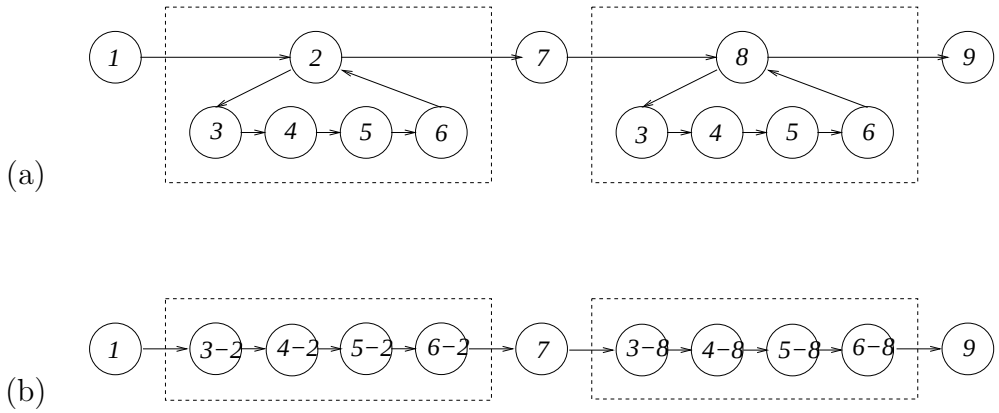


Figure 1.2: Example of HHMM and HMM

HHMM can be converted to a HMM by creating a state for every possible observation, a process is often referred to as "flattening". Figure 1.2 demonstrates the flattening process. Figure 1.2(a) is the HHMM version and Figure 1.2 is the HMM version. After this process there is no difference between parent states and child states (Chapter 4 elaborates on this process).

This thesis proposes that, because of the fundamental difference in structure between HMMs and HHMMs, an HHMM should be capable of outperforming the non-hierarchical version at mark-up tasks such as reference tagging, text chunking or clause identification [Moline and Pla, 2001]. In general, a well-trained model requires a substantial amount of training data before it can deliver acceptable levels of accuracy. Hierarchical hidden Markov models offer the potential to overcome this problem by exploiting repeated structures in the training data. Specifically, building and re-using sub-models for repeated structure better utilises the training data, which allows HHMMs to converge on more accurate predictions more quickly than conventional HMMs can.

This thesis explains the shortcomings of existing methods, and explains what is needed to make HHMMs work correctly. It proposes an efficient probabilistic estimation method to calculate the transition distribution between different levels of hierarchical structure, along with several new approaches to their construction and use, namely merging and simplifying. These techniques are applied to a variant of the HHMM called the merged and

simplified hierarchical hidden Markov model (MSHHMM). Its performance is evaluated in terms of accuracy and processing time efficiency against RHHMMs (regular HHMMs), SHHMMs (HHMMs that have only had the simplification technique applied) and regular HMMs. The RHHMM is constructed in a fashion similar to that laid out by Fine [1998], while the HMMs were developed in accordance with the work of Rabiner [1986]. Figure 1.3 displays the taxonomy of HHMMs evaluated during this thesis, in order to illustrate the relationship between the different variations of the model.

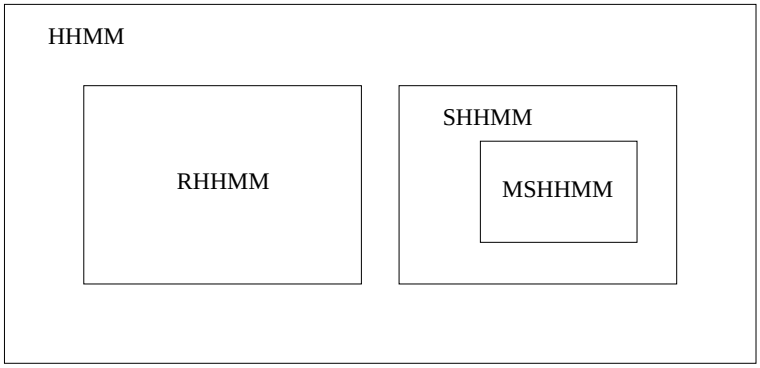


Figure 1.3: Types of hierarchical hidden Markov model

During the training process, the system takes input from user-defined data to construct the model. Optimising the transition between state and observation matrix within each state for better extraction accuracy can be a difficult task. This thesis presents four new techniques to improve the performance of the model: *smoothing*, *pattern generalisation*, *structure formation* and *partial flattening*. These are investigated based on building efficiency and prediction accuracy (discussed in Chapter 6). To validate the main proposal of the thesis, the results of applying each of the models to several different natural language tasks are compared in terms of the performance cost of applying the model to the test data, and in terms of accuracy of the final model in processing unseen data.

1.3 Thesis Statement

This thesis investigates the use of the merging and simplification techniques for improving the performance of hierarchical hidden Markov models (MSHHMMs). It proposes that since MSHHMMs can make use of repeated sub-state sequences within the model, thus increasing the number of observations within each state, they should produce more efficient and accurate results than regular HHMMs (RHHMMs). It will be demonstrated that the MSHHMM is a more accurate model than the HMM for tasks that are based upon information of a hierarchical nature. Furthermore, by taking advantage of repeated sub-models, the MSHHMM shall be shown to require less processing time than either the RHHMM or the HMM.

1.4 Contributions

The thesis makes contributions in the form of new techniques for improving hierarchical hidden Markov models, and applying these techniques to a range of sequences.

- A novel *Merging*, applied during training, that combines the information in sub-models that represent repeated structure in the input data, thus increasing observation counts and accuracy.

- A novel *Simplification*, applied during testing, that calculates a summary of the sub-model elements of a particular internal state, simplifying them into three transformed states: *state-in*, *state-stay* and *state-out*. This decreases the processing time required.

- A novel *Smoothing* that increases the extraction accuracy. The technique involves adjusting the observation probability distributions for states in the model to account for unseen events.

- A novel *Pattern Generalisation* that reduces the amount of training data required for a reliable model.

- A novel *Structure Formation* that reduces the dependency of observation symbols

within each state. This process involves splitting states that contain different significant features.

- A novel *Partial Flattening* that reduces the complexity of the model. This process involves the application of a phrase extraction algorithm followed by observation and state dependency calculation.

1.5 Thesis Structure

Chapter 2 begins with a brief review of existing research relating to HHMMs. The chapter is divided into five key areas: hidden Markov models, hierarchical hidden Markov models, Stochastic Context-Free Grammars, Dynamic Bayesian Networks, and other related techniques.

Chapter 3 describes the details of the HMM. The first section introduces its elements. Section 3.2 explains the basic algorithms, and the problems associated with HMM use: evaluation, alignment and training. All three problems are studied, along with algorithms to solve them. A discussion of the limitations of the model is provided in Section 3.4. The discussion also addresses difficulties inherent in source data with certain characteristics, such as large numbers of states.

Chapter 4 describes HHMMs, including the method for training and testing the MSHHMM used during this research and information on how its hierarchical structure is created. The chapter begins with an introduction to the model and then describes its standard notation in Section 4.2. Section 4.3 introduces the elements of the sub-model and details an efficient way of reforming the transition matrix for the sub-model. Section 4.4 describes the process of combining sub-states. Section 4.5 discusses the structural issues of the HHMM, and Section 4.6 applies the HHMM to extract information from unstructured data.

Chapter 5 applies MSHHMMs to various natural language tasks. Section 5.2 discusses the reference tagging task. Section 5.3 shows how HHMMs can be applied to the text

chunking task, where the sequence of observations is represented by part-of-speech tags. In both sections the results of applying the MSHHMM are compared to the results from equivalent RHHMMs, SHHMMs and HMMs.

Chapter 6 discusses techniques for further refining HHMMs. Section 6.1 introduces a new smoothing technique, called *C-smoothing*. The approach is based on estimating the probability distribution of how frequently low count states can be expected to encounter an unseen variable. The results are compared against various smoothing techniques, such as Bayesian smoothing using Dirichlet Priors [Berger, 1985], Absolute Discounting [Ney et al., 1994] and Jelinek-Mercer [Jelinek and Mercer, 1980]. Section 6.2 demonstrates a pattern generalisation for input data when applied to the reference tagging task. Section 6.3 illustrates a new technique to improve the HHMMs on reference tagging tasks, called *structure formation*. The process involves splitting a state when it contains different features, thus providing more independent features for each resulting state. Section 6.4 presents a new automatic partial flattening process that uses a term extractor method [Pantel and Lin, 2001]. The partial flattening process can reduce the depth of the hierarchical structure for HHMMs by moving sub-trees from one node to another.

CHAPTER 2

Background

Computing already provides well established automated mechanisms for storing, indexing, searching and retrieving structured data. However the proliferation of other formats of data, such as natural language media, has created great interest in automated systems that would offer accessibility similar to that of structured data. The problem is that, while computers can process the rigid structure of fixed formats, they are far less accurate in understanding unstructured natural language formats—something humans are better at. This gap between human and machine understanding is the focus of several areas of study; for example, Machine Learning and Natural Language Processing are used to assign structure to natural language media.

This thesis focuses on the use of hierarchical hidden Markov models (HHMMs) as a method of automated NLP and text mining. However there are several other interesting approaches that were considered during the development of this model. Each of these approaches has various advantages and disadvantages which are discussed in this chapter. By combining lessons learned from previous research, the HHMM attempts to address the drawbacks of other models and to exploit the structural nature of NLP in order to model the structure in a way that is more robust and offers deeper understanding of the problems.

This chapter gives a brief history of the Markov Chain, the HMM and the HHMM, then discusses two related hierarchical models: Stochastic Context-free Grammars and

Dynamic Bayesian Networks, along with other related techniques. All of these sections describe the theory behind each technique and give examples of some real-world applications.

2.1 Markov Chains

A Markov chain is a sequence of $X_0, X_1, \ldots, X_t$ of random variables, which has the property that the conditional distribution of X_{n+1} is given by:

$$P(X_{n+1} = k | X_0 = h, \ldots, X_n = j) = P(X_{n+1} = k | X_n = j), \tag{2.1}$$

where $h, j, \ldots, k$ are values belonging to the discrete state space [Rabiner, 1989], and the probability distribution of X_{n+1} is only dependent upon the previous state probability distribution X_n. If the state space is finite, the transitional distribution can be represented in matrix form, called the *transition matrix*, where the element in row i and column j is equal to:

$$P(X_{n+1} = i | X_n = j). \tag{2.2}$$

Example: Consider a two-state Markov process with state variables a and b. The transition probabilities can be written in matrix form:

$$P = \begin{array}{c} \\ a \\ b \end{array} \begin{array}{c} \begin{array}{cc} a & b \end{array} \\ \begin{bmatrix} \alpha & 1-\alpha \\ 1-\beta & \beta \end{bmatrix} \end{array} \tag{2.3}$$

where $\alpha, \beta \in [0, 1]$, and the value in cell (i,j) represents the conditional probability of transition from state i at time t to state j at time $t + 1$. For example in Figure 2.1, the probability $1 - \alpha$ represents the transition probability from state a to state b. ■

A Markov Chain is a specialized type of Finite State Automata (FSA) [Carroll and Long, 1989]. An FSA is a computational model that consists of a finite number of states and the transitions between them. When the transitions between the states are controlled by probability distributions, the model is called a Probabilistic Finite State Automata [Ron

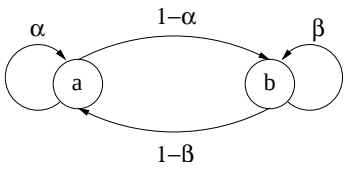


Figure 2.1: Two-state Markov process

et al., 1994]. Probabilistic Finite State Automata have been widely used in many areas of computational linguistics because of their benefits in the process of representing a series of lexical rules.

2.2 Hidden Markov Models

The models discussed in the previous section contain only one observation per state—that is, each state would be associated with a single 'event' in the source data. That said, the question arises about where it is more useful and efficient to construct a model where each state contains several different observations.

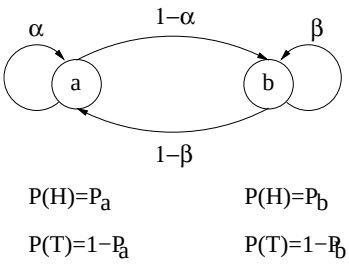


Figure 2.2: Two-coins hidden Markov model

Consider a simple coin tossing experiment on a two-state Markov process as in Figure 2.1, where state a represents person performing the coin tossing experiment and state b represents a second person performing the coin tossing. Figure 2.2 represents the two-state model with the observation symbols being heads (H) or tails (T). The transition represents the probabilities of changing the person throwing the coin. Each state may output either heads or tails based on the probability distribution (P_a or P_b) for each state. When

the model transitions into one of these states, the state will match against one of its observations according to some probabilistic function—a function based upon the state itself. Such a model makes more efficient use of repeated structure in the source data, rather than becoming distracted by the individual 'events'. However as a consequence of constructing a model in this fashion, observations are no longer directly observable from the model but, instead, can only be observed by making a transition through the set of states. This, in turn, will produce the sequence of observations. For example, in the coin tossing experiment the person throwing the coin is known (i.e. the state) but the outcome of a coin throw (the observations head and tails) are hidden. A model of this form is called a *hidden Markov model* (HMM), named so because of the 'hidden' nature of its observations.

HMMs are often used to address sequential problems [Chang and Martinsek, 2004], where the states must sometimes be matched against unobserved 'events'. The Markov model component provides the transition probabilities between these states. When encountering an unobserved event, the model can predict a matching state by knowing the state prior to and immediately following the event.

The basic theory of HMM was published by Baum et. al. [1966; 1967; 1968; 1970] in the late 1960s. It built upon the standard Markov Model by providing the means for constructing a model for the situation where only the sequence of observations is visible, with the events themselves being hidden. However the theory, published in mathematical journals, was not easily understood and did not attract the attention of the speech recognition engineers—who would ultimately make use of its modelling power—until several years later. Indeed, it was the later efforts of Rabiner [1986; 1989] and others, who developed tutorials to explain the theory of hidden Markov models and illustrate how they could be applied in speech recognition, that generated for HMMs a huge amount of interest in the computer science world. The tutorials further helped foster the understanding of the theoretical aspects of the various types of HMM, by giving detailed instructions for their implementation. As a result, several research labs, now armed with a better understanding of the basic mathematical theory, began their work using HMMs in speech processing applications and other related research areas.

Now that the foundations of HMM were understood, researchers could begin experimenting with methods of model construction that were either more accurate (correct information extracted) or more efficient (in terms of training and processing time, and in the amount of training data required). McCallum [1999; 2000] and others [Seymore et al., 1999; Freitag and McCallum, 2000] investigated training a model from data using an HMM, where the accuracy of the HMM extractor is affected by how the transition states are selected. The goal of the research was to develop an automated process for selecting the best model structure in order to maximise the extraction accuracy, the hypothesis being that state structure is a major factor in the final model accuracy. The process itself would begin with a simple model and then split into one or more states and evaluate the resulting model in terms of its accuracy, repeating this process until a maximum result is achieve (as determined by a hill-climbing algorithm). Freitag and McCallum [1999] also developed an HMM which made use of *shrinkage*—a statistical technique capable of improving HMM parameter estimation—to address the problem of balancing the need to construct complex models against sparse training data, a problem that this research also addresses.

Bikel [1999] and Borkar [2001] illustrated the use of HMMs to extract names and numeric tags from a free text document. For example, their model extracted tags like *Title*, *Author*, and *Affiliation* from the titles of computer science research papers. Bikel used the HMM to recognise and classify names, dates, times and numerical quantities from English, Spanish and speech inputs. This application of an HMM is repeated in this research, and is called Reference Tagging. Borkar made use of a two-level nesting of HMMs to improve accuracy. The 'lower' models are small and capture the information surrounding a sequence of highly dependant words (such as a name), whereas the 'higher' model captures the overall structure of the observation sequence using the lower models as building blocks. By doing so, Borkar tied together the smaller independent models into one sequence—an idea that is analogous to the structured use of the sub-groupings of states in this research, although the use of sub-groupings is not limited to two levels.

2.2.1 Techniques

In most applications of HMMs, the model parameters are estimated by an Expectation-Maximization (EM) procedure called the *Baum-Welch* algorithm [Baum and Petrie, 1966; Baum and Egon, 1967; Baum and Sell, 1968; Baum et al., 1970]. The method is based on calculating and utilising the maximum-likelihood estimation for the model parameters until those parameters have been locally maximised. A model trained in such a way can then be used to find the most likely hidden state sequence.

Given an observation sequence $O = \{o_1, o_2, \ldots, o_t\}$, the HMM system must identify the best sequence of states $S = \{s_1, s_2, \ldots, s_t\}$ that would produce such a sequence. There are many different ways of determining the best sequence of states. The most common algorithm is the *Viterbi* algorithm [Viterbi, 1967], which uses a set of model parameters to generate the most likely probabilistic model and state sequence. The Viterbi algorithm involves finding the highest likelihood path among those states searched. Rabiner [1986] has shown that this is an effective process for general language processing tasks.

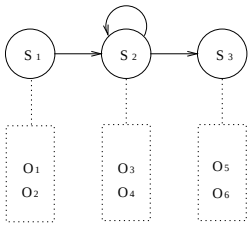


Figure 2.3: Example of a standard HMM

Figure 2.3 shows a simple HMM with three states $\{S_1, S_2, S_3\}$, with observation symbols inside the rectangles, where state S_1 contains $\{O_1, O_2\}$ as observations. The intention of this model is to correctly 'fit' a sequence of values, or observations. Each state of the sequence contains an observation, and thus the sequence is really just some path transition through one or more states. For example, a sequence of observations with its corresponding sequence of states for the model in Figure 2.3 might be:

$$O = \{o_1, o_3, o_4, o_5\} \rightarrow S = \{s_1, s_2, s_2, s_3\}$$

where there is a one-to-one correspondence between the observations and states. A more detailed description of the model will be presented in Chapter 3.

2.2.2 Application

HMMs are useful when considering a part-of-speech tagging task. A model can be constructed by associating each word with a tag, such as assigning "apples" a *Noun* tag, where each tag will then become a state in the finished model. Each state will have as its observations a set of words whose likelihood of being matched is controlled by a probability distribution. The model learns this distribution from the relative number of occurrences of each word in the training data. In order to label a new sentence with part-of-speech tags, the sentence is treated as a sequence of observations, one per word, against which the model attempts to match the most likely state sequence. It does so by using the Viterbi algorithm as mentioned previously. Each state in the resulting state sequence represents the part-of-speech tag that will be associated with the word from the sentence.

Figure 2.4 (from McCallum talk [McCallum, 2004]) shows a list of part-of-speech tags with their corresponding tag symbols and some examples of words. Consider an example of part-of-speech ambiguities as shown in Figure 2.5. The figure shows that the word "flies" can be tagged as either *VBP* or *NN*. Likewise the word "like" can be tagged as either *JJR* or *VBP*. The sequential nature of the HMM allows it to resolve such ambiguities by identifying the context of the word in terms of its position in the sequence of part-of-speech tags. It does so by first deciding upon a state, taking into account the previous and following states. In sentence (a), the word "flies" is tagged as *VBP* in the proximity of the previous state *NN* and the following state *JJR*, whereas, in sentence (b), it is tagged as *NN* in the proximity of the previous state *NN* and the following state *VBP*.

Part-of-speech	TAG	Examples
Adjective	JJ	cold, kind
Adjective, comparative	JJR	less, more
Adjective, cardinal number	CD	2, eleven
Adverb	RB	sideways, differently
Conjunction, coordination	CC	for, and, nor, but
Conjunction, subordinating	IN	after, although, if
Determiner	DT	a, an, the
Determiner, post-determiner	JJ	all, few, many
Noun	NN	engine, corpus
Noun, plural	NNS	men, apples
Noun, proper, singular	NNP	Hamilton, John
Noun, proper, plural	NNPS	Australians, gods
Pronoun, personal	PRP	you, we, she, it
Pronoun, question	WP	who, whoever
Verb, base present form	VBP	walk, jump
Verb, the -*s* form of the verb	VBZ	is, 'is

Figure 2.4: Example of part-of-speech tags

sentence (a):	time	flies	like	a	banana
tags	NN	VBP	JJR	DT	NN
sentence (b):	fruit	flies	like	a	banana
tags	NN	NN	VBP	DT	NN

Figure 2.5: Example of part-of-speech ambiguities

2.3 Hierarchical Hidden Markov Models

An HHMM is a structured multi-level stochastic process that can be visualised as a tree structured variant of the HMM. There are two types of hidden states: production states (a child state or leaf node of the tree structure); and internal states (a parent state or node

that contains the leaf node), which contain production states or other internal states. The difference between a standard HMM and a hierarchical HMM is that individual states in the hierarchical model can contain sequences of nested states or observations, whereas each state in the standard HMM can contain only linear sequences of observations, due to its linear nature.

HHMMs were first proposed by Fine [1998] to resolve the complex multi-scale structures that pervade natural language, such as speech [Rabiner and Juang, 1986], handwriting [Nag et al., 1985], and text. The main idea is to allow HHMMs to correlate structures that are arbitrarily far apart and to handle the statistical inhomogeneities for different sub-models. The sub-models of HHMMs can be used to spot frequent letter combinations, punctuation and the ending of phrases in natural English text. For example, the states at different levels can be seen as expert. The production states produce high probabilities on short frequent strings, such as: *ing th wh ou*. Internal states produce frequent words and phrases, such as 'will', 'such', 'not', and the root state produces a sentence. Based on Fines' work, Skounakis [2003] describes the HHMM as multiple "levels" of HMM states, where the lower levels represent each individual output symbol, and upper levels represent the combinations of lower level sequences. Skounakis [2003] also introduced a novel modification of HHMMs, where phrases and states must have matching types, and that phrase states must contain complete phrases. The observations can be treated as feature vectors, which lead to the observation and grammatical knowledge cooperating within IE models.

The original HHMM by Fine [1998] is restricted to a tree structure that does not allow the sharing of common sub-structures in the model. Bui [2004] presents a general HHMM in which the state hierarchy can be a lattice allowing the arbitrary sharing of a sub-structure at the lower levels of the model; for example, the models for hand-written scripts where different sub-models of word level states should be able to share the same sub-models of letter level states. For the alignment aspect, the inside-outside algorithm [Lari and Young, 1990] is used to determine the beginning and the end for each of sub-model. The inside-outside algorithm is similar to a technique used in probabilistic context-free gram-

mar, where the outside observations are grouped into one.

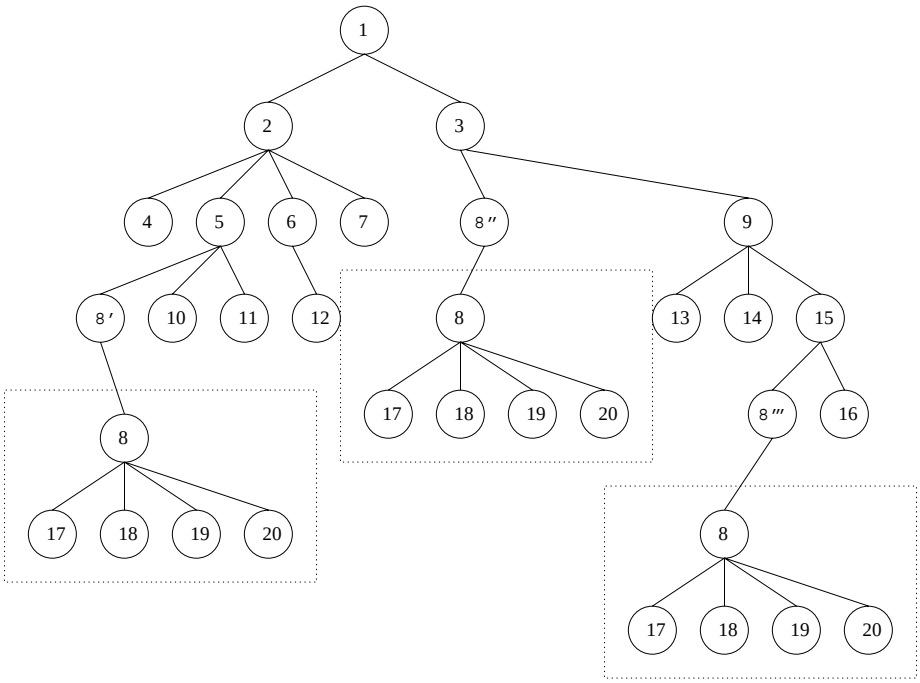


Figure 2.6: Example of an HHMM

Using a hierarchical structure has the advantageous of being able to merge together repeated parts of the structure. Figure 2.6 shows that state 8 is repeated at three different positions in the HHMM, denoted as state $8'$, $8''$ and $8'''$, which are the children of states 5, 3 and 15 respectively. The three states share the common child states 17, 18, 19 and 20. For the HHMM, the system is only required to train state 8 once, instead of states $8'$, $8''$ and $8'''$ separately, resulting in less computational costs in building compared to the HMM. Other advantages of the HHMM over the HMM include a smaller total number of states needing to be identified and the fact that each merged state has an increased amount of training data ('events' from all three positions in the model) leading to better probability estimates, and, therefore the system is capable of more stable predictions.

2.3.1 Techniques

In most applications of HHMMs, the model parameters are estimated by an Expectation-Maximisation (EM) procedure called the *Baum-Welch* (or forward-backward) algorithm

[Baum and Petrie, 1966] (which is the same algorithm that is used in HMMs), and trained models are used to find the most likely hidden state sequence by the Viterbi [Viterbi, 1967] search algorithm. However, due to the multi-level nature of the HHMM, the parameters used in these algorithms are more complicated to estimate. As a consequence the system needs to re-calculate the observational and transitional probabilities for each level. This procedure involves calculating the observation distribution for each level and then reforming the transition matrix by including estimated values for each level.

The probabilistic information for each internal state is determined by a bottom-up algorithm, where lower levels of the hierarchy tree are calculated first to provide information for the upper level states. Once all the internal states have been calculated, the top-level of the hierarchy tree can be used to estimate the probability sequences. This means the model will now become a linear HMM for the final Viterbi search process [Viterbi, 1967]. A more detailed description of this process will be provided in Chapter 4.

2.3.2 Application

In recent years automated reference linking has become widely developed in documents like journal articles, digital libraries, hypertext and web publishing. For example, the Open Journal Project [Hitchcock et al., 1998] provides a web based repository that systematically marks up documents with reference links, thus providing faster access to related information, such as associated journals or other online information. Another example of journal reference linking is the popular dynamic delivery system IngentaConnect[1], which extends the search content to full text, abstract and indexing services. The "Forward citation linking" agent provides a pathway to other articles that cite the article currently being read. Connetea[2] provides a free online reference management service created by the Natural Publishing Group[3]. Connetea allows users to store and share their reference lists online. The Open Citation Project[4] analyses the operational semantics of documents

[1]IngentaConnect, *http://www.ingentaconnect.com/about/publishers/reference_linking_services*
[2]Connotea, http://www.connotea.org/
[3]Nature Publishing Group, http://npg.nature.com/
[4]Open Citation project, http://opcit.eprints.org/

to determine the features for 'optimal linking'. Many digital library communities have adopted the OpenURL [Sompel, 2001] standard as a basis for enabling reference linking, for example the National Information Standards Organization[5] (NISO).

Reference

[1] T. Moloney, A. C. Lea, and C. Kowalchuk. Manufacturing and packaged goods. In G. H. Castle, editors, Profiting from a Geographic Information System. GIS World, Inc., Fort Collins, CO, 1993.

States

author: T. Moloney, A. C. Lea, and C. Kowalchuk.
title: Manufacturing and packaged goods.
editor: In G. H. Castle, editors,
booktitle: Profiting from a Geographic Information System.
publisher: GIS World, Inc.,
address: Fort Collins, CO,
year: 1993.

Figure 2.7: Example of the reference tagging task

Traditionally, the reference linking task is solved using an HMM, where HMM states mark up the reference [Bikel et al., 1999; Miller et al, 1998; Zhou and Su, 2002]. The reserach will show that an HHMM, when applied to this task, is more efficient and accurate because it can make use of repeated structure of sub-groups that are only exposed by a hierarchical model.

[5]National Information Standards Organization, http://www.niso.org/

In Figure 2.7, the reference is segmented into useful fragments as; *author*, *title*, *editor*, *booktitle*, *publisher*, *address* and *date*. Figure 2.8 illustrates a hierarchical relationship between *author*, *editor* and *persons' name*, where both *author* and *editor* can be the parent node of the *person's name*, and the model can also split the *name* into different sub-states, such as *first name* and *last name*. By identifying these properties, the system can make use of the information to perform the Reference Linking tasks.

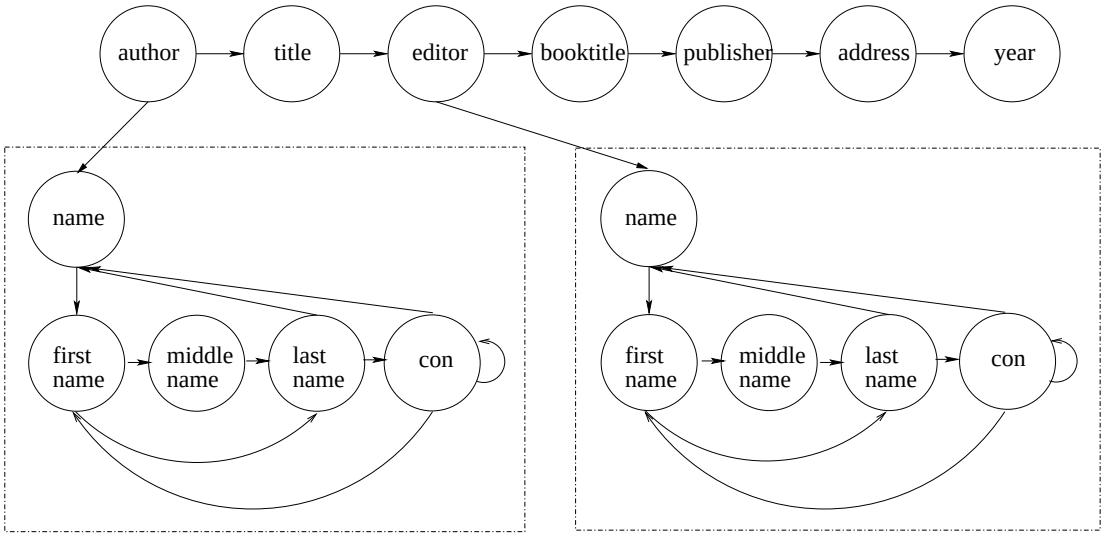


Figure 2.8: Example of a copied model

2.4 Related Work

2.4.1 Stochastic Context-Free Grammar

A stochastic context-free grammar (SCFG) is a variant of an ordinary context-free grammar where each production is assigned a probability. SCFGs express context-free grammars in the same way that HMMs express regular grammars, where regular grammars are the weakest class of grammar in the Chomsky Hierarchy [Chomsky, 1965]. The model is frequently used in tasks such as natural language processing [Stolcke, 1994], speech recognition [Jelinek and Lafferty, 1991] and RNA sequencing [Jurafsky and Martin, 2000].

Rule			Probability
S	$\rightarrow$	AB	1.0
A	$\rightarrow$	a	0.6
A	$\rightarrow$	CS	0.4
B	$\rightarrow$	b	1.0
C	$\rightarrow$	a	1.0

Figure 2.9: SCFG for the language $a^n b^n$

A context free grammar (CFG) is a formal grammar that consists of a set of rules to describe the language grammar, and production rules are represented in the form:

$$\begin{aligned} R_i &\rightarrow s_i \\ R_i &\rightarrow R_j R_k \end{aligned}$$

where R_i represents a non-terminal symbol and s_i represents the terminal symbol. The non-terminal symbol R_i can always be replaced by terminal symbols s_i in context-free form. Figure 2.9 shows an example of a simple SCFG, where each production is assigned with a probability in the second column, and the probability for a non-terminal symbol must sum to one. The SCFG model generates the language of $\{a^n b^n\}$ with $n \geq 1$. The probability of a string ab can be calculated as product of probabilities for rules used. For example,

$$\begin{array}{ccccccc} \underline{a}\ b & \rightarrow & A\ \underline{b} & \rightarrow & \underline{AB} & \rightarrow & S \\ (0.6) & & (1.0) & & (1.0) & & \end{array}$$

where the process starts from the beginning of the string, and recursively replaces the production rules provided in Figure 2.9 until the end of the string. The probability of string ab is equal to $P(ab) = 0.6 \times 1.0 \times 1.0 = 0.6$, and the probability for string $aabb$ is equal to $P(aabb) = 0.24$. The probabilities of all possible strings sum to 1. The probabilities are calculated using the recursive Inside-Outside algorithm [Lari and Young, 1990]. The Inside-Outside algorithm is related to the Forward algorithm and Backward algorithm, and is used to calculate the probability of a sequence based on some SCFG rules.

A particular implementation of the SCFG model makes use of push-down automata, where possible rules are recursively refined as they are encountered on a stack of unbounded size. By doing so, the SCFG accurately uncovers the hidden embedded structure within speech and other natural language data—exactly the ability required to bridge the understanding gap when automatically analysing and processing natural language. A second advantage of SCFGs is that they are robust against ambiguities created when forming the grammar rules. Ambiguity is caused when there is more than one possible parse tree, and thus multiple trees that could match the input data. A SCFG addresses this problem by selecting the highest probability tree, as determined during the recursive process.

However this same ability becomes a disadvantage when a SCFG is applied to data which does not have a complex hidden structure—such as those that have the property of a finite regular grammar. In such a case the SCFG will still use the complex and expensive Inside-Outside recursive algorithm. The algorithm involves a complexity of $\mathrm{O}(N^3)$, whereas a simple modelling language could be used to solve the problem more efficiently.

SCFGs, much like HHMMs, have the potential to identify patterns within natural language data, such as speech. The main uses of the SCFGs are:

- solving sequential problems that contain hierarchical structures.
- identifying the occurrence of sub-sequences.
- predicting combinations of sub-sequences, e.g. the secondary structure in RNA sequence.

There are issues to consider when applying these models to data, problems common to both HHMMs and SCFGs. These problems can be grouped into three distinct categories: evaluation, alignment and training.

1. **Evaluation Problem:** This problem involves estimating the probability $P(O|\lambda)$ of an observed sequence $O = \{O_1, O_2, \ldots, O_T\}$. This procedure allows the system to choose a suitable model (λ) for the observation sequence. For the HHMM, this problem can be solved by the *Forward Algorithm*. For the SCFG model, the problem

is solved by providing an estimation for the set of rules by using the *Inside-Outside* algorithm [Lari and Young, 1990].

2. **Alignment Problem:** This problem involves assigning the most likely state sequence $Q = \{q_1, q_2, \ldots, q_T\}$ for a given observation sequence $O = \{O_1, O_2, \ldots, O_T\}$ using model parameters λ. This procedure identifies the most likely state sequence and estimates the best possible solution. For the HHMMs model, this problem can be solved by the *Viterbi algorithm* described in Chapter 4, where the algorithm computes the most likely state sequence given an observation sequence. For the SCFGs model, this problem is solved by the Cocke-Younger-Kasami (CYK) algorithm [Lari and Young, 1990], where the process proceeds bottom-up by assigning non-terminal symbols to combination of terminal symbols.

3. **Training Problem:** This problem involves finding the most likely set of model parameters λ to maximise $P(O|\lambda)$. For the HHMM, this problem can be solved using the *Baum-Welch algorithm* [Baum and Petrie, 1966], where the procedure reiterates several times to reach to a local maximum set of λ^*. For the SCFG model, the observation sequence is determined by the set of grammar rules, where the rule probabilities can be estimated iteratively using the *Expectation-Maximisation* (EM) procedures to find the local maximum likelihood value for the model parameters.

2.4.1.1 Techniques

To find the best model for a given sentence, the SCFG uses a variant of the *Cocki-Younger-Kasami* algorithm [Lari and Young, 1990] to determine the most likely parse tree (which is formed by both terminal and non-terminal symbols). This algorithm is commonly used to logically fold RNA sequences [Durbin et al., 2001], where the important grouping factor is not the sequence itself but the underlying secondary structure. Thus it can be applied to natural language problems in much the same way to uncover the hidden structure. The Cocki-Younger-Kasami variant is similar to the Viterbi algorithm for HMMs in that it calculates the most likely path between different hidden states. This variant is widely known as the *Inside-Outside* algorithm [Lari and Young, 1990] and is comparable to the forward algorithm [Baum and Egon, 1967] in the HMM. The procedure involves calcu-

lating the probability of a given sequence by summing up the probabilities of the most probable parse trees. The processing complexity of this algorithm is $O(N^3)$, where N is the number of non-terminal symbols.

The application of the Inside-Outside algorithm is again similar to that of the Forward and Backward algorithm in the HMM. The process uses the Expectation-maximisation algorithm to compute the maximum likelihood of the grammar parameters, and each step of the process produces an estimation of the model parameters for the SCFG model. The Inside-Outside algorithm is a computationally expensive process that runs in $O(N^3T^3)$ time, where N is the number of non-terminal symbols and T is the length of the sentence.

2.4.1.2 Application

The following is an example of a SCFG built to address the well known Natural Language task of text chunking. Consider a sentence from CoNLL-2004[6] for text chunking process:

> He__PRP reckons__VBZ the__DT current__JJ account__NN deficit__NN will__MD narrow__VB to__TO only__RB #__# 1.8__CD billion__D in__IN September__NNP .__.

where the part-of-speech tag associated with each word is attached with an underscore. Text parsed is represented by the part-of-speech tags as:

> PRP VBZ DT JJ NN NN MD VB TO RB # CD D IN NNP .

This thesis addresses the problem of modelling the structure of the sentence. Only the part-of-speech tags and grammar information are considered for the extraction tasks. The output of the tagged text is expressed as:

> (S (NP He__PRP) (VP reckons__VBZ) (S (NP the__DT current__JJ account__NN deficit__NN) (VP will__MD narrow_VB) (PP to_TO) (NP only__RB #__# 1.8_CD billion_D) (PP in_IN) (NP September__NNP)) (O .__.)).

[6]The 2004 Conference on Computational Natural Language Learning, Boston, MA, USA, 2004, *http://cnts.uia.ac.be/conll2004*

where the sentence involves the clause information identified by the *S* symbol, and the chunking information is identified by the symbols of *NP* (noun phrase), *VP* (verb phrase), *PP* (prepositional phrase) and *O* (null complementizer). The brackets are in Penn Treebank II style.[7]

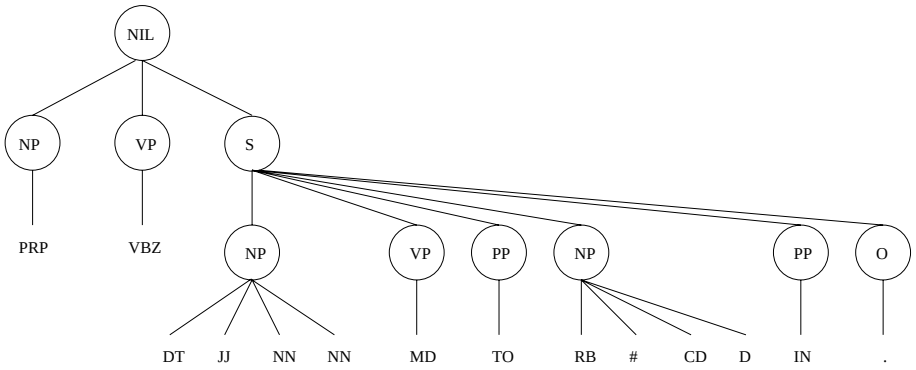


Figure 2.10: Diagram of the parse tree from Penn treebank

Figure 2.10 shows the structure of the sentence in the tree form, where the leaf nodes of the tree are the part-of-speech tags, which are in turn are associated with the words of the sentence, and *NIL* represents the root of the sentence. The sentence contains only the part-of-speech tags and syntactic information. The SCFG production rules for this sentence can be represented in the form:

NIL	→	NP VP S	(1.0)
S	→	NP VP PP NP PP O	(1.0)
NP	→	**PRR**	(0.33)
NP	→	**DT JJ NN NN**	(0.33)
NP	→	**RB # CD D**	(0.33)
VP	→	**VBZ**	(0.5)
VP	→	**MD**	(0.5)
PP	→	**TO**	(0.5)
PP	→	**IN**	(0.5)
O	→	**.**	(1.0)

[7]The Penn Treebank Project, *http://www.cis.upenn.edu/˜ treebank/home.html*

where the items in bold represent the terminal symbols (i.e. part-of-speech tag) in the sentence, and the probability associated with each rule is given in parentheses. The probabilities of a set of productions are calculated by the possible non-terminal rules that are associated to it. For example, a non-terminal symbol *NP* is associated with three rules; *PRR*, *DT JJ NN NN* and *RB # CD D*, and the sum of all probabilities must be equal to one ($\frac{1}{3} + \frac{1}{3} + \frac{1}{3} = 1$).

2.4.2 Dynamic Bayesian Networks

A Bayesian network is a directed acyclic graph that represents the dependence relations between the variables in a probabilistic model. The graph is constructed with nodes and arcs; a node represents a variable that contains an observation and/or probability distribution parameters; an arc represents the dependency relations amongst the variables; and the network is formed by the nodes and arcs between them. The probability distribution for a sequence of variables can be estimated by the joint probability distribution under the Bayesian formalism [Heckerman et al., 1995],

$$p(x_1, \ldots, x_n|\lambda) = \prod_{i=1}^{n-1} p(x_i|\prod\nolimits_i, \lambda) \tag{2.4}$$

where $\prod_i$ represents the set of variables $\{x_1, \ldots, x_{n-1}\}$ and λ represents the model information.

The Dynamic Bayesian Network (DBN) was introduced by Schaefer [1997] as an extension of Bayesian Networks. The network graph of DBN represents a model that deals with sequential time events. Murphy [2002] describes different models that can be represented as DBNs, and how to approximate inference and learn DBN models from sequential events. For example, Murphy considers HHMMs as a special kind of dynamic Bayesian network and derives a simpler inference algorithm. The complexity of such a DBN is linear in time, but exponential in relation to the depth of the HHMM.

The network graph of a DBN represents event flow over time [Ghahramani and Jordan, 1997], where each variable within the model is assigned a time index as an extra parameter. This index parameter allows the probability distribution to be modified depending

on the current time. Note that the model does not change over time; only the probability distribution within the variables changes.

Consider the case of DBN where the set of variable at time t is dependent on only the previous time index's variable $(t-1)$. In this instance, the model does not require 'history', which allows it to be approximated to a Markov model. Figure 2.11 represents this DBN, which is now structured similar to a Markov model, where the model contains states $\{x_1, x_2, x_3, x_4\}$ associated with observations $\{y_1, y_2, y_3, y_4\}$ respectively.

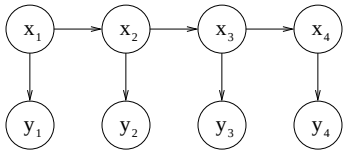


Figure 2.11: Markov process as a Bayesian network

The main difference between a DBN model and a HMM is that of the representation of state and state history. In a HMM the hidden state is assigned a probability distribution based on the variables which have occurred in a single, discrete, state. In contrast, a DBN state represents the hidden state with a set of probability distributions that are dependant on the variables of the previous states. Figure 2.12 is a three level HMM represented as a DBN (as described by Murphy [2002]), where Q_t^d is the state at time t and level d. The F_t^d is the binary indicator variable to identify the status of the current node for time t, where the current node is still at level d or finished level d.

As with all techniques mentioned in this section, there are advantages and disadvantages to the DBN technique. The DBN model makes use of historical parameter information; it can also make use of temporal dependency controls, whereas a HHMM cannot. Consider the case where a certain sequence has a far greater probability of occurring after another sequence—a fact that could be leveraged in the DBN model as the model remembers the previous state. In terms of computational complexity, the DBN model performs relatively well, with its cost being $\mathrm{O}(Q^D T)$, where D is the total number of levels and T is the time

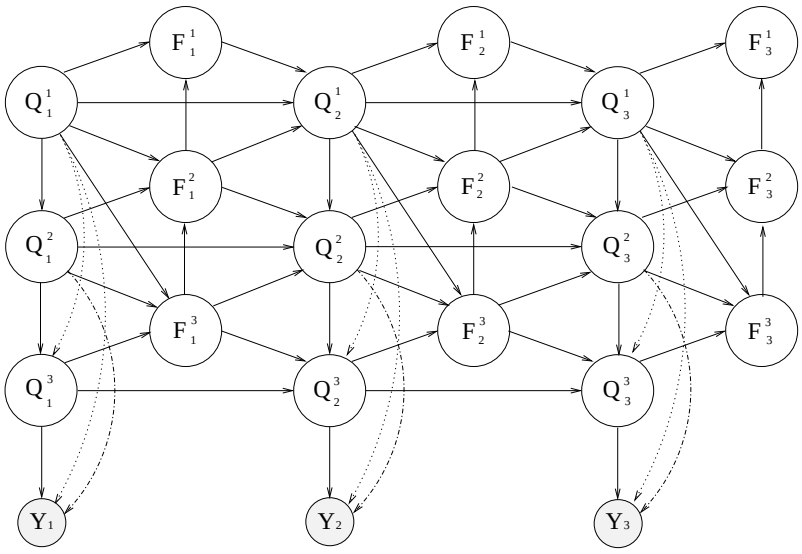


Figure 2.12: Markov process as a Bayesian network

period. Also the structure of the model is easily changed, for example, by removing arcs between states.

The disadvantages of a DBN arise when learning a probabilistic distribution for a large sparse network, which contains a large set of parent variables. The learning process for each individual relation can be computationally expensive.

2.4.2.1 Techniques

Constructing a Bayesian Network model involves learning the structure of the network, and estimating the parameters for each variable within the model. Given a set of training data, the model attempts to learn the distribution of probabilities for transition between variables, and also the probability distribution within each state, which is equivalent to the transition distribution and observational distribution for the hidden Markov model.

Different tasks require different methods to compute the distribution. Cheng [1997] describes an algorithm for learning structure by using the *mutual information* value to calculate the joint probability of adjacent variables. Mutual information is a measurement of the information of event X_i that is shared by event X_j. The formula for mutual

information (I) is given as:

$$I(X_i, X_j) = \sum_{x_i, x_j} P(x_i, x_j) \log \frac{P(x_i, x_j)}{P(x_i)P(x_j)}, \tag{2.5}$$

where $P(x_i)$ represents the probability of variable x_i within the training data, and the higher values of $I(X_i, X_j)$ indicate that more information is shared between these two variables (x_i and x_j).

The probability distribution within states can be estimated by calculating the expected value of each event encountered. The *Dirichlet distribution* is the conjugate prior of the parameters of the multinomial distribution [Gelman et al., 1995]. The Dirichlet distribution is often used to adjust probability distributions for each event, and the algorithm provides the flexibility to set the probability density in each individual variable.

2.4.2.2 Application

A DBN can be used when attempting to solve the Robot Localisation task—a popular topic in AI—which concerns the task of orienting a robot with respect to its surroundings. The problem can be summed up as identifying the location of a robot based on possibly inaccurate sensor data [Thrun, 2002]. There are two aspects of the localization task; estimating the global position and estimating the local position. The global position estimation task involves estimating the position of the robot on a predefined map (e.g., robot navigation [Borenstein et al., 1996; Kortenkamp et al., 1998]). The local position estimation task is where the robot keeps track of its position relative to the starting point over a period of time. Theocharous [2001] uses a DBN to provide a faster learning algorithm than the Hierarchically Partially Observable Markov Decision process (HPOMPDs) for indoor robot navigation, and Murphy [2001] demonstrates the usage of DBN on particle filters, which can provide a good learning algorithm to determine colour—an important observation when attempting to estimate the local position of a robot.

Consider a local position estimation task involving a robot which can only move in one dimension; left or right depending on the control unit, U_t, where the location of the robot can be obtained from a range of data, such as, sonar, stereo camera, or laser range-finders.

When the initial location of the robot is known for a given map, the system can deduce the local position after movement by using a DBN model.

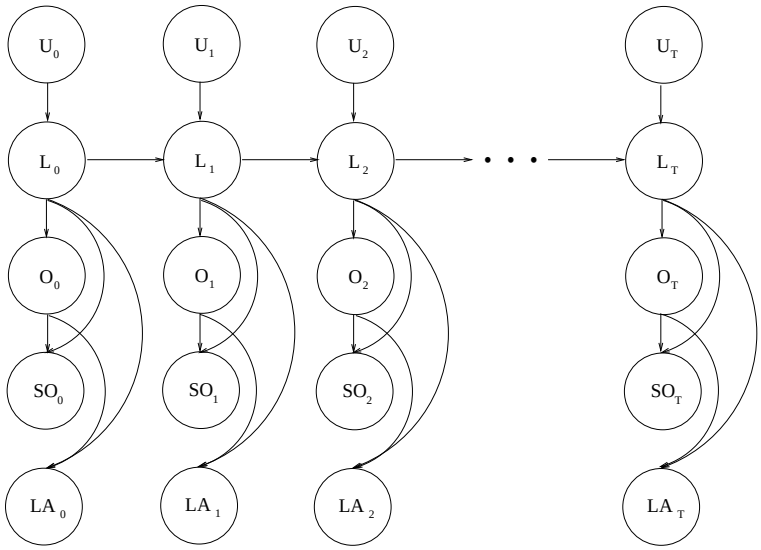


Figure 2.13: Dynamic Bayesian Network for robot mapping task

Figure 2.13 shows the DBN model used in the map learning task, where L_t represents the location at time t, O_t represents the observation, U_t represents the control unit and SO_t, LA_t represent the sources for information about the location of the robot, for instance the sonar sensor and the laser range-finder. In each time step t, the robot collects the data from SO_t and LA_t, where these two sources of data are independent of each other, and then interprets them using a set of information from the observation O_t (as the sensor inputs). The robot gathers information from each input source, and then identifies the location based on the learned DBN model.

2.4.3 Other Related Techniques

A Markov process is governed by the Markov property, which states that the future behaviour of the process given its path depends only on its present state. There are many related variations on the Markov process and this section will briefly outline a few of these. This is done in order to provide the reader with an idea of some of the other techniques for improving or modifying an HHMM.

A *Markov Decision Process* (MDP) [Berteskas and Tsitsiklis, 1989] is a four-tuple with parameters (S, A, P, R), where S represents the state space, A is the action space, P is the state-transition and R represents the reward function. This process is widely applied to planning under uncertainty [Boutilier et al., 1999] and reinforcement learning [Sutton and Barto, 1998]. For example, a robot navigates the environment based upon some form of sensor data and reacts to the environment accordingly. In general, the information that a robot can identify is restricted to observations obtained from the sensor data—observations that may be less than perfect. There is a model called *Partially Observable Markov Decision Process* (POMDP) to deal with this type of problem. When a task involves a large state-space with a more complex structure, a hierarchical model called the *Hierarchically Partially Observable Markov Decision Process* (HPOMDP) [Theocharous and Mahadevan, 2002] can be applied. *Relational Markov Models* (RMM) [Anderson et al., 2002] are a variation of Markov Models that attempt to expose other hidden relationships by clustering together states that represent observations that share some similar feature. These 'related' states can then further be processed by having their parameters clustered together. The resulting model can be effective under large sparse data, and makes good predictions of unseen events.

Freitag [2000] implements a statistical technique called 'shrinkage' into the HMM, which provides a more robust HMM on sparse data, where the data is represented in some hierarchical form with similar sub-states. Raiko [2002] presents the *logical hidden Markov model* (LHMM) to enhance the rules learned from the model and give more specific constraints. *Coupled hidden Markov models* (CHMM) [Brand et al., 1997] consist of modelling N processes for N HMMs, which models the system with multiple interacting processes to suit the needs of systems that have a structure in both time and space. Meanwhile, Ghahramani [1997] generalises an HMM state by factoring in multiple state variables and representing them in a distributed manner. This is called a *factorial hidden Markov model*. The model consists of multiple independent causes or factors by collaborating with a learning architecture called *Cooperative Vector Quantization* (CVQ) by Zemel [1993]. The process generalises mixture models by allowing the mixture components to cooper-

ate in modelling the data set. The *abstract hidden Markov model* (AHMM) [Bui et al., 2002] consists of HMM states, where the state variables depend on a hierarchy of action variables. The structure contains the properties of DBN models and can model in a similar manner. Vogler [1999] presents a novel approach to American Sign Language based on *parallel hidden Markov models* (PaHMMs). It is able to model the parallel processes independently as the process does not require consideration of different combinations at training time. Adibi [2001] uses embedded information about self-similar structure to reduce the complexity of learning and increase the accuracy of the learned model, creating a *self-similar layered hidden Markov model* (SLHMM).

These approaches were considered when developing this research, but their model construction methods were too dissimilar to integrate with the ones implemented.

2.5 Summary

This chapter outlines a brief history of the HHMM and discusses two closely related models: Stochastic Context-free Grammars (SCFG) and Dynamic Bayesian Networks (DBN). The theory behind each of these techniques is discussed along with some real-world applications of those techniques. The chapter has also made mention of several other related hierarchy based models. Table 2.1 represents the summary of four main models in the chapter, along with the algorithm used for each of the three fundamental problems, and the time complexity of testing using the model. The following chapters will discuss both the HHMM and its predecessor, the HMM, in greater depth.

Model	Evaluation	Alignment	Training	Time Complexity
HMM [Rabiner and Juang, 1986]	Forward	Viterbi	Baum-Welch	O(N^2T)
HHMM [Fine et al., 1998]	Foward	Viterbi	Baum-Welch	O(N^3T)
SCFG [Lari and Young, 1990]	Inside	CYK	EM	O(N^3T^3)
DBN [Murphy, 2002]	Bayesian Dirichlet	Forwards-Backwards	EM	O($N^{2D}T$)

Table 2.1: Summary of four main models

CHAPTER 3

Hidden Markov Models

Hidden Markov Models (HMMs) were introduced in the late 1960s, and are widely used as a probabilistic tool for modelling sequences of observations [Rabiner and Juang, 1986]. They have proven to be capable of identifying semantic labels and assigning appropriate tokens over a wide variety of input types. This is useful for text-related tasks that involve some uncertainty, including part-of-speech tagging [Brill, 1995], reference tagging [Seymore et al., 1999], text segmentation [Borkar et al., 2001], event tracking [Theocharous et al., 2001], named entity recognition [Bikel et al., 1999] and information extraction tasks [McCallum et al., 1999; Craven et al., 2000; Eikvil, 1999].

In a regular Markov model, the state transition probabilities are the only parameter since the state is directly observed—in other words each state will output exactly one observation symbol. The "hidden" aspect of the HMM arises from the fact that the state is no longer directly observed because each state does not hold a single observation symbol. Instead each state may output one of several symbols dependent on a second probability distribution. As previously mentioned, transitions from one state to another are represented by a set of *transition probability distributions*. The system starts with an initial state, then transitions to a new state as "predicted" by the transition probabilities. As it enters each state, the system produces an observation symbol from a set of possible observation symbols as in proportion to the *observation probabilities distribution*. Once completed, the model can be applied to a previously unseen sequence of observation sym-

bols in order to calculate the most likely sequence of states required to output such a sequence.

This chapter presents a brief overview of HMMs by first introducing the building blocks or elements of HMMs in Section 3.1. Section 3.2 then describes the algorithms applied to the HMM. These algorithms are separated into three distinct stages: training, alignment and evaluation. A brief discussion of the limitations follows in Section 3.3. Finally, Section 3.4 provides a summary for this chapter.

3.1 The Elements of HMM

The basic building blocks of any Markov model can be represented as a set of discrete states and the transitions between them. These building blocks are often called "elements", and a typical HMM generally consists of five elements.

With respect to the Hidden Markov model process, the model is characterised by the following elements [Rabiner and Juang, 1986]:

- Set of hidden states:

  $$S = \{S_1, S_2, \ldots, S_N\},$$

  where N is the number of states in the model.

- Set of observation symbols per state:

  $$V = \{v_1, v_2, \ldots, v_M\},$$

  where M is the number of distinct observation symbols per state.

- State transition probability distribution:

  $$A = \{a_{i,j}\}, \qquad i = 1, 2, \ldots, N, j = 1, 2, \ldots, N$$

  where $a_{i,j}$ is the probability of transition from state i to j.

- Observation symbol probability distribution:

$$B = \{b_j(k)\}, \qquad j = 1, 2, \ldots, N, k = 1, 2, \ldots, M$$

  where $b_j(k)$ is the probability of k^{th} observation in state j.

- Initial state distribution

$$\pi = \{\pi_i\}, \qquad i = 1, 2, \ldots, N$$

  where π_i is the probability that the first observation starts in state i.

The complete set of parameters for a HMM involves identifying the number of states N and the number of observation symbols M and calculating the probability distribution for A, B and π. The complete parameter set of the HMM can be represented in a compact notation of the form:

$$\lambda = (A, B, \pi). \tag{3.1}$$

The model itself is limited to a certain set of states—termed hidden states in a HMM—and S is this set. V is the finite set of symbols emitted by the states as observations. The A and B are the transition probability distribution and the observation probability distribution. The mathematical control is the matrix that predicts the transition between two states. This state transition probability distribution matrix is used by the Markov process, along with knowledge of the states prior to and following the event being modelled to predict which state the model will enter next. When the model focus is in a particular state, the observation emitted by that state can be predicted by the observation probability distribution matrix for that state. The final element π of a HMM is a probability distribution that predicts the state the model will initially be in. This is called the initial state probability distribution matrix.

Consider an observation sequence:

$$O = \{O_1, O_2, \ldots, O_T\}$$

with corresponding set of states $S = \{S_1, S_2, \ldots, S_T\}$. For a HMM, the probability of the

sequence can be calculated as:

$$\begin{aligned} P(O|\lambda) &= P(S_1, S_2, \ldots, S_T|\lambda) \\ &= P(S_1) \cdot P(S_2|S_1) \cdot P(S_3|S_2) \ldots P(S_T|S_{T-1}) \\ &= \pi_1 \cdot a_{1,2} \cdot a_{2,3} \ldots a_{T-1,T} \end{aligned} \tag{3.2}$$

where π_1 represents the probability of initially starting in state S_1 and $a_{1,2}$ represents the transition probability from state S_1 to S_2. The transition probability of state S_T at time T is only dependent on previous state S_{T-1} at $T-1$ (as $P(S_T|S_{T-1})$).

When using an HMM to perform an information extraction task, the goal is to determine the most likely sequence of states that generates the required output, where the output is broken into a sequence of observation symbols taken from the set of all possible observation symbols. Thus given a sequence of observation symbols, O, the system calculates the most likely state sequence, S, which maximises the model parameter $P(O|S, \lambda)$ (Equation (3.2)). For example in a reference tagging task, the main goal is to identify reference tags (such as author, editor, title, year etc.) as output symbols. This process was introduced in Chapter 2. The model treats the reference tags as states in the HMM and the total number of reference tags will be set as the total number of states, $N = 8$, as in Figure 3.1. The system takes an observation sequence from the references section of a research paper, then tries to determine the best possible state sequence S for the given observation O which maximises the model $P(O|S, \lambda)$.

Figure 3.1 shows a simple HMM for a reference tagging task. The model contains a total of eight states with the set of hidden states given by $S = \{$*author, title, booktitle, volume, page, number, month, year*$\}$. In Figure 3.1, the transition probabilities (A) are labelled on the edges in the diagram. For example, the probability of transitioning from *author* to *title* is 0.19. The total probability of transitioning out of a state S_i is equal to $\sum_{j=1}^{N} a_{i,j} = 1$. In the model, the state transition starts with an initial state *author*, then transitions to the state *title*, then to *pages* and so on. Each state outputs an observation from a set of observation symbols in a probabilistic way. For example, state *Author* might contain observation symbols representing names (*A. C. Smith, M. Banko* and *E. Brill.*), where

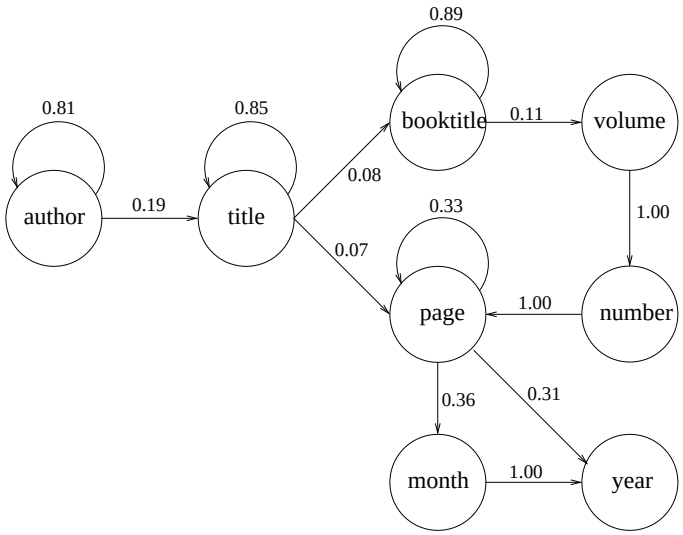


Figure 3.1: Simple HMM for reference tagging task

an observation symbol of "A." (as first name) is more likely to appear in state *Author* as part of the first name symbol. Here is an example of an input/output sentence for testing process:

> **Input:** S. K. Gadia. Weak temporal relations. pages 70-77, year 1986.

The output of the system gives the state sequence corresponding to the input sentence:

> **Output:** (author S. K. Gadia.) (title Weak temporal relations.) (pages pages 70-77,) (year year 1986.)

The output state sequences are generated by applying the *Viterbi Algorithm* [Viterbi, 1967] to predict the most likely state sequence. The details of this algorithm and several others used during the HMM process will be described in the next section.

3.2 The Basic Algorithms of HMM

The basic algorithms for HMMs were developed to provide solutions to the three fundamental problems associated with the HMM: calculating the probability of a state sequence and observation with given model parameters; predicting the most likely state sequence for a given observation; and determining the model parameters. Rabiner [1986] describes the three fundamental problems for HMM as:

- Evaluation
- Alignment
- Training

Evaluation provides a measurement of how close a given observation sequence matches against the model (Equation (3.2) in Section 3.1), alignment determines the most likely state sequence (path) for a given observation sequence, and training involves estimating the model parameter, λ, to maximise the probabilistic function $P(O|\lambda)$ against given observation sequences.

3.2.1 Evaluation Problem

The evaluation problem arises when attempting to compute $P(O|\lambda)$, which is the probability that the model will output a certain observation symbol sequence. This can be viewed as the similarity of a given sequence to the model—the higher the probability, the closer it matches the model. For example, consider a general research paper which is processed through a HMM as part of a reference tagging task (as mentioned in the previous section). The results might show that the last section of the paper contains higher probabilities against the model—an expected result considering that references generally appear at the end of a paper. Calculating the evaluation problem of the HMM allows the system to select the most similar model for a given observation sequence. The most common way to solve the evaluation problem is to use the Forward-Backward algorithm [Baum and Egon, 1967], which calculates the forward (α) and backward (β) probability measures variables for each time step. The forward and backward variables are the probability of the sequence $P(O|\lambda)$ against the model in two directions, one from the beginning of the sequence, and the other from the end of the sequence.

3.2.1.1 Forward Algorithm

The forward algorithm involves computing the probability of a particular output sequence given the model parameters $\lambda = (A, B, \pi)$ and the observation sequence $O = \{O_1, O_2, \ldots, O_T\}$. This can be solved by the *forward algorithm* [Baum and Egon, 1967].

The forward algorithm consists of calculating the probability of an observation sequence with forward variable α_t for each time t:

$$\alpha_t(i) = P(O_1, O_2, \ldots, O_t, q_t = S_i|\lambda) \tag{3.3}$$

where q_t represent the state (S_i) at time t at given model λ. The parameter $\alpha_t(i)$ can be calculated using induction:

1 Initialization:

$$\alpha_1(i) = \pi_i b_i(O_1), \qquad i = 1, 2, \ldots, N. \tag{3.4}$$

2 Induction:

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i)a_{ij}\right] b_j(O_{t+1}), \qquad t = 1, 2, \ldots, T-1 \text{ and } j = 1, 2, \ldots, N. \tag{3.5}$$

3 Termination:

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i). \tag{3.6}$$

Equation (3.4) computes the initial forward probability for each state S_i. Equation (3.5) calculates the probability for every possible state transition from S_i to S_j at $t+1$, where the formula $\alpha_t(i)a_{ij}$ calculates the joint probability for a particular observation O_t. The last step of the procedure, known as termination, involves summing the joint probabilities $\alpha_T(i)$ for each state. The resulting probability provides an estimate of a given observation sequence of $\{O_1, O_2, \ldots, O_t\}$ for $P(O|\lambda)$, determined using Equation (3.6). The computation involved in the calculation of $\alpha_t(j)$, $1 \leq t \leq T$, $1 \leq j \leq N$, requires in the order of N^2T calculations.

Figure 3.2 illustrates the possible state transitions from S_i to S_j at $t+1$, where $\alpha_{t+1}(j)$ is the joint probability estimation of the observation sequence $\{O_1, O_2, \ldots, O_{t+1}\}$ and state S_j at time $t+1$. Calculating the forward variable allows the system to evaluate the sequence for the resulting HMM.

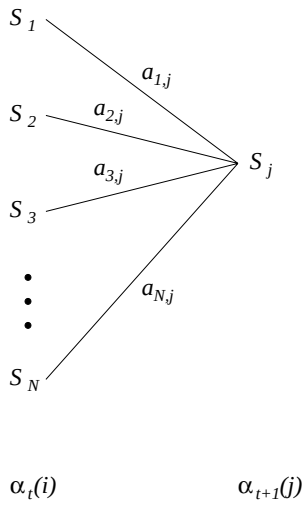


Figure 3.2: Forward Algorithm

3.2.1.2 Backward Algorithm

The backward procedure consists of calculating the probability of an observation sequence. The backward variable β_t for the observation sequence calculates the probabilities from time $t+1$ to the final output at time T. The backward variable $\beta_t(i)$ can be expressed as:

$$\beta_t(i) = P(O_{t+1}, O_{t+2}, \ldots, O_T, q_t = S_i|\lambda) \tag{3.7}$$

where q_t represents the state (S_i) at time t in the model λ. The parameter $\alpha_t(i)$ can be calculated using induction:

1 Initialization:

$$\beta_T(i) = 1, \qquad i = 1, 2, \ldots, N. \tag{3.8}$$

2 Induction:

$$\beta_t(j) = \sum_{i=1}^{N} a_{ij}\beta_{t+1}(j)b_j(O_{t+1}), \qquad t = T, T-1, \ldots, 1 \text{ and } i = 1, 2, \ldots, N. \tag{3.9}$$

3 Termination:

$$P(O|\lambda) = \sum_{i=1}^{N} \pi_i b_i(O_1)\beta_1. \tag{3.10}$$

Equation (3.8) computes the initial backward probability for each state S_i at the last step $t = T$. Equation (3.9) calculates the probability of state S_j being the destination of a

transition from state S_i for each state at time $t+1$ for times $t = T, T-1, T-2, \ldots, 1$, and the formula $a_{ij}\beta_{t+1}(j)$ calculates the joint probability for a transition from state S_i to state S_j as well as the observation probability in state S_j. As with the forward algorithm, the computation involved in the calculation of $b_t(j)$, $t = 1, 2, \ldots, T$, $j = 1, 2, \ldots, N$, requires on the order of $\mathrm{O}(N^2T)$ calculations.

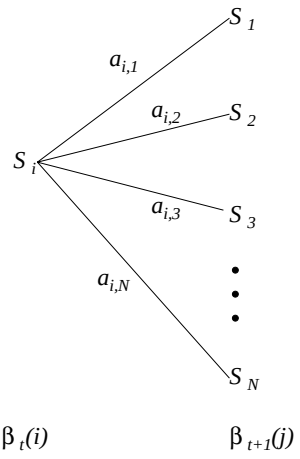


Figure 3.3: Backward Algorithm

3.2.2 Testing Algorithm of HMM

Every state in the HMM is associated with exactly one observation; therefore each symbol is associated with the state that outputs the symbol. In order to solve the alignment problem for the observation sequence $O = \{O_1, O_2, \ldots, O_T\}$ the system finds a state sequence with length T from the initial state to the end state, such that the i^{th} symbol O_i is output by the i^{th} state in the path. The Viterbi algorithm is then used to find the highest probability state path generated by the observation symbols.

Given the model parameters $\lambda = (A, B, \pi)$ with N being the total number of states, and the observation sequence $O = \{O_1, O_2, \ldots, O_T\}$, find the most likely sequence of states that could have output the given observation sequence. This can be solved by the *Viterbi Algorithm* [Viterbi, 1967]. The four formulae for the Viterbi algorithm are:

1 Initialization:

$$\delta_1(i) = \pi_i b_i(O_1), \qquad i = 1, 2, \ldots, N \tag{3.11}$$

$$\phi_1(i) = 0 \tag{3.12}$$

2 Recursion:

$$\delta_t(j) = \max_{1 \le i \le N} [\delta_{t-1}(i) a_{ij}] b_j(O_t), \qquad t = 2, \ldots, T \text{ and } j = 1, 2, \ldots, N \tag{3.13}$$

$$\phi_t(j) = \arg \max_{1 \le i \le N} [\delta_{t-1}(i) a_{ij}], \qquad t = 2, \ldots, T \text{ and } j = 1, 2, \ldots, N \tag{3.14}$$

where $b_j(O_t)$ is the probability of outputting symbol O_t at state S_j, and a_{ij} is the transition probability from state S_i to S_j. The parameter $\delta_t(j)$ takes the maximum value among all states of HMM at time t, where $\phi_t(j)$ records the most probable path for each state S_j.

3 Termination:

$$P_T^* = \max_{1 \le i \le N} \delta_T(i) \tag{3.15}$$

$$q_T^* = \arg \max_{1 \le i \le N} \delta_T(i) \tag{3.16}$$

where P_T^* records the maximum probability distribution for parameter δ, and q_T^* keeps track of the path based on the value calculated by $\delta_T(i)$.

4 Path tracking:

$$q_t^* = \phi_{t+1}(q_{t+1}^*), \qquad t = T-1, T-2, \ldots, 1 \tag{3.17}$$

The most likely path can be determined by finding q_T^* for $\arg \max_{1 \le i \le N} \delta_T(i)$ at each time step.

Given an HMM with N states, the computational cost of this process is $\mathrm{O}(TN^2)$. Compare this to the *brute force* full path search algorithm which searches every possible path for every state at time t, whose cost is $\mathrm{O}(T^N)$.

Figure 3.4 shows the path tracking of five output symbols ($t = 5$) with four states ($S = \{S_1, S_2, S_3, S_4\}$). The solid line between states indicates the most probable path predicted by the Viterbi algorithm, and the states touched upon by the path being the 'preferred' or most likely state sequence.

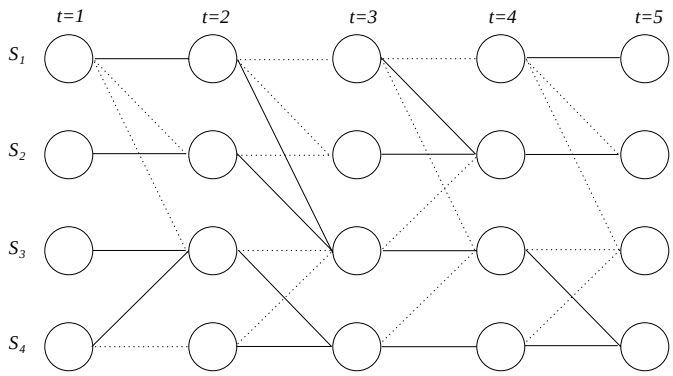


Figure 3.4: Viterbi Algorithm

3.2.3 Training Algorithm of HMM

The goal of training the model is to determine values for the model parameters (A, B, π) that maximise the probability of matching the observation sequences in training data. That is, find the most likely set of state transitions and output probabilities by adjusting the model $\lambda = (A, B, \pi)$ to maximise $Pr(O|\lambda)$ for a given output sequence O. This can be solved by the *Baum-Welch algorithm* [Baum and Petrie, 1966]. The process involves re-estimating the parameters A and B to improve the model.

Firstly, the system defines $\xi_t(i, j)$ as the probability of transitioning from state S_i to S_j at time $t + 1$ (given the model and the observation sequence) as:

$$\xi_t(i, j) = P(q_t = S_i, q_{t+1} = S_j|O, \lambda) \tag{3.18}$$

where ξ is calculated from the forward and backward variables, as shown by:

$$\begin{aligned} \xi_t(i, j) &= \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{P(O|\lambda)} \\ &= \frac{\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)}{\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_t(i)a_{ij}b_j(O_{t+1})\beta_{t+1}(j)} \end{aligned} \tag{3.19}$$

The probability of being in state S_i at time t is:

$$\gamma_t(i) = \sum_{j=1}^{N}\xi_t(i, j) \tag{3.20}$$

The model re-estimation can be calculated using Equations (3.19) and (3.20), where the model parameters are now defined as:

$$\bar{\pi}_i = \gamma_1(i) \tag{3.21}$$

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{3.22}$$

$$\bar{b}_j(k) = \frac{\sum_{\substack{t=1 \\ s.t.\ O_t=v_k}}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)} \tag{3.23}$$

After the parameters have been re-calculated, the compact notation for the model becomes $\bar{\lambda} = (\bar{A}, \bar{B}, \bar{\pi})$. By iteratively applying the Baum-Welch algorithm, the parameters improve to a point where a local maximum point is found based on Baums' auxiliary function:

$$Q(\lambda, \bar{\lambda}) = \sum_{Q} P(Q|O, \lambda) \log(P(O, Q|\bar{\lambda})). \tag{3.24}$$

This implies:

$$Q(\lambda, \bar{\lambda}) \geq Q(\lambda, \lambda) \Longrightarrow P(O|\bar{\lambda}) \geq P(O|\lambda) \tag{3.25}$$

3.3 Limitions of HMM

As with most mathematical models, there are drawbacks to the HMM. For instance the model cannot easily handle problems with recursive characteristics, and it becomes computationally expensive when there are a large number of states. The limitations of HMM are:

- the observation probability distribution is independent between states, because of the independence assumption; thus the model cannot take advantage of the situation when there are relationships between observations for different states.
- the states in the HMM cannot be dynamic, which means the probability distribution within a state cannot be changed over a period of time.

- the probability of the model being in a particular state at time t only depends on the state at $t-1$. This is known as the Markov property. For example, in the reference tagging task, the knowledge of a name like "A. C. Smith" forming the state sequence of { *first name, middle name, family name* }. When the sequence has transited to the third observation "Smith", the model does not know that it is in the third state of the sequence. The model only remembers the previous state. Also, because of the independence assumption, states cannot share information.

3.4 Summary

The chapter started with the fundamental elements of HMM, and discussed the training and testing process for HMMs, followed by a brief discussion of the limitations of HMMs. This research explores the use of hierarchical hidden Markov models (HHMMs), which offer to incorporate and resolve some of the limitations of HMMs.

CHAPTER 4

Hierarchical Hidden Markov Models

This chapter expands upon the ideas covered in the introduction to HHMMs presented in Chapter 2, and puts forward a new approach to their construction and use in order to improve processing time and extraction accuracy. The improvement is based on merging repeated state sub-models during training, and the simplifying process involves re-expressing the various transitions within the sub-models during the testing process. A model that has undergone this merging and simplification process is referred to as a merged and simplified hierarchical hidden Markov model (MSHHMM).

The concept of the hierarchical hidden Markov model (HHMM) was introduced by Fine [1998] as a generalisation of the HMM modelling tool described in Chapter 2. The model itself makes use of hierarchical structure and gives the potential benefit of reusing sub-state information when the same subsequence occurs. For example, in reference tagging problems, the *author* and *editor* fields both contain names as their state observations (this will be discussed further in the Chapter 5). The model also provides a better fit than a linear model to data that is hierarchically structured, such as DNA sequences [Hu et al., 2000], handwriting [Fine et al., 1998], robot navigation [Theocharous et al., 2001], and natural language processing [Rabiner and Juang, 1986]. Some of these examples were discussed in Chapter 1.

This chapter describes the nature of HHMMs and presents an efficient probabilistic estimation method to calculate the transition distribution between different levels of the hierarchical structure. Section 4.1 provides a description of HHMMs. Section 4.2 describes the notation of the HHMM. Section 4.3 describes the algorithm to compute the output distribution for each level. Section 4.4 describes the likelihood relations between the parameters and the models. Section 4.5 discusses structural issues of HHMMs with some comparison against the stochastic context-free grammar (SCFG) model, and Section 4.6 gives a summary explaining the processes of training and testing HHMMs.

4.1 Model Description

An HHMM is a structured multi-level stochastic process and can be visualised as a tree structured HMM (see Figure 4.1). For a regular HMM, there is only one type of state: that which contains an output observation. There are two types of states for the HHMM:

- **Production state**: a leaf node of the tree structure that contains only output observations (represented in Figure 4.1 as the empty circle $\bigcirc$).
- **Internal state**: a node that contains production states or other internal states (represented in Figure 4.1 as a circle with a cross inside $\bigoplus$).

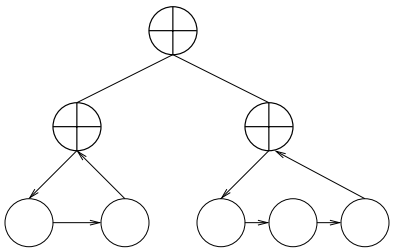


Figure 4.1: HHMM tree structure

The output of an HHMM is generated by a process of traversing some sequence of states within the model. At each internal state, the process traverses down the tree, possibly through further internal states, until it encounters a production state. Thus, as it continues through the tree, the process matches a sequence of observations. The process ends

when a final state is reached. The difference between a standard HMM and a hierarchical HMM is that individual states in the hierarchical model can traverse to a sequence of production states, whereas each state in the regular HMM corresponds to a single production state, which, in turn, can only contain a single observation.

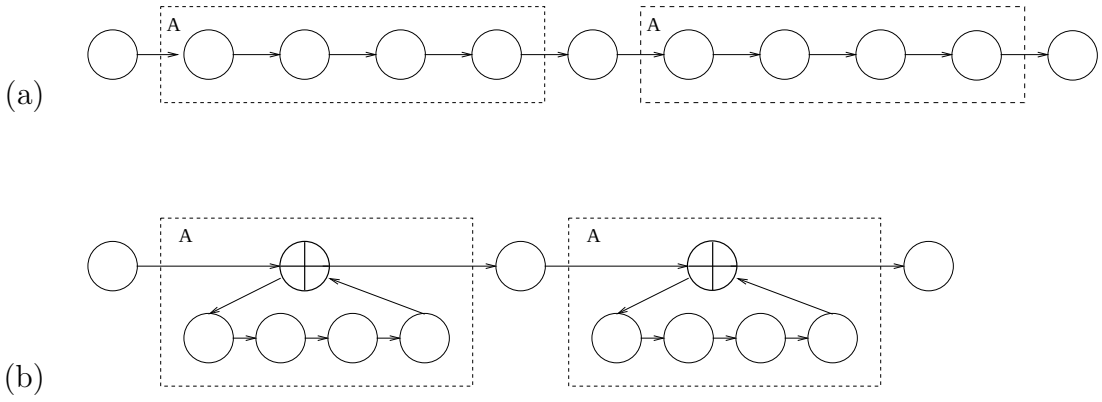


Figure 4.2: Example of (a) HMM (b) HHMM

Figure 4.2(a) and Figure 4.2(b) illustrate the process of reconstructing a HMM as a hierarchical HMM. Figure 4.2(a) shows an HMM with 11 states. The two dashed boxes (A) indicate regions of the model that have a repeated structure. These regions are furthermore independent of the other states in the model. Figure 4.2(b) models the same structure as a hierarchical HMM, where those repeated structures are now grouped together under a state called an internal state. This HHMM uses a two level hierarchical structure to expose more information about the transitions and probabilities within the internal states. These internal states, as discussed earlier, produce no observation of their own. Instead, this is left to the child production states within them. Figure 4.2(b) is an example of HHMM with two internal states that each contain four production states.

Example: Consider the reference tagging task from Figure 2.7, where the reference is segmented into useful fragments. The output of the model can expressed as:

> [1] (author T. Moloney, A. C. Lea, and C. Kowalchuk.) (title Manufacturing and packaged goods.) (editor In G. H. Castle, editors,) (booktitle Profiting

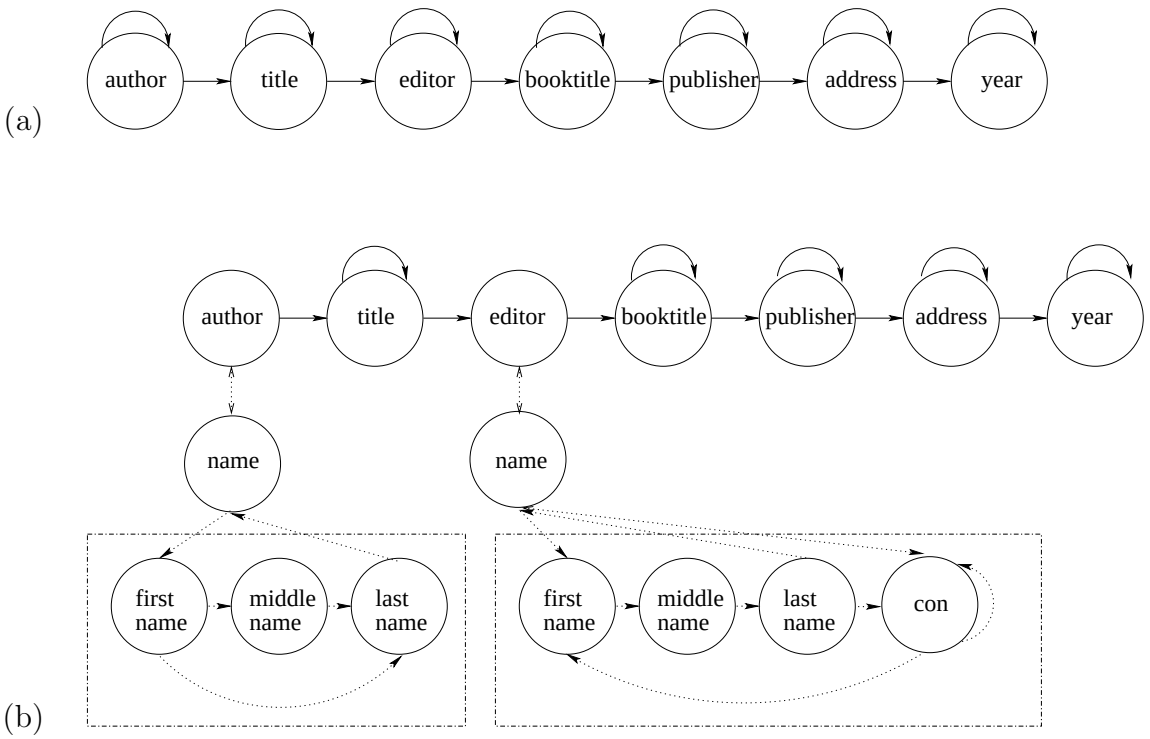


Figure 4.3: Example of reference tagging task with (a) HMM (b) HHMM

from a Geographic Information System.) (publisher GIS World, Inc.,) (address Fort Collins, CO,) (year 1993.)

where both *author* and *editor* contain a sequence of people's name as part of their observation. ■

Figure 4.3(a) shows an HMM with seven states that represents the example. Figure 4.3(b) represents an equivalent model built in a hierarchical manner. The dashed boxes in Figure 4.3(b) indicate regions of the model that have a repeated structure, which correspond to the dashed box (A) in Figure 4.2(b). From Figure 4.2, there is a hierarchical relationship between *author*, *editor* and name, where name is represented by the states; *first name*, *middle name*, *last name* and *con*. The state *con* corresponds to observations that do not involve actual names, but instead describe connecting words, such as the observations "In" and "editors,".

During the construction of an HHMM, two key factors considered are:

- merging repeated sub-models (M)
- simplifying the sub-models (S)

This research makes use of these two factors, and constructs a new Merged and Simplified hierarchical hidden Markov model (MSHHMM). The merging process involves merging repeated sub-models during training to increase prediction accuracy for the extraction task. The simplifying process involves re-expressing the various transitions within the sub-models during the testing process.

4.1.1 Merging Repeated Sub-models (M)

In some cases, different internal states of a hierarchical HMM correspond to exactly the same structure in the output sequence. This repeated structure was denoted by the dashed box in Figure 4.2. A hierarchical model lends itself to the merging of this repeated structure, the act of which improves the model in several ways. It makes more observations available for each internal state of the sub-model and, at the same time, requires fewer observations in order to achieve a given level of accuracy. However, a merged model no longer adheres to the Markov assumption as the model would now have to remember which parent state the sub-model was entered from. Also, such a model would no longer be, strictly speaking, a tree, as it would contain cyclic paths. Thus the MSHHMM provides a post-training stage where each sub-model is cloned so each parent node receives its own copy, thus restoring both the tree and the Markov assumption.

The sharing of model states has been put forward in previous HHMM research. Work by Fine [1998] and Murphy [2001] used repeated states by temporarily transforming the tree into a lattice. The subsequent transform from lattice back into a tree, unfortunately, incurred an exponential growth in model size and complexity. Recall that, in Figure 4.3(b), both the states *author* and *editor* contained the state *name* as their child state. In a hierarchical model, the state *name* is needs to be expanded into two sub-states as *name-author* and *name-editor*, so that each relation within the model structure satisfies the $1 : N$ mapping constraint (where N represents the number of child states). Recent research by Bui [2004] once again addressed the matter of shared states by developing new

methods to allow processing against the lattice model itself (rather than reverting to a tree form). The new methods included formulas for handling multiple parent states. This new lattice research was published in 2004, a year after the development of the merging employed in this thesis.

The merged model makes use of a series of repeated states—termed a sub-model—where all sub-models of equivalent structure are merged into one. The system introduces control transitions into the sub-model, via calculated probabilities. This is done before the merging process. For example, in Figure 4.4(a) both internal states have the same sequence of states (A), and the model can make use of repeated parts to gain more information for the sub-models.

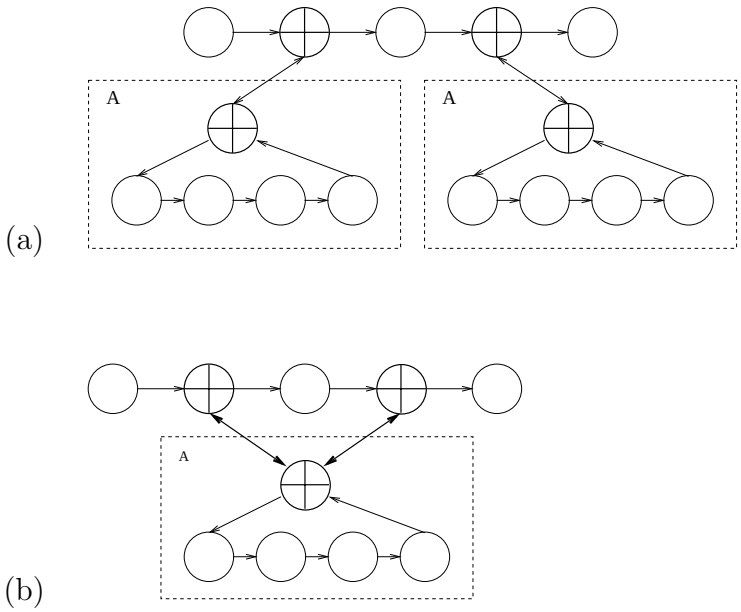


Figure 4.4: Example of (a) HHMM with (b) a copied model

Figure 4.4(b) is a simple representation of Figure 4.4(a), where the system has merged the two repeating sub-models into one (that is, state A). When two repeating sub-states are merged, the model requires less computation for the building process, with a fewer total states needing to be identified than the regular HMM. It produces a more reliable system by having an increased amount of training data (observations) for those merged states. A bold double arrow ($\longleftrightarrow$) is used in the figure to indicate that the original state

of the transition is (temporarily) remembered so that the transition from the sub-model can be directed back to the correct parent state. Figure 4.5 shows an example for the reference tagging task, where the HHMM contains three internal states *author*, *editor* and *name* and the state *name* is the shared internal state. The eight production states are; *first name*, *middle name*, *last name*, *title*, *booktitle*, *publisher*, *address* and *year*.

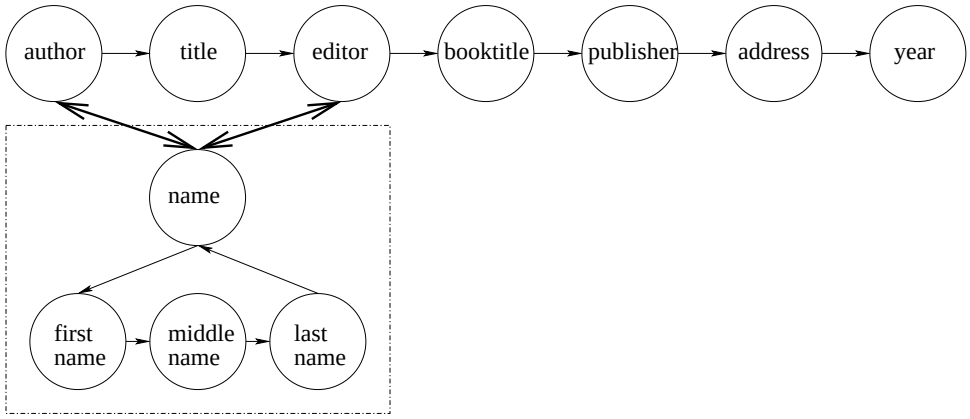


Figure 4.5: Example of a reused sub-model for the reference tagging task

Using a hierarchical structure has the benefit of merging repeated parts of the structure, whereupon a larger number of observations are available for each sub-state. Figure 4.6 illustrates the probability distribution for the merging process for two repeating sub-states A. The top part of the figure shows the probability distribution before the merging process, and the bottom part of the figure shows the probability distribution after the model is merged. The observation count is shown inside each production state. The dotted lines represent merged nodes from both sub-states A. For example, the first sub-state A contains an observation count of 30 after states are merged. It combines the observations from the two sub-states A before they are merged, where 20 observations are collected from the first sub-state of A and 10 from the second. The combined sub-states that make up A, contains a higher observation count, which leads to a more accurate and stable model when applied to extraction tasks.

Figure 4.7 shows a HMM which is visually similar to the one shown in Figure 4.4(b).

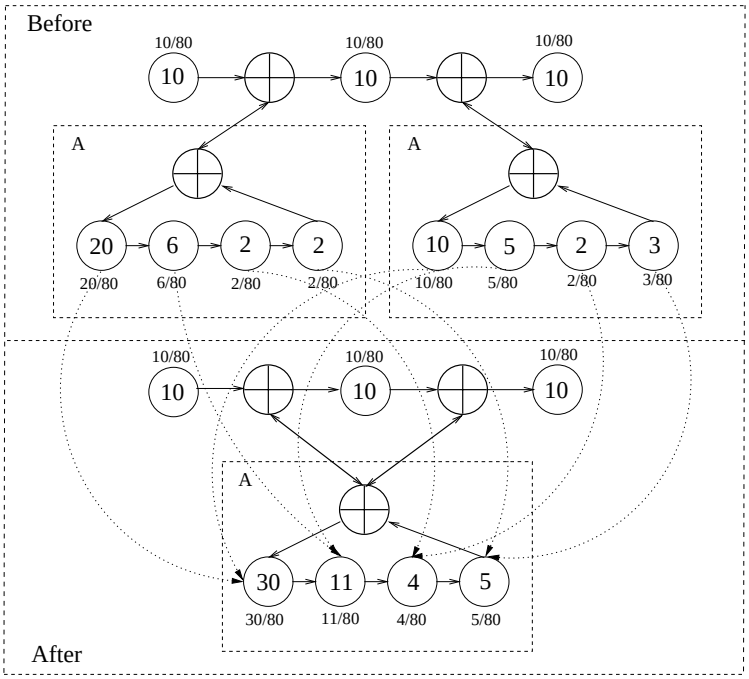


Figure 4.6: Example of probability distribution for repeated sub-model (A)

However there are significant differences between these two diagrams. Each state in the HMM outputs a single probabilistic observation, whereas an internal state for an HHMM can output either another internal state or a production state. Also in the HMM, the transition can be made from one state to another state without the restriction of the bold double arrow as shown in Figure 4.4(b), where the HHMM must transit back to the parent state from which the process originates.

So far, this chapter has discussed the HHMM as a hidden Markov model extended by allowing a hierarchy of states similar to the form of a tree. However a pure tree structured model would be inefficient where each sub-state can only contain a $1 : N$ relation. Natural language contains many repeated sequences of observations and the regular model cannot gain any efficiency from this fact as it does not allow the merging of these repeated sub-models. Thus an improvement can be made to the HHMM by merging sub-models, but only at the expense of the non-cyclic property of pure trees. This consequence of the merging process, means that the model is no longer a tree as at least one node of the tree now has two or more parents.

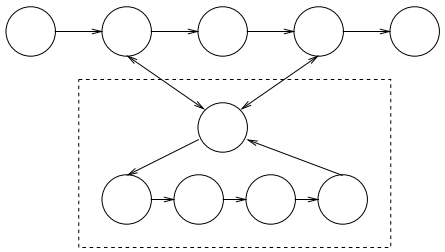


Figure 4.7: Example of a HMM

Strictly speaking, the model shown in Figure 4.4(b) no longer adheres to one of the Markov properties [Cox and Miller, 1965], that the current state should only depend on the previous state: the bold arrow (⟷) is used so that the transition is remembered. If a model is to conform to the Markov property, the probabilities for each state should be independent of each other. For example, if the system used the structure in Figure 4.7 without restoring the tree property, it would need to keep track of the previous state (at least while in the repeated sub-states), so that the process could return to the appropriate position in the 'root level' sequence of states.

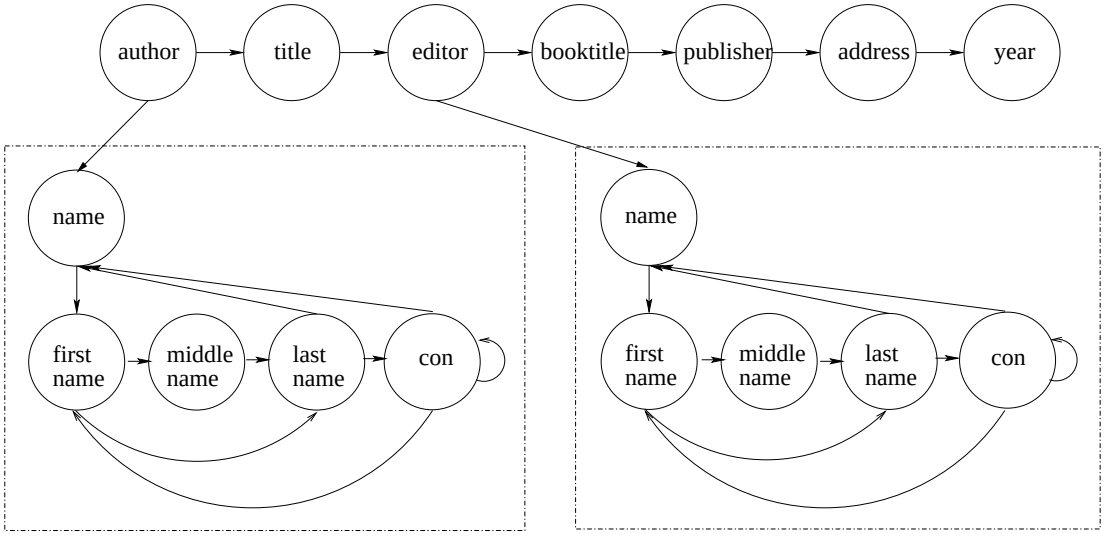


Figure 4.8: Example of an HHMM with duplicated sub-models

In order to restore the Markov property to a model that has been merged, it is necessary

to clone multiple copies of the shared sub-model—one for each parent state. Figure 4.8 shows a hierarchical model that no longer needs to remember the parent states of the repeated sub-models, because the sub-model *name* has been duplicated into two sub-models. The information within each sub-model is still shared, but the sub-model holds two positions in the HHMM. Therefore the model conforms to the Markov property, but still retains the benefits of reusing sub-model information.

4.1.2 Simplifying the Sub-models (S)

During the course of this research, a method was developed to simplify the sub-models during the testing process. It was primarily developed to re-express the various transitions within the sub-models. The simplification involves transforming an internal state q_i into three transformed states:

$$\{q_{in}^{(i)}, q_{stay}^{(i)}, q_{out}^{(i)}\}$$

where $q_{in}^{(i)}$ represents the vertical transition of moving into sub-model i, $q_{stay}^{(i)}$ represents the horizontal transition of moving within the sub-model and $q_{out}^{(i)}$ is the vertical transition of moving out the sub-model. In this manner, the sub-model information, and in particular the calculations involved in generating the transition probability distribution matrix, can be summarised in just three transformed states.

Consider an example of an output sequence that iterates two times through the states of a sub-sequence A (as shown in Figure 4.9). The first sub-sequence of A is $\{p_2, p_3, p_4, p_5\}$ and the second sub-sequence of A is $\{p_6, p_3, p_4, p_4\}$. Notice the first sub-sequence A starts with sub-state p_2, and the second starts with sub-state p_6. The two sub-sequences A have states p_4 and p_5 as the exit state respectively.

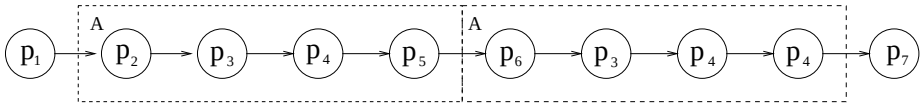


Figure 4.9: Example of a repeat sequence

Figure 4.10 shows the general graphical representation of the HHMM for the state sequence from Figure 4.9. The top level of the model consists of three states $\{p_1, q_A, p_7\}$ with five production states $\{p_2, p_3, p_4, p_5, p_6\}$ as the child states of the internal state q_A.

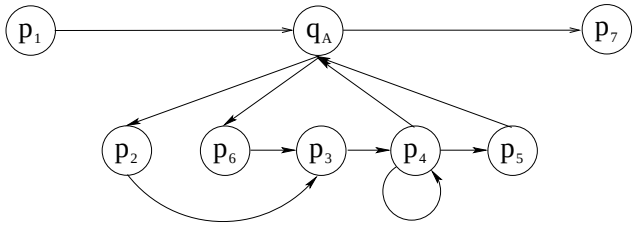


Figure 4.10: The general representation of HHMM

When two repeated sub-sequences exist in the state sequence, a single representation is no longer able to define when a sub-sequence finishes or starts another sub-sequence. A simple solution is to use further representations for the internal state, so as to clearly denote the entry and exit points of the sequence. Figure 4.11 demonstrates the structure of the internal states of sub-model A, where the states have been simplified into three transformed state $q_{in}^{(A)}$, $q_{stay}^{(A)}$ and $q_{out}^{(A)}$. The transformed state $q_{in}^{(A)}$ represents the entry point of the sub-model (A), and the observation probability is dependent on the two entry states of p_2 and p_6. The transformed state $q_{stay}^{(A)}$ represents the situation where the sub-sequence is neither entering nor leaving the sub-model (A). The observation probability of transformed state $q_{in}^{(A)}$ is dependent on two production states $\{p_3, p_4\}$. The transformed state $q_{out}^{(A)}$ represents the exit point of the sub-model (A), and the observation probability is dependent on two exit states of p_4 and p_5. The simplified states, *in*, *stay* and *out*, become placeholders for multiple 'actual' states. The transition between these three internal states are estimated by summing the transition between child states. Detailed calculation of each sub-model is discussed in Section 4.3.

The three transformed internal states ($q_{in}^{(A)}$, $q_{stay}^{(A)}$ and $q_{out}^{(A)}$) represent the child state transition for the sub-model A (as shown in Figure 4.11). They represent the status of the sub-model: whether it is the beginning of the sub-state, staying within the sub-state or leaving the sub-state. Transitions from the internal state *stay* are limited to the sub-

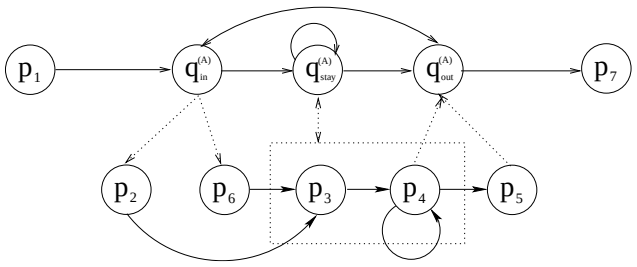


Figure 4.11: Example of a simplified model

states of A, thus making transition only to other states within the same sub-model. In this manner, the model adheres to the rules of a tree-like structure: child states of the sub-model cannot make transitions to other sub-models. The top level of the model in Figure 4.11 is now equivalent to a regular HMM with five states (p_1, $q_{in}^{(A)}$, $q_{stay}^{(A)}$, $q_{out}^{(A)}$ and p_7).

In the reference tagging task, the entry state to the author sub-model (state *first name*) depends on how the reference is written. Figure 4.12 lists references for three published science research papers. The first reference starts with the first name "T" under the *author* state, the second starts with the last name "Nahm" and the third starts with something different again. In order to correctly merge the initial sub-models that represent the *name*, the internal state must allow multiple entrances into the sub-model, including transitions to the state *first name* ("T"), and *last name* ("Nahm").

While the structure, and hence observations, of a sub-model may be repeated several times within a particular output sequence, no transitions are shared between occurrences of the sub-model. Thus each transition of the sub-model is independent of previous or future transitions. This property extends to the probabilities and other information calculated for the sub-model.

4.2 Notation

According to [Rabiner and Juang, 1986], a HMM can be defined in terms of five elements, as mentioned in Chapter 3; the set of hidden states S, the transition probability distribu-

T. Moloney, A. C. Lea, and C. Kowalchuk. Manufacturing and packaged goods. In G. H. Castle, editors, Profiting from a Geographic Information System. GIS World, Inc., Fort Collins, CO, 1993.

Nahm, U. Y., and Mooney, R. J. 2000. Using information extraction to aid the discovery of prediction rules from texts. In *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining (KDD-2000) Workshop on Text Mining*, 51-58.

[*Wilks et al., 1989*] Yorick Wilks, D. Fass, C. Guo, J. McDonald, T. Plate and B. Blator. A tractable machine dictionary as a resource for computational semantics. In Bran Boguraev and Ted Briscoe, editors, *Computational Lexicography for Natural Language Processing*, pages 193-228. Longman Group UK Limited, London, 1989.

Figure 4.12: List of three science research published papers

tion A, the set of observation symbols V, an observation probability distribution B and the initial state probability distribution π. For the HHMM, there is one extra element: the final probability distribution over every internal state τ, which represents the probability of leaving a sub-model. In Table 4.1, most of the notations of the HHMM (and the sub HHMM) are similar to the HMM, but with an extra set of internal states (q). For example the set of hidden states now becomes $\{p_1, \ldots, p_N, q_1, \ldots, q_D\}$, where p is the set of production states (those that contain observations) and q is the new set of internal states (those that contain other states). N represents the total number of production states p, D represents the total number of internal states q, and M represents the total number of observation symbols. The transition probability distribution for the HHMM is $\bar{a}_{ij}$, where $\bar{a}$ contains a standard transition probability from state to state, plus the probability of transitioning to, and within, internal states. The initial state probability includes π_i as the probability of a sequence starting with production state p_i, and $\bar{\pi}_i$ as the probability of entering internal state q_i.

	HMM	HHMM
set of hidden states S:	$\{s_1, \ldots, s_N\}$	$\{p_1, \ldots, p_N, q_1, \ldots, q_D\}$
set of output symbols V:	$\{v_1, \ldots, v_M\}$	$\{v_1, \ldots, v_M\}$
transition distribution A:	$\{a_{ij}\}$	$\{\bar{a}_{ij}\}$
observation distribution B:	$\{b_j(1), \ldots, b_j(N)\}$	$\{b_j(1), \ldots, b_j(N), \bar{b}_j(1), \ldots, \bar{b}_j(D)\}$
initial distribution π:	$\{\pi_1, \ldots, \pi_N\}$	$\{\pi_1, \ldots, \pi_N, \bar{\pi}_1, \ldots, \bar{\pi}_D\}$
final distribution τ:	-	$\{\bar{\tau}_1, \ldots, \bar{\tau}_D\}$
Compact notation:	$\lambda = (A, B, \pi)$	$\Lambda = (\bar{A}^{(i)}, \bar{B}^{(i)}, \bar{\pi}^{(i)}, \bar{\tau}^{(i)})$

Table 4.1: Table of notations

Table 4.1 shows the elements of the HMM and the HHMM, and it can be seen that the HHMM has one extra element—the final state probability distribution τ:

$$\bar{\tau} = \{\bar{\tau}_1, \ldots, \bar{\tau}_D\} \tag{4.1}$$

where $\bar{\tau}_d$ represents the final probability distribution of leaving the internal state q_d, and D is the total number of internal states in the model. The corresponding $\bar{\pi}_d$ expresses the probability of entering the internal state q_d.

4.3 Sub-models

Each internal state q_i in a simplified hierarchical hidden Markov model (SHHMM) is simplified by resolving each child production state into one of three transformed states to

identify the status of either entering, staying or exiting a sub-model, such that:

$$q_i \Rightarrow \{q_{in}^{(i)}, q_{stay}^{(i)}, q_{out}^{(i)}\} \tag{4.2}$$

These transformed states are not production states themselves but instead contain either production states or other internal states. The three transformed states are:

- $q_{in}^{(i)}$: the entry state to the sub-model, where the model transitions from a state in the upper level model into this model,
- $q_{stay}^{(i)}$: a state that represents all other states within the sub-model that are neither entry nor exit states,
- $q_{out}^{(i)}$: the exit state, where the model transitions from a state in the sub-model out into the upper level model.

Each model, and the sub-models within it, can be expressed in terms of a state set to represent the structure of the model and a parameter set to determine the best sequence of model states. These elements of the model combine to produce both the observation and transition probabilities for a particular state. But as a consequence of the transformation method mentioned above, the model will need to re-calculate the new observation distributions and the transition probabilities from the elements of each of these transformed states. While determining the state set is straightforward—in that each internal state is transformed into three states—the estimation process of the parameter set for a multi-level HHMM is a complex process. This section will first introduce the elements for a sub-model, followed by a probability estimation method for internal states that works by transforming each internal state into three production states.

4.3.1 Elements of a Sub-model ($\bar{\lambda}^{(i)}$)

The elements of a SHHMM sub-model are represented using the basic notation and the superscript label (d) to represent the internal state q_d. For example, $\bar{\pi}^{(3)}$ represents the initial probability distribution of entering state q_3, $\bar{\tau}^{(3)}$ represents the initial probability distribution of leaving state q_3, and $\bar{a}_{ij}^{(3)}$ represents the transition probability distribution

within the internal state q_3.

The elements of a sub-model (i) can be characterised as follows.

1 **Transition probability distribution (transition matrix)** $\bar{A}^{(i)}$:

$$\begin{aligned}
\bar{A}^{(i)} &= \{\bar{a}_{ij}^{(3)}\} \qquad \text{for } i = 1, 2, \ldots, n_i, j = 1, 2, \ldots, n_i, \\
&= \begin{bmatrix} a_{11} & \ldots & a_{1n_i} \\ \vdots & \ddots & \vdots \\ a_{n_i1} & \ldots & a_{n_in_i} \end{bmatrix} \\
&\Rightarrow \begin{bmatrix} \bar{a}_{in,in}^{(i)} & \bar{a}_{in,stay}^{(i)} & \bar{a}_{in,out}^{(i)} \\ \bar{a}_{stay,in}^{(i)} & \bar{a}_{stay,stay}^{(i)} & \bar{a}_{stay,out}^{(i)} \\ \bar{a}_{out,in}^{(i)} & \bar{a}_{out,stay}^{(i)} & \bar{a}_{out,out}^{(i)} \end{bmatrix} \qquad (4.3)
\end{aligned}$$

Here, $\bar{a}_{i,j}$ represents the transition probability for the child states $\{p_1, \ldots, p_{n_i}\}$ of the internal state q_i, n_i is the total number of child states for internal state q_i and $\bar{a}^{(i)}$ represents the 3×3 transformed matrix.

2 **Observation probability distribution (observation matrix)** $\bar{B}^{(i)}$:

$$\begin{aligned}
\bar{B}^{(i)} &= \{\bar{b}_j^{(i)}(k)\} \qquad \text{for } j = 1, 2, \ldots, n_i, k = 1, 2, \ldots, m_j, \qquad (4.4) \\
&= \begin{bmatrix} b_{1,1} & \ldots & b_{1,T} \\ \vdots & \ddots & \vdots \\ b_{n_i,1} & \ldots & b_{n_i,T} \end{bmatrix} \qquad (4.5) \\
&\Rightarrow \begin{bmatrix} \bar{b}_{in,1}^{(i)} & \ldots & \bar{b}_{in,t}^{(i)} \\ \bar{b}_{stay,1}^{(i)} & \ldots & \bar{b}_{stay,t}^{(i)} \\ \bar{b}_{out,1}^{(i)} & \ldots & \bar{b}_{out,t}^{(i)} \end{bmatrix} \qquad \text{for } t = 1, 2, \ldots, T \qquad (4.6)
\end{aligned}$$

Here, m_j represents the total number of observation symbols in state p_j and $\bar{a}^{(i)}$ represents the transformed observation probability matrix having been reformed into a $3 \times T$ matrix to correspond to three transformed states for each internal state, where T represents the length of the test observations.

3 **Initial state probability distribution** $\bar{\pi}^{(i)}$:

$$\bar{\pi}^{(i)} = \{\bar{\pi}_j^{(i)}\} \qquad \text{for } j = 1, 2, \ldots, n_i. \qquad (4.7)$$

Here, $\bar{\pi}_j^{(i)}$ represents the probability of entering internal state q_i by way of child state p_j, and n_i is the number of sub-states for state q_i.

4 **Final state probability distribution**:

$$\bar{\tau}^{(i)} = \{\bar{\tau}_j^{(i)}\} \qquad \text{for } j = 1, 2, \ldots, n_i. \tag{4.8}$$

Here, $\bar{\tau}_j^{(i)}$ represents the probability of leaving internal state q_i at state p_j.

The compact notation for a sub-model of an SHHMM is:

$$\bar{\lambda}^{(i)} = (\bar{A}^{(i)}, \bar{B}^{(i)}, \bar{\pi}^{(i)}, \bar{\tau}^{(i)}) \tag{4.9}$$

The structure for a SHHMM contains the property that each internal state is sub-divided into three transformed states $\{in, stay, out\}$. When a state sequence appears, the model must identify the entry and exit point of each internal state. Consider a SHHMM trained using the repeated sequences of A in Figure 4.9, where these sequences are represented by three transformed states as shown in Figure 4.11. Given such a state sequence, Figure 4.13 demonstrates the output representation for the SHHMM, where the internal states are labelled by $\{q_{in}^{(A)}, q_{stay}^{(A)}, q_{out}^{(A)}\}$. Notice that the sequences of A are linked one after the other, with a transition from $q_{out}^{(A)}$ to $q_{in}^{(A)}$. The sequence of top level states consists of six transformed states $\{q_{in}^{(A)}, q_{stay}^{(A)}, q_{out}^{(A)}\}$ and two production states $\{p_1, p_7\}$. The sequence of each transformed state consists of four of the five production states $\{p_2, p_3, p_4, p_5, p_6\}$.

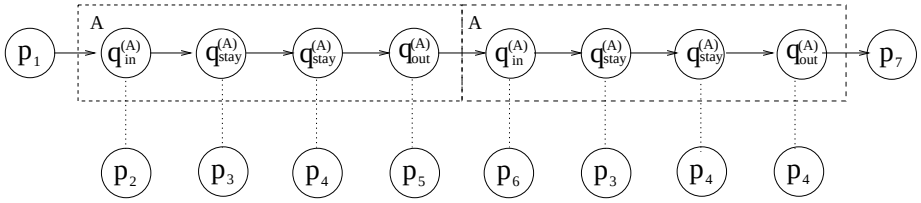


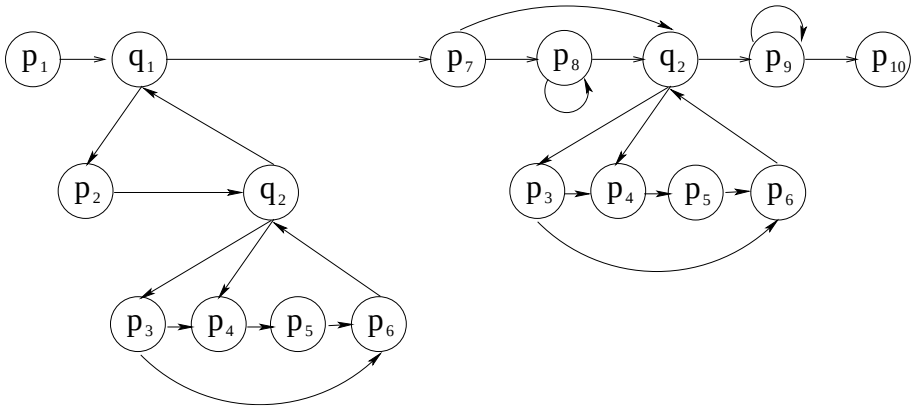
Figure 4.13: Example of state sequences for a simplified HHMM.

Figure 4.14(a) shows a more complex HHMM with a depth of three. The sub-model of q_2 has two occurrences in the model: as a child state of sub-model q_1 and at the root level of the model. When the sub-models are merged, the total number of observations

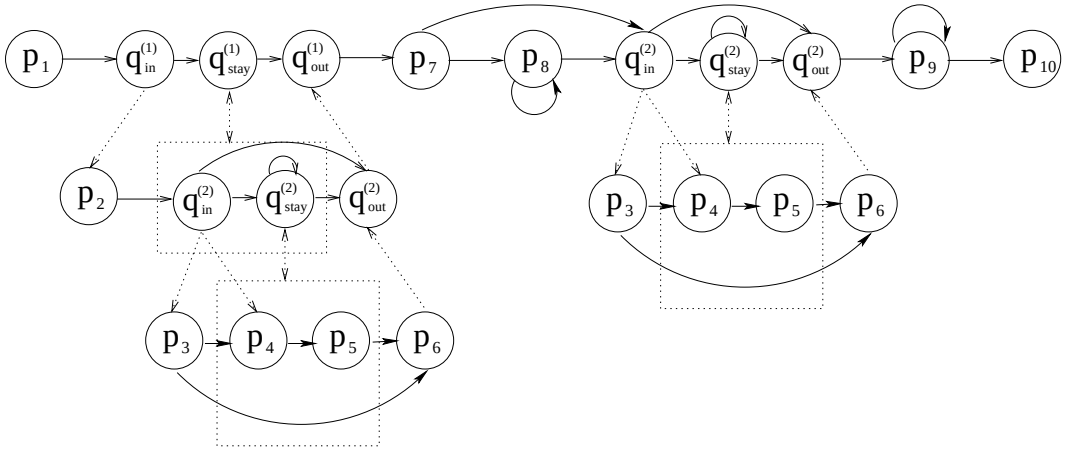
in sub-model q_2 is increased. Figure 4.14(b) illustrates the model after the simplification is applied. Sub-models are transformed into three states:

$$\begin{aligned} q_1 &\rightarrow \{q_{in}^{(1)}, q_{stay}^{(1)}, q_{out}^{(1)}\} \\ q_2 &\rightarrow \{q_{in}^{(2)}, q_{stay}^{(2)}, q_{out}^{(2)}\} \end{aligned}$$

State $q_{in}^{(1)}$ represents the entry activities for sub-state q_1, and the probability of $q_{in}^{(1)}$ occuring is dependent on state p_2. State $q_{stay}^{(1)}$ represents the inner transitions of q_1 and the observation probability is dependent on the two internal states of $q_{in}^{(2)}$ and $q_{stay}^{(2)}$.



(a)



(b)

Figure 4.14: A three level HHMM: (a) regular HHMM (b) simplified HHMM

Example: Transformed state sequence for the SHHMM. An example of state sequence for a three level SHHMM as shown in Figure 4.14(b). Consider a sequence of production

states:

$$\{p_1, p_2, p_3, p_6, p_7, p_8, p_4, p_4, p_5, p_6, p_9, p_{10}\}$$

After the model had transformed into a SHHMM, each internal state is represented by three sub-states as the status of the child states. Figure 4.15 illustrates how the state sequence would be labelled using the model from Figure 4.14(b), where the observational probability of $q_{in}^{(2)}$ is dependent on state p_3, and the observational probability of $q_{stay}^{(1)}$ is dependent on state $q_{in}^{(2)}$. ■

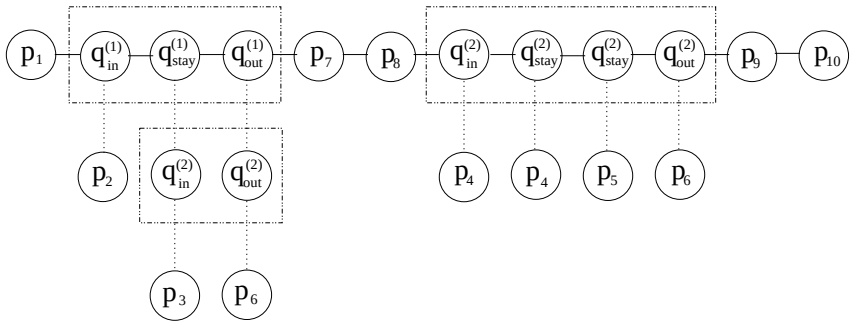


Figure 4.15: Example of state sequences for a three level SHHMM.

The detailed calculation of each sub-model is discussed in the next section.

4.3.2 Sub-model Calculation

As mentioned earlier, each internal state q_i has its child production states p_j resolved into one of three transformed states, $q_i \Rightarrow \{q_{in}^{(i)}, q_{stay}^{(i)}, q_{out}^{(i)}\}$. The transformation requires re-calculating the new observational and transition probabilities for each of these transformed states. The procedure to transform internal states is: 1) reform the transition matrix by including estimated values for additional transformed internal states; then 2) apply the Viterbi algorithm to estimate the state probabilities for the three transformed states.

1 **Reform transition probability** $\bar{A}^{(i)}$

Each internal state q_i is transformed into a new 3×3 transition probability matrix $\bar{A}^{(i)}$, which records the transition state for the transform matrix. The formula for

the estimated cells in $\bar{A}^{(i)}$ are:

$$\bar{a}^{(i)}_{in,stay} = \sum_{j=1}^{n_i} \pi_j \tag{4.10}$$

$$\bar{a}^{(i)}_{in,out} = \sum_{j=1}^{n_i} \frac{\pi_j}{2} \tag{4.11}$$

$$\bar{a}^{(i)}_{stay,stay} = \sum_{k=1,j=1}^{n_i,n_i} a_{k,j} \tag{4.12}$$

$$\bar{a}^{(i)}_{stay,out} = \sum_{j=1}^{n_i} \tau_j \tag{4.13}$$

$$\bar{a}^{(i)}_{out,stay} = a_{i,i} \tag{4.14}$$

where n_i is the number of child states for state q_i and the remaining cells of the transition matrix $\bar{A}^{(i)}$ are equal to zero.

Example: Consider the simple HHMM from Figure 4.10, where the model has a depth of two. The top level of the model contains three states $\{p_1, q_{\mathbf{A}}, p_7\}$, and the bottom level contains the production states for internal state $q_{\mathbf{A}}$. In this example the internal state $q_{\mathbf{A}}$ will be labelled as q_1. The transition matrix of the top level can be expressed as:

$$A = \begin{array}{c} \\ p_1 \\ q_1 \\ p_7 \end{array} \begin{array}{c} \begin{array}{ccc} p_1 & q_1 & p_7 \end{array} \\ \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & \boxed{a_{2,2}} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \end{array} \tag{4.15}$$

where the transition only involves three states. The boxed cell represents the location for the internal state which will be expanded further into a 3×3 matrix for those transformed states $\{q^{(1)}_{in}, q^{(1)}_{stay}, q^{(1)}_{out}\}$. The transition matrix for the internal state q_1 consists of the five production states $\{p_2, p_3, p_4, p_5, p_6\}$, and the matrix can

be expressed as:

$$A^{(1)} = \begin{array}{c} \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \end{array} \begin{array}{c} \begin{array}{ccccc} p_2 & p_3 & p_4 & p_5 & p_6 \end{array} \\ \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} \\ a_{5,1} & a_{5,2} & a_{5,3} & a_{5,4} & a_{5,5} \end{bmatrix} \end{array} \tag{4.16}$$

The transition matrix records transition probabilities for the child states of the internal state q_1, and the purpose of this matrix is to treat each internal state as an individual HMM. This allows application of the Viterbi search algorithm for the internal state q_1. To coordinate the sub-model of HMM with the three transformed states, the transition matrix is reformed into a transformed matrix for the internal state q_1:

$$\bar{A}^{(1)} = \begin{array}{c} \\ a_{in}^{(1)} \\ a_{stay}^{(1)} \\ a_{out}^{(1)} \end{array} \begin{array}{c} \begin{array}{ccc} a_{in}^{(1)} & a_{stay}^{(1)} & a_{out}^{(1)} \end{array} \\ \begin{bmatrix} 0 & a_{in,stay}^{(1)} & a_{in,out}^{(1)} \\ 0 & a_{stay,stay}^{(1)} & a_{stay,out}^{(1)} \\ a_{2,2} & 0 & 0 \end{bmatrix} \end{array} \tag{4.17}$$

The values of each element of $\bar{A}^{(i)}$ are calculated by Equations (4.10)-(4.14), and this can then be combined into a new transformed matrix of $\bar{A}$:

$$\bar{A} = \begin{array}{c} \\ p_1 \\ q_{in}^{(1)} \\ q_{stay}^{(1)} \\ q_{out}^{(1)} \\ p_7 \end{array} \begin{array}{c} \begin{array}{ccccc} p_1 & q_{in}^{(1)} & q_{stay}^{(1)} & q_{out}^{(1)} & p_7 \end{array} \\ \begin{bmatrix} a_{1,1} & a_{1,2} & 0 & 0 & a_{1,3} \\ a_{2,1} & 0 & a_{in,stay}^{(1)} & a_{in,out}^{(1)} & a_{2,3} \\ 0 & 0 & a_{stay,stay}^{(1)} & a_{stay,out}^{(1)} & a_{2,3} \\ 0 & a_{2,2} & 0 & 0 & a_{2,3} \\ a_{3,1} & a_{3,2} & 0 & 0 & a_{3,3} \end{bmatrix} \end{array} \tag{4.18}$$

The value in the second, third and fourth rows contain the transformed matrix for internal state q_1, and the first and fifth rows contain the original transition from Equation (4.15). The resulting matrix can be used to keep track of entering and leaving the sub-model. ■

2 **Apply the Viterbi algorithm to estimate the transformed observation value $\bar{B}^{(i)}$**

Each internal state q_i reforms a new $3 \times T$ observation matrix $\bar{B}^{(i)}$, which records the probabilities of the three transformed states. First, calculate the adjustment weight for each internal state $\bar{O}_t^{(i)}$ by summing up all the observation probabilities in each production state p_j:

$$\bar{O}_t^{(i)} = \sum_{j=1}^{n_i} O_{j,t}, \tag{4.19}$$

where time t corresponds to a position in the sequence, $O_{j,t}$ is the observation probability for state p_j at t, and n_i represents the number of production states for the internal state q_i. The transformed observation values are simplified to $\{\bar{b}_{in,t}^{(i)}, \bar{b}_{stay,t}^{(i)}, \bar{b}_{out,t}^{(i)}\}$, which are then given as the observation values for the three transformed states $\{q_{in}^{(i)}, q_{stay}^{(i)}, q_{out}^{(i)}\}$. The probability of entering state q_i at time t (i.e. production state $q_{in}^{(i)}$) is given by:

$$\bar{b}_{in,t}^{(i)} = \max_{j=1..n_i} \left[\pi_j \times O_{j,t}\right], \tag{4.20}$$

where π_j represents the transition probabilities of entering child state p_j. The second probability, that of staying in state q_i at t (i.e. production state $q_{stay}^{(i)}$), is given by:

$$\begin{aligned} \bar{b}_{stay,t}^{(i)} &= \max_{j=1..n_i} \left[A_{\hat{j}^*,j} \times O_{j,t}\right], \\ \hat{j} &= \arg\max_{j=1..n_i} \left[A_{\hat{j}^*,j} \times O_{j,t}\right], \end{aligned} \tag{4.21}$$

where $\hat{j}^*$ is the state corresponding to $\hat{j}$ calculated at the previous time $t-1$, and $A_{\hat{j}^*,j}$ represents the transition probability from state $p_{\hat{j}^*}$ to state to p_j. The third probability, that of exiting state q_i at time t (i.e. production state $q_{out}^{(i)}$), is given by:

$$\bar{b}_{out,t}^{(i)} = \max_{j=1..n_i} \left[A_{\hat{j}^*,j} \times O_{j,t} \times \tau_j\right], \tag{4.22}$$

where τ_j is the transition probability for leaving state p_j.

After Equations (4.20)-(4.22) have been calculated, the model then normalises the matrix and multiplies it by Equation (4.19) to estimate the value for $\bar{B}^{(i)}$:

$$\bar{B}^{(i)} = \frac{\bar{b}_{j,t}^{(i)}}{\sum_{j=1}^{n_i} \bar{b}_{j,t}^{(i)}} \times \bar{O}_t^{(i)} \qquad \text{for } j = 1, 2, 3, \text{ and } t = 1, 2, \ldots, T. \tag{4.23}$$

Example: Consider the simple HHMM from Figure 4.10. The observation matrix for each internal state is transformed into three states as previously mentioned. The observation matrix for the top level states can be expressed as:

$$B = \begin{matrix} p_1 \\ q_1 \\ p_7 \end{matrix} \begin{bmatrix} b_{1,1} & b_{1,2} & \dots & b_{1,T} \\ b_{2,1} & b_{2,2} & \dots & b_{2,T} \\ b_{3,1} & b_{3,2} & \dots & b_{3,T} \end{bmatrix} \tag{4.24}$$

where the state q_1 is an internal state that contains sub-model information. The probability distribution of the internal state q_1 is stored in the middle three rows in the observation matrix.

$$\bar{B} = \begin{matrix} p_1 \\ q_{in}^{(1)} \\ q_{stay}^{(1)} \\ q_{out}^{(1)} \\ p_7 \end{matrix} \begin{bmatrix} b_{1,1} & b_{1,2} & \dots & b_{1,T} \\ b_{in,1}^{(1)} & b_{in,2}^{(1)} & \dots & b_{in,T}^{(1)} \\ b_{stay,1}^{(1)} & b_{stay,2}^{(1)} & \dots & b_{stay,T}^{(1)} \\ b_{out,1}^{(1)} & b_{out,2}^{(1)} & \dots & b_{out,T}^{(1)} \\ b_{3,1} & b_{3,2} & \dots & b_{3,T} \end{bmatrix} \tag{4.25}$$

The value for the transformed state is calculated using Equations (4.19)-(4.23), where the values in row 1 and row 5 remain the same. ■

Each internal state is calculated by a bottom-up algorithm using the values from Equations (4.19)-(4.22), where the lower levels of the tree are calculated first to provide information for upper level states. Once all the internal states have been calculated, the process need only use the top-level of the tree to solve the alignment problem for the sequence. This means the model can be approximated by a linear HMM for the final Viterbi search process [Viterbi, 1967].

4.4 Sequence Likelihood

Consider an observation sequence $O = \{O_1, O_2, \dots, O_T\}$. The task is to determine the most likely sequence ($\hat{S}$) of states $S = \{s_1, s_2, \dots, s_T\}$ within the model:

$$\hat{S} = \arg\max_S P(S|O, \lambda). \tag{4.26}$$

From Bayes' Theorem, the formula of Equation (4.26) can be expressed as:

$$P(S|O, \lambda) = \frac{P(O|S, \lambda)P(S, \lambda)}{P(O|\lambda)}, \tag{4.27}$$

where the probability of observations, given that sequence, is:

$$\begin{aligned} P(O|S,\lambda) &= P(s_1 s_2 \ldots s_t = S_i | O_1 O_2 \ldots O_T, \lambda) \\ &= b_{s_1}(O_1) b_{s_2}(O_2) \ldots b_{s_T}(O_T). \end{aligned} \tag{4.28}$$

The probability of such a state sequence S can be written as:

$$P(S,\lambda) = \pi_{q_1} a_{s_1,s_2} a_{s_2,s_3} \ldots a_{s_{T-1}s_T}. \tag{4.29}$$

The joint probability of O and S is written as:

$$P(O|\lambda) = \sum_{\forall S} P(O|S,\lambda) P(S|\lambda) \tag{4.30}$$

$$= \pi_{s_1} b_{s_1}(O_1) a_{s_1,s_2} b_{s_2}(O_2) a_{s_2,s_3} \ldots b_{s_T}(O_T) a_{s_{T-1},s_T}. \tag{4.31}$$

For every internal state s_i and a given observation sequence O, the model evaluates $\arg\max_S P(O|S,\lambda^{(i)})$ by finding the best path with a sequence of states that only involves child states. The model uses the Viterbi search algorithm to determine the best path for each internal state (mentioned in Section 4.3). After the values have been calculated, the internal state provides three estimated values for each observation to match the transformed states ($\bar{b}^{(i)}_{in}(O_t)$, $\bar{b}^{(i)}_{stay}(O_t)$, $\bar{b}^{(i)}_{out}(O_t)$).

The computational cost of applying the Viterbi search algorithm to each of the internal states is $O(M^2T)$ calculations, where M represents the total number of child states for that internal state and T is the length of the observation sequence. The cost of applying the algorithm to the entire model is approximately $O(NM^2T)$, where N represents the number of states in the model. In the worst case the number of child states M will approach the total number of states N, and thus the worst case computation cost for an HHMM can be simplified to be in the order of $O(N^3T)$ calculations.

Let $g(\theta^i)$ represent the observation probability distribution:

$$g(\theta^i) = b_{q_i}(O_i) \qquad \text{for } i = 1, 2, \ldots, T \tag{4.32}$$

where the probability of the observations given a sequence is:

$$P(O|S,\lambda) = \prod_{i=1}^{T} g(\theta^i) \tag{4.33}$$

Taking the logarithms of both sides, the log probability of the observations equals the sum of $f(\theta^i)$, where $f(\theta^i) = \log g(\theta^i)$:

$$\log P(O|S,\lambda) = \sum_{i=1}^{T} \log g(\theta^i), \tag{4.34}$$

$$= \sum_{i=1}^{T} f(\theta^i). \tag{4.35}$$

Definition 4.1 *Assume $\theta^1, \ldots, \theta^N$ to be i.i.d. Then*

$$E_N[f(\theta)] = \frac{1}{N}\sum_{i=1}^{N} f(\theta^i) \tag{4.36}$$

with a standard deviation of

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(f(\theta^i) - E_N[f(\theta)])^2 \tag{4.37}$$

Theorem 4.1 *Assume that $\{\theta^{(j)}\}_{j=1}^{\infty}$ is a stationary, irreducible, aperiodic Markov chain with limiting distribution π, and that $Var[f(\theta)] < \infty$. Let a positive integer M be given, and suppose that a geometric subsample of size M is used to estimate $E[f(\theta)]$. Formally, let $v_1, \ldots, v_{M+1}$ be i.i.d. geometric variates with mean p_1^{-1}, $j_k = \sum_{i=1}^{k} v_i$, $N = \sum_{i=1}^{M+1} v_i - 1$, $e_1 = (1/N)\sum_{j=1}^{N} f(\theta^{(j)})$, and $e_3 = (1/M)\sum_{j=1}^{M} f(\theta^{(j_k)})$. Then $Var_N[e_3] \geq Var_N[e_1]$, for every $N \geq M$, and $Var[e_3] \geq Var[e_1]$. [MacEachern and Peruggia, 2000].*

For a HHMM, when two or more different sub-states contain the same probabilistic distributions, then the combined sub-states will produce more stable predictions. This is expressed by the following lemma.

Lemma 4.1 *For any $M \leq N$ which contains the same distribution of X and Y, $var(\bar{X}) \geq var(\bar{Y})$.*

Proof: Given two normal distributions of X and Y:

$$X \sim Normal(M, p)$$

$$Y \sim Normal(N, p)$$

with $M \leq N$. According to Equation (4.37), the variance of expected values can be written as:

$$\begin{aligned} var(\bar{X}) &= \frac{s^2}{M-1} \\ var(\bar{Y}) &= \frac{s^2}{N-1} \end{aligned}$$

Hence, for any $M \leq N$, the $var(\bar{X}) \geq var(\bar{Y})$. ■

4.5 Structural Issues of HHMMs

HHMMs were introduced to manage the complexity of data structures occurring in many extraction tasks. For the HHMM, the model itself makes use of hierarchical structure and the potential benefit is that sub-state information can be reused.

When reusing sub-state information extracted from pre-tagged source information, there are situations where different sub-models have the same parent state. This research takes a simple approach and combines those sub-models, reusing the information within them. While this method is simple and provides some increase in accuracy and stability, it may also obfuscate the importance of sub-states, which should carry greater weight during alignment. Consider the tree structure representation of a HHMM in Figure 4.16, where different sub-states that share the same parent state are labelled state 8. Notice that state $8''$ contains internal state 8 with sub-states 21, 22 and 23, whereas the other occurrences of state 8 contain sub-states of 17, 18, 19 and 20. In order to resolve this issue the system needs to be able to identify when merging of such sub-states has occurred and rename the internal state 8 to a different name.

Due to the Markov property, the transition probabilities depend only on the current event, and are independent of past events; therefore the HHMM itself does not provide a strong relationship between long sequential events. For example, the transition probabilities depend only on the current state, and this leads to the side effect of losing relationship information; not from the previous state, but from past states.

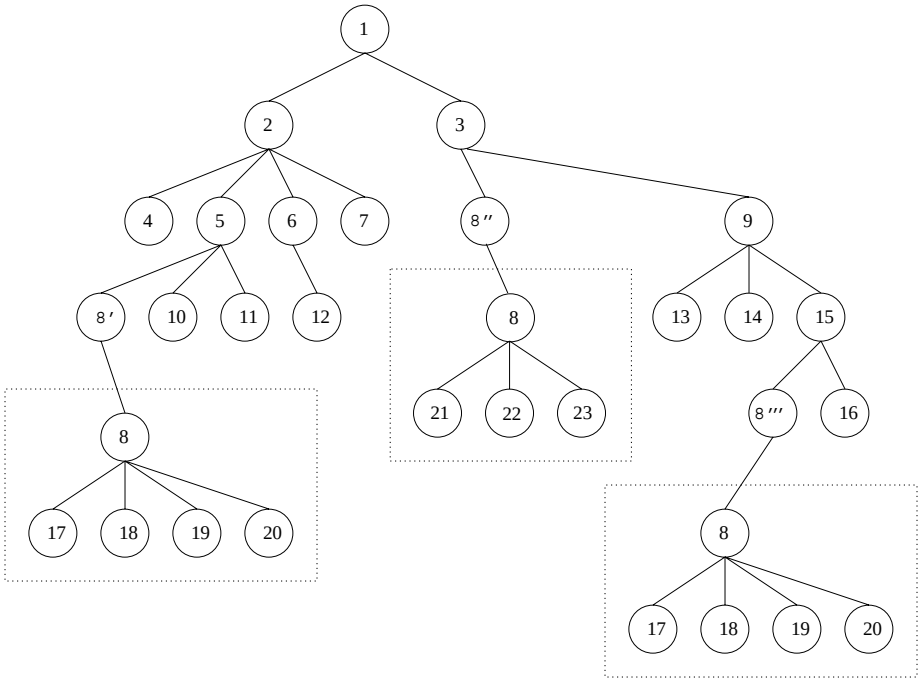


Figure 4.16: Example of different sub-states

Another issue encountered when building models of natural language text is recursive patterns within a text. During the HHMM building process described above, the model could be applied to text that contains recursive structure. For example, a language text with the following rules:

$$S \rightarrow ab$$
$$S \rightarrow aSb$$

The Markov model is ill-suited to handle recursive situations, due to the independence property—which states that the next state is only dependent upon the current state. Thus an HHMM is unlikely to correctly identify—and resolve—recursive behaviour. This is one instance where a CFG model may be better suited for this task. Furthermore, in keeping with the required property of independence between sub-models, the current model building process does not merge any repeating sequences within an existing sub-model; in other words it never descends more than one layer of recursion. Therefore the HHMM is ill-suited to model recursively structured data.

The main applications of HHMMs are:

- solving sequential problems that may contain hierarchical structures, such as text chunking, which marks up the logical structure of the sentence.
- characterising states and sub-state information. For example, in the reference tagging problem the MSHHMM can make use of repeated structure under the state *author* and the state *editor*, such as the state *first name*, state *middle name*, state *last name* and the state *con*, to provide more accurate prediction for states that contain names.
- identifying new sequences as states or sub-models, such as identifying words or terms which do not appear in the training data (unseen terms).

4.5.1 Comparison between HHMM and SCFG model

A very similar modelling technique to HHMMs is Stochastic Context-Free Grammars (SCFG) [Jelinek and Lafferty, 1991]. An interesting way to compare an HHMM to a SCFG is to look at the similarities and differences in the algorithms used during their application. First, consider the task of determining the probabilities of a particular observation sequence against the model. An HHMM uses a standard forward algorithm to accomplish this, as explained above, while a SCFG uses a similar method called the inside algorithm. Moreover variants of each of these algorithms, called the Viterbi algorithm and the CYK algorithm respectively, can be applied to the model to calculate the best path or most likely sequence. Finally the forward and backward variables of the HHMM are used in conjunction with EM methods to re-estimate the probability parameters in much the same way as inside and outside variables are used in the SCFG. Thus the inside-outside algorithm [Lari and Young, 1990] for a SCFG is analogous to the forward-backward algorithm of HHMMs.

However, there is one critical difference between the two algorithms. While the forward-backward algorithm used in HHMMs is computationally straightforward, in that each calculation performed is independent of any other calculation within the model, the

inside-outside algorithm uses dynamic calculations based upon sub-model values, calculated recursively. This dynamic, recursive nature means that the computational cost of the inside-outside algorithm is substantially greater.

	HHMM	SCFG
optimal alignment	Viterbi	CYK
$P(X\|\lambda)$	forward	inside
EM parameter estimation	forward-backward	inside-outside
memory complexity	$O(NT)$ [Fine et al., 1998]	$O(NT^2)$ [Lari and Young, 1990]
time complexity	$O(N^3T)$ [Fine et al., 1998]	$O(N^3T^3)$ [Lari and Young, 1990]

Table 4.2: Information for HHMM and SCFG models

Table 4.2 shows that more memory and computation time are required for the SCGF model than the HHMM. Lari [1990] suggests that the solution is to decrease the time complexity by introducing more non-terminals to reduce the re-estimation process in the inside-outside algorithm.

4.6 Processes Involved in HHMM Modeling

The aim of the HHMM is to extract or mark up some target information in data as part of an automated process. The HHMM process generally consists of two stages.

- Training, which involves performing some calculation on a set of hand processed data to extract information and structures, then using that information to create the model.

- Testing, which makes use of this model by applying it to unprocessed data to extract or mark up the intended target information.

This section discusses these two processes and presents flow charts to illustrate them.

4.6.1 The Training Process

The goal of the HHMM training process is to form an extraction model that is capable of exploiting hierarchical structure (in contrast to HMMs). In general, parameter estimation for HMM uses the Baum-Welch re-estimation algorithm, where the model is given initial seed parameters, and trained iteratively with the Baum-Welch algorithm to maximise $P(O|\lambda)$ as discussed in Section 3.

There are various types of pre-tagged data that can be easily accessed on the Internet. For example, the Penn Treebank[1] has large quantities text pre-tagged with syntactic and semantic information for natural language tasks. This thesis concentrates only on training from pre-tagged data to estimate model parameters.

```
(A (N (F T. )(L Moloney, ) ) (N (F A. C. ) (L. Lea, ) )
(C and )(N (F C. ) (L Kowalchuk. ) ) )(T Manufacturing
and packaged goods. ) (ED (N (C In )) (N (F G. H. ) (L
Castle, ) ) (C editor, ) ) (BT Profiting from a Geographic
Information System. ) (PU GIS World, Inc., ) (AD Fort
Collins, CO, ) (Y 1993. )
```

Figure 4.17: Example of pre-processed data for the reference tagging task

In order to build an HHMM, the user provides pre-processed data. Consider the reference tagging example previously shown in Section 4.1, where the tag associated with each word begins with an open parenthesis "(" followed by the tag. Table 4.3 provides a complete list of the possible types of tag. Figure 4.17 illustrates a sequence contains three internal states, such as *author* (A), *editor* (ED) and *name* (N), where the state *name* has the shared internal state, and the remaining states are production states as mentioned in Figure 4.13. The training process for a MSHHMM includes simplification, which involves transforming the internal state into three transformed states. For example, state *name* (N)

[1]The Penn Treebank Project, http://www.cis.upenn.edu/~ treebank/home.html

Types of tag	tag
author	A
title	T
booktitle	BT
volume	V
number	NUM
pages	P
month	M
year	Y
editor	ED
publisher	PU
address	AD
name	N
first name	F
last name	L
conjugate	C

Table 4.3: Types of tag for the reference tagging task

can be transformed into $name_{in}^{(3)}$, $name_{stay}^{(3)}$ and $name_{out}^{(3)}$ as described earlier in Section 4.3.

Training consists of taking pre-tagged data to form the hierarchical model, and using the hierarchical structure proposed in Section 4.1 to build the model. Figure 4.18 shows the flowchart of the HHMM training process. Once the training process has completed, it outputs the model parameters λ. The steps involved are as follows:

1 **Model Initialisation:** Collect pre-processed training data from the user.

2 **Data Conversion:** Select the type of conversion for the input text. A simple approach for this task is not to convert any input text, but keep it in its original form. In practice, keeping the text in its original form can be a problem when the amount of text is large. The system can reduce the memory required to store the input text by converting it in some way, such as transforming observations into

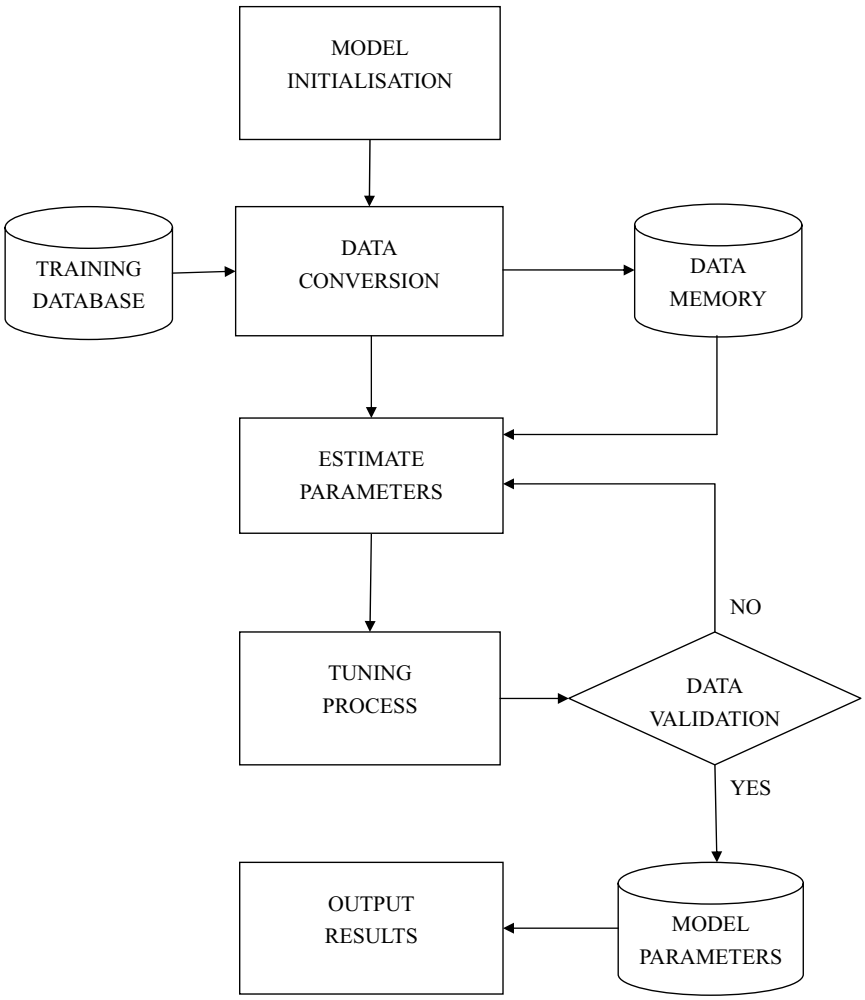


Figure 4.18: Flowchart of training process

patterns, then searching for identical patterns within the training data (a process described in Chapter 6).

3 **Estimate Parameters:** Model parameters are formed by calculating the transition probability distribution and the observation probability distribution. The system records each transition path for each state and updates the model parameters λ when the process has finished.

4 **Tuning Process:** Model parameters are tuned using model smoothing. Smoothing provides probabilistic estimation for occurrences of unseen events. This technique will be further described in Chapter 6.

5 **Output Results:** The final step outputs the validated model parameters to a text file. The output data provides the initial transition probability (π), the transition probability distribution (A), and the observation probability distribution (B) for each state, along with observation symbols that has occurred in that state. The structure file contains the hierarchical structure of the model with a list of state relationships between parent and child states.

4.6.2 The Testing Process

Testing the HHMM involves using the model to extract untagged data, assigning tag information to the testing data and producing output with tagged data. The goal of this process is to extend knowledge about the sentence by providing meaningful information about the input data.

Consider, as an example, the following observation sequence:

> Nahm, U. Y., and Mooney, R. J. 2000. Using information extraction to aid the discovery of prediction rules from texts. In *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining (KDD-2000) Workshop on Text Mining*, 51-58.

The task is to determine the most likely sequence ($\hat{S}$) of states (A, T, EM, V, NUM, P, M, Y, ED, PU, AD, N, F, L, C), where the model can be expressed as:

$$\hat{S} = \arg\max_{S} P(S|O, \lambda). \tag{4.38}$$

The output is:

> (A (N (L Nahm,) (F U. Y.,)) (C and) (N (L Mooney,) (F R. J.))) (Y 2000.) (T Using information extraction to aid the discovery of prediction rules from texts.) (EM In Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining (KDD-2000) Workshop on Text Mining,) (P 51-58.)

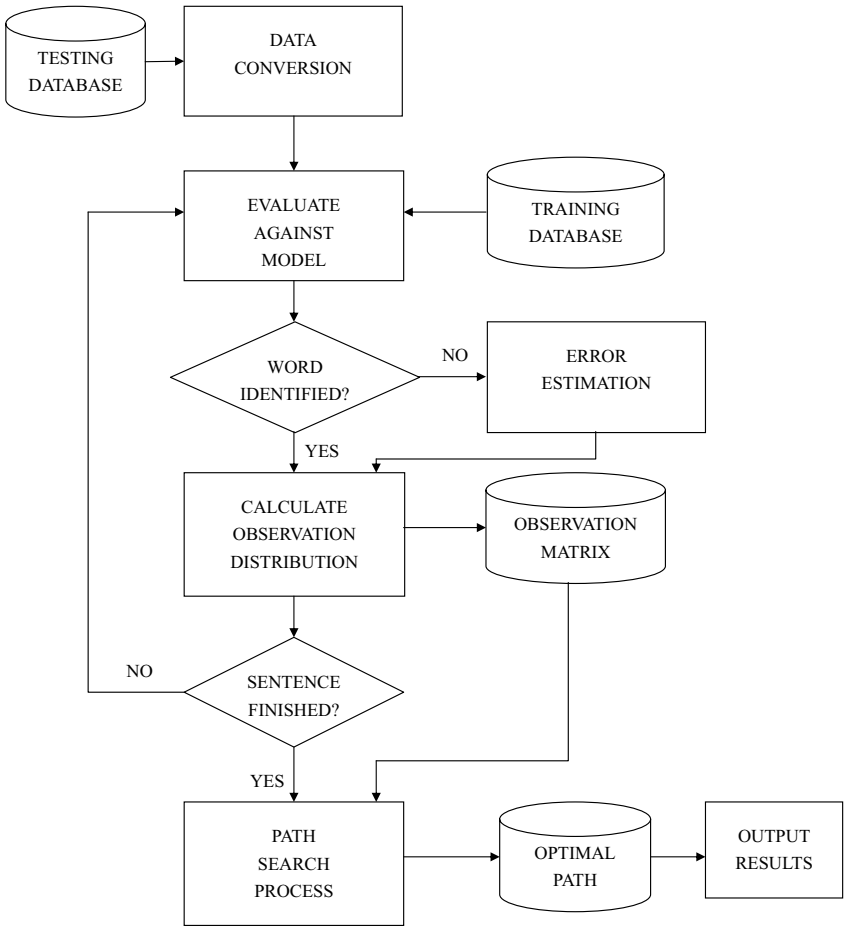


Figure 4.19: Flowchart of the HHMM testing process

Figure 4.19 shows the flowchart of the HHMM testing process, where the system starts with the test sequence, reforms the transition matrix $\bar{A}$, then reforms the observation matrix for each internal state ($\bar{B}^{(i)}$) to create a HMM of transformed states. This is followed by a standard Viterbi search process. Finally, the system outputs the results. The processes involved are list as follow:

1 **Data Conversion:** This step is the same as in training. It takes input text and converts it into user pre-defined format as required for each task.

2 **Evaluate Against Model:** The system then evaluates against the trained model by matching each word with an existing state. When words are not identified by

the trained model, the system estimates the probability using the error estimation method.

3 **Error Estimation:** The error estimation method is used to predict the probability of the unseen observation. The method and the threshold value are evaluated during the training process.

4 **Calculate Observational Distribution:** Once all the observation values are collected for each state, the system normalises the observation matrix B so that the probability of the matrix for each column is summing up to 1 and stores the data.

5 **Path Search Process:** For every internal state q_i and observation sequence O, the single best state sequence can be found by using the Viterbi algorithm, which maximises $P(O|Q, \lambda)$.

6 **Output Result:** Output the result to a file. In the reference tagging task, the output for a test sequence is labelled with reference tags, such as:

> (A P. Borras, J. C. Mamou, D. Plateau, B. D. Tallot.) (T Building user interfaces for database applications: the O/sub 2/ experience.) (BT SIGMODRecord (ACM Special Interest Group on Management of Data),) (V 21) (NUM (1):) (P 32-38,) (M Mar.) (Y 1992.)

where each output result is compared against the actual answer to estimate the extraction accuracy of the model.

4.7 Summary

This chapter outlines the process by which HHMMs, and in particular the Merged and Simplified hierarchical hidden Markov models, are built, and explains the difference between HHMMs and non-hierarchical HMMs. It started by explaining the fundamental properties of an HHMM, where the information in sub-states with the same parent can be re-used, and then followed with an explanation of the merging technique used for the MSHHMM to increase this information reuse. The idea of using copied sub-states to

restore the Markov property is then discussed. In this way the model can both adhere to the Markov property and also take advantage of the benefits of repeated sub-states. This chapter also describeed a new transformation for each internal state:

$$q_i \Rightarrow \{q_{in}^{(i)}, q_{stay}^{(i)}, q_{out}^{(i)}\}. \tag{4.39}$$

along with how the transformed states were calculated. The transformed internal state represents the status of the sub-model, whether entering, staying or exiting the sub-model. Section 4.4 describes how variance of expected values can be reduced by increasing the number of observations available for a state. This leads to a gain in stability for the model. Thus a second benefit of the merging process is to increase the number of observations per state. Chapter 5 will develop these ideas and apply them to two different applications and Chapter 6 will suggest some techniques to improve the model.

In summary, the fundamental properties of the HHMM require the sub-models to be independent of each other, and the current state can only depend on previous states. This research proposes that when repeated sub-models occur, the information within those states should be shared, as part of a 'merging process', whereupon a larger number of observations are available for each sub-state. In order to restore the Markov property, the system provides duplicate copies for combined sub-states, so that those sub-states can still retain the Markov property of being independent of each other without needing to remember which sub-states have been merged.

CHAPTER 5

Applications

Chapter 4 discusses the theoretical aspects of the merged and simplified hierarchical hidden Markov model (MSHHMM) and its construction method. This chapter will describe the practical application of the MSHHMM to two problems:

- reference tagging, and
- text chunking.

The results are compared to those of the regular tree structured hierarchical hidden Markov model (RHHMM) and the tree structured HHMM with the simplified *in-stay-out* states (SHHMM). The results are also compared to a simple HMM to determine the effects of using a hierarchical structure as compared to the linear structure of the HMM. Section 5.1 introduces the evaluation process that will be used for these two tasks. Section 5.2 then evaluates the performance of all four types of model when applied to the reference tagging task. The reference tagging task contains only a small number of layers in the hierarchical structure. The second evaluation moves to a task that involves a higher number of layers in the hierarchy structure, the text chunking task, as explored in Section 5.3.

5.1 Background to the Evaluation Process

When evaluating any extraction task, the performance of a model can be gauged using two standard measures: *Precision* (P) and *Recall* (R) [Rijsbergen, 1979]. Precision mea-

sures the proportion of the extracted tags that were correct, while recall measures the proportion of the correct tags that were extracted. Although the best situation is a model that exhibits both high precision and high recall, in practice these two measures are often inversely related. In other words, achieving high precision often requires having a smaller result set and thus lower recall. Conversely high recall is easier to achieve with a larger result set but only at the sacrifice of precision. Because of the relation between precision and recall, we can derive a combined performance measure called the *F-measure* [Rijsbergen, 1979], which is the geometric mean of recall and precision. The formula for the F-measure is:

$$F = \frac{2 \times P \times R}{P + R} \tag{5.1}$$

where F $\in [0, 1]$, with 1 being the best score.

There are two possible ways of averaging this measure, called the *macro-average* and *micro-average* [Rijsbergen, 1979]. The macro-average is a measurement based on each individual state. The average performance is calculated by summing over all individual states, where each state carries equal weight. The micro-average value is a measurement of global effectiveness. The average performance is calculated by globally summing all of the individual observations, where each state carries a weight in proportion to its size. In this research the model is measured by micro-average, in other words based on total effectiveness rather than that of individual states. The results of all four models; the MSHHMM, the SHHMM, the RHHMM and the HMM, are compared using the micro-averaged F-measure.

During the evaluation process a significance test is performed to determine the validity of comparing the results of two different models. There are two types of issue that can be addressed by the test of significance:

- the probability that a relationship exists, and
- the strength of said relationship.

In general the significance test can be obtained by re-sampling methods such as cross-validation. The $k \times n$ cross-validation test [Bouckaert and Frank, 2004] is used to compare

the performance estimates, where the evaluation process repeats n-fold cross-validation k times. During the evaluation process, a Type I error (false positive) occurs when a positive result is reported where none really exists, and a Type II error (false negative) occurs when a negative result is reported when it was really present. By using a $k \times n$-fold cross-validation the results are more resilient to Type I and Type II errors for each individual data set.

A N-fold cross validation involves partitioning the input data into n subsets. The first $n - 1$ sub-sets are put aside as training data leaving one sub-set as testing data. The cross-validation process is then repeated n times, with each of the n sub-sets used exactly once as the validation data.

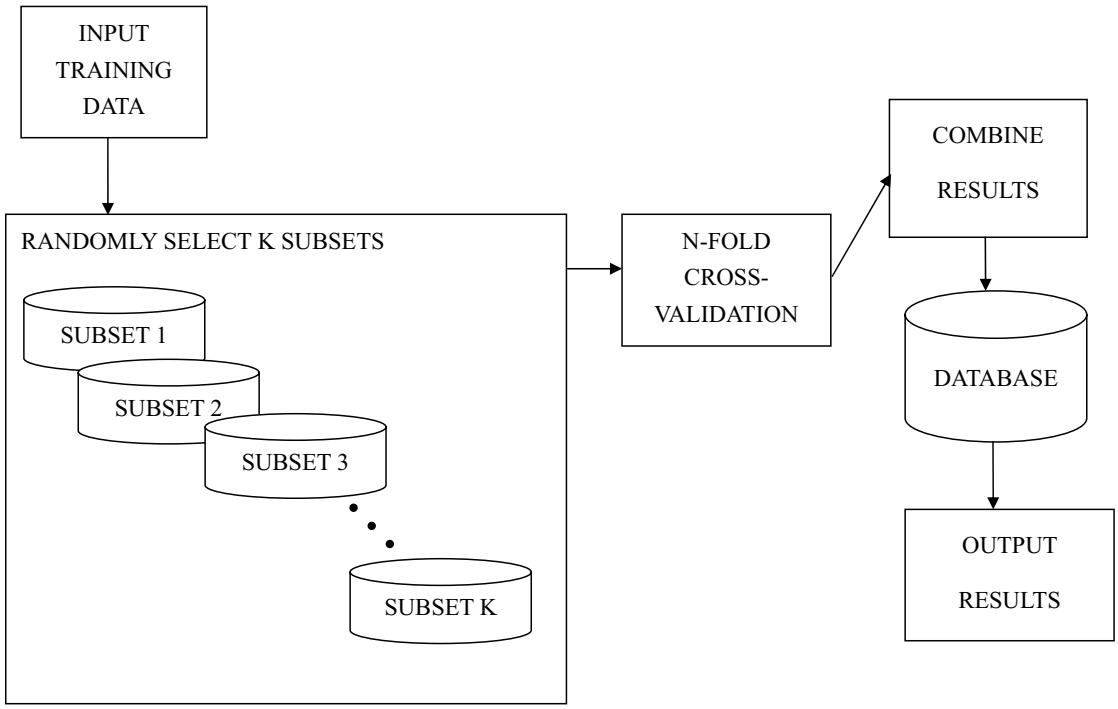


Figure 5.1: Evaluation process

In this research, $k \times n$ cross-validation is used to compare performance among different models. Figure 5.1 illustrates the evaluation process of $k \times n$ cross-validation. The process begins by randomly selecting k sub-sets of data, then each sub-set is used in n-fold cross

validation against the evaluation model. The amount of training data for each fold is proportional to $\frac{n-1}{n}$. The purpose of this evaluation is to examine the amount of training data that is required for a stable model and also to measure the variance of accuracy.

The significance test [Wild and Seber, 1995] applied during this research, a $t-test$, is used to test whether there are differences between two sets of data (d_1 and d_2). The t-test hypothesis states that the extraction accuracy from the two sets of data is the same at a 95% confidence interval. Formal calculation is then performed in order to determine the threshold for accepting or rejecting this hypothesis. The test formula is:

$$t_0 = \frac{\bar{x}_{d_1} - \bar{x}_{d_2}}{se(\bar{x}_{d_1} - \bar{x}_{d_2})} \tag{5.2}$$

The two separate sets of data should be independent to each other. Therefore the standard error ($se(\bar{x})$) is calculated as:

$$se(\bar{x}_{d_1} - \bar{x}_{d_2}) = \sqrt{\frac{sd_{d_1}^2}{N_{d_1}} + \frac{sd_{d_2}^2}{N_{d_2}}} \tag{5.3}$$

where sd_{d_1} is the standard deviation for the data set d_1, and N_{d_1} is the total number of results. In a 5×10-fold cross-validation, the evaluation process produces 50 results for each model. The value of t_0 for a 95% confidence interval is thus equal to

$$t_{49}(0.025) = 2.0009$$

so the significance test will reject the t-test hypothesis when the absolute value of $|t_0|$ is greater than 2.0009.

5.2 Reference Tagging

This section applies the various HMMs to the reference tagging task and compares the results. The section begins by explaining how reference tagging has been applied. It then explains how the reference tagging problem is tested against the four model variations—MSHHMM, SHHMM, RHHMM and HMM. The section concludes with the evaluation of these models on the same set of data supplied by Seymore [1999].

The reference tagging task involves parsing a document in order to extract the information required to determine what other documents are related to this one—information which can then be used, for instance, to automatically generate web links to related online documents. This linking also enables the reader to gain other information about the related documents, such as the publisher or year of publication. For example, the reference section of an online research paper allows for relevant *linking information* [Bergmark, 2000] to be extracted. These reference sections often appear at the end of a document under a section heading, such as *Reference*, *Bibliography*, or *List*.

Reference

[1] T. Moloney, A. C. Lea, and C. Kowalchuk. Manufacturing and packaged goods. In G. H. Castle, editors, Profiting from a Geographic Information System. GIS World, Inc., Fort Collins, CO, 1993.

Figure 5.2: Example of a raw reference form

Example: Figure 5.2 is an example of a reference from the end of a research paper, where the problem is to segment the reference into useful fragments such as *author*, *title*, *editor*, *booktitle*, *publisher*, *address* and *date*. Both *author* and *editor* contain a person's name as their observation. Figure 5.3 represents the HHMM for the reference entry, and shows the hierarchical similarity between the states *author* and *editor*, where both can be the parent of a shared sub-tree rooted at the state *name*. The model also displays the splitting of the single state *name* into a sequence of sub-states; *first name*, *middle name*, *last name* and *con* (which represents connecting observations such as "and" and "In"). This splitting should allow the structure to model more correctly the underlying sentence structure, and the purpose of this splitting process is to define a model structure that more accurately reflects the sentence structure.

Table 5.1 summarises the states and observations in the example. By identifying which parts of the hierarchical structure have the potential for repeated sub-states the system can promote sub-model reuse, thus hopefully increasing the efficiency of the model created

during training.

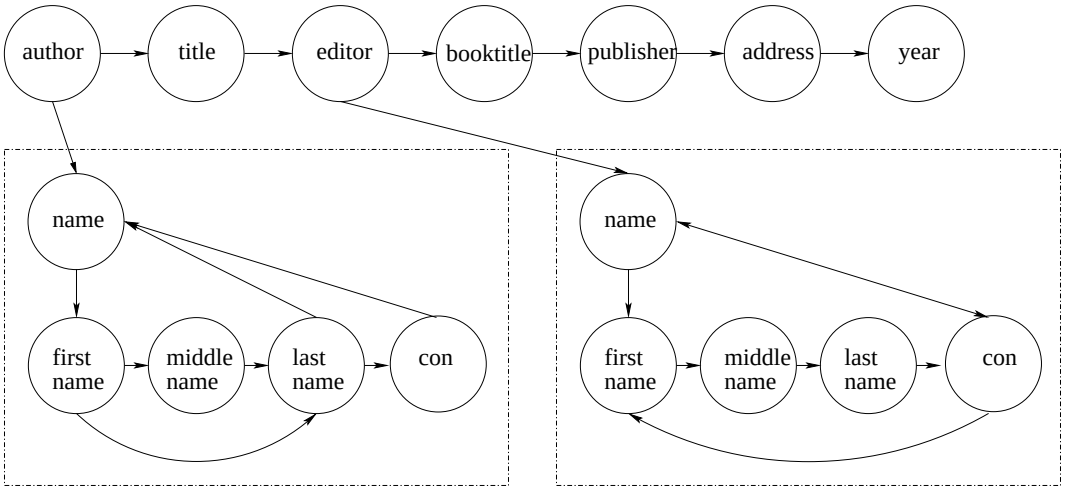


Figure 5.3: Example of the reference-tagging task

state	observations
first name	T. C. G.
middle name	C. H.
last name	Moloney, Lea, Kowalchuk. Castle,
con	and In editors,
title	Manufacturing and packaged goods.
booktitle	Profiting from a Geographic Information System.
publisher	GIS World, Inc.,
address	Fort Collins, CO,
year	1993.

Table 5.1: Summary of states and observations for Figure 5.3

■

It is worth mentioning that, because of the small depth of the underlying structure of this task, it was anticipated that the HMM would, in some cases, perform better than the HHMMs. The HMM should have the advantage in terms of both accuracy measure,

as the simple linear model is less likely to misidentify state boundaries as compared to HHMMs, and in performance time, as the algorithms run on the linear HMM are significantly less complex than those of HHMMs. To explain the possible confusion over state boundaries, consider the example in Table 5.1. An HMM, having correctly matched "C. Kowalchuk." to a name state need only consider whether "Manufacturing" is a name or a title state, whereas the HHMMs would have to consider whether it were a new name, another part of the name they were currently matching, or a new title. This extra complexity should offer another opportunity where HHMMs could misidentify a state leading to extra processing time and lower accuracy. However, the HMM did not exhibit these properties, as evidenced in the following results.

5.2.1 Evaluation

In order to explore the modelling potential of the HHMM for reference tagging, this section evaluates the various HHMMs and compares their performance in terms of accuracy (as measured by the micro-average F-measure), stability (as measured by the standard deviation of F-measure) and processing time (measured in seconds). Each of these factors was further plotted against different volumes of training data (as measured in number of sentences). The models were evaluated on a dataset from Seymore [1999], which contains 600 references with 13 types of pre-inserted tags: *title*, *author*, *institution*, *location*, *note*, *editor*, *publisher*, *date*, *pages*, *volume*, *journal*, *booktitle*, and *technical report*.

It is already known that the volume of training data has a significant influence on the stability of the model. In general, the model's stability increases with the size of the training data. The first experiment tested the impact of varying volumes of training data on accuracy over the four types of model (MSHHMM, SHHMM, RHHMM, HMM). Figure 5.4 shows a graph of the micro-averaged F-measure for four types of models when evaluated using 5×10-fold cross-validation and with the volume of training data ranging from 60 to 600 sentences. The results show that the MSHHMM achieves better accuracy than the other three models when the training data is less than 300 sentences. For the same range of training data volume, the HMM has the lowest accuracy. As the number of

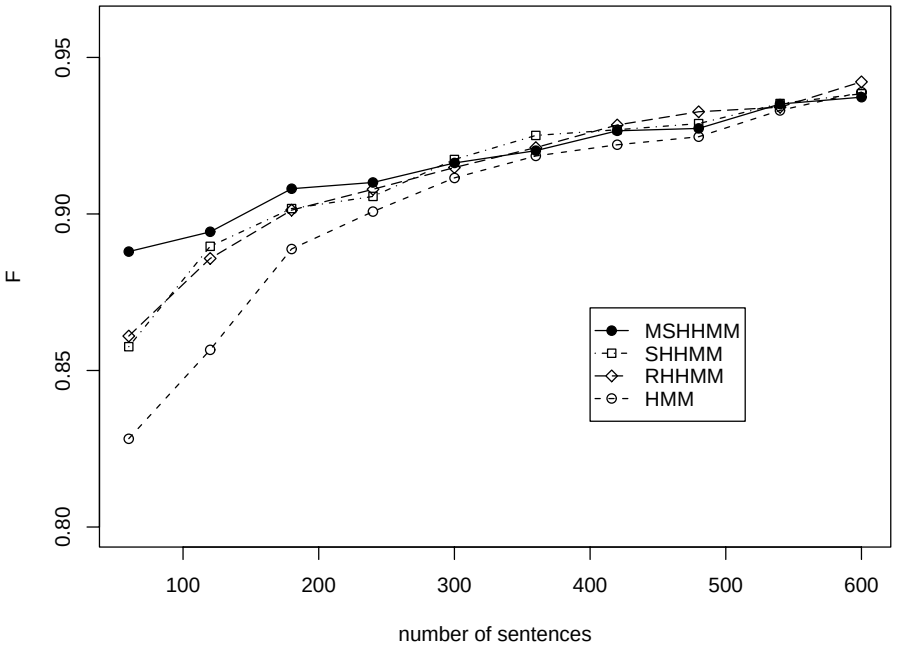


Figure 5.4: Graph of F-measure for 5×10-fold cross validation

training sentences increases, the accuracy for the different models converges to a similar performance. The results for individual states can be found in Appendix A.2.

In reality, training data is sometimes very expensive to obtain. Furthermore, many of the interesting applications of HHMM involve small training and testing data or situations where resources (memory andor processing time) are limited, such as speech recognition [Chien, 1999].

To further explore the ability of the MSHHMM to perform even over limited volumes of training data, experiments with training data volumes ranging from 10 to 100 were carried out. The results, given in Figure 5.5 show the MSHHMM performs better than the other three models when applied to very small volumes of training data. For example, the MSHHMM achieves 0.783 on micro-averaged F-measure with only 10 sentences, compared to the results of 0.678, 0.706 and 0.634 for RHHMM, SHHMM and HMM respectively.

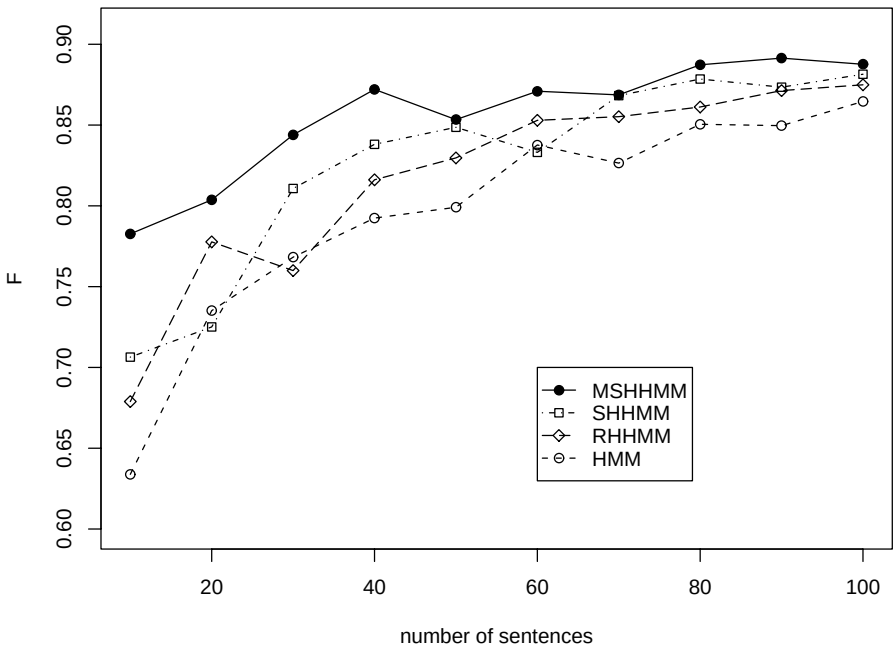


Figure 5.5: Graph of F-measure for 5×10-fold cross validation for small volumes of training data

Figure 5.6 shows the value of standard deviation on F-measure for MSHHMM, SHHMM, RHHMM and HMM for different volumes of dataset when again evaluated using 5×10-fold cross-validation. The results show that as the volume of dataset increases, the accuracy of all models increase, and the variances decrease, as they approach the point where they converge. Note though, that the MSHHMM achieved better accuracy and less variance with smaller amounts of training data than the three other models—SHHMM, RHHMM and HMM.

Table 5.2 shows the t-test for three pairs of comparisons; MSHHMM verses RHHMM, MSHHMM verses SHHMM and MSHHMM verses HMM. When the volume of training data is 60 sentences the MSHHMM has strong evidence of better performance over the RHHMM, as the value of t_0 is equal to 2.392 (greater than 2.0009 under 95% confidence

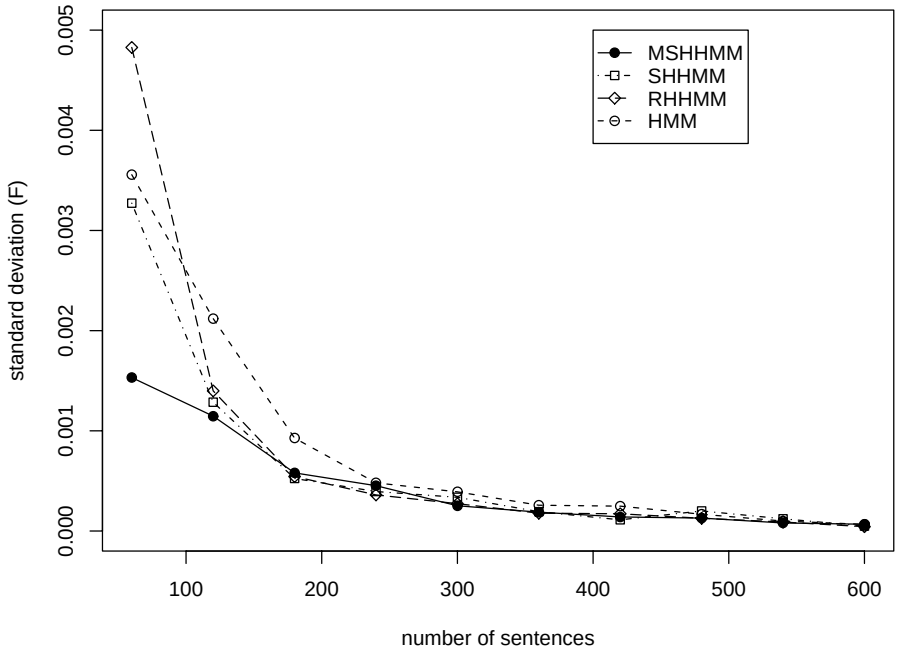


Figure 5.6: Graph of standard deviation on F-measure for 5×10-fold cross validation

interval). The results show the MSHHMM displays strong evidence of better accuracy than other models on sparse data (when there is an insufficient data during the training process). The remaining results show weak evidence that the MSHHMM achieved better performance than RHHMM, SHHMM and HMM for volumes of training data greater than 300 sentences.

For each cross validation, $\frac{1}{10}$ was used as testing data and the remaining $\frac{9}{10}$'s were used as training data. Figure 5.7 represents the average processing time of testing (in seconds) for the 5×10-fold cross validation. The x-axis represents the number of sentences that have been used for 10-fold cross validation. The tests were carried out on a dual P4-D computer running at 3GHz and with 1Gb RAM. The results show that, in general, there is little processing time difference in using a MSHHMM rather than a HMM for the processing task, due to there being only a small amount of sub-model sharing between different sub-states. This is an interesting result given that the algorithms used on the MSHHMM,

	MSHHMM vs RHHMM		MSHHMM vs SHHMM		MSHHMM vs HMM	
no. of sentences	t_0	$P-value$	t_0	$P-value$	t_0	$P-value$
60	2.392	0.019	3.099	0.003	5.9238	6.572^{-8}
120	1.191	0.236	0.662	0.509	4.661	1.084^{-5}
180	1.443	0.152	1.349	0.181	3.504	0.001
300	0.451	0.653	−0.327	0.748	1.328	0.188
360	−3.362	0.718	−1.761	0.081	0.569	0.570
420	−0.744	0.459	−1.151	0.880	1.600	0.113
480	−2.351	0.021	−0.600	0.550	1.093	0.277
540	0.553	0.581	−0.076	0.937	1.076	0.285
600	−3.289	0.001	−0.719	0.474	−0.979	0.330

Table 5.2: Significances test for different size of datasets

including the various improvements, are more complex than those of the RHHMM.

Table 5.3 lists the extraction accuracies for the different models when applied to the reference tagging task. 400 sentences were used for training and 200 sentences for testing. The HMM achieved an extraction accuracy of 0.902, while the regular HHMM had poorer performance only managing an accuracy of 0.714. Although the data exhibited a shallow hierarchy, the MSHHMM provided better extraction accuracy than HMM with a F-measure of 0.912. However, Seymore [1999] used a HMM with distantly-labelled data to set the model parameters, thus improving the performance of the model to achieve an extraction accuracy of 0.929. The performance of the MSHHMM can be further improved by applying the techniques of structure formation, pattern generalisation and smoothing. After the techniques are applied the MSHHMM achieves an extraction accuracy of 0.958, which is the best result surpassing even the Seymore model. This improvement technique will be discussed later in Chapter 6.

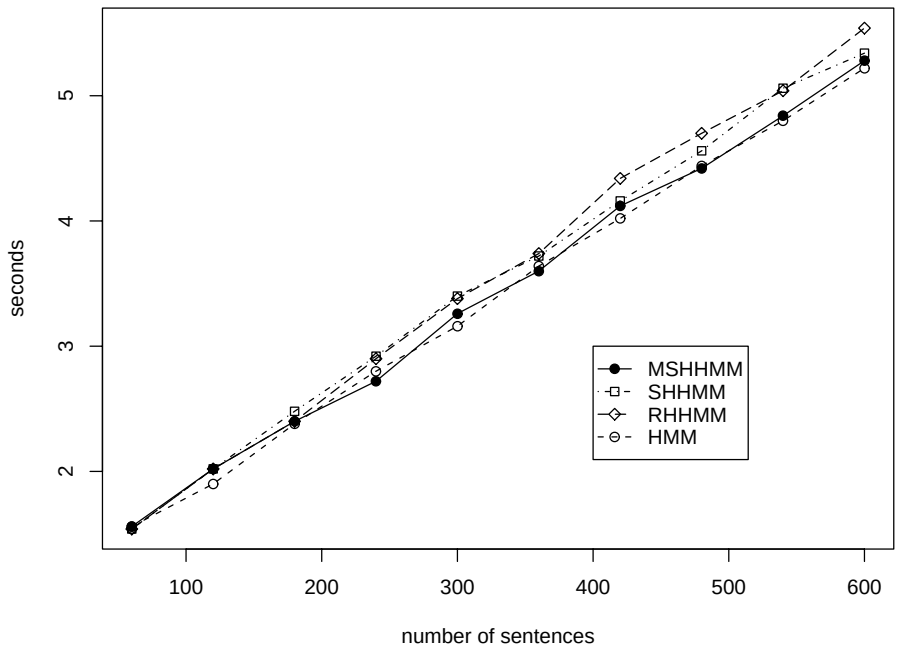


Figure 5.7: Average processing time of reference tagging

5.2.2 Accuracy Performance on Individual Sentences

In order to uncover what properties of sentences favour one model above another, as series of experiments were conducted using a set of test sentences where each sentence was uniquely identified by a number. The accuracy performance of one model when attempting to resolve each sentences is plotted against the performance of a second model on the same sentence—thus quantifying which model had superior performance when tagging that sentence. In theory, sentences with more complex and repeated structure should be better handled by a hierarchical model, while simplier, shorter sentences should lend themselves to the linear HMM. To examine what type of sequence is better suited for what model, 400 sentences were used for training and 200 sentences for testing.

Figure 5.8 illustrates the extraction performance for 200 test sentences for two different models—HMM and MSHHMM. The numbers on Figure 5.8 are the unique identifier given to each of the 200 test sentences. The sentences that are located above the diagonal line

Model	F-measure
RHHMM	0.714
HMM	0.902
SHHMM	0.912
MSHHMM	0.914
Seymore [Seymore et al., 1999]	0.929
MSHHMM(structure formation + pattern generlisation + smoothing)	0.958

Table 5.3: F-measure for different models

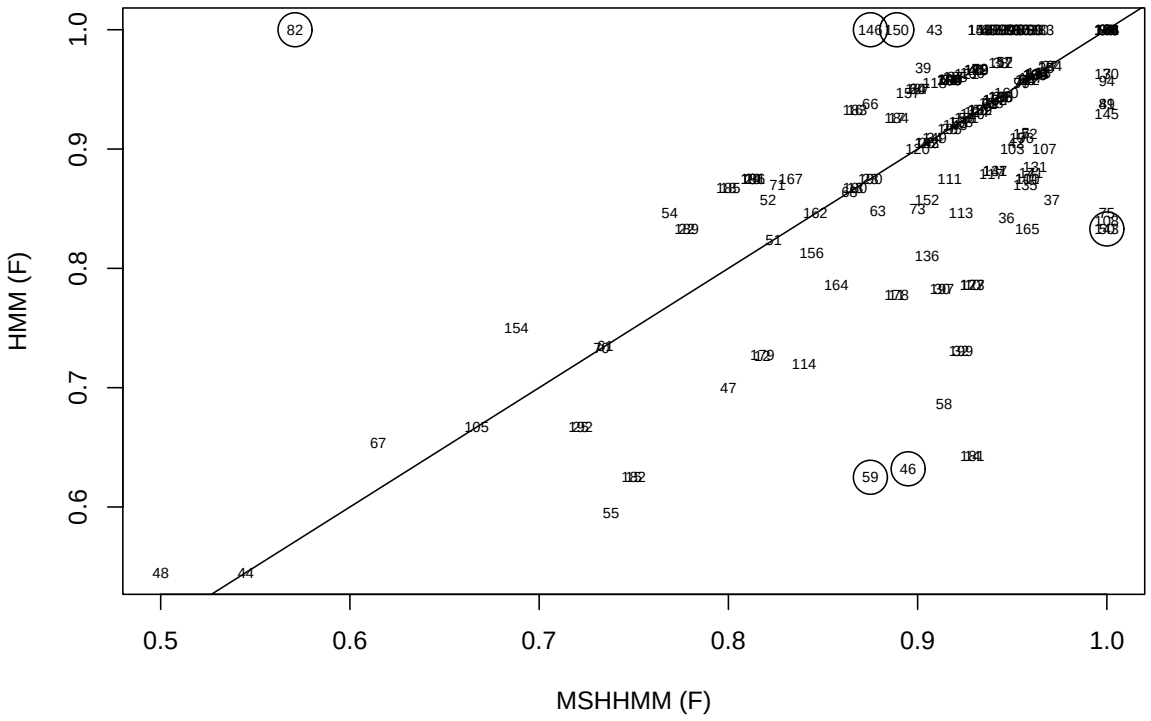


Figure 5.8: F-measure for MSHHMM against HMM

indicate the test sentences that resulted in better extraction accuracy under the HMM. The test sentences 82, 146 and 150 achieved 1.0 of F-measure when processed by the HMM, but performed more poorly under the MSHHMM. On the other hand, the test sentences 46, 59 and 143 performed poorly under the HMM but could be extracted accu-

rately under the MSHHMM.

82	(author L. Press.) (title Software export from developing nations.) (booktitle IEEE Computer,) (volume 26) (number (12):) (pages 62,) (month Dec.) (year 1993.)
146	(author G. Gardarin.) (title Relational database and knowledge base systems.)
150	(author M. Stonebraker.) (title The INGRES Papers.) (publisher Addison-Wesley,) (address Reading, MA,) (year 1986.)
59	(author G. Wright and P. Ayton.) (title Eliciting and Modelling Expert Knowledge.) (booktitle DecisionSupport Systems,) (volume 3) (number (3):) (pages 13-26,) (year 1987.)
46	(author S. Jaenischen, G. Hommel, and C. H. A. Koster.) (title Methodisches programmieren: Algorithmenentwicklung durch schrittweise Verfeinerung.) (publisher DeGruyter Verlag,) (address Berlin,) (year 1983.)
143	(author P. A. Bernstein, V. Hadzilacos, and N. Goodman.) (title Concurrency Controland Recovery in Database Systems.) (publisher Addison-Wesley,) (address Reading, MA,) (year 1987.)

Table 5.4: Six test sentences from reference tagging task

Table 5.4 contains the six testing sentences mentioned, where the top three sentences represents the three sentences above the diagonal line from Figure 5.8, and the bottom three sentences represents the three sentences from below. In the reference tagging task, hierarchical structure only occurs during the states *author* and *editor*. The results indicated that the HMM had better extraction accuracy when encountering shorter names within the state *author*, such as "(author L. Press.)". Meanwhile the MSHHMM performs better when sentence contains a longer length of the state *author*, such as "(author P. A. Bernstein, V. Hadzilacos, and N. Goodman.)". The hierarchy labelling present in training sequence enhances the extraction accuracy for the MSHHMM as it can make

use of both the child state, which controls the observation, and also has the advantage of using the parent state, which provides a strong coupling relation between the child states.

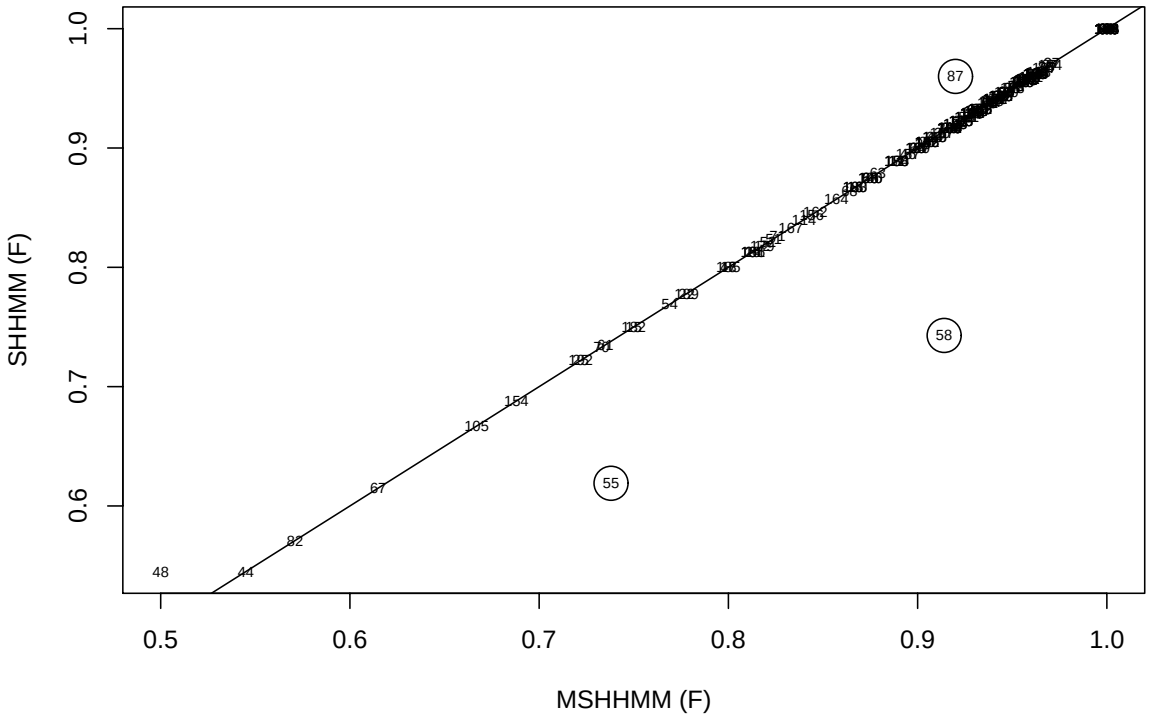


Figure 5.9: F-measure for MSHHMM against SHHMM

In the reference tagging task, there is only small amount of data that could be reused, where the state name combined observation from both the *author* and *editor* states. Figure 5.9 shows the results of F-measure for the SHHMM against the MSHHMM. Most sentences lie on the diagonal line, as the extraction accuracy is the same for both of the methods. However, the sentences numbered 55 and 58 favour the MSHHMM while the sentence 87 instead favours the SHHMM.

Table 5.5 lists the three test sentences of 55, 58 and 87. Note that sentences 55 and 58 have both an author and an editor field. Given this fact, and the model's high accuracy when tagging these sentences, the results suggest that the merging process allows the MSHHMM to benefit from the repeated name structure.

55	(author U. Hohenstein, L. Neugebauer, G. Saake, and H.-D. Ehrich.) (title Three-Level - Specification of Databases using an extended Entity-Relationship Model.) (editor In R. R. Wagner, R. Traunm?uller and H. C. Mayr, editors,) (booktitle Informationsbedarfsermittlungand -analyse f?ur den Entwurf von Informationssystemen,) (pages pages 58-88,) (address Berlin, Germany,) (year 1987.) (publisher Springer-Verlag.)
58	(author D. J. Penney and J. Stein.) (title Class Modification in the GemStone Object - OrientedDBMS.) (editor In N. Meyrowitz, editor,) (booktitle Proceedings of the ACM Conferenceof Object-Oriented Systems, Languages and Applications (OOPSLA),) (pages pages 111-117,) (address Orlando, Florida,) (month Oct.) (year 1987.)
87	(author S. J. Finkelstein, M. Schkolnick, and P. Tiberio.) (title DBDSGN - A physicaldatabase design tool for system R.) (booktitle IEEE Data Eng. Bull.,) (volume 5) (number (1),) (month Mar.) (year 1982.)

Table 5.5: Three test sentences from reference tagging task

Figure 5.10 plots the relationship of F-measure for RHHMM against MSHHMM on 200 test sentences. The graph shown the MSHHMM provides better extraction accuracy than RHHMM, where most of sentences lie on or below the diagonal line. There is only one sentence that achieved better extraction accuracy under the RHHMM. Here is that sentence:

> (editor T. W. Olle, H. G. Sol, and A. A. Verrijn-Stuart, editors.) (title Information SystemsDesign Methodologies: A Comparative Review.) (publisher North-Holland/IFIP,) (address Amsterdam, The Netherlands,) (year 1982.)

In sentence number 48, the reference starts with state *editor* and doesn't contains an *author* state at all—a unique case within this dataset. While the RHHMM has a lower F-measure—compared with other sentences—it still has a higher extraction accuracy of

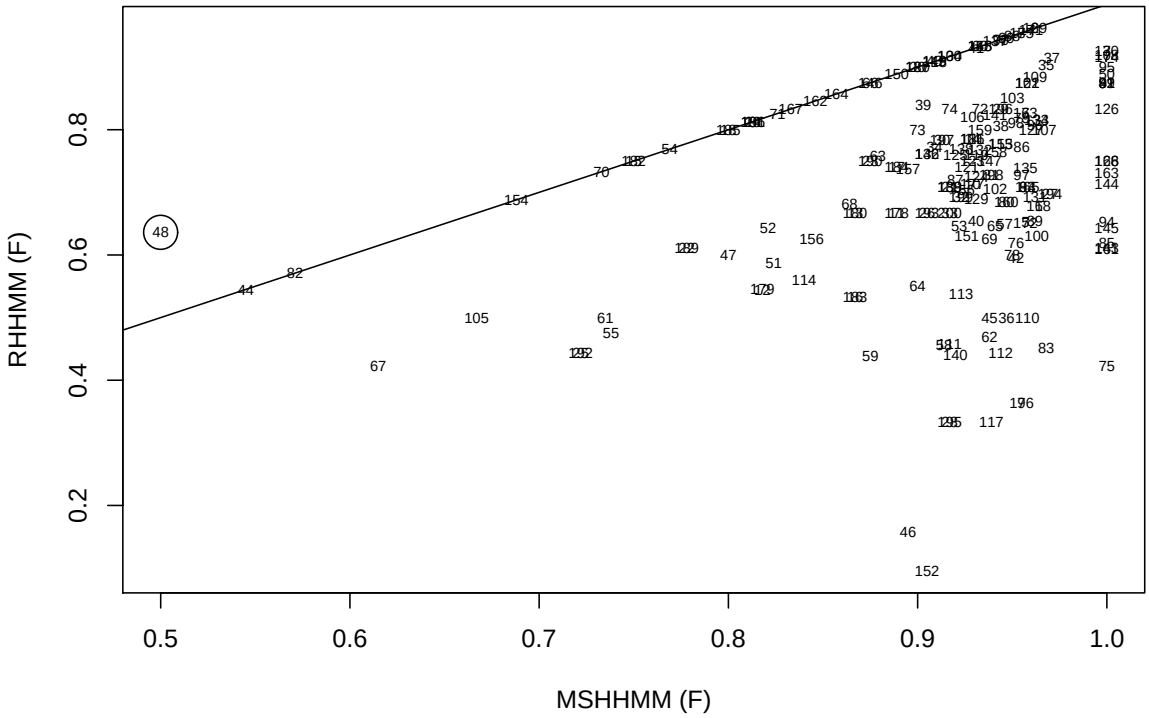


Figure 5.10: F-measure for HHMM against MSHHMM

0.636 compared to 0.500 for the MSHHMM. This is one case where the simplification technique applied to the MSHHMM, which results in tighter coupling between the child states of an internal state, results in poorer performance. While both models initially mis-identify the names as being from the *author* state, the RHHMM more easily 'breaks out' when it encounters the "editors." observation moving quickly to a correctly identified *editor* state. The MSHHMM, meanwhile, has a higher chance of incorrectly indentifying the observation "editors." due to its position in the child states of the *author*.

5.2.3 Discussion

The results above show that the MSHHMM typically outperforms either of the other two HHMM models, in terms of accuracy, stability and processing time. Furthermore it suggests that the MSHHMM can also outperform the HMM when there is a lim-

ited volume of training data—at least in accuracy. For larger training data sets, the MSHHMM, at worst, performs comparably to the HMM. This evidence would suggest that the MSHHMM presents a better 'general' model for tasks such as reference tagging. This advantage is due to two optimisations or improvements over the standard HHMM; Merging and Simplification.

Merging involves finding two or more states which display the same sub-model structure, and then merging the sub-models, with the resulting combined sub-model shared between the parent states. In the experiment above, the MSHHMM, by way of its hierarchical nature, can take advantage of the repeated structure in the *editor* and *author* states by merging the probabilistic information and observations within them into a combined sub-model under the single shared internal state *name*.

The overall results show that the MSHHMM provides better extraction accuracy, stability and processing performance than the RHHMM's. Furthermore the MSHHMM often performs the same as, or better than, the HMM.

It is again worth mentioning that when training data is limited, the MSHHMM has the most significant increase in performance, as compared to the other models—SHHMM, RHHMM and HMM. Remember that training data sparsity is one of the fundamental problems in modelling natural language, and so any techniques that increases the accuracy of a MSHHMM under these circumstances is most desirable.

However, the results also show that, when the volume of the training data is increased, the benefits of using a MSHHMM for the reference tagging task decrease. Eventually all of the models converge at a certain accuracy and variance. Moreover the results for the SHHMM do not show any significant advantage in using the simplification improvement during this task despite the reasoning that it might. This is due to the reference tagging task only having one level of hierarchy, with the only repeated structure being a person's name—a repeated hierarchy, which while well-defined, tends to contain only three states. These two properties result in little opportunity for the simplification and merging of

repeated sub-models.

The results shown by the individual analysis of sentences reveal that certain properties of a sentence favour one model above another. For example, a sentence with a repeated structure of longer lengths typically is processed more accurately by the MSHHMM. This is because it can reuse this repeated structure to merge sub-model information. Furthermore, by focusing on the boundary observations that demark entry and exit points to a sub-sequence, the model gains accuracy by having tighter coupling between the parent and child states.

The final point to discuss regards the seemingly poor performance of the HMM when applied to this task. It was earlier mentioned that because of the shallow hierarchy and simple information it was believed the HMM should perform well on this task. However the results show the HMM is often outperformed by the HHMM. The cause of this discrepancy was investigated and it was found that the HMM was more easily confused about the boundary between name information (such as *editor* and *author*) and title information and that, once it incorrectly identified a first name as a title, it had no way to recover from this error. In contrast, the MSHHMM, due to the advantage of more observations for the entry state to a person's name, was less susceptible to making the identification error in the first place. Even if the MSHHMM did identify the wrong state it was able to recover, by use of its hierarchical nature, and correctly match the remainder of the reference. This investigation uncovered another interesting benefit of using a hierarchical approach rather than a linear one. It also highlights that, although its algorithms are more complex, a MSHHMM can still have a lower processing cost due to it being able to more correctly match the testing data, thus avoiding the extra processing time caused by a model attempting to match a testing sentence once the model has already misidentified a state.

5.3 Text Chunking

A mathematical model can be used to identify syntactic roles in text chunking for the purpose of propositional analysis. The aim of text chunking is to divide a sentence into non-overlapping part-of-speech phrases, for instance, noun phrases and verb phrases. Due to the nesting nature of such phrases, this task involves modelling a much deeper hierarchy than that shown in the previous application.

Noun phrases are often used for extraction and retrieval purposes, and it is often beneficial to explore text documents in terms of these phrases rather than individual words. For instance, searching for the noun phrase "stock market" may be more useful compared to search for the single word "stock". This approach can be useful for tasks such as document analysis and document indexing. The subject of a text document can often be distilled from a set of noun phrases. Thus, if the noun phrases of the text are collected, they can provide a better understanding of the text. Noun phrases are used in text chunking, to avoid having to develop a complete parse tree of the text, a process which can prove to be very difficult and computationally expensive. Research has shown that a shallow parsing tree such as that applied during text chunking can extract enough information for the basic understanding of text. Church [1988], for example, proposes a noun phrase extractor that makes estimates of the position at each noun phrase within a sentence according to the two probability matrices: the starting noun phrase matrix, and the ending noun phrase matrix. There are also more recent approaches, such as shallow parsing [Thollard and Clark, 2002], where the algorithm finds the most likely chunking sequences and uses them as a preliminary step to full parsing.

Text chunking involves producing non-overlapping segments of low-level noun groups. The system uses clause information to construct the hierarchical structure of text chunks, where clauses represent the phrases within the sentence. Clauses can be embedded in other clauses but cannot overlap one another. Furthermore, each clause contains one or more text chunks.

Consider a sentence from a CoNLL-2004[1] corpus:

> (S (NP He__PRP) (VP reckons__VBZ) (S (NP the__DT current__JJ account__NN deficit__NN) (VP will__MD narrow_VB) (PP to_TO) (NP only__RB #__# 1.8_CD billion_D) (PP in_IN) (NP September__NNP)) (O .__.)).

The part-of-speech tag associated with each word is attached with an underscore. The clause information is identified by the S symbol and the chunk information is identified by the remaining symbols: *NP* (noun phrase), *VP* (verb phrase), *PP* (prepositional phrase) and O (null complementizer). The brackets are in Penn Treebank II style[2]. The sentence can be re-expressed in terms of its part-of-speech tags:

> PRP VBZ DT JJ NN NN MD VB TO RB # CD D IN NNP

The part-of-speech tags are the inputs for the text chunking task. This is done so that the system can minimise the computational cost involved in learning a large number of observation symbols. This approach maximises the efficient use of the training data by learning the syntactic pattern that underlies the words rather than the words themselves. The part-of-speech tag is determined by using tagging software, such as Brill's part-of-speech tagger (available from the Microsoft research home website[3]).

Figure 5.11 shows the HHMM tree representation for the text chunking task. This example involves a hierarchy with a depth of three. Note that the state *NP* appears in two different levels of the hierarchy, a common occurrence given the nested nature of the phrases. In order to build a HHMM the sentence shown above must be restructured as:

> (S (NP PRP) (VP VBZ) (S (NP DT JJ NN NN) (VP MD VB) (PP TO) (NP RB # CD D) (PP IN) (NP NNP)) (O .))

where the model makes no use of the word information contained in the sentence. Once the training data has been converted to the input format it is used to train the tree-

[1]The 2004 Conference on Computational Natural Language Learning, Boston, MA, USA, 2004, *http://cnts.uia.ac.be/conll2004*

[2]The Penn Treebank Project, http://www.cis.upenn.edu/~ treebank/home.html

[3]http://research.microsoft.com/~ brill/

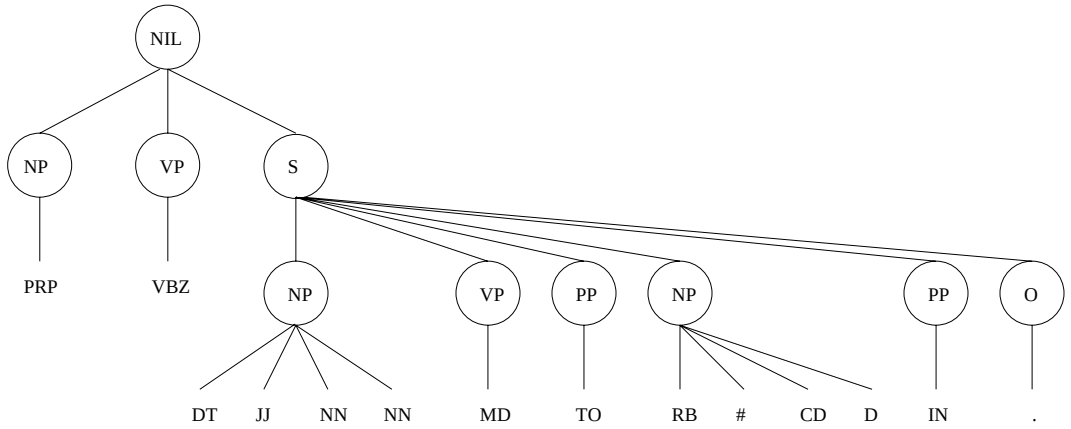


Figure 5.11: HHMM for syntax roles

structured HHMM.

The second HHMM, the simplified HHMM (SHHMM), is created during testing, by transforming the children of each internal state S_i into three production states, $S_{in}^{(i)}$, $S_{stay}^{(i)}$ and $S_{out}^{(i)}$, as described in Chapter 4. Finally, the merging technique is applied in this research are applied to the RHHMM to generate the MSHHMM.

5.3.1 Evaluation

When an English sentence is converted into a part-of-speech sequence, the number of observation symbols reduces dramatically. In order to explore the stability of the various HHMMs, the following set of experiments were conducted. First, the affect of the volume of training data was evaluated. Second, the performance of each of the three models was analysed on sentences of different lengths. The models were evaluated by analysing the results of performing the text chunking task on the data from CoNLL-2004. The dataset contains 8936 training sentences and 1671 test sentences.

As shown earlier, the amount of training data dramatically affects prediction accuracy. When developed from training data with a limited number of observation symbols, the resulting model suffers. In the text chunking task the number of observation symbols is

the number of part-of-speech tags in the training data.

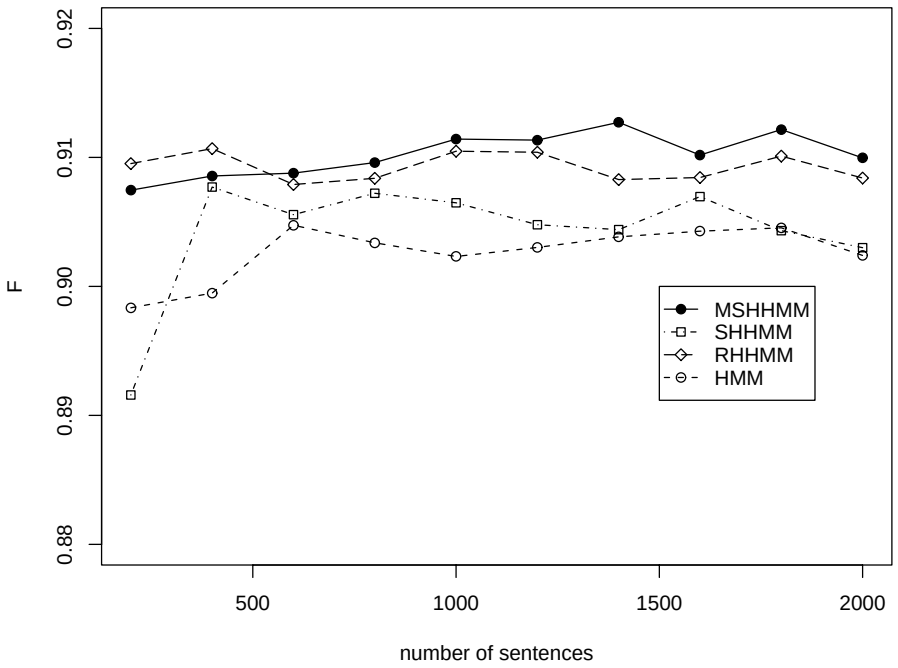


Figure 5.12: Overall results of F-measure for text chunking task

Figure 5.12 plots the graph of micro-averaged F-measures for 5×10-fold cross-validation for different volumes of dataset ranging from 200 to 2000, for each of the models. In Figure 5.12, the MSHHMM achieves the best performance, due to the large amount of repeated structure within the text chunking data. Figure 5.14 plots the standard deviations for each subset when evaluated using the four models (MSHHMM, SHHMM, RHHMM and HMM). The MSHHMM generally has higher extraction accuracy according to micro-averaged F-measure on volumes of data ranging from 200 to 2000, compared to the other models. During this range the MSHHMM also has a lower standard deviation in its results implying a more stable model. Results for some of the individual states are listed in Appendix A.3.

There is a limited number of observations within each state for the text chunking task,

due to there being only 38 types of part-of-speech tags for the set of data. Later results, during significant testing, show there is no strong evidence of difference on F-measure between the MSHHMM and the RHHMM.

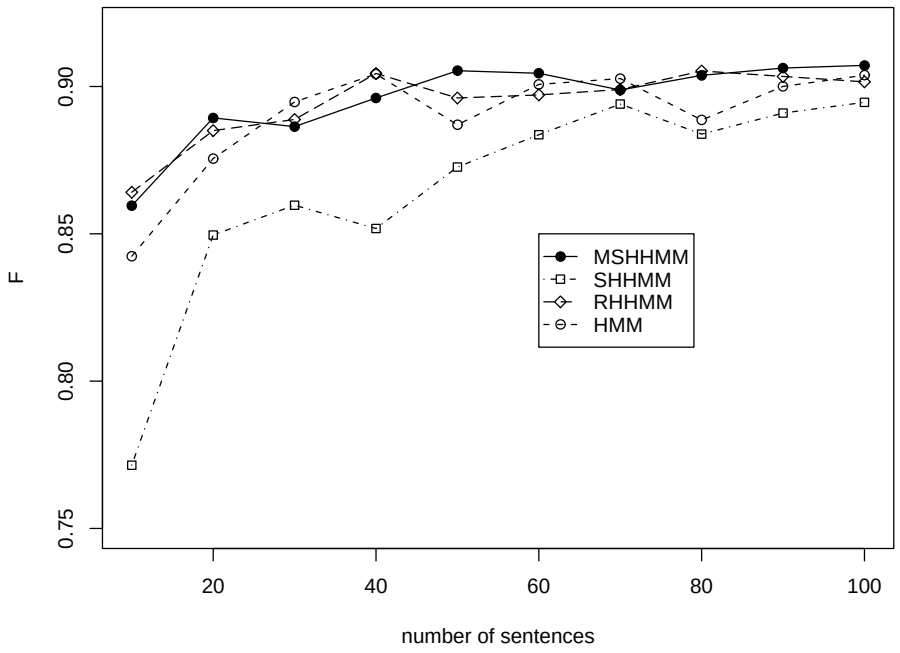


Figure 5.13: Overall results of F-measure for text chunking task on small volume of data

In order to more closely determine the various models performance when applied to sparse datasets they were applied to smaller volumes of training data. Figure 5.13 plots the relationship of micro-averaged F-measure on the small volumes of data, which ranged from 10 to 100 sentences. The figure shows that although the MSHHMMs accuracy may be slightly better than the other models, the advantage of its use is not as significant as it was for the reference tagging task. It is interesting to note that the SHHMM has the poorest accuracy of all four models.

Significance testing allows us to determine whether or not the results are a genuine difference between two (or more) groups, or whether it is just due to chance. For the text

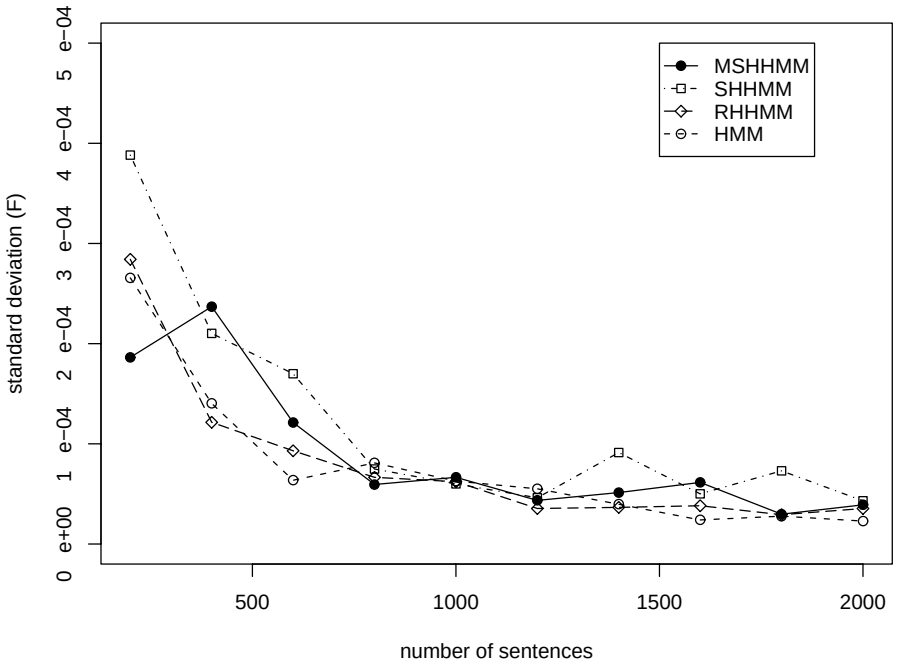


Figure 5.14: Standard deviation of F-measure for text chunking task

chunking task, the null hypothesis of this test is to find out whether the MSHHMM is better than the other three models—SHHMM, RHHMM and HMM. The significance evaluation process for text chunking is done in the same manner as in the reference tagging task. For a 5×10-fold cross-validation, the value of t_0 to reject the hypothesis within 95% confidence interval is when $|t_{49}(0.025)| \leq 2.0009$.

Table 5.6 provides a summary of t_0 values and p-value over different volumes of training data. The result shows that there is strong evidence that MSHHMM performs better than the RHHMM on training data of 1400 sentences, as the values of t_0 are greater than 2.0009. The difference between MSHHMM vs. RHHMM starts with negative values of t_0 (-2.237 and -0.792), then gradually increases to 0.425. The MSHHMM gains extraction accuracy as the volume of data is increased. The results also show there is strong evidence that the MSHHMM performs better than the SHHMM and the HMM, as the t_0 values are greater than 2.0009 for most of the different volumes of data.

	MSHHMM vs RHHMM		MSHHMM vs SHHMM		MSHHMM vs HMM	
no. of sentence	t_0	$P-value$	t_0	$P-value$	t_0	$P-value$
200	−2.237	0.028	2.825	0.001	1.089	0.279
400	−0.792	0.431	0.288	0.774	3.305	0.001
600	0.425	0.672	1.334	0.185	2.100	0.038
800	0.769	0.444	1.452	0.150	3.712	0.001
1000	0.586	0.559	3.105	0.002	5.657	1.527^{-7}
1200	0.749	0.456	4.897	3.834^{-6}	5.924	4.811^{-8}
1400	3.348	0.001	4.926	3.749^{-6}	6.581	2.433^{-9}
1600	1.858	0.066	2.792	0.006	5.459	4.408^{-7}
1800	1.896	0.061	5.468	4.676^{-7}	7.101	1.994^{-10}
2000	1.278	0.204	5.427	4.169^{-7}	6.7892	1.094^{-9}

Table 5.6: Significance test for different volume of data sets

Figure 5.15 plots the relationship of computational cost (in seconds) of processing each task, in respect to the alignment problem. The tests were carried out on the same dual $P4-D$ computer running at $3GHz$ and with $1GB$ RAM. These results show the dramatic effect of applying the improvement techniques to a HHMM. Both the tree-structured RHHMM and the SHHMM require extra processing time due to the greater complexity of the models. However the MSHHMM offsets the extra complexity by reusing parts of the model. Even though there is further cost involved in merging and later cloning out the repeated models, the MSHHMM outperforms the other hierarchical models and the linear HMM.

Table 5.7 lists the extraction accuracies for the different models with applied to the text chunking tasks on CoNLL-2000[4] data set. The results show the MSHHMM can provide

[4]http:www.cnts.ua.ac.beconll2000chunking

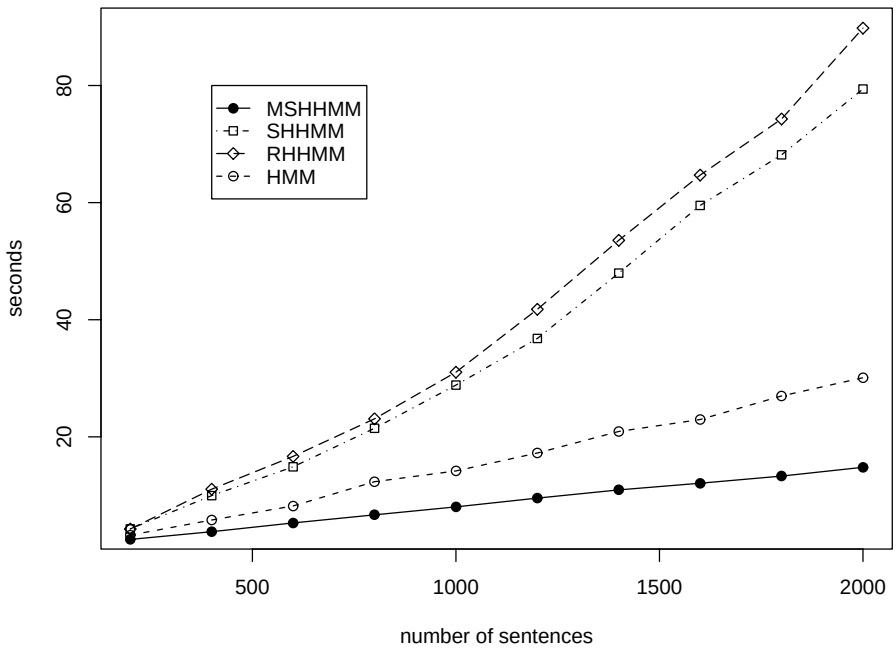


Figure 5.15: Average processing time of testing for text chunking tasks

model	Chunking Development	Chunking Test
HMM	0.907	0.895
HHMM	0.915	0.911
HMM by Molina [2001]	0.922	0.921

Table 5.7: F-measures of text chunking for CoNLL-2000.

better extraction accuracy than the simple HMM. But, the HMM system by Molina [2001] achieved higher accuracy than the MSHHMM. Moline's system takes advantage of selected words corresponding to certain tags (such as SBAR and PP), then adds those word towards the original tag. For example,

observation	POS	chunk tags	→	observation	POS	chunk tags
You	PRP	NP		You	PRP	PRP-NP
shows	NNS	NP		shows	NNS	NNS-NP

where the chunk tag transformed to new chunk tag of "PRP-NP" and "NNS-NP". During

the transformation process, the chunk tag "NP" is divided into different types of "NP" by joining with POS tag, which results in one-to-one relationship between chunk tag and observation symbol (rather than one-to-many relationship between chunk tag and observation symbols). In other words, the chunk tag "NP" is sub-divided along the boundaries of part-of-speech tags to reduce the number of observation symbols within each tag. In this manner, the model provided higher quality results than the MSHHMM on text chunking task. Conversely, when the same system is applied to the task of clause identification on the ConLL-2001[5] data set the MSHHMM outperformed the HMM [2001]. Table 5.8 lists the extraction accuracies for the clauses identification on two different models. The results indicated that the MSHHMM achieved a better extraction accuracy for F-measure of 0.7621 compared to 0.7068.

Model	P	R	F
MSHHMM	0.8047	0.7238	0.7621
HMM by Molina [2001]	0.7085	0.7051	0.7068

Table 5.8: Results of the top four systems that participated in shared task and the MSHHMM results for CoNLL-2001.

Hierarchical model has benefit of flexibility for merging repeated sub-models, also provided better handling for data that has hierarchical representation of tagged data. Overall, the results have indicated that the MSHHMM is better suited on the clause identification compared to the linear HMM.

5.3.2 Accuracy Performance on Individual Sentences

To explore properties of sentences for text chunking task, a series of experiments were conducted using a set of data from CoNLL-2000. The dataset contains 8936 training sentences and 1671 test sentences. The experiment was performed in a manner similar to those from the previous section, where the accuracy performance of one model is plotted

[5]http:www.cnts.ua.ac.beconll2001clauses

against the performance of a second model on the same sentence.

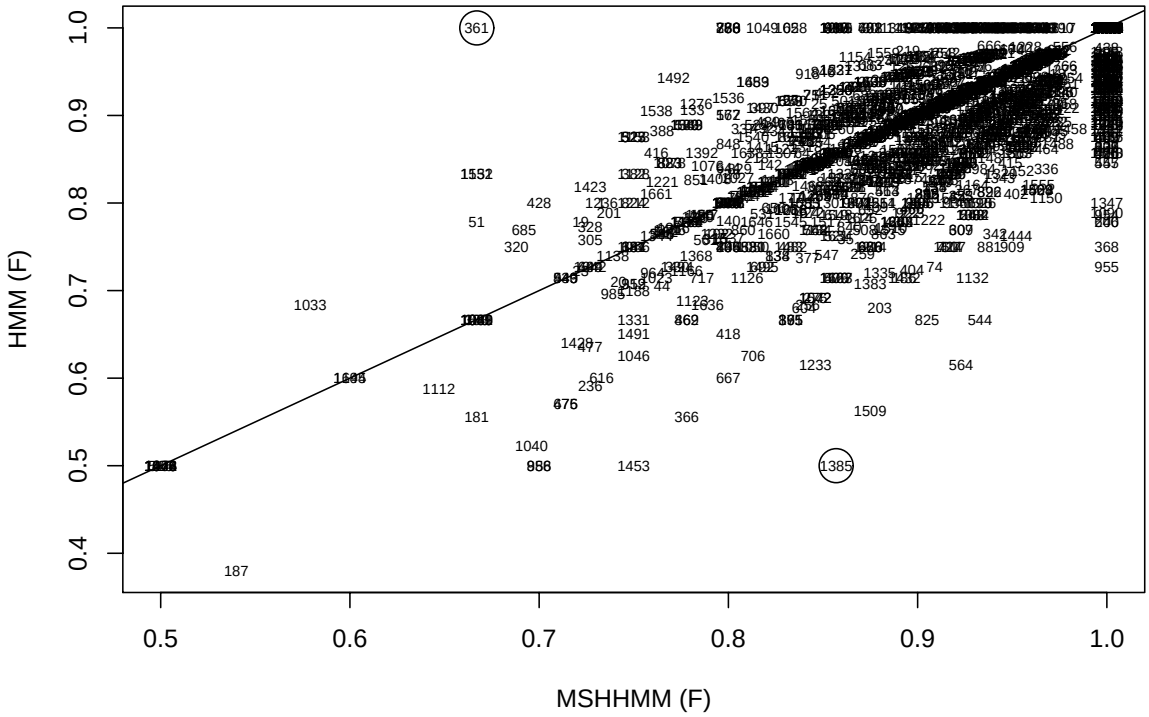


Figure 5.16: F-measure for MSHHMM against HMM

Figure 5.16 plots the relationship of F-measure for the HMM against the MSHHMM on 1671 test sentences. There are 706 sentences below the diagonal line, 373 sentences above the diagonal line and 592 sentences lie on the diagonal line. This indicates that the MSHHMM is more suitable then the HMM. The two circled sentences, 361 and 1385, are the extreme cases for these two models. Here is the tagged data for these two sentences.

361	(NP NNS CC NNPS)
1385	(S (O CC) (NP PRP) (VP VBZ) (PP IN) (S (VP VBG) (PRT RP) (NP DT NN) (O CC) (S (VP VBG) (ADVP RB) (PP IN))))) (O . ”)))

The result once again show that shorter sentences are more likely to be better handled

by the linear HMM model, where longer sentences are better suited to the MSHHMM.

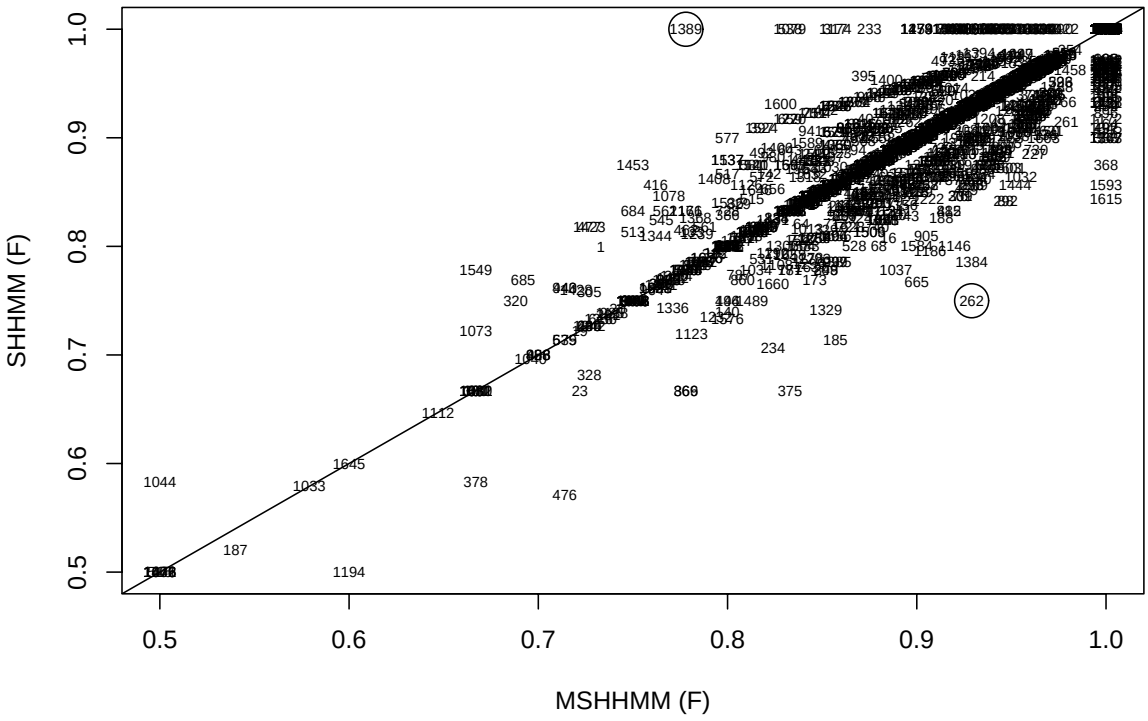


Figure 5.17: F-measure for MSHHMM against SHHMM

During the merge process, the information between equivalent sub-models are merged to provide better extraction accuracy. The results in Figure 5.17 show that there are large number of sentences which are set on the diagonal line. From 1671 test sentences, there are 380 sentences below the diagonal line, 268 sentences above it and the 1023 sentences lie on the diagonal line. The text chunking task is based on part-of-speech tags as observation symbol and there are only 45 different part-of-speech tags within the training data. Therefore, the merge process does not have as great an affect on the merged information compared to the reference tagging task. Here are the two sentences that were circled in Figure 5.17:

262	(S (S (O CC) (NP DT NN) (VP VBD) (ADJP JJR) (PP IN) (NP PRP$ NNS) (S (VP TO VB) (PRT RP) (PP IN) (NP PRP$ JJ NNS))))) (O , CC) (S (PP IN) (S (NP JJ NN) (VP VBD) (ADJP JJR) (PP IN) (NP PRP) (O IN) (NP VBD NNS))))) (O .)))
1389	(S (O “) (NP EX) (VP VBZ) (NP NN) (PP IN) (NP NN) (PP TO) (NP NN) (O .)))

The results again show that MSHHMM better handles the longer length sentence than the SHHMM.

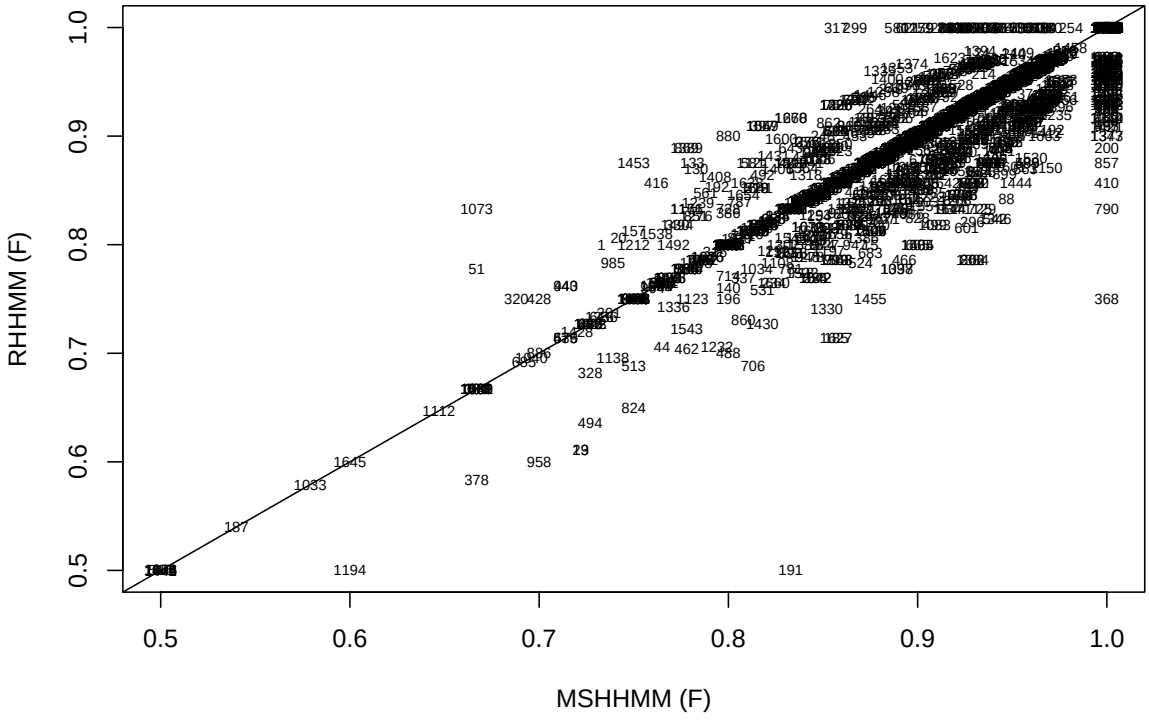


Figure 5.18: F-measure for MSHHMM against RHHMM

Figure 5.18 plots the relationship of F-measure for the RHHMM against the MSHHMM. The figure shows a similar trend as the previous Figure 5.17, where most of the sentences lies on the diagonal line. There are 958 sentences that have achieved the same

F-measure on both models, with 487 sentences having higher extraction accuracy under the MSHHMM, and 226 sentences have higher extraction accuracy under the RHHMM.

5.3.3 Discussion

The accuracy results shown in the previous section are not as promising for the MSHHMM as hoped. The accuracy of the merged and simplified model is shown to be comparable to that of the RHHMM. Only the performance of the SHHMM is significantly different, in that it has poorer accuracy than the other three models.

However, the significance test results suggest that the MSHHMM has achieved better overall performance than the SHHMM and the RHHMM. This is due to the MSHHMM having the advantage of reusing information for repeated sub-models, with all the accuracy benefits provided by combining observations. More importantly, these results show that the MSHHMM also achieves major gains in processing efficiency particularly for large training sets by sharing sub-model information—an action that reduces the complexity of the model while maintaining the benefits of hierarchical modelling. This results in a fewer number of states needing to be identified by the model and significant improvements in processing time.

Simplification, to recap, is a method for expressing the vertical model transitions (*in* and *out* of sub-models) and horizontal transitions (which *stay* in the sub-model). These three simplify the transition probability matrices that must be calculated during model testing. It was theorized that this would lead to improvements in accuracy, variance and processing time. During the reference tagging task, the SHHMM achieves a small increase in accuracy over the RHHMM when applied to a limited volume of training data. But, during the text chunking task, the results indicate that the SHHMM did not perform significantly better than the RHHMM. This is due to the type of data, where the data contains only part-of-speech tags. In this case the data contains a number of observation symbols that occur in more than one state. Therefore, when these child states are summarised, as in the simplification process, information about which states are shared

is lost, making correctly matching a sequence harder. Thus applying the simplification process on a model built from such data results in poorer accuracy than the RHHMM.

The individual analysis of each sentence showed similar trend as in reference tagging task, where the sentence with longer lengths typically are processed more accurately by the MSHHMM. However the improvement in performance, in terms of accuracy and stability, of the MSHHMM over the other HHMMs is less marked than those shown in the previous task. This is due to this task being generally more suited to hierarchical modelling—as evidenced by the HMM's lower performance during the micro-averaged F-measure. In such a situation, the hierarchical models should have better performance.

5.4 Summary

In this chapter the MSHHMM was applied to two applications; reference tagging and text chunking. The results were compared to the SHHMM, the regular tree structured HHMM (RHHMM) and the simple linear HMM. The MSHHMM benefits in accuracy and stability by reusing both observation and probabilistic information for those areas of the model where the hierarchical structure is repeated. It does so by merging these repeated sub-models during construction, and then simplifying sub-model elements during testing. This action also increases the MSHHMM's efficiency as, by merging the repeated states, fewer states need to be matched in the model, thus resulting in less computation cost when applying the MSHHMM to the task of phrase extraction.

In the reference tagging task, the MSHHMM can take advantage of reused information within the states *author* and *editor*. While there are only a small number of opportunities to reuse state information, the MSHHMM still receive increased extraction accuracy. In terms of computation cost, the MSHHMM does not prove any more efficient than the other models since there is only one level of hierarchy in this particular task.

For the text chunking task, the MSHHMM has many opportunities to gain accuracy by re-using information in different states, thus taking full advantage of the benefits of a

model structure that more closely models the underlying hierarchical sentence structure. But, because only 38 part-of-speech tags are present in the training data, there is no significant increase in the number of observation symbols within each state as a consequence of the merging process. Thus the increase in accuracy is not as high as it might be given data containing more observations. In terms of computational cost, the merging of repeated sub-models within the MSHHMM results in fewer states in the model. In contrast the RHHMM is required to identify every single path, leading to more states within the model and higher computation cost. During the experiment of calculating computational time, the RHHMM required significantly more processing time than the MSHHMM as the number of sentences increased. The extra costs of constructing a HHMM, which will also have the same number of production states as the HMM plus extra internal states, make it the least efficient.

In summary, a hierarchical model trained using limited data—as encountered in the problem of data sparseness—can benefit (in terms of accuracy) from the merging and simplification process. An example of data sparseness is the reference tagging task, where the training data contains a smaller proportion of observation symbols as compared to those found in the testing data. Although its structure is more complex, the MSHHMM gains in time efficiency as the model shares sub-state information thus reducing the amount of time required to process the testing data.

CHAPTER 6

Techniques for Refining HHMMs

This chapter focuses on four techniques for improving hierarchical hidden Markov models:

1 smoothing,

2 pattern generalisation,

3 structure formation, and

4 a partial flattening process.

The first two focus on observational aspects of the model and last two on structural aspects.

C-smoothing, as discussed in Section 6.1, was developed for the purpose of calculating error estimations for unseen events. The technique predicts which states are most likely to encounter unseen events. By doing so, the model can benefit from better probability estimates within each state. C-smoothing is compared against three well-known smoothing techniques: *Bayesian smoothing* [Berger, 1985], *Absolute discounting* [Ney et al., 1994] and *Jelinek-Mercer* [Jelinek and Mercer, 1980].

Section 6.2 examines the benefits of pattern generalisation, a method for simplifying input data, in terms of speed of training and the accuracy of prediction. The generalisation process reduces the total number of unique observation symbols within states, and reduces

the amount of training data required for a stable model.

Section 6.3 concentrates on optimising the structure of the model using a technique called *structure formation*. The process involves splitting states that contain several disparate features and re-expressing them as individual states. The process builds on the idea of capturing the boundary observation for each state in order to increase the extraction accuracy.

Section 6.4 investigates how to improve the structure of the model by using a partial flattening process to reduce its complexity. The flattening process is capable of transforming the HHMM to a new, simpler structure resulting in more accurate information extraction and lower processing cost. The technique involves calculating the dependency value, between either states or observations, over a range of sequences, then determining where the flattening process can be applied for maximum benefit. The state/observation dependency values are calculated using both *mutual information* and *log-likelihood*. The process also provides an estimate of the transition probability distributions for the model by transforming its states.

The final section presents a summary of the techniques mentioned above and their effects on model accuracy on two different tasks—reference tagging and text chunking.

6.1 Smoothing Techniques

The two fundamental problems in model estimation are the paucity of the data set [Banko and Brill, 2001], in that there is not a sufficiently large training corpus to provide a good estimation for the model, and data sparseness [Katz, 1987; Banko and Brill, 2001], where unseen events appear in the test data but not in the training data. One simple solution to resolve the first problem is to enlarge the training corpus. However this is not always feasible in that further training data is not always available. The second problem is generally addressed by providing an estimated value for the unseen event using some systematic

technique. Upon the occurrence of an unseen event, the system uses the technique chosen to assign a non-zero probability to it, then adjusts the probability distribution for each observation symbol. This process is collectively called *smoothing* [Zhai and Lafferty, 2001].

In general, smoothing techniques prevent an unseen event having zero probability. A zero probability would cause the model to 'break' as there would be no further state transitions available which matched the unseen event. They are also used to adjust the maximum likelihood estimates to improve the model. Smoothing produces a balanced training model by increasing the probability for infrequent events and decreasing those of frequent events. This improves the extraction performance of the trained model. A basic smoothing technique estimates the probability for unseen events by uniformly assigning a small value to every event. For example, Jeffreys [1948] applied *Additive Smoothing* to assign probabilities for each bigram in the corpus, where

$$P(w_i|w_{i-1}) = \frac{c(w_{i-1}w_i)+1}{\sum_w [c(w_{i-1}w_i)]+1} \tag{6.1}$$

$$\approx \frac{c(w_{i-1}w_i)+1}{\sum_w [c(w_{i-1}w_i)]+|V|} \tag{6.2}$$

where $c(w_{i-1}w_i)$ is the term frequency count for term $w_{i-1}w_i$, and $|V|$ is the size of the vocabulary in the training corpora. By introducing the parameter $|V|$, the system is able to increase the probability for low count events. More complex methods, such as Katz smoothing [Katz, 1987] and Good-Turing smoothing [Good, 1953], calculate a different value for each state depending on factors such as the number of previous unseen events, and the number of words of the same frequency throughout the training set. The estimations are based on applying the technique to a large corpus. The formula of Good-Turing is written as:

$$P(o|q) = \frac{N_o+1}{T} \times \frac{E(N_o+1)}{E(N_o)}, \tag{6.3}$$

where o is the event, N_o is the total number of events o, T is the sample size and $E(n)$ is an estimate of how many different events happened exactly n times. The equation takes into consideration the difference between observations with n counts and $n+1$ counts.

When encountering an unseen event, the probability is often set to approximately $\frac{N_1}{T}$. The Good-Turing smoothing is focused on the difference between the observation that occurred in n and $n-1$ counts for the entire data set, compared to the Markov model where the emphasis is on the observation distribution among different states. Therefore the Good-Turing method is not an appropriate method for the MSHHMM.

6.1.1 The Methods

Several smoothing methods were considered before attempting to calculate a new variation to suit extraction tasks. These methods were chosen primarily because of their efficient implementation.

This section discusses several smoothing techniques:

- Bayesian smoothing using Dirichlet priors (μ) [Berger, 1985]
- Absolute discounting (δ) [Ney et al., 1994]
- Jelinek-Mercer (λ) [Jelinek and Mercer, 1980]

The objective is to design an estimation method to suit different types of model, and simultaneously determine an optimal value to replace the observation value for the unseen variable.

6.1.1.1 Bayesian Smoothing using Dirichlet Priors

The Dirichlet distribution [Berger, 1985] of the prior distribution associates the variables $p = (p_1, \ldots, p_n)$ with parameters $u = (u_1, \ldots, u_n)$. The re-estimation formula for each probability distribution is:

$$P_\mu(o|q) = \frac{c(o,q) + u(o|C)}{\sum_{i=1}^{M} c(o_i,q) + u(o|C)} \tag{6.4}$$

and the Dirichlet parameters are estimated by:

$$u(o|C) = \frac{\mu}{\sum_{i=1}^{M} c(o_i, q) + \mu},$$

where μ is a parameter to be set by the user and μ must be greater than zero. M is the total number of the observation symbols in the training corpus, and $c(o_i, q)$ is the number of counts for the observation symbol o_i appearing in state q. This formula scales the probability distribution according to the size of the state (in terms of number of observations), where a larger state has a lower probability value of $u(o|C)$ and smaller states have higher probability value of $u(o|C)$. Note that $u(o|C)$ is never 0, not even for unseen events, as μ is greater than 0, thus addressing the zero probability problem.

6.1.1.2 Absolute Discounting

The idea of Absolute discounting [Ney et al., 1994] smoothing is to subtract a constant from the seen words count to lower the probability rather than multiplying it by a parameter. The formula can be written as:

$$P_\delta(o|q) = \frac{\max(c(o;q) - \delta, 0)}{\sum_{i=1}^{M} c(o_i;q)} + \sigma p(o|C) \tag{6.5}$$

where $\delta \in [0, 1]$ is a discount constant and $\sigma = \delta|o|_u/|o|_q$. The $|o|_u$ represents the number of unique symbols in state q, $|o|_q$ is the total number of observation symbols in state q, and $p(o|C)$ is the probability of observation o over the entire training set. The formula also places emphasis on an additional probability that considers the observation symbols within each state σ, along with observation probability distribution for entire training set (as calculated by $p(o|C)$). If an unseen event occurs during the testing process, $P_\delta(o|q)$ will be set to a small probability value so as to prevent the zero probability problem.

6.1.1.3 Jelinek-Mercer

Jelinek-Mercer [1980] is a technique for smoothing n-grams in language modelling for speech recognition. This method uses a coefficient λ to control the maximum likelihood

model and the collection model as:

$$P_\lambda(o|q) = (1 - \lambda)P_{ml}(o|q) + \lambda P(o|C) \tag{6.6}$$

where o represents the observation event, q represents each individual state, and C represents the entire training data. The focus for this smoothing technique is a combination estimation of probability distribution within the state, plus the probability distribution for the entire training data, with the estimation adjusted by using the parameter λ. Zero probability events are handled in the same way as they are in *absolute discounting* smoothing.

6.1.1.4 C-smoothing

When applied to a basic Markov model, a smoothing technique typically takes an approach based on assigning a small value, such as:

$$P^*(o|q) = P(o|q) + 10^{-8}$$

to every state, in order to prevent an unseen event having zero probability. Such a method should, preferably, focus on the transition between states. In this section, *C-smoothing* is applied to provide a better estimation for unseen events for each individual state.

Consider the methods mentioned above. The problem with using a technique like the Dirichlet Priors on HMMs is that a state containing a large number of observations does not necessarily have a lower probability of encountering an unseen event. Likewise, a state containing fewer observations does not absolutely imply a higher probability of encountering an unseen event. Some states might just have fewer observation symbols.

Table 6.1 summaries the number of observation symbols within each state. The last column contains the number of observation symbols which have appear less than 2 times within each state for reference tagging task. For example, the state *month* in a reference tagging task, may only contain 15 observation symbols with total number of observation count of 258, where there are only 4 observation symbols which have observation count less than 2. It contrast, the state *publisher* contains 71 observation symbols with total

state	number of observation count	number of observation symbols	number of observation symbols (≤ 2)
author	2935	846	784
title	3811	1650	1444
booktitle	2460	296	189
volume	378	59	33
number	146	12	4
pages	730	329	319
month	258	15	4
year	536	27	8
editor	116	50	39
publisher	150	71	57
address	419	86	55
note	4	2	2
techtype	32	12	9
technumber	11	11	11
institution	59	30	25
organisation	7	5	5
series	12	6	4
thesistype	2	2	2

Table 6.1: Number of observation symbols for reference tagging task

observation count of 150, where there are 57 observation symbols that contains less than two observation count. In this example, the state *publisher* is more likely to encounter an unseen event than the state *month*. This indicates that the number of observations within a state is not necessarily proportional to the error estimation for unseen events. Instead the probability estimation for unseen events should based on number of low count observation symbols within each state.

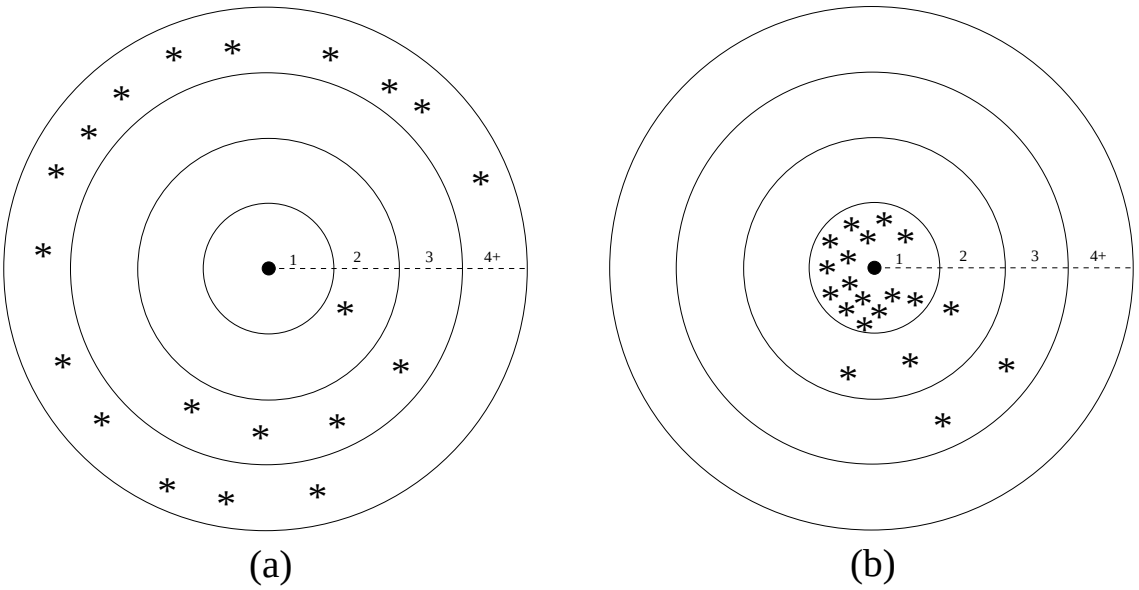


Figure 6.1: Distribution of observation symbols with (a) low probability (b) high probability of encoutering unseen events

Figure 6.1 illustrates two states during the training process, where each state contains 20 observation symbols. Each star ($*$) in the figure represents an observation symbol and the radius of the circle represents the observation count. While there is one observation symbol in Figure 6.1(a) that has a count less than two, most of observation symbols occur four or more times. Figure 6.1(b) contains 18 observation symbols that have an observation count less than two, with most of the observation symbols lying within a radius of one. In terms of occurrence of unseen events for these two states, Figure 6.1(b) should have a higher probability distribution for unseen events, because most of the observation symbols have only occurred once during the training process. There is more likely to be a greater variance of observations for this state, and thus more chance of encountering an unseen event.

Using this fact, the estimation formula for C-smoothing is derived as:

$$P(o|q) = \frac{c(o,q)}{\sum_{i=1}^{M} c(o_i,q)} + P_\epsilon(q) \tag{6.7}$$

where the error prediction $P_\epsilon(q)$ calculates the estimate probability distribution of the

infrequent events within state q_k. The formula for the estimation method is written as:

$$P_\epsilon(q_k) = \frac{\sum_{j=1}^{M} f(o_j, q_k)}{\sum_{j=1}^{M} c(o_j, q_k) \times \sum_{i=1}^{M}\sum_{j=1}^{N} c(o_j, q_i)} \tag{6.8}$$

and,

$$f(o_j, q_k) = \begin{cases} 1 & c(o_j, q_k) \leq \epsilon \\ 0 & \text{otherwise,} \end{cases} \tag{6.9}$$

where $c(o_j, q_k)$ represents the number of occurrences for observation symbol o_j in state q_k, $c(o_j, q_k) \leq \epsilon$ represents the number of observations of symbol o_j that have occurred less than ϵ times in the state q_k, M represents the total number of states, and N represents the total number of observation symbols. The maximum occurrence threshold allowed for $c(o_j, q_k)$ can be set to other values depending on the type of training data. The formula is designed to estimate the probability of how frequently the state q_k can be expected to encounter an unseen variable. For example, in reference tagging task, which uses discrete measurements, the value of ϵ is set to be 1. When a state contains only observation symbols whose occurrence count is greater than ϵ, then the probability of an unseen event will be very low.

6.1.2 Evaluation

This section investigates, empirically, the performance of the smoothing techniques discussed in Section 6.1. The aim of this evaluation is to compare the performance in accuracy of smoothing techniques with the constant smoothing of

$$p_\epsilon(q_i) = 10^{-8} \qquad \text{for } \forall i, \tag{6.10}$$

when applied to the reference tagging task. The techniques are evaluated on 600 sentences of hand tagged data by Seymore [1999]. The experiment was carried out using 5×10-fold cross validation, where the data set is randomly divided into equal sized sub-sets, and 10-fold cross validation is performed on each subset to evaluate the affect on F-measure. This process is repeated 5 times. The results are displayed as the distribution of the

F-measurement for different smoothing methods, different smoothing parameters and different volumes of training data.

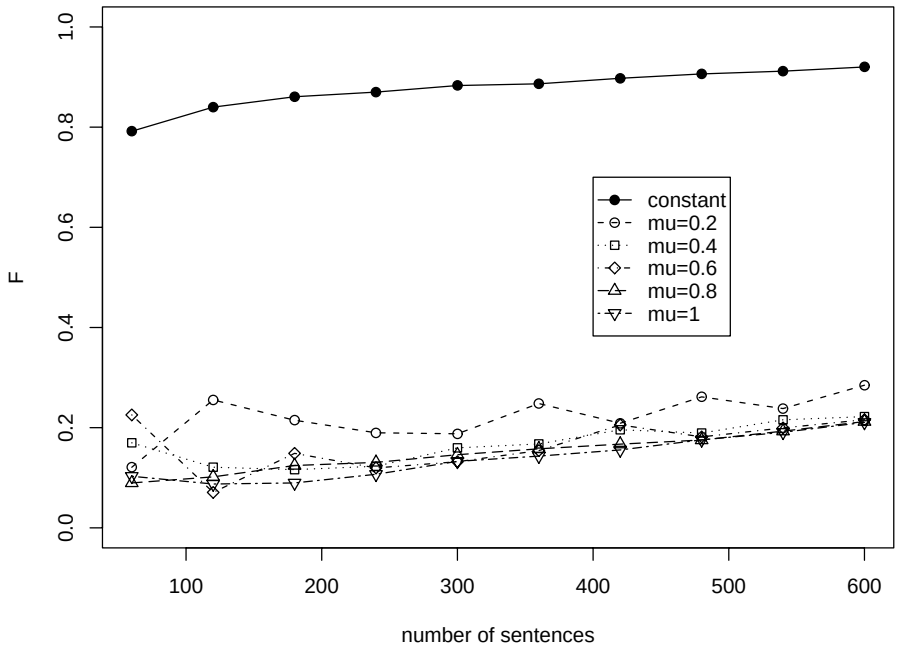


Figure 6.2: Smoothing results for Dirichlet Priors

Dirichlet Priors smoothing adjusts the probability of observations for each state by combining the probability of observations within the state adding some factor derived from the probability for the entire data set, $u(o|C)$, as shown in Equation (6.4). Figure 6.2 shows the Dirichlet Priors smoothing fails to provide better extraction accuracy for MSHHMM model. The constant smoothing obtains higher extraction accuracy than the Dirichlet Priors technique regardless of the value of the parameter μ (mu). The volume of training data has only a small effect on the F-measure across all variations.

Figure 6.3 plots the F-measure against volume of training sentences for different absolute discount constants δ (delta). The method lowers the probability of seen observations then sums the probability of observations over the entire data set ($\sigma p(o|C)$), where the size

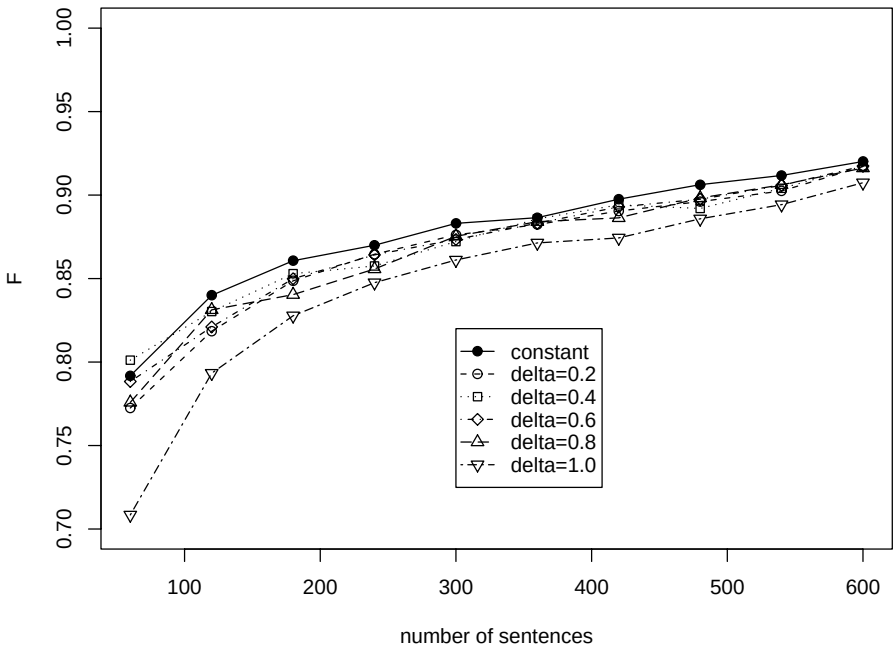


Figure 6.3: Smoothing results for Absolute Discounting

of $\sigma p(o|C)$ is proportional to the discount constant δ. The Figure 6.3 shows that size of δ is inversely related to accuracy. Figure 6.3 also shows that the constant smoothed model achieved better extraction accuracy than any of the absolute discounting smoothing variations, thus the absolute discounting smoothing does not provide good probability estimation for the MSHHMM. Most of the techniques showed only slight improvement as the volume of training data increased, although volume did have a more pronounced effect for $\delta = 1.0$.

Each of the previous two smoothing methods (Bayesian smoothing and Absolute discounting) involve the observation probability for entire data set $p(o|C)$ as part of the observation probability within each state $p(o|q)$. The extraction accuracy tends to drop when $p(o|q)$ contains a higher weight of $p(o|C)$. In Jelinek-Mercer, the coefficient λ controls the weight of $p(o|C)$, which contributes to the observation probability $p(o|q)$. The results show that the higher the value of λ, the lower the extraction accuracy performance on the reference

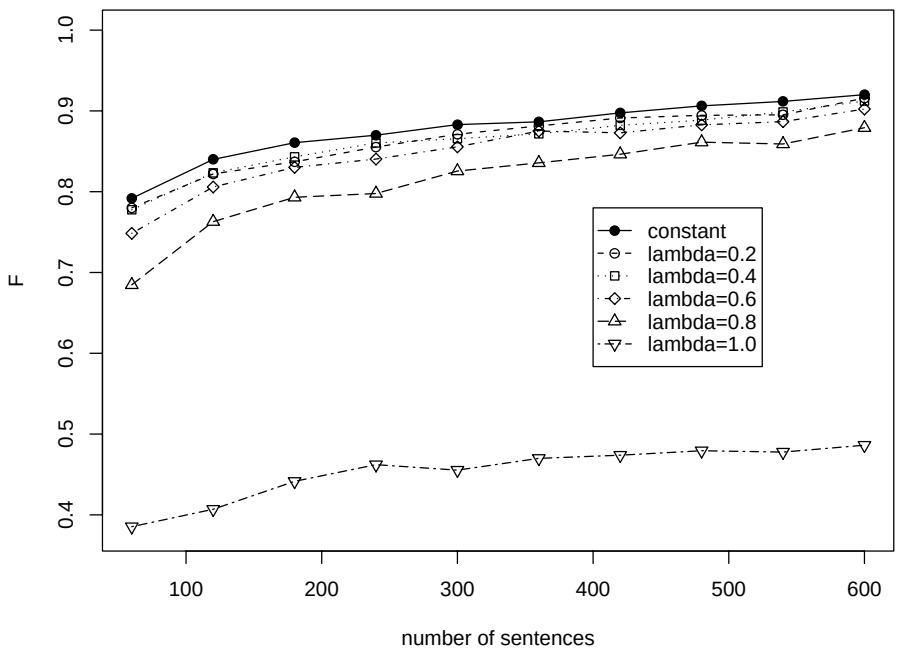


Figure 6.4: Smoothing results for Jelinek-Mercer

tagging task, with some significant reduction in accuracy as λ approaches 1. The overall results show the 'constant' model outperforms the Jelinek-Mercer smoothing. Again, only slight gains in accuracy can be attributed to the volume of training data.

The C-smoothing method focuses on the prediction of rare events within each state rather than using a smoothing factor of the observation probability for the entire data ($p(o|C)$). Figure 6.5 shows the results of different values of threshold (ϵ epsilon) for the C-smoothing method against several volumes of training sentences. When the threshold ϵ is equal to 1, the smoothing achieves the highest improvement from amongst the other threshold settings ($\epsilon = \{2, 3, 4, 5\}$). When the volume of sentences reaches 300, the C-smoothing method achieves 2% increases in accuracy as compared to the 'constant' method. This result indicates that the C-smoothing can provide better estimation when a reasonable volume of training data is reached. When there is an insufficient volume of training sentences, the C-smoothing can not correctly estimate the low count observation symbols

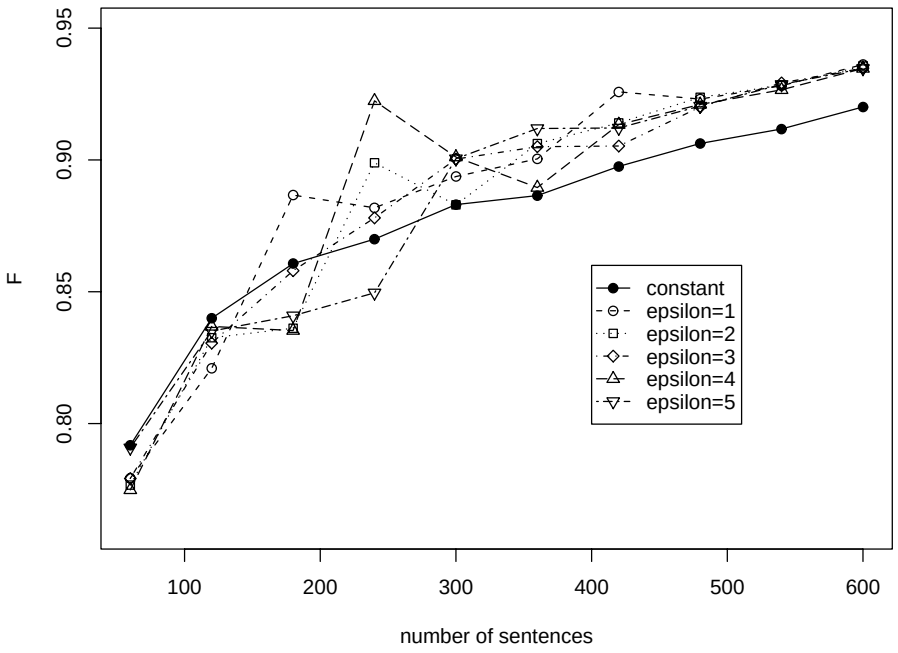


Figure 6.5: C-smoothing of different threshold of ϵ (epsilon)

within each state. The results show that the 'constant' model achieves better extraction performance than C-smoothing method under sparse training data. On the other hand, the C-smoothing method provides a good estimation for the probability for unseen events when given larger volumes of training data.

Method	parameter	Micro-average F-measure
constant	-	0.920 ± 0.00009
C-smoothing	$\epsilon = 2$	0.936 ± 0.00008
Bayesian smoothing	$\mu = 0.2$	0.285 ± 0.00267
Absolute discounting	$\delta = 0.4$	0.917 ± 0.00016
Jelinek-Mercer	$\lambda = 0.2$	0.916 ± 0.00016

Table 6.2: F-measure for different smoothing results at 600 sentences.

Table 6.2 summarises the overall results for the four different smoothing techniques at a

training data volume of 600 sentences. It shows the benefit of using C-smoothing on the reference tagging task. C-smoothing achieved 9.936 on micro-averaged F-measure compared to 0.920 for the MSHHMM without the smoothing. Furthermore, the C-smoothing has the smallest standard deviation among other smoothing methods, indicating good stability.

Method	constant	smoothing
MSHHMM	0.920 ± 0.009	0.936 ± 0.009
SHHMM	0.938 ± 0.007	0.937 ± 0.011
RHHMM	0.753 ± 0.017	0.769 ± 0.015
HMM	0.939 ± 0.010	0.949 ± 0.008

Table 6.3: F-measure of C-smoothing and constant model at 600 sentences.

In Chapter 5, the results stated that MSHHMM can outperform the HMM when there is a limited volume of training data. Table 6.3 summaries the F-measure of C-smoothing with parameter of $\epsilon = 2$ against the constant model ($P_\epsilon(q_i) = 10^{-8}$) for the four different models. There are 600 reference tagging sentence and the results are calculated by 5×10-fold cross validation. The results show that not only the MSHHMM can be benefit from the C-smoothing error estimation for the unseen event, where two other models—RHHMM and HMM can also benefit from it.

6.1.3 Discussion

The C-smoothing method is shown to be beneficial for the MSHHMM when applied to the reference tagging task. The overall result shows higher prediction accuracy on micro-average F-measure than the other smoothing techniques evaluated. The other smoothing techniques such as the Dirichlet Priors, the Absolute Discounting, and the Jelinek-mercer smoothing do not achieve better accuracy than the basic constant method. The results also indicate that the smoothing techniques for MSHHMM depend on the probability distribution of the observations within each state, and not the probability distribution of states within the entire training data set.

The size of training data controls the observation probability distribution within each state, where the larger the training data, the better the estimation. On the other hand, the C-smoothing is ineffective when a small volume of training sentences is used. The smoothing method is unable to determine a good estimate for the probability of unseen events as it cannot identify the states most likely to encounter unseen events. This is due to the fact that this state identification depends on observation frequency counts within the state, and obviously with fewer training sentences most of these observation counts are low. Therefore, when encountering a limited amount of training data, the constant smoothing technique should be used during extraction.

6.2 Pattern Generalisation

When a model is applied to a particular sentence it must use some form of matching algorithm to identify which words in the input data match some observation in the model. Sometimes it is better to simply use plain-text matching, which takes user input in its original form and matches it directly against the word in the training data. The method is accurate when the training data includes the complete lexicon of words. However, in practice, it is difficult to obtain the complete lexicon of words. Pattern generalisation allows approximate matching against training data. Where such generalisation is applied the process benefits from a decreased number of training symbols, which in turn increases the system efficiency as there are less observation symbols to be matched within the model.

Comparing observations within trained data can be a difficult task. For a general text extraction task, the pattern matching algorithm involves transforming observations into patterns, then searching for identical patterns within the training data. When a pattern matching algorithm is not applied, then the system needs more training data to support accurate extraction, as observations from testing data may not be included within training data. While increasing the training data can solve this problem, as mentioned earlier, large training sets containing an exhaustive list of observations are unlikely to be available. Moreover, lack of training data can cause the serious problem of misidentifying

the test observations, which leads to invalid states and an unreliable model. So instead of word-by-word matching, the system can make more efficient use of the training data by representing observations as a pattern using a small, fixed alphabet. The resulting model requires less training data to become reliable and has a greater accuracy when recognising the state the correctly matches.

Consider an example of pattern generalisation from the reference in Figure 2.7:

> (A (N (F T.__A.)(L Moloney,__Aaaaaaa,)) (N (F A.__A. C.__A.) (L. Lea,__Aaa,)) (C and__aaa) (N (F C.__A.) (L Kowalchuk.__Aaaaaaaaa.))) (T Manufacturing__Aaaaaaaaaaaaa and__aaa packaged__aaaaaaaa goods.__aaaaa.) (ED (N (C In__Aa)) (N (F G.__A. H.__A.) (L Castle,__Aaaaaa,)) (C editor,__aaaaaa,)) (BT Profiting__Aaaaaaaaa from__aaaa a__a Geographic__Gaaaaaaaaa Information__Aaaaaaaaaaa System.__Aaaaaa.) (PU GIS__AAA World,__Aaaaa, Inc.,__Aaa.,) (AD Fort__Aaaa Collins,__Aaaaaaa, CO,__AA,) (Y 1993.__iiii.)

The reference tag associated with each word begins with an open parenthesis followed by the tag name. Each word is then attached by an underscore to its pattern generalisation, as outlined in Table 6.4. For example, symbol "(*A*" represents state *author* and "(*T*" represents *title*. The pattern generalisation involves transforming the input data by using "A" to represent the capital letters, "a" for lower-case letters, "i" for integers and all other symbols, and in particular punctuation, remain in the same form.

6.2.1 The Methods

There are two novel types of pattern generalisation that were implemented to improve the extraction performance on the MSHHMM. The methods were based on the nature of input characters for the reference tagging task. These methods are:

- character class pattern generalisation (CCPG), and
- regular expression pattern generalisation (REPG).

state	tag
author	A
title	T
booktitle	BT
volume	V
number	NUM
pages	P
month	M
year	Y
editor	ED
publisher	PU
address	AD
name	N
first name	F
last name	L
conjugate	C

Table 6.4: Annotated tags for reference tagging task

The CCPG converts each word based on its character class, and the REPG converts each word on its character class and occurrence. This leads to a model with fewer observation symbols for each state; thus computational cost is decreased, while the efficiency of the training data is increased.

The punctuation between words and the case characters for each word are often the most significanty influencing factors when identifying types of tag in the reference tagging task. For instance, the first name's pattern consists only of a capital letter and a full stop after it (A.), while the title state often starts with a quotation mark ("Aaa). The CCPG converts each word to a pattern based on; the number of characters, whether a character is a letter or a number, the capitalisation and its punctuation. The conversion rules are:

- capital letters → "A"

- small letters → "a"
- integer → "i"

and the remaining symbols are unchanged. For example, "T. Moloney" is converted to "A. Aaaaaaa". The final step involves grouping each part of the pattern with the hand-tagged markup. Thus "A. Aaaaaaa" is further processed into "(F A.) (L Aaaaaaa)" with "F" representing the state *first name* and "L" the state *last name*.

A logical extension of character class its to further generalise the pattern using a similar method to that found in regular expressions. The REPG converts each word based on its character class and occurrence. The model provides a more efficient model than CCPG with less observation symbols are required for the extraction tasks. The REPG conversion rules are:

- a capital letter → "A"
- two or more capital letters → "A+"
- a small letter → "a"
- two or more small letters → "a+"
- a integer → "i"
- two or more integers → "i+"

and the remaining symbols are unchanged. For example, "T. Monloney" is converted to "A. Aa+", and "1993" is convert to "i+".

6.2.2 Evaluation

The goal of pattern generalisation is to reduce the number of observation symbols in each state, which at the same time increases the total number of observation counts. In order to identify situations where generalisation is beneficial, this section compares the affects on accuracy of two types of pattern generalisation—the CCPG and the REPG—against different volumes of training data. The different generalisation techniques are evaluated

on 600 sentences of hand tagged data by Seymore [1999], with three variations of the hierarchical hidden Markov model; the RHHMM, SHHMM and MSHHMM. The effects of pattern generalization are also shown when applied to a HMM. The experiments were carried out using 5×10-fold cross validation, where the data set is randomly divided into equal size subsets, 5 times, and 10-fold cross validation is performed on each subset on F-measure.

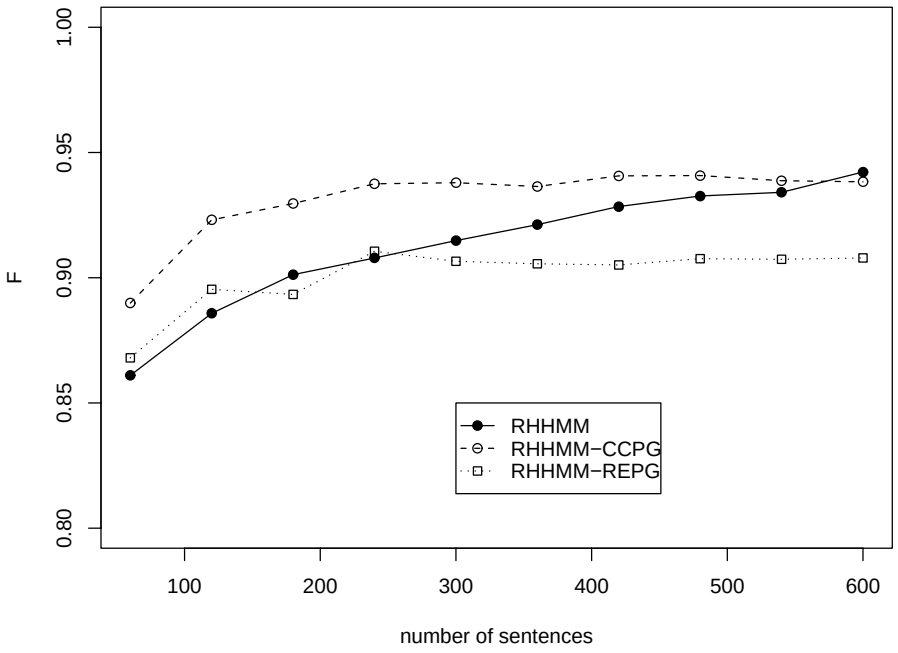


Figure 6.6: F-measure of the RHHMM for pattern generalisation.

Figure 6.6 shows the RHHMM achieves better extraction accuracy after the CCPG is applied. The RHHMM-CCPG exhibits a small rate of increase after the volume of training sentences reaches to 240 compared to RHHMM, which achieves same as RHHMM-CCPG accuracy at 600 sentencs, which is slightly quicker than the RHHMM. The REPG can improve extraction accuracy for RHHMM only under smaller volumes of training data, as the results indicate an ungeneralised RHHMM performs better than RHHMM-REPG after the volume of training data reaches 250 sentences.

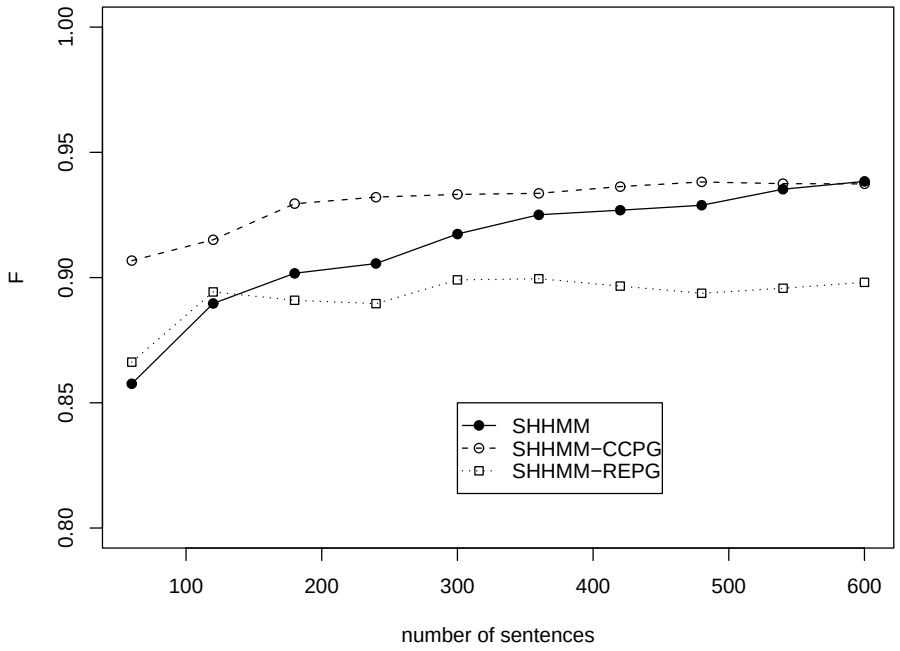


Figure 6.7: F-measure of the SHHMM for pattern generalisation.

Figure 6.7 illustrates the result of applying the pattern generalisation techniques on the SHHMM. Again, the performance indicates the benefit of using the CCPG method on the reference tagging task over a range of different volumes of training data. The SHHMM-REPG is again shown to have lower accuracy, on average, than the unaltered SHHMM.

Figure 6.8 shows the F-measure for the MSHHMM without pattern generalisation, and then with the two types of generalisation denoted MSHHMM-CCPG and MSHHMM-REPG. It indicates that both pattern generalisations can provide better exaction accuracy than the original MSHHMM for small volumes of training data, but, as the volume increases, the difference between the MSHHMM and the MSHHMM-CCPG decreases. The MSHHMM-REPG is less consistent over number of sentences, and is generally outperformed by MSHHMM-CCPG and an unaltered MSHHMM.

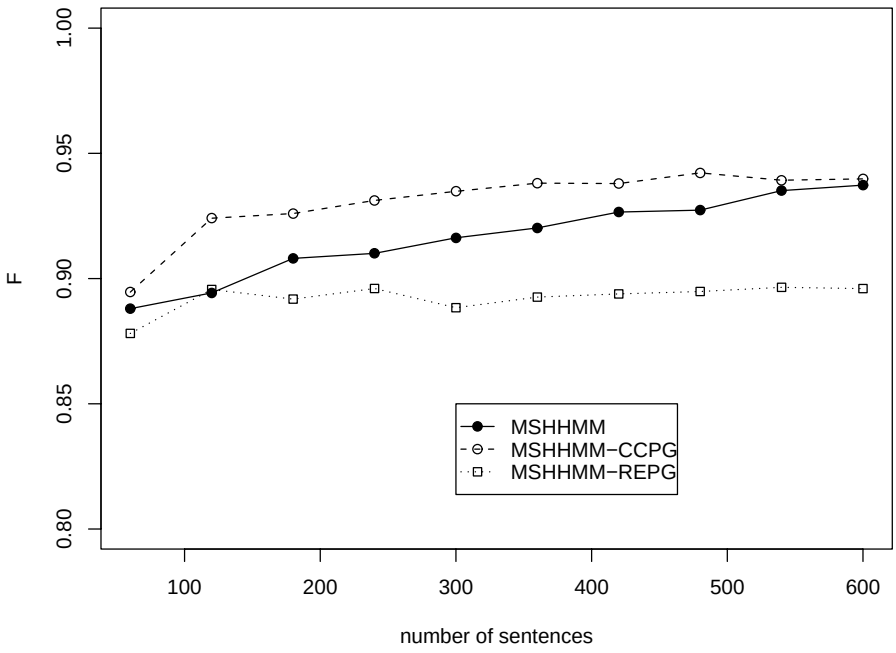


Figure 6.8: F-measure of the MSHHMM for pattern generalisation.

Figure 6.9 shows the influence of pattern generalisation on the HMM. The results show that while the volume of data is under 180 sentences, both pattern generalisation (HMM-CCPG and HMM-REPG) outperform the ungeneralised HMM. When there is an insufficient volume of training sentences, the HMM can benefit from gaining observations within each state during the pattern generalisation process. The HMM-CCPG reaches a stable result after 300 training sentences. However as the volume of training data increases, the results for the HMM accuracy continues to increase. Eventually, the accuracy of the HMM and HMM-CCPG converged, while the HMM-REPG fails to improve from its initial values. In comparison with the hierarchical models, the HMM requires more training data to achieve a stable result.

The results shown in this section indicate that pattern generalisation helps the model reach a steady result quicker, but that the REPGs performance is eventually surpassed by the ungeneralised data. Table 6.5 provides a summary of the overall results of micro-averaged F-measure for the four different models at a training data volume of 600 sentences. The

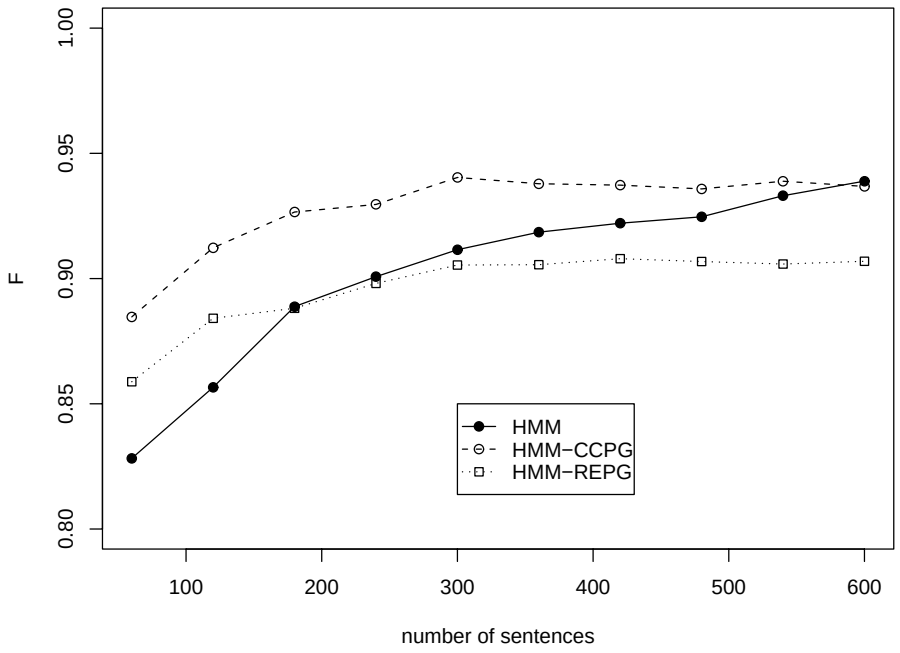


Figure 6.9: F-measure of the HMM for pattern generalisation.

results indicate that the CCPG improves the model accuracy when applied to the reference tagging task. The REPG method, while initially offering some improvement to accuracy, is shown to be less favourable for this task.

Method	original	CCPG	REPG
RHHMM	0.942 ± 0.00004	0.938 ± 0.00004	0.908 ± 0.00017
SHHMM	0.938 ± 0.00005	0.937 ± 0.00005	0.898 ± 0.00019
MSHHMM	0.937 ± 0.00007	0.940 ± 0.00008	0.896 ± 0.00011
HMM	0.939 ± 0.00007	0.937 ± 0.00005	0.907 ± 0.00015

Table 6.5: Micro-averaged F-measure of 5×10 fold cross-validation with 600 sentences

6.2.3 Discussion

The experimental results show the effectiveness of pattern generalisation. The pattern generalisation process can contribute to model performance by reducing the number of observation symbols that have to be matched. This also leads to an increase in observation symbol counts, the benefits of which have been previously discussed. Furthermore, pattern generalisation increases the chance of testing observations matching against the training data. The results of a model that has had pattern generalisation applied outperform a model built simply on the original, ungeneralised, training data. The difference becomes more apparent if there are fewer training sentences. Theoretically, when there is sufficient training data to describe all the observation information within each state, the original data gives a more accurate model. This situation, however, is difficult to achieve, as there is unlikely to be sufficient training data to be assured of encountering every possible observation.

The experimental results also show that the REPG fails to improve performance on reference tagging task. This is due to the regular expression generalisation converting the observation to a point where there is not enough information accurately to differentiate between states. Although the generalisation reduces the amount of observation symbols within each state, it makes it harder for the model to identify to which state the observation belongs.

6.3 Structure Formation

Structure formation is the process that transforms the structure of some part of the model, those that contain an appropriate production state, to a new structure that improves extraction accuracy. The process involves splitting the child states of the HHMM in order to capture detailed information during the state transition.

For example, the *title* state in Figure 5.3 contains:

title: Manufacturing and packaged goods.

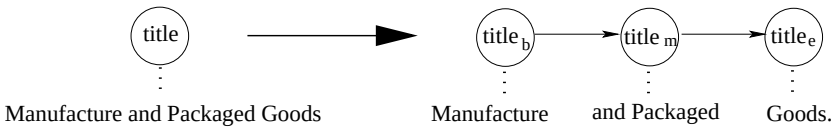


Figure 6.10: Structure formation for state *title*

The beginning of the state *title* often starts with a uppercase word (as in "Manufacturing") and ends with a full stop (as in "Goods."). The words between the first and last tend to be lowercase. Because of these features, the single state *title* can be made to more closely model the underlying structure by splitting it into beginning, middle and ending states. Figure 6.10 illustrates the formation process for state *title*. The formation involves splitting each production state to three sub-states:

$$\text{Title} \rightarrow \{\ \text{title}_b, \text{title}_m, \text{title}_e\ \}$$

the beginning state $title_b$, the middle state $title_m$ and the ending state $title_e$. The remaining states, such as *publisher*, *date* and *pages*, are treated in the same manner. When a state contains only one observation, such as state *year*, the splitting process assigns one sub-state, the beginning state $year_b$, to represent the state.

The main idea of the splitting process is to capture the types of observation symbols that are more likely to occur at the starting point and the ending point of the state. Figure 6.11 illustrates the process of structure formation for a simple HHMM. Figure 6.11(a) represents the simple HHMM with five production states (p_1, p_2, p_3, p_4 and p_5) and one internal state (q_1). The splitting process divides each production state p_i into three sub-states of $\{p_{i,b}, p_{i,m}, p_{i,e}\}$ as shown in Figure 6.11(b).

6.3.1 Evaluation

To explore the potential of structure formation, an experiment was conducted to compare performance on accuracy between the RHHMM, SHHMM, MSHHMM and HMM and versions of the models that have undergone structure formation. The method is evaluated on a set of data by Seymore [1999]. The data contains 600 references with 13 types of tag: *title*, *author*, *institution*, *location*, *note*, *editor*, *publisher*, *date*, *pages*, *volume*, *journal*,

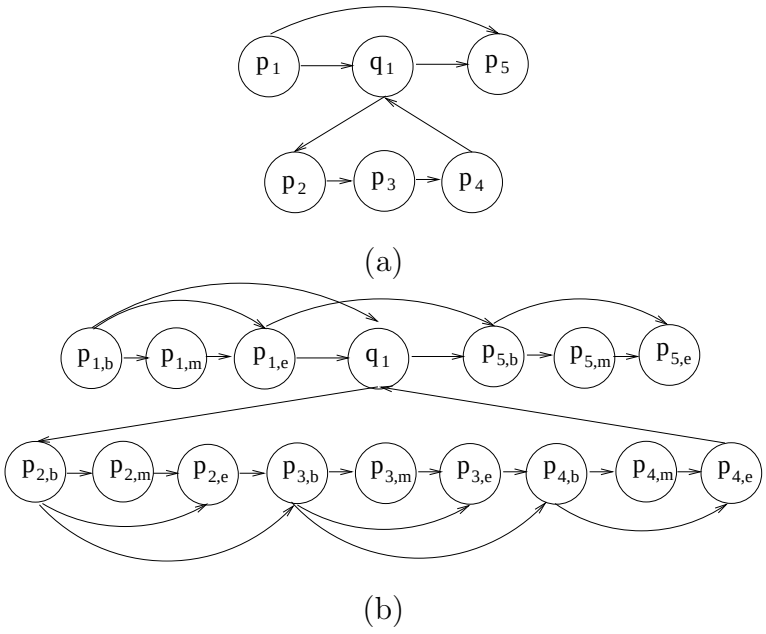


Figure 6.11: Process of structure formation; (a) original HHMM (b) HHMM with structure formation

booktitle, and *technical report*.

Figure 6.12 shows the overall F-measure for the RHHMM in two different forms; one is the model under structural formation (RHHMM-s), and the other is the RHHMM without the structure formation. The figure shows an average 3% increase on micro-averaged F-measure when structure formation is applied regardless of the volume of training data.

The structure formation has exhibits a greater improvement on the SHHMM shown in Figure 6.13. The SHHMM-s appears to be a more stable model as the F-measure reaches a stable result more quickly than the SHHMM. The F-measure increases when structure formation is applied. Once the volume of data has reach to 400 sentence, the SHHMM-s reaches to a steady results of 0.95 for its micro-averaged F-measure.

The MSHHMM can also benefit from structure formation, as shown in Figure 6.14, where MSHHMM-s achieved better extraction accuracy than MSHHMM on volumes of training data ranging from 60 to 600 sentences. The MSHHMM-s reaches a stable result

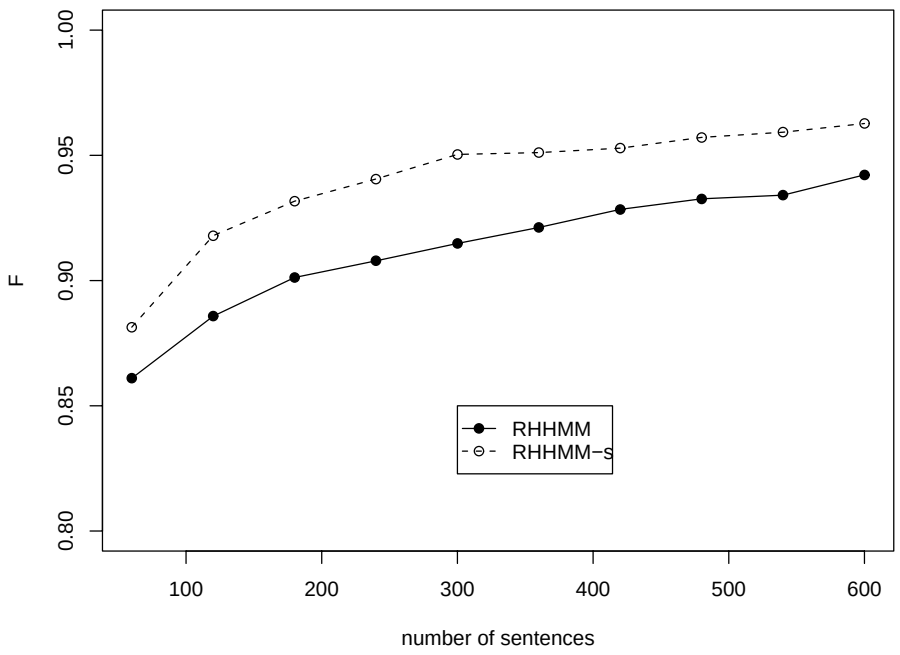


Figure 6.12: Graph of F-measure for structure formation.

more quickly than the unaltered MSHHMM. However, in comparison with the results for other models, the structure formation provides less benefit in extraction accuracy on the MSHHMM. During the testing process of the MSHHMM, each internal state is simplified into three different transformed states $\{in, stay, out\}$, such that the benefits of structure formation are minimal for the MSHHMM.

Structure formation shows the greatest improvement in accuracy when applied to the HMM. The HMM-s achieved a 7% increase in F-measure when the volume of training data is approximately 60 sentences.

Table 6.6 shows the F-measure of identified tags on reference tagging task with data volume of 600 sentences. The results show the highest extraction accuracy of 0.964 belongs to the HMM having undergone structure formation. Due to the shallow nature, in terms of hierarchical depth, involved in reference tagging task, the HMM was able to achieve

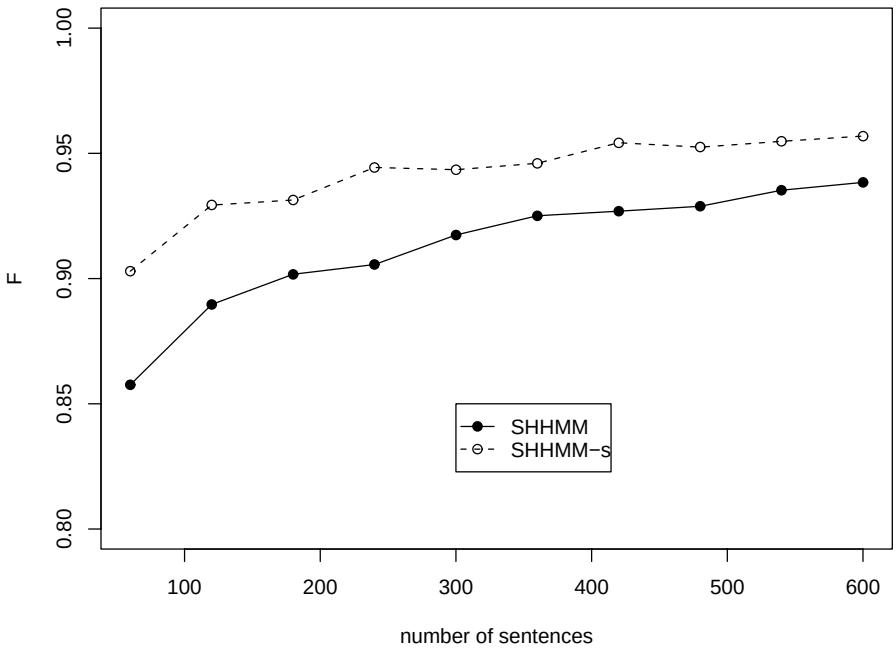


Figure 6.13: Graph of F-measure for structure formation.

better performance than the hierarchical models (RHHMM, SHHMM, MSHHMM) with data volume of 600 sentences. The effect on the HMM is more marked simply because structure formation can only be applied to production states and the HMM contains only production states.

model	original	structure formation
RHHMM	0.942 ± 0.00004	0.963 ± 0.00003
SHHMM	0.938 ± 0.00005	0.957 ± 0.00004
MSHHMM	0.937 ± 0.00007	0.954 ± 0.00006
HMM	0.939 ± 0.00007	0.964 ± 0.00005

Table 6.6: Micro-averaged F-measure of structure formation for reference tagging task

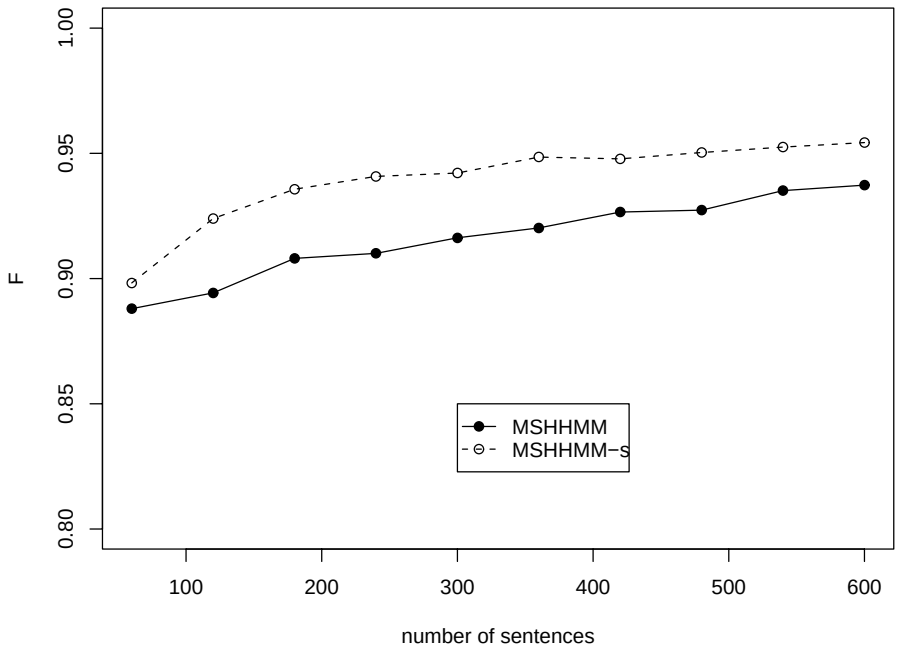


Figure 6.14: Graph of F-measure for structure formation.

6.3.2 Discussion

The experiments show the benefit of structure formation on HHMMs. The technique involves splitting production states into three sub-states, *state-beginning*, *state-middle* and *state-end*, in order to identify the occurrence of where the state is likely to begin and end. The results from the evaluation section show that the reference tagging task benefits from structure formation, a technique which increased accuracy by 2% on average. The results indicate the importance of being able to identify the state start and end, rather than the observation itself. This technique builds on work originally proposed by Chou [2006].

6.4 Partial Flattening Process for HHMM

Partial flattening is a process for reducing the depth of hierarchical structure trees. The process involves moving sub-trees from one node to another, with the aim of moving sub-trees to 'higher' nodes in the tree, thus reducing depth and complexity. By using this

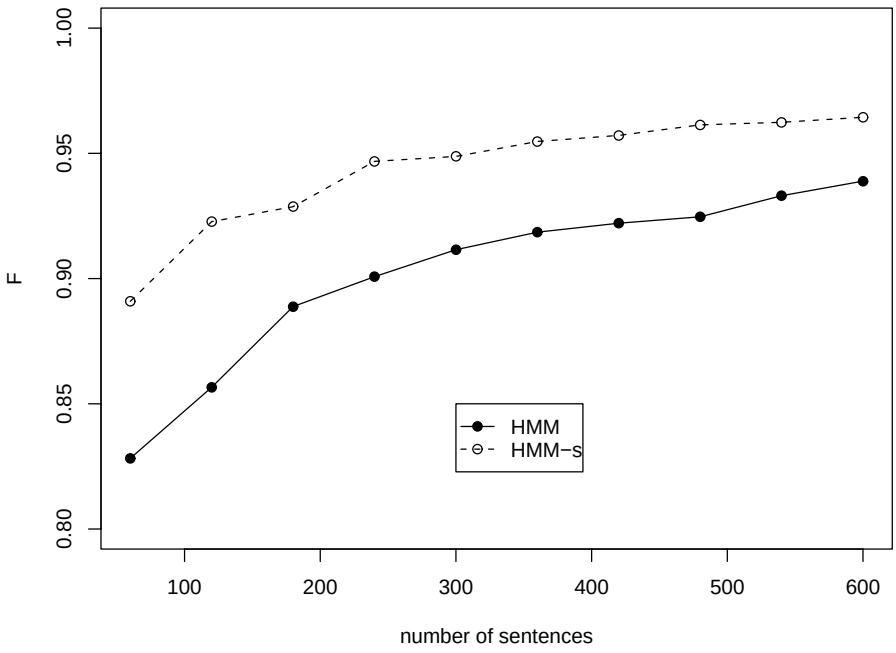


Figure 6.15: Graph of F-measure for structure formation.

process, model construction requires less training time and the model generated is more suitable for the intended task, leading to more accurate extraction. The basic idea is to transform the training data to suit the training model.

Brill [1995] applied a transformation-based error-driven learning approach to a number of natural language problems. The approach used learned linguistic information and transformed linguistic rules to improve the model for certain analytical features of the text. Drawing upon previous work, Krotov [1999] applied a probabilistic algorithm to compact the derived grammar by eliminating rules. For example,

$$VP \rightarrow VP\ NP\ PP \tag{6.11}$$

$$VP \rightarrow VP\ NP \tag{6.12}$$

$$NP \rightarrow NP\ PP \tag{6.13}$$

where (6.11) is the rule resulting from (6.12) and (6.13). In this case, the system has eliminated the need for (6.13).

6.4.1 Developing a Partial Flattening Process

This section presents an automatic partial flattening process that uses the term extractor method [Pantel and Lin, 2001]. The method discovers ways of tightly coupling observation sequences within sub-models, thus eliminating rules from the HHMM, resulting in a more efficient model. This technique uses the dependency measure between elements in an observation or state sequence. This process involves calculating dependency values, which measure the joint probability between the elements in the state sequence (or observation sequence).

This method uses *mutual information* and *log-likelihood*, which Dunning [1993] used to calculate the dependency value between words. Where there is a higher dependency value between words they are more likely to be treated as a term. The process involves collecting bigram frequencies from a large dataset, and identifying the possible two word candidates as terms. The first measurement used is *mutual information*, which is calculated using the formula:

$$mi(x, y) = \frac{P(x, y)}{P(x)P(y)} \tag{6.14}$$

where x and y are words adjacent to each other in the training corpus. The second measurement, log-likelihood, is the ratio between x and y and is defined as:

$$\begin{aligned} \log L(x, y) &= ll(\frac{k_1}{n_1}, k_1, n_1) + ll(\frac{k_2}{n_2}, k_2, n_2) \\ &\quad -ll(\frac{k_1 + k_2}{n_1 + n_2}, k_1, n_1) \\ &\quad -ll(\frac{k_1 + k_2}{n_1 + n_2}, k_2, n_2) \end{aligned} \tag{6.15}$$

where $k_1 = C(x, y)$, $n_1 = C(x, *)$, $k_2 = C(\neg x, y)$, $n_2 = C(\neg x, *)$, $C(x, y)$ is the frequency of the two words $\{x, y\}$, $*$ represents all the words in entire training corpus and

$$ll(p, k, n) = k \log(p) + (n - k) \log(1 - p) \tag{6.16}$$

The system computes dependency values between states (tree nodes) or observations (tree leaves) in the tree in the same way. The mutual information and log-likelihood values

are highest when the words are adjacent to each other throughout the entire corpus. By using these two values, the method is more robust against low frequency events.

Consider a sentence from the CoNLL-2004[1] corpus:

> Although the bidding group has n't had time to develop its latest idea fully or to discuss it with banks , it believes bank financing could be obtained .

where the sentence can be re-expressed just as its part-of-speech tags and grammar information thus:

> (S (S (SBAR IN) (S (NP DT NN NN) (VP VBZ RB VBD) (NP NN) (S (VP TO VB) (NP PRP$ JJS NN) (ADVP RB) (O CC) (VP TO VP) (NP PRP) (PP IN) (NP NNS)))) (O ,) (NP PRP) (VP VPZ) (S (NP NN NN) (VP MD VB VBN)) (O .))

Figure 6.16 is a tree representation of the HHMM, which illustrates the flattening process for the sentence. Figure 6.16(a) shows the original structure of the sentence, and the Figure 6.16(b) shows the transformed structure. The model's hierarchy is reduced by one level, where the state *NP* has become a sub-state of state S at the upper level of the tree. The process is likely to be useful when state *NP* is highly dependent on state *SBAR*.

The flattening process can be applied to the model based on two types of sequence dependency; observation dependency and state dependency.

- **Observation dependency :** The observation dependency value is based upon the observation sequence, which in Figure 6.16 would be the sequence of part-of-speech tags {IN DT NN NN VBZ RB VBD NN TO VB PRP$ JJS NN RB CC TO VP PRP IN NNS , PRP VPZ NN NN MD VB VBN .}. Given observations *IN* and *DT*'s as terms with a high dependency value, the model then re-constructs the sub-tree at *DT* parent state *NP* moving it to the same level as state *SBAR*, where the states *SBAR* and *NP* now share the same parent state S.

[1]The 2004 Conference on Computational Natural Language Learning, Boston, MA, USA, 2004, *http://cnts.uia.ac.be/conll2004*

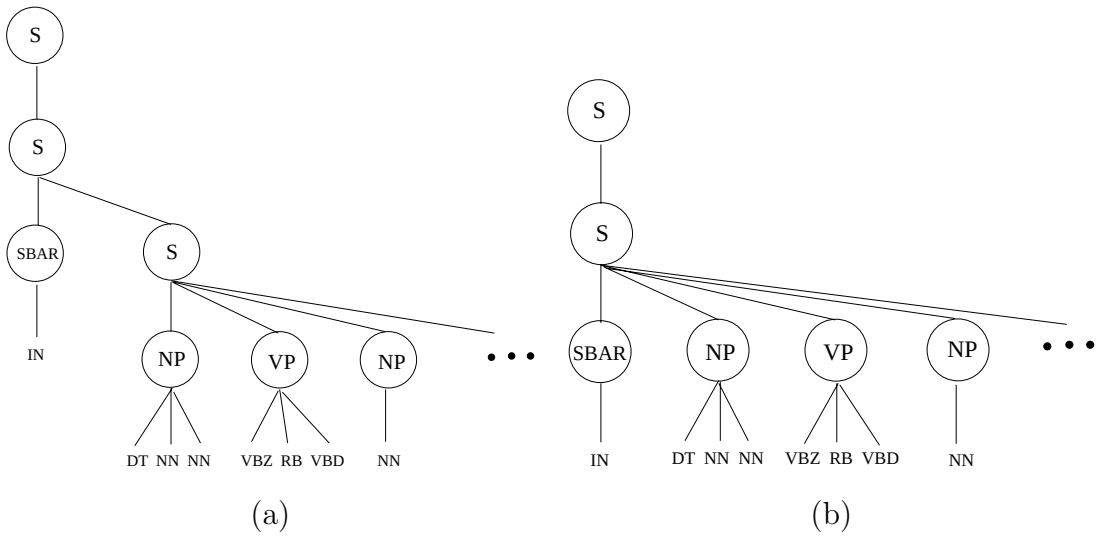


Figure 6.16: Partial flattening process for state $SBAR$ and NP.

- **State dependency :** The state dependency value is based upon the state sequence, which in Figure 6.16 would be {SBAR NP VP NP VP NP ADVP O VP NP PP NP O NP VP NP VP}. The flattening process occurs when the current state has a high dependency value with the previous state, say $SBAR$ and NP.

term	dependency value
IN DT	4532.798
DT NN	4532.798
NN VBZ	4532.798
VBZ RB	362.250
VBD NN	193.961
RB VBD	23.161
NN NN	2.334

Table 6.7: Observation dependency values of part-of-speech tags

High dependency values are determined by selecting the top n values from a list of all terms ranked by either observation or state dependency, where n is a parameter that can be configured by the user for better performance. Table 6.7 shows the dependency values of terms for part-of-speech tags from the previous example. The term {IN DT} has an

equal dependency value to {DT NN}, therefore the state *NP* is then joined with the state *SBAR* as a sub-tree of state *S* at the second level of the tree as shown in Figure 6.16(b).

This technique continues on work originally proposed by Chou [2006], which presents preliminary evidence that the partially flattened hierarchical hidden Markov model can assign propositions to language texts (grammar parsing) at least as accurately as the HMM.

6.4.2 Evaluation

To evaluate the effectiveness of the partial flattening process on HHMM, the model is applied on the text chunking task from Section 5.3, using the data from CoNLL-2004. The evaluation process is carried out upon three partially flattened hierarchical hidden Markov model—PFRHHMM, PFSHHMM and PFMSHHMM—with 2000 training sentences and 1671 test sentences.

dependency value	PFRHHMM/PFSHHMM/PFMSHHMM (770)
3000+	15
1000+	23
800+	41
600+	47
400+	56
200+	122
100+	186
50+	238

Table 6.8: Number of ranked terms for observation dependency flattening

As mentioned previously, partial flattening can be performed on two types of sequence within the HHMM. Each calculates the dependency value, either by adjacent observations or adjacent states, then transforms the training data for the system. Table 6.8 provides

a summary of the number of ranked terms that lie above a certain dependency value in the training data, where the three models share common dependency values as they are based on same set of training data. The number of ranked terms indicates the number of dependency terms that will be used, each being selected from the observation sequence—which in this task is the part-of-speech tags—according to the dependency measure of Equation 6.15. There are a total of 770 ranked terms for this set of data with 15 terms containing a dependency value greater than 3000.

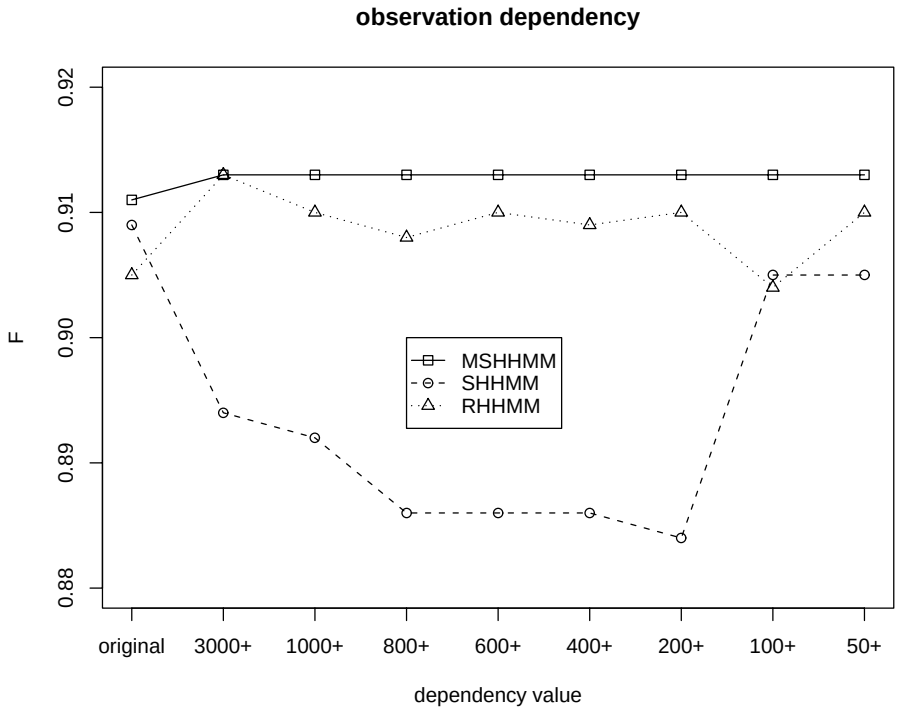


Figure 6.17: F-measure for observation dependency

The first evaluation presents the effects of applying a partial flattening process, based on observation dependency, to various hierarchical hidden Markov models, as shown in Figure 6.17. The dependency value represents the threshold chosen for a particular flattening process, where the number of ranked terms from the table above represents the number of terms that passed this threshold value. The results show that there is small increase in accuracy for the MSHHMM and RHHMM, while the threshold of the dependency value is

set at the 3000 mark. As the dependency value decreases, the accuracy of the MSHHMM remains consistent at slightly increased accuracy, while the RHHMM appears to become unstable. Meanwhile the SHHMM's performance is highly erratic regardless of dependency value.

dependency value	PFSHHMM/PFRHHMM (240)	PFMSHHMM (37)
1000+	27	5
800+	30	6
600+	34	8
400+	42	9
200+	64	10
100+	93	17
50+	124	20

Table 6.9: Summary of state dependency value

Table 6.9 provides a summary of state dependency values of ranked state terms ranging from 1000 to 50, while the number within the parentheses represents the total number of ranked terms available in the training data. The SHHMM and RHHMM share the same dependency value for their training data, because both models are required to construct each individual path of the tree in the same manner. However, due to the merging of repeated sub-model, the MSHHMM has fewer states and thus contains only 37 ranked terms for this training data set.

The second evaluation in Figure 6.18 shows that applying the partial flattening process, based on state dependency, has improved extraction accuracy on two hierarchical models; MSHHMM and RHHMM. At a dependency value of 200+ the RHHMM achieved 0.917 on extraction accuracy—the highest performance from amongst all the models. This indicates that a complex structure model (such as the RHHMM) can achieve higher extraction accuracy when partial flattening is applied. There is a less significant impact on

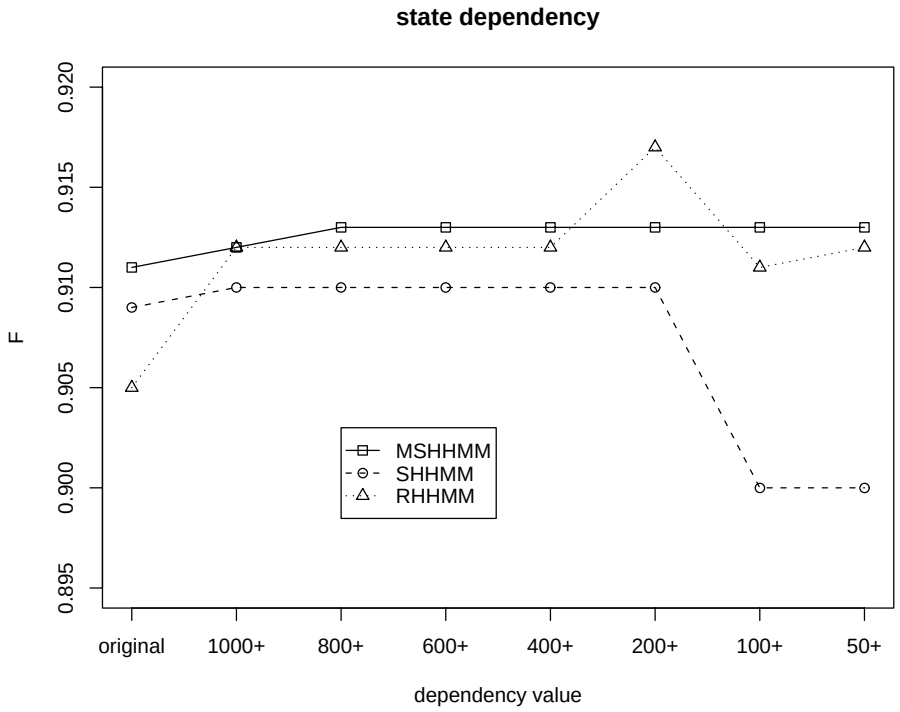


Figure 6.18: F-measure for state dependency

the simplified and merged MSHHMM during the flattening process. From Figure 6.18 there is a significant drop in accuracy when encountering dependency values ranging from 200+ to 100+ for both models, which indicates that false flattening terms (as dependency state terms) were used. False flattening occurs when states, that are highly independent to each other, are incorrectly merged.

6.4.3 Discussion

Improvements on the HHMM's performance can be achieved by applying a partial flattening process, thereby reducing the hierarchical complexity and resulting in better performance. Furthermore, this method addresses an important issue when dealing with small datasets: by using the hierarchical model to uncover less obvious structures we are able to increase performance even over more limited source materials. The experimental results have shown the potential of this method in refining a hierarchical model and providing

better handling of states with fewer observation counts. The flattening process has been shown to be effective and could be applied to many kinds of data with hierarchical structure. The method is especially appealing where the model involves complex structure or there is a shortage of training data.

6.5 Summary

This chapter discusses techniques for refining the HHMMs. These techniques fall into two categories:

- observational aspects:

  smoothing technique

  pattern generalisation.

- structural aspects:

  structure formation

  partial flattening process.

A smoothing technique was developed for the purpose of improving the model extraction accuracy. Section 6.1 demonstrates the benefit of the C-smoothing technique. The technique calculates an approximation for the unseen observation for each state, so that the model can provide better probabilistic estimations. The results show that the smoothing technique can be beneficial when processing sparse data. In the reference tagging task, the results show the advantage of using the C-smoothing technique, as an observation in the test sequence is less likely to have been seen in the training data. The overall results show the C-smoothing has a significant positive impact on the performance of the MSHHMM, the RHHMM and the HMM.

Section 6.2 illustrates two types of pattern generalisation on the reference tagging task. The methods were character class pattern generalisation and regular expression pattern generalisation. The purpose of the generalisation is to reduce the total number of unique

observation symbols within each state. The rules of the pattern generalisation involve transforming the input data by using "A" to represent all the capital letters in the observation sequence, "a" for lower-case letters, "i" for integers and all other symbols, and in particular punctuation, remain in the same form. The process reduces the chance of unseen events occurring when approaching a new data set, and results in a faster algorithm due to there being fewer observation symbols to compare. The overall results show that pattern generalisation improves extraction accuracy on the reference tagging task.

Section 6.3 demonstrates the benefit of using structure formation for the extraction accuracy of the model. States are split into three sub-states; state-beginning, state-middle and state-end, where the observation symbols in each state are independent from each other. The results show an improvement for the reference tagging task, and also highlight the importance of being able to correctly identify state boundaries—the entry and exit points from a state.

Section 6.4 presents the automatic partial flattening process. This process is capable of transforming the HHMM to a simpler structure, resulting in higher performance in terms of accuracy for the extraction task. The main idea of the partial flattening process is to capture the repeated pattern within sequences of events (i.e. state sequences or observation sequences) and re-construct the model depending on those sequences. The process uses both *mutual information* and *log-likelihood* values to transform states for a better model structure.

In summary, by applying one of more of the above techniques to the two applications that were previously mentioned in Chapter 5, the extraction accuracy of the model improves. For example, pattern generalisation is used to increase the number of observations within each state, whereas structure formation provides a more solid structure to identify the starting and the ending of a state. By utilising a smoothing method, the model is able to provide a reasonable estimation of parameters for unseen events, and partial flattening is a flexible way to reduce the complexity of the model.

Chapter 5 shows the MSHHMM improves extraction accuracy by using sub-model information to produce a more stable and more accurate model. In this chapter, the techniques discussed offer further improvement on the extraction accuracy for the MSHHMM and other hierarchical models (SHHMM and RHHMM).

CHAPTER 7

Conclusions

HMMs are widely used probabilistic tools for modelling sequences of observations. This thesis investigates modelling sequences that contain hierarchical structure, and how to make use of the structure to provide a more reliable model. The hierarchical hidden Markov model (HHMM) is appropriate when repeated sub-state structures occur within the sequence. By combining the repeated sub-models within hierarchies, the merged and simplified hierarchical hidden Markov model (MSHHMM) has the benefit of being able to merge together repeated parts of the structure, which can lead to a more reliable and accurate model for extraction tasks.

7.1 Contributions

This thesis is based on previous work on regular hierarchical hidden Markov models (RHHMMs) by Fine [1998], and sets out to prove that merged and simplified hierarchical hidden Markov models (MSHHMMs) improve efficiency and accuracy over the equivalent HMM and the RHHMM. The original contributions of this thesis are:

- *Merging technique* (Section 4.1). This was developed after the observation that by combining the repeated parts of models. The process allows for repeated sub-models to be combined during training providing more observations for that internal state, resulting in greater extraction accuracy as shown in Section 5.2.

- *Simplification technique* (Section 4.1). This made the testing process more efficient by summarising the probabilities of a sub-model into three transformed states; *state-in*, *state-stay*, and *state-out*. In Section 5.2, the results show that the SHHMM achieves a small increase in accuracy over the RHHMM.

- *C-Smoothing technique* (Section 6.1). This was used to calculate an approximation for an unseen observation, so that the model provided a better probabilistic estimation. The results have proven that the novel C-Smoothing technique achieved greater extraction accuracy than other related smoothing techniques.

- *Pattern generalisation* (Section 6.2). This was applied to simplify observation symbols. The method reduced the chances of unseen events occurring when processing a new data set. The results in Section 6.2 show that the character class pattern generalisation yielded greater extraction accuracy than the original data, and results in a faster algorithm due to there being few observation symbols to compare. This makes it appealing for tasks such as Reference Tagging that contain significant repeated patterns in their observation symbols.

- *Structure formation* (Section 6.3). This was built on the idea of emphasising boundary information between states to improve extraction accuracy of the model. The results showed an improvement for the reference tagging task, and also highlighted the importance of being able to correctly identify state boundaries—the entry and exit points from a state.

- *Partial flattening process* (Section 6.4). This reduced the complexity of the structure by transforming the HHMM to a simpler structure, resulting in higher performance in terms of accuracy for the extraction tasks.

The combination of the methods shown above, when applied to the HHMM, have been shown to result in an increase in accuracy and efficiency.

7.2 Applications

Reusing sub-state information extracted from pre-tagged source information measurably increases the accuracy and stability of a MSHHMM. Chapter 4 discusses the benefits of reusing repeated sub-models, which yields more observations for each state than in the equivalent HMM and HHMM. The overall result of Chapter 5 is that both the reference tagging and text chunking tasks benefit from the use of models that more closely reflect the hierarchical structure that exists in the source data. By increasing the number of observations for a state—a consequence of hierarchical modelling and sub-state reuse—the model is more likely to produce a stable result than its non-hierarchical counterpart.

In the reference tagging task in Section 5.2, the MSHHMM reuses the state information within *author* and *editor*, resulting in a more stable model than the HMM and RHHMM. Since only a small part of the model is reused in this task, there is only a modest increase in prediction accuracy.

The MSHHMM performs well on the text chunking task, where the model is able to make use of the hierarchical structure, and is shown in Section 5.3 to be a more stable model than the HMM. By using part-of-speech tags as the observation sequence instead of the words themselves, the total number of unique observation symbols is dramatically decreased; this results in a more stable algorithm. The number of training observations available also affects extraction accuracy. There is a direct relationship between the number of observations within each state, in proportion to the entire data, and the extraction accuracy, where states with more observations have superior performance. This factor has a greater effect on the linear HMM than the MSHHMM, because the hierarchical model provides stronger structure for each internal state. The MSHHMM also has the best efficiency, in terms of processing time, of all the models evaluated.

For either task, the greatest gain in accuracy when using MSHHMMs occurs when the training data is limited. This property makes MSHHMMs more suitable when dealing with the problem of sparse data sets. Once the volume of the training data is increased,

the performance of the MSHHMM is comparable to—or better than—that of the equivalent RHHMM or HMM.

7.3 Techniques

Due to the large vocabulary in a language, it is impossible to ensure that every word appears in the training data. Various smoothing techniques have been devised to prevent a zero probability for an unseen event. These techniques also try to adjust the probability distributions by decreasing probabilities for larger states and increasing those for smaller states. The model predicts how often the unseen event is likely to occur in each state. Because the Markov model is highly sensitive to the probability distribution within each state, over-fitting can occur when the smoothing technique is applied. To avoid this, the MSHHMM uses the C-smoothing technique to provide a small estimate for each unseen event (as shown in Section 6.1). This smoothing process is more likely to obtain the true distribution of seen events, while also producing a reliable estimation for unseen events.

The reference tagging task applies two techniques to improve extraction accuracy: pattern generalisation and structure formation (as shown in Section 6.2 and Section 6.3). Pattern generalisation involves a pattern matching technique to reduce the total number of unique observation symbols within each state. This results in a faster algorithm due to there being fewer observation symbols to compare. It can also reduce the chance of unseen events occurring when dealing with a new data set. Pattern generalisation involves transforming the input data by using "A" to represent all capital letters in the observation sequence, "a" for lower-case letters and "i" for integers. The remaining symbols remain unchanged. By applying pattern generalisation, the MSHHMM achieves better extraction accuracy than the HMM, and both models benefit from pattern matching when applied to a small training data set. Structure formation involves separating each state into sub-states to suit the nature of the state. For example, the state *title* contains structure that indicates a capital letter for the first word and a full stop after the last word, such as:

title: Manufacturing and packaged goods.

The middle states contains words with lower-case letters. At this stage separating the states proved to be a better way of handling the structure. The technique is implemented by splitting the *title* state into three sub-states; *state-beginning*, *state-middle* and *state-end*. For example, the state *title* would be split into *title-beginning*, *title-middle* and *title-end*. Applying structure formation using the state splitting technique improves extraction accuracy. This may also indicate that in a sequential extraction task it is more important to identify the beginning and end of the state than the state itself.

The partial flattening process implements a term extraction technique to expose structural patterns within natural language text. It is an automated process that can reduce the complexity of HHMMs by moving sub-trees from one node to another. It discovers ways of more tightly coupling sequences within sub-models. The overall results have shown that by applying the flattening process the HHMM can achieve high quality performance by reducing hierarchical structure. Furthermore, it addresses an important issue when dealing with small datasets: by using the hierarchical model to uncover less obvious structures, the system is able to increase model performance even for limited source materials. The experimental results show the potential of the HHMM in building a hierarchical model and providing better handling of states with smaller observation counts.

7.4 Future Work

Several topics for future research are raised by the issues discussed in this thesis. While it is beneficial to reuse sub-state information when common parent states occur, there is a notable case when sub-states should not be reused. Figure 7.1 demonstrates a situation when information within sub-states should not be reused, where the sub-models share the same parent state, but contain different child states. In this situation, the sub-model does not gain stability by combining non-repeated structure. Therefore a strategy is required for identifying which sub-states should be reused, and what notation should be used to differentiate between unmerged sub-models of the same structure.

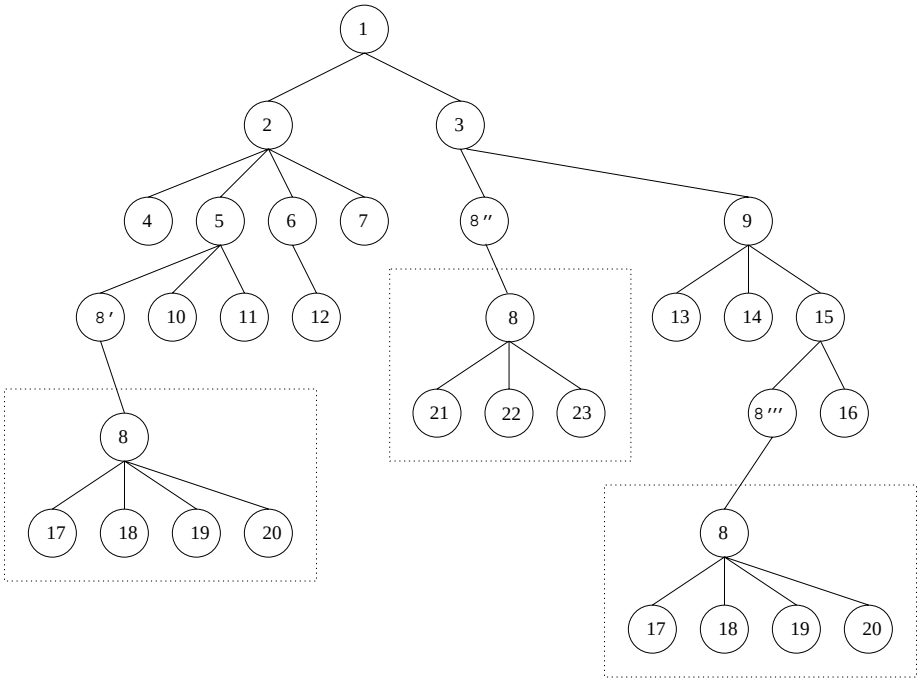


Figure 7.1: Example of different sub-states. (*Same as Figure 4.16*)

The partial flattening process has been shown to be effective and could be applied to many kinds of data with hierarchical structure. The method is especially appealing when the model involves complex structure or there is a shortage of training data. Further research is needed in both experimental and theoretical aspects of this work, specifically in the area of reconstructing hierarchies where recursive formations are present, and formal analysis and testing of our flattening techniques.

During structure formation for the reference tagging task some states have been split into three states, for example *title* is split into:

$$\text{title} \rightarrow \{ \text{ title-beginning, title-middle, title-end} \}$$

where states are split according to patterns within them. Chapter 6 indicates the importance of being able to identify the start and end state. The boundaries of sequential events give vital information to the formation process, and thus deserve further investigation. Future research may uncover further proof that the boundaries between events are more informative than the events themselves.

CHAPTER A

Appendix

A.1 Input

(A (N (F P. J. B.) (L Brown.))) (T Exploring geodemographics.) (ED (N (C In)) (N (F I.) (L Masser)) (N (C and)) (N (F M.) (L Blakemore,)) (N (C editors,))) (BT Handling Geographical Information: Methodology and PotentialApplications,) (P pages 221-258.) (PU Longman Scientific & Technical,) (AD Essex,) (Y 1991.)

(A (N (F P.) (L Borras,)) (N (F J. C.) (L Mamou,)) (N (F D.) (L Plateau,)) (N (F B.) (L Poyet,)) (N (C and)) (N (F D.) (L Tallot.))) (T Building userinterfaces for database applications.) (BT SIGMODRecord (ACM Special Interest Group on Management of Data),) (V 21) (NUM (1):) (P 32-38,) (M Mar.) (Y 1992.)

(A (N (F S.) (L Conrad)) (N (C and)) (N (F M.) (L Gogolla.))) (T An annotated bibliography on object - orientationand deduction.) (BT SIGMOD Record (ACM Special Interest Groupon Management of Data),) (V 21) (NUM (1):) (P 123-132,) (M Mar.) (Y 1992.)

(A (N (F M. P.) (L Consens,)) (N (F I. F.) (L Cruz,)) (N (C and)) (N (F A. O.) (L Mendelzon.))) (T Visualizing queries andquerying visualizations.) (BT SIGMOD

Record (ACM Special Interest Group onManagement of Data),) (V 21) (NUM (1):) (P 39-46,) (M Mar.) (Y 1992.)

(A (N (F C. R.) (L Costilla,)) (N (F M. J.) (L Bas,)) (N (C and)) (N (F J.) (L Villamor.))) (T SIRIO: A distributed informationsystem over a heterogeneous computer network.) (BT SIGMOD Record (ACMSpecial Interest Group on Management of Data),) (V 22) (NUM (1):) (P 28-33,) (M Mar.) (Y 1993.)

(A (N (F M. M.) (L David.))) (T Advanced capabilities of the outer join (SQL).) (BT SIGMODRecord ACM Special Interest Group on Management of Data,) (V 21) (NUM (1):) (P 65-70,) (M Mar.) (Y 1992.)

(A (N (F O.) (L Etzion.))) (T Pardes — A data driven oriented active database model.) (BT SIGMOD Record (ACM Special Interest Group on Management of Data),) (V 22) (NUM (1):) (P 7-??,) (M Mar.) (Y 1993.)

(A (N (F O.) (L Etzion.))) (T PARDES-a data-driven oriented active database model.) (BT SIG - MOD Record (ACM Special Interest Group on Management of Data),) (V 22) (NUM (1):) (P 7-14,) (M Mar.) (Y 1993.)

(A (N (F B. B.) (L Flynn)) (N (C and)) (N (F D.) (L Maier.))) (T Supporting display generation for complexdatabase objects.) (BT SIGMOD Record (ACM Special Interest Group on Managementof Data),) (V 21) (NUM (1):) (P 18-25,) (M Mar.) (Y 1992.)

(A (N (F S. K.) (L Gadia.))) (T Parametric databases: seamless integration of spatial, temporal,belief and ordinary data.) (BT SIGMOD Record (ACM Special InterestGroup on Management of Data),) (V 22) (NUM (1):) (P 15-20,) (M Mar.) (Y 1993.)

A.2 Results for Reference Tagging Task

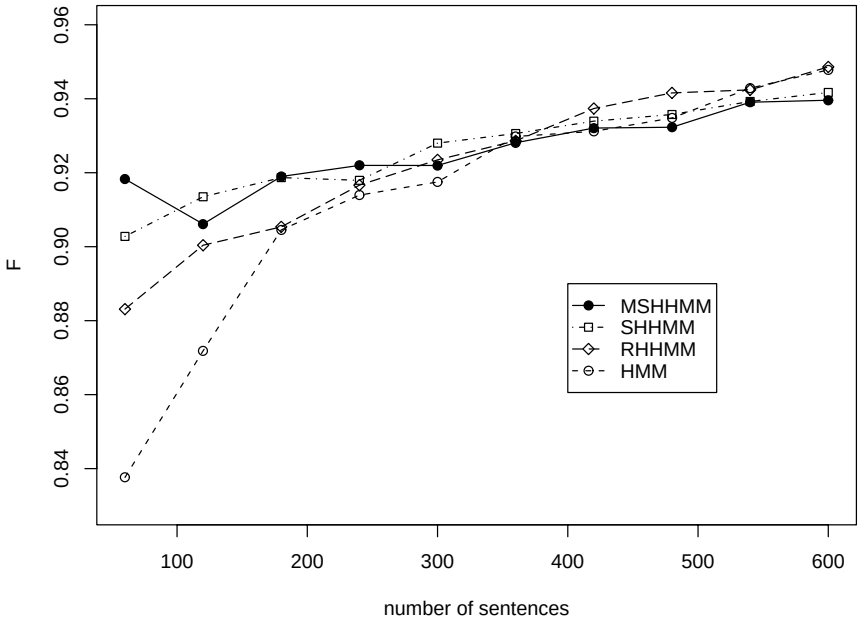


Figure A.1: Graph of F-measure for state *author*

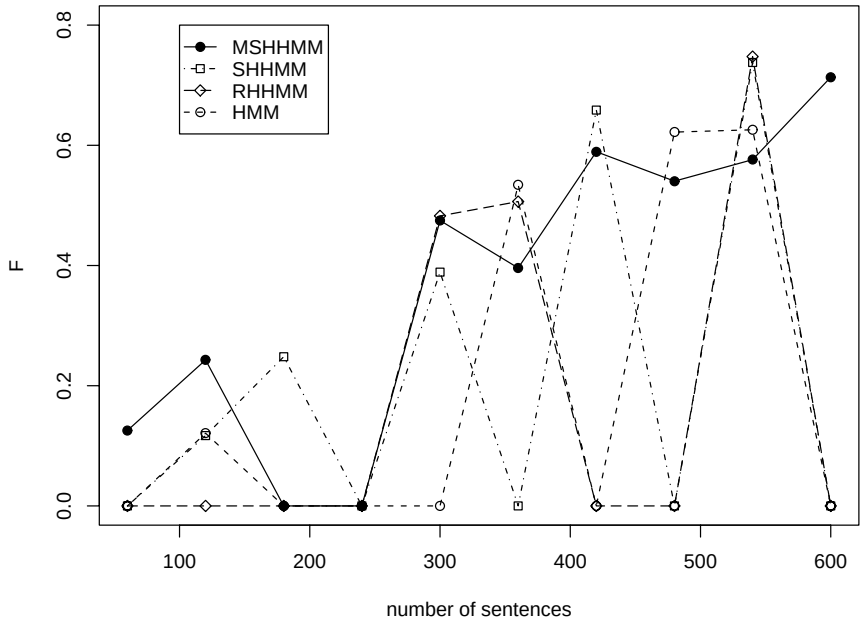


Figure A.2: Graph of F-measure for state *editor*

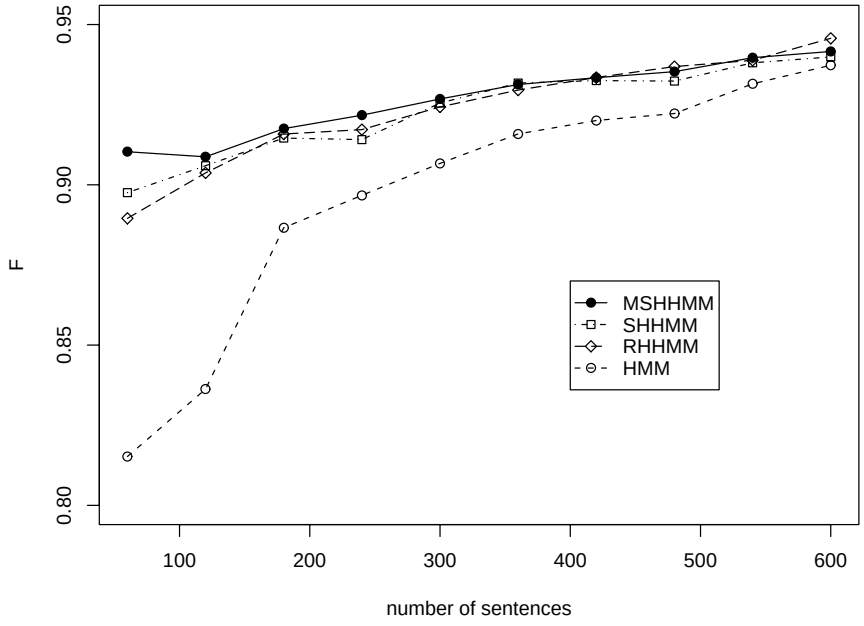


Figure A.3: Graph of F-measure for state *title*

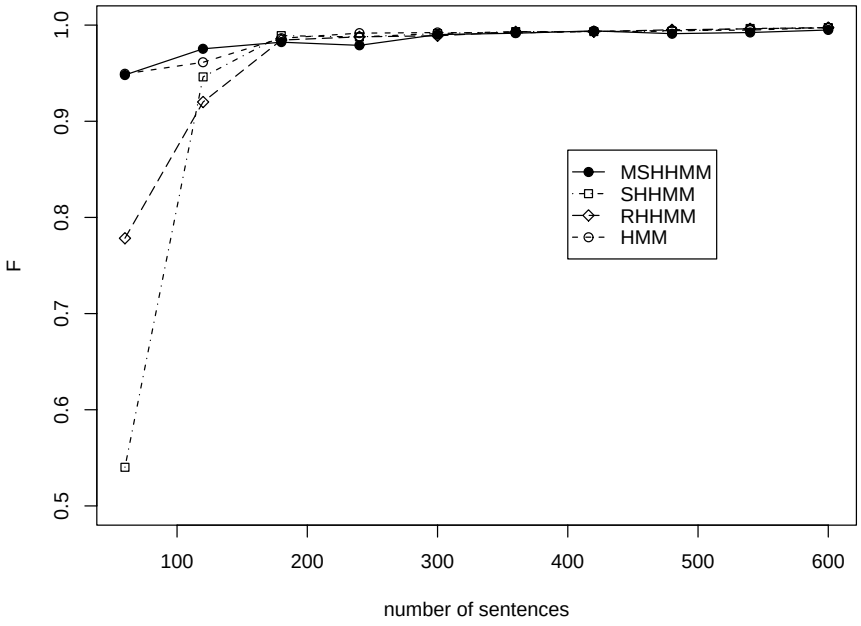


Figure A.4: Graph of F-measure for the state *year*

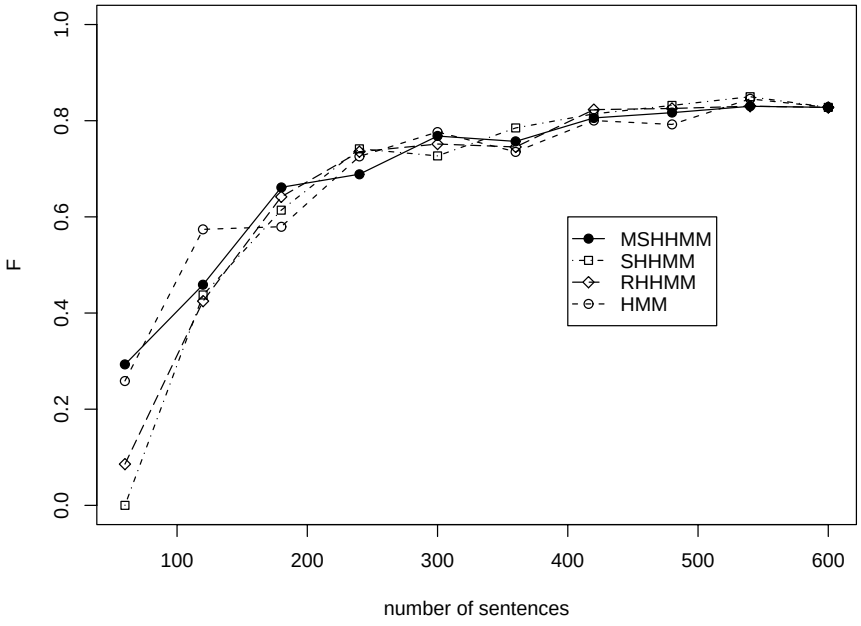


Figure A.5: Graph of F-measure for state *publisher*

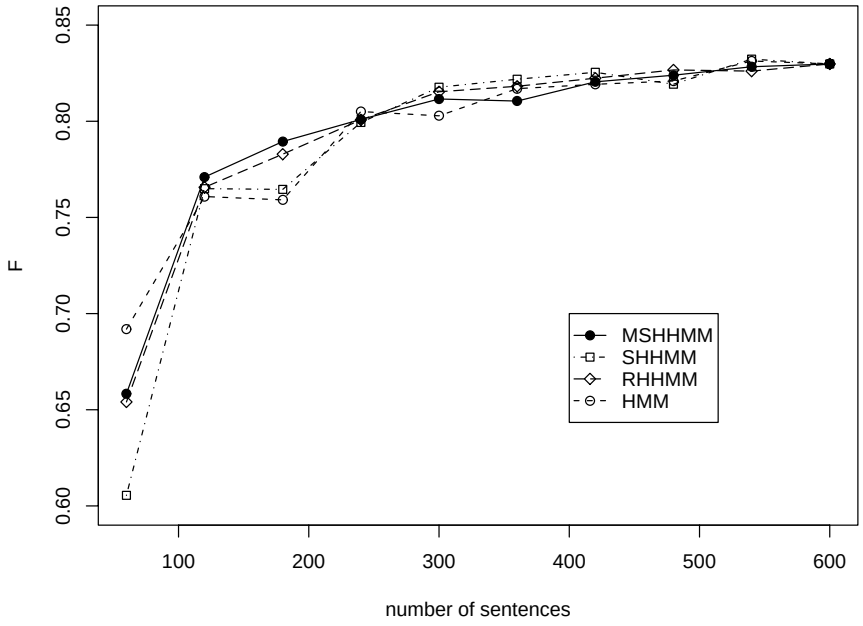


Figure A.6: Graph of F-measure for state *address*

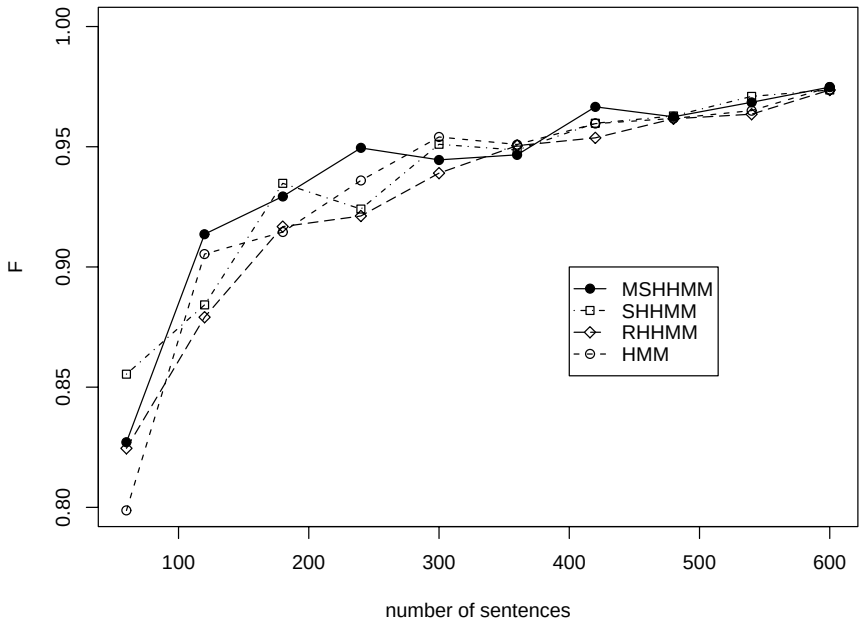


Figure A.7: Graph of F-measure for state *volume*

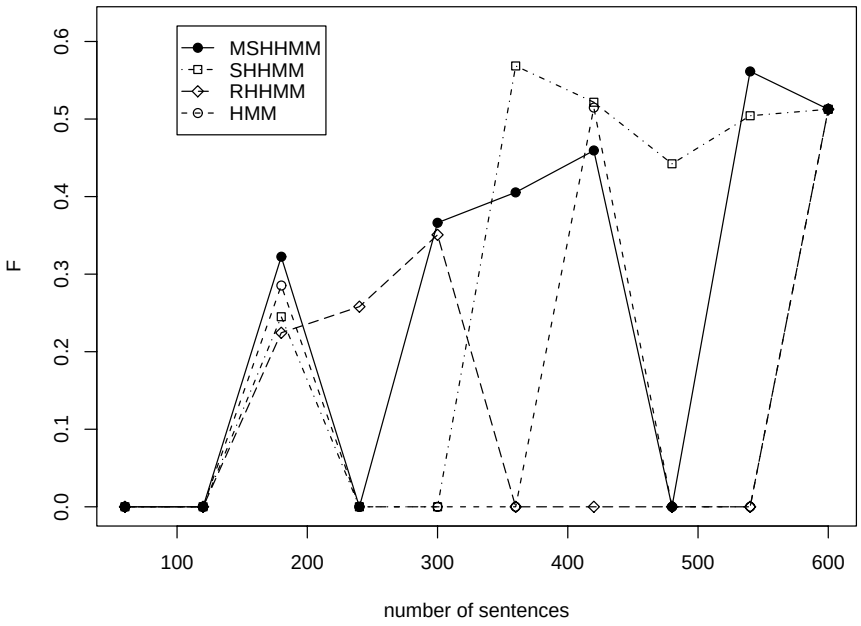


Figure A.8: Graph of F-measure for state *institution*

A.3 Results for Text Chunking Task

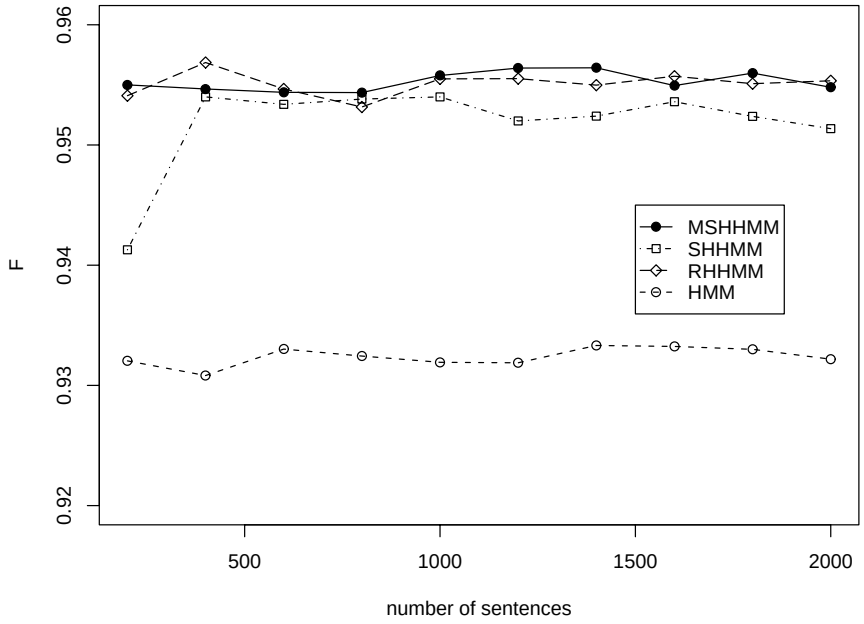


Figure A.9: Graph of F-measure for state *NP*

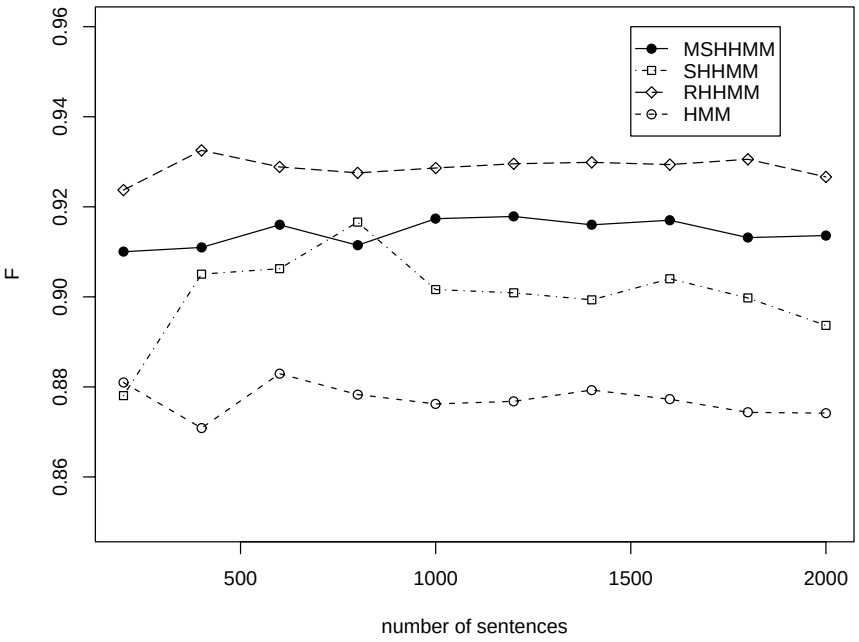


Figure A.10: Graph of F-measure for state O

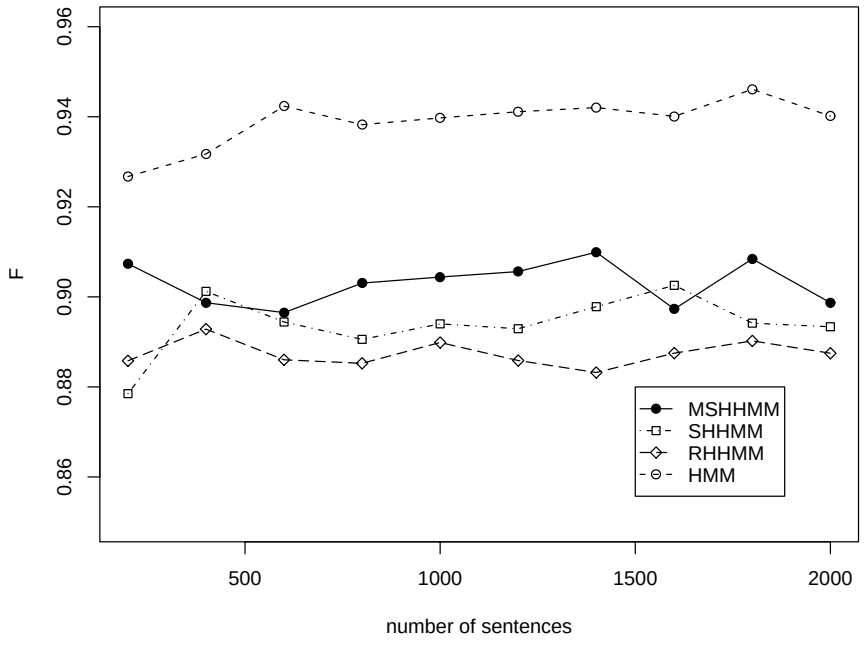


Figure A.11: Graph of F-measure for state *VP*

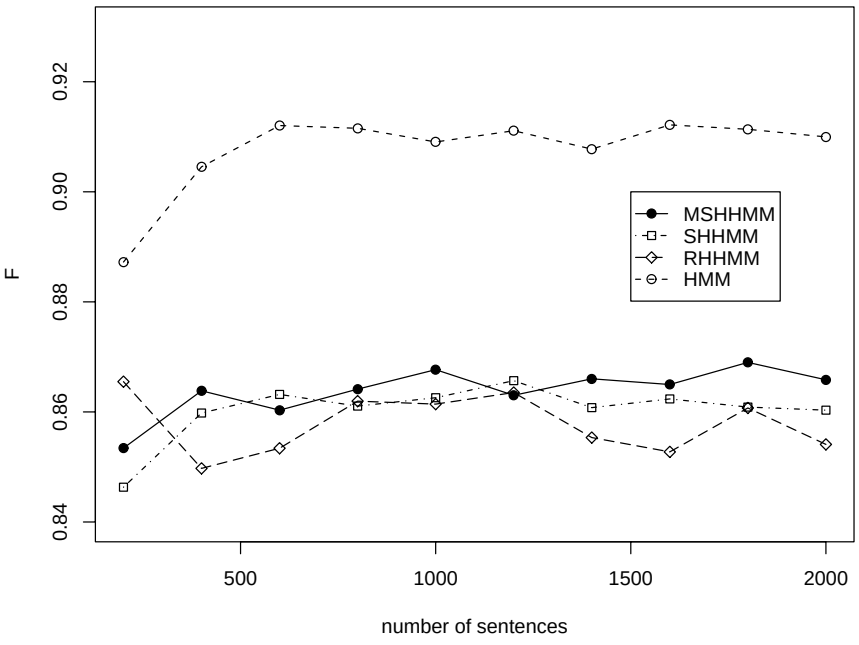


Figure A.12: Graph of F-measure for state *PP*

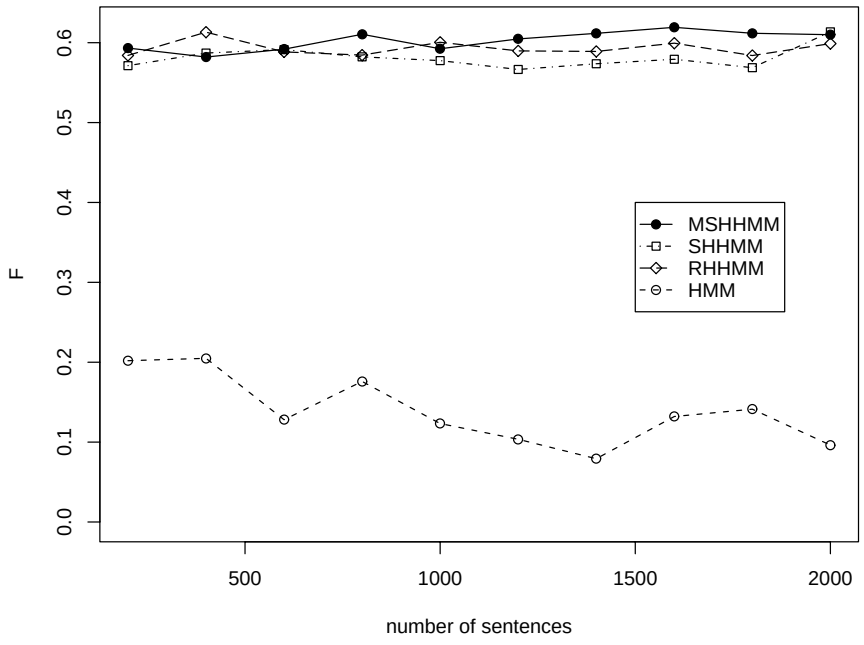


Figure A.13: Graph of F-measure for state *ADVP*

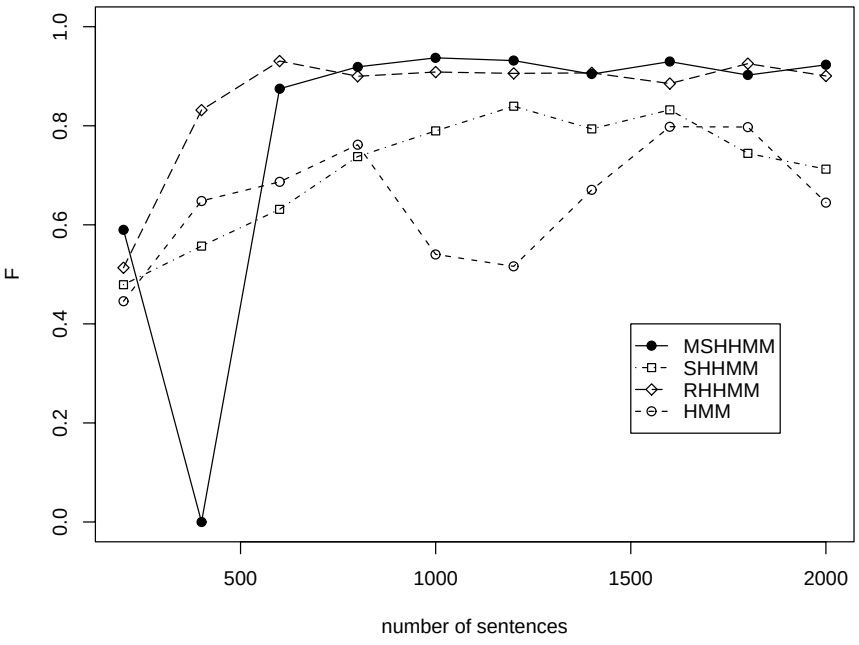


Figure A.14: Graph of F-measure for state *PRT*

Bibliography

[Abney, 1991] S. P. Abney. Parsing by chunks. In R. C. Berwick, S. P. Abney, and C. Tenny, editors, *Principle-Based Parsing: Computation and Psycholinguistics*, pages 257-278. Kluwer, Dordrecht, 1991.

[Adibi and Shen, 2001] J. Adibi and W. Shen. Self-Similar Layered Hidden Markov Model. 12th European Conference on Machine Learning (ECML'01) and 5th European Conference on Principles and Practice of Knowledge Discovery in Databases, PKDD'01, pp. 3-7, September 2001, Freiburg, Germany.

[Anderson et al., 2002] C. R. Anderson, P. Domingos and D. S. Weld Relational Markov Models and their Application to Adaptive Web Navigation. *In Proc. 8th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, pages 143-152, Edmonton, Canada, 2002.

[Banko and Brill, 2001] M. Banko and E. Brill. Mitigating the Paucity of Data Problem: Exploring the Effect of Training Corpus Size on Classifier Performance for Natural Language Processing. *In: Proc. of the Conference on Human Language Technology*, 2001.

[Banko and Brill, 2001] M. Banko and E. Brill. Scaling to very very large corpora for natural language disambiguation. *In Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pages 26–33. Association for Computational Linguistics, 2001.

[Baum and Petrie, 1966] L. E. Baum and T. Petrie. Statistical Inference for Probabilistic Functions of Finite State Markov Chains. *Annals of Math. Statistics*, vol. 37, pp. 1554-1563, 1966.

[Baum and Egon, 1967] L. E. Baum and J.A. Egon. An Inequality with Applications to Statistical Estimation for Probabilistic Functions of a Markov Process and to a Model for Ecology. *Bull. Amer. Meteorology Soc.*, vol. 73, pp. 360-363, 1967.

[Baum and Sell, 1968] L. E. Baum and G. R. Sell. Growth Functions for Transformations on Manifolds. *Pacific J. Math*, vol. 27, no. 2, pp. 211-227, 1968.

[Baum et al., 1970] L. E. Baum, T. Petrie, G. Soules and N. Weiss. A Maximization Technique Occuring in the Statistical Analysis of Probabilistic Functions of Markov Chains. *Annals of Math. Statistics*, vol. 41, no. 1, pp. 164-171, 1970.

[Berger, 1985] J. Berger. Statistical Decision Theory and Bayesian Analysis. New York: Springer-Verlag, 1985.

[Bergmark, 2000] D. Bergmark. Automatic Extraction of Reference Linking Information from Online Documents. *Technical Report TR 2000-1821*, Cornell Computer Science Department, November 2000.

[Berteskas and Tsitsiklis, 1989] D. C. Bertsekas and N. J. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*, Englewood Clifffs, New Jersey: Printice-Hall, 1989.

[Bikel et al., 1999] D. M. Bikel, R. Schwartz and R. M. Weischedel. An Algorithm that Learns What's in a Name. *Machine Learning*, vol. 34, pp. 211-231, 1999.

[Borenstein et al., 1996] J. Borenstein, B. Everett, and L. Feng. Navigating Mobile Robots: Systems and Techniques. A. K. Peters, Ltd., Wellesley, MA, 1996.

[Borkar et al., 2001] V. R. Borkar, K. Deshmukh and S. Sarawagi. Automatic Segmentation of Text into Structured Records. *Proceedings of SIGMOD*, 2001.

[Bouckaert and Frank, 2004] R. Bouckaert and E. Frank. Evaluating the replicability of significance tests for comparing learning algorithms. *Proc Pacific-Asia Conference on*

Knowledge Discovery and Data Mining. H. Dai, R. Srikant and C. Zhang (eds), LNAI 3056, Sydney, Australia, 3-12. Springer-Verlag, 2004.

[Boutilier et al., 1999] C. Boutilier, T. Dean and S. Hanks. Planning under uncertainty: Structural assumptions and computational leverage. *In JAIR*. To appear. 1999. http://citeseer.ist.psu.edu/boutilier96planning.html

[Brand et al., 1997] N. Brand, N. Oliver and A. Pentland. Coupled Hidden Markov Models for complex action recognition. in CVPR. 1997.

[Brill, 1995] E. Brill. Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part-of-Speech Tagging. *Computational Linguistics*, 21(4):543–565, 1995.

[Bui et al., 2002] H. H. Bui, S. Venkatesh and G. West. Policy Recognition in the Abstract Hidden Markov Model. *Journal of Artificial Intelligence Research*, 17:451-449, 2002.

[Bui et al., 2004] H. H. Bui, D. Q. Phung and S. Venkatesh. Hierarchical Hidden Markov Models with general state hierarchy. *19th National Conference on Artificial Intelligence* (AAAI-04), 25-29 July 2004, San Jose, California, USA.

[Caplan, 2001] P. Caplan. Reference Linking for Journal Articles: Promise, Progress, and Perils. *Portal: Libraries and the Academy*, vol. 1, no. 3, pp 352-356, 2001.

[Carroll and Long, 1989] J. Carroll and D. Long. Theory of Finite Automata with an Introduction to Formal Languages. Prentice Hall. Englewood Cliffs, 1989.

[Chang and Martinsek, 2004] Y-C I. Chang and A. Martinsek. Sequential Approaches to Data Mining. *In Sequential Methodologies*, Ed. Mukhopadhyay, Datta and Chattopadhyay, Marcel Dekker Inc.

[Chen and Chen, 1994] K-H Chen and H-H Chen. Extracting Noun Phrases from Large-Scale Texts: A Hybrid Approach and its Automatic Evaluation. *Meeting of the Association for Computational Linguistics*, 234-241, 1994.

[Cheng et al., 1997] J. Cheng, D. Bell and W. Liu. Learning Bayesian Networks from Data: An Efficient Approach Based on Information Theory. *Proceeding of the sixth ACM International Conference on Information and Knowledge Management*, 1997.

[Chien, 1999] J-T Chien. Online hierarchical transformation of hidden Markov models for speech recognition. *IEEE Transactions on Speech and Audio Processing*, vol. 7, no. 6, pp. 656-667, November 1999. (NSC89-2213-E-006-151).

[Chomsky, 1965] N. Chomsky. Aspects of the Theory of Syntax. MIT Press, Cambridge, MA.

[Chou, 2006] L. Chou. Techniques to incorporate the benefits of a Hierarchy in a modified hidden Markov model. *Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions*, July, Sydney, Australia, 2006. Association for Computational Linguistics, pp 120-127.

[Church, 1988] K. Church. A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text. *Proceedings of Second Conference on Applied Natural Language Processing*, pp. 136-143, 1988.

[Cox and Miller, 1965] D. R. Cox and H. D. Miller. The theory of stochastic processes. London, Methuen, 1965.

[Craven et al., 2000] M. Craven, D. DiPasquo, D. Freitag, A. K. McCallum, T. M. Mitchell, K. Nigam and S. Slattery. Learning to construct knowledge bases from the World Wide Web. *Artificial Intelligence*, 118(1-2): 69-113, 2000.

[Dunning, 1993] T. Dunning. Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics*, 19(1), pp 61-74, 1993.

[Durbin et al., 2001] R. Durbin, S. R. Eddy, A. Krogh and G. Mitchison. Biological sequence analysis: Probabilistic Models of Proteins and Nucleic Acids. Cambridge University Press, Cambridge UK, April 1998.

[Eikvil, 1999] L. Eikvil. Information extraction from the world wide web: a survey. Technical Report 945, Norwegian Computing Center, 1999.

[Fine et al., 1998] S. Fine , Y. Singer and N. Tishby. The Hierarchical Hidden Markov Model: Analysis and Applications. *Machine Learning*, Vol 32, pp. 41-62, July 1998.

[Freitag and McCallum, 1999] D. Freitag and A. McCallum. Information extraction using HMMs and shrinkage. In Papers from the AAAI-99 Workshop on Machine Learning for Information Extraction, pp. 31-36, 1999.

[Freitag and McCallum, 2000] D. Freitag, A. McCallum. Information Extraction with HMM Structures Learned by Stochastic Optimization. Proceedings of the Eighteenth Conference on Artificial Intelligence (AAAI-2000), pp. 584-589.

[Gelman et al., 1995] A. Gelman, J. B. Carlin, H. S. Stern and D. B. Rubin. *Bayesian Data Analysis*. Chapman & HallCRC, Boca Raton, 1995

[Ghahramani and Jordan, 1997] Z. Ghahramani and M. Jordan. Factorial hidden Markov Models. *Machine Learning*, 2: pp.1-31, 1997.

[Good, 1953] I. J. Good. The Population Frequencies of Species and the Estimation of Population Parameters. *Biometrika*, volumn 40, part 3, 4, pp. 237-264, 1953.

[Greengrass, 2001] E. Greengrass. Information retrieval: A survey. *DOD Technical Report* TR-R52-008-001, 2001.

[Hitchcock et al., 1998] S. Hitchcock, L. Carr, W. Hall, S. Harris, S. Probets, D. Evans, and D. Brailsford. Linking electronic journals: Lessons from the Open Journal project. *D-Lib Magazine*, December 1998.

[Heckerman et al., 1995] D. Heckerman, D. Geiger and D. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20(3):197–243, 1995.

[Hu et al., 2000] M. Hu, C. Ingram, M. Sirski, C. Pal, S. Swamy, and C. Patten. A Hierarchical HMM Implementation for Vertebrate Gene Splice Site Prediction. Technical report, Department of Computer Science, University of Waterloo, 2000.

[Jeffreys, 1948] H. Jeffreys. Theory of Probability. Clarendon Press, Oxford, second edition, 1948.

[Jelinek and Mercer, 1980] F. Jelinek and R. Mercer. Interpolated estimation of markov source parameters from sparse data. In S. Gelsema and L. N. Kanal, editors, *Pattern Recognition in Practice*, pages 381-402, 1980

[Jelinek and Lafferty, 1991] F. Jelinek and J. D. Lafferty. Computation of the probability of initial substring generation by stochastic context-free grammars. *Computational Linguistics*, 17(3):315-323, 1991.

[Jurafsky and Martin, 2000] D. Jurafsky and J. H. Martin. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. Prentice Hall series in Artificial Intelligence. Prentice-Hall, Englewood Cliffs, N. J. 2000.

[Katz, 1997] B. Katz. From sentence processing to information access on the World Wide Web. In Natural Language. *Processing for the World Wide Web: Papers from the 1997 AAAI Spring Symposium*, pages 77-94, 1997.

[Katz, 1987] S. M. Katz. Estimation of probabilities from spares for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing*, volumn ASSP-35, pp. 400-401, March 1987.

[Kortenkamp et al., 1998] D. Kortenkamp, R.P. Bonasso, and R. Murphy, editors. *AI-based Mobile Robots: Case studies of successful robot systems*, Cambridge, MA, 1998. MIT Press.

[Krotov et al., 1999] A. Krotov, M Heple, R. Gaizauskas and Y. Wilks. Compacting the Penn Treebank Grammar. *Proceedings of COLING-98* (Montreal), pages 699-703, 1999.

[Lari and Young, 1990] K. Lari and S. J. Young. The Estimation of Stochastic Context-Free Grammars Using the Inside-Outside Algorithm. *Computer Speech and Language*, vol. 4, pp. 35-56, 1990.

[MacEachern and Peruggia, 2000] S. N. MacEachern and M. Peruggia. Subsampling the Gibbs sampler: variance reduction. *Statistics & Probability Letters 47*, pp. 91-98, 2000.

[McCallum et al., 1999] A. McCallum, K. Nigam, J. Rennie and K. Seymore. Building Domain-Specific Search Engines with Machine Learning Techniques. In AAAI-99 Spring Symposium on Intelligent Agents in Cyberspace, 1999.

[McCallum, 2004] A. McCallum. Part-of-speech Tagging & Hidden Markov Model introduction. Talk: Introduction to Natural Language Process, University of Massachusetts Amherst, CMPSCI 585, Spring 2004.

[MacCallum et al., 2000] A. McCallum, D. Freitag and F. Pereira. Maximum Entroply Markov Models for Information Extraction and Segmentation. Proceedings 17th International Conferebce on Machine Learning, pp.591-598, 2000.

[Miller et al, 1998] S. Miller, M. Crystal, H. Fox, L. Ramshaw, R. Schwartz and R. Stone, R. Weischedel, and the Annotation Group. Algorithms that learn to extract information—BBN:Description of the SIFT System as Used for MUC-7 *In Proceedings of the Seventh Message Understanding Conference (MUC-7)*, MUC-7. Fairfax, Virginia. 1998.

[Moline and Pla, 2001] A. Molina and F. Pla. Clause Detection using HMM. *In Proceedings of CoNLL-2001*, Toulouse, France, 2001.

[Moline and Pla, 2001] A. Molina and F. Pla. Shallow Parsing using Specialized HMMs. *In Journal of Machine Learning Research*, volume 2 (March), 2002, pp. 595-613.

[Murphy and Paskin, 2001] K. P. Murphy and M. A. Paskin. Linear Time Inference in Hierarchical HMMs. In T. Dietterich, S. Becker, and Z. Gharahmani eds., Advances in Neural Information Processing Systems 14. Cambridge, MA: MIT Press, 2001.

[Murphy et al., 2001] K. Murphy, S. Russell, A. Doucet. Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks. *In Sequential Monte Carlo Methods in Practice* N. de Freitas and N.J. Gordon, editors., Springer-Verlag, New York, 2001.

[Murphy, 2002] K. P. Murphy. Dynamic Bayesian Networks: Representation, Inference and Learning. UC Berkeley, Computer Science Division, July 2002.

[Nag et al., 1985] R. Nag, K. H. Wong, and F. Fallside. Script Recognition Using Hidden Markov Models. *Proc. of ICASSP 86*, pp. 2071-1074, Toyko , 1986.

[Ney et al., 1994] H. Ney, U. Essen and R. Kneser. On structuring probabilistic dependencies in stochastic language modeling. *Computer Speech and Language*, 8:1-38, 1994.

[Pantel and Lin, 2001] P. Pantel and D. Lin. A Statistical Corpus-Based Term Extractor. In Stroulia, E. and Matwin, S. (Eds.) AI 2001. *Lecture Notes in Artificial Intelligence*, pp. 36-46. Springer-Verlag, 2001.

[Rabiner and Juang, 1986] L. R. Rabiner and B. H. Juang. An Introduction to Hidden Markov Models. *IEEE Acoustics Speech and Signal Processing ASSP Magazine*, ASSP-3(1): 4-16, January 1986.

[Raiko et al., 2002] T. Raiko, K. Kersting, J. Karhunen, and L. De Raedt. Bayesian Learning of Logical Hidden Markov Models. *In the proceedings of the Finnish Artificial Intelligence Conference*, STeP 2002, Oulu, Finland, December 2002, pp. 64-71.

[Rabiner, 1989] L. R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77 (2), pp. 257-286, 1989.

[Rijsbergen, 1979] C. J. Rijsbergen. *Information Retrieval*, second edition. Butterworths, 1979.

[Ron et al., 1994] D. Ron, Y. Singer and N. Tishby. Learning probabilistic authomata with variable memory length. *In Proceedings of the Seventh Annual Workshop on Computational Learning Theory*. 1994.

[Saggion et al., 2004] H. Saggion, R. Gaizauskas, M. Hepple, I. Roberts and M. Greenwood. Exploring the Performance of Boolean Retrieval Strategies for Open Domain Question Answering. *Proc 2004 SIGIR Workshop on Information Retrieval for Question Answering*, Sheffield, UK, July, 2004.

[Schaefer, 1997] R. Schaefer and T. Weyrath. Assessing Temporally Variable User Properties with Dynamic Bayesian Networks. In: A. Jameson, C. Paris and C. Tasso (eds.).

User Modeling - Proceedings of the Sixth International Conference UM. Springer, New Work, 1997.

[Seymore et al., 1999] K. Seymore, A. McCallum and R. Rosenfeld. Learning Hidden Markov Model Structure for Information Extraction. AAAI'99 Workshop on Machine Learning for Information Extraction, 1999.

[Slomin et al., 2002] N. Slonim, G. Bejerano, S. Fine, and N. Tishby. Discriminative feature selection via multiclass variable memory Markov model. *In Proceedings of ICML*, 2002.

[Skounakis et al., 2003] M. Skounakis, M. Craven and S. Ray. Hierarchical Hidden Markov Models for Information Extraction. *In Proceedings of the 18th International Joint Conference on Artificial Intelligence*, Acapulco, Mexico, Morgan Kaufmann, 2003.

[Sompel, 2001] H. V. D. Sompel. Generalizing the OpenURL Framework beyond References to Scholarly Works: The Bison-Futè Model. Oren Beit-Arie. *D-Lib Magazine* 7(7/8), 2001.

[Stolcke, 1994] A. Stolcke. Bayesian Learning of Probabilistic Language Models. PhD thesis, University of California, Berkeley, 1994.

[Sutton and Barto, 1998] R. S. Sutton and A. G. Barto. Reinforcement learning: An introduction. Cambridge, MA: MIT Press. 1998.

[Theocharous et al., 2001] G. Theocharous, K. Rohanimanesh, and S. Mahadevan. Learning Hierarchical Partially Observable Markov Decision Processes for Robot Navigation. *Proceedings of the IEEE Conference on Robotics and Automation* (ICRA), Seoul, South Korea, May 2001.

[Theocharous and Mahadevan, 2002] G. Theocharous and S. Mahadevan. Approximate planning with hierarchical partially observable markov decision processs for robot navigation. *In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2002.

[Thollard and Clark, 2002] F. Thollard and A. Clark. Shallow Parsing using PRobabilistic Grammatical Inference. *Proceedings of the 6th Internaltional Colloquium on Grammatical Inference: Algorithms and Applications*, pp. 269-282, Springer-Verlag, London, UK, 2002.

[Thrun, 2002] S. Thrun. Robotic mapping: A survey. In G. Lakemeyer and B. Nebel, editors. *Exploring Artificial Intelligence in the New Millenium.* Morgan Kaufmann, 2002.

[Viterbi, 1967] A. J. Viterbi. Error bounds for convolutional codes and an asymtotically optimum decoding algorithm. *IEEE Transactions on Information Theory* IT-13:260-267, 1967.

[Vogler and Metaxas, 1999] C. Vogler and D. Metaxas. Parallel Hidden Markov Models for American sign Language Recognition. In International Conference on computer Vision, Kerkyra, Greece, 1999.

[Wild and Seber, 1995] C. J. Wild and G. A. F. Seber. Introduction to Probability and Statistics. Lecture Notes for U. of A. papers 528.18x and W. U. paper 0655.121, Auckland, NZ, 1995.

[Xie et al., 2002] L. Xie, S-F Chang, A Divakaran, H. Sun. Learning hierarchical hidden Markov models for video structure discovery. *Tech. Rep. 2002-006*, ADVENT Group, Columbia University, December 2002.

[Zemel, 1993] R. Zemel. A minimum description length framework for unsupervised learning. Ph.D. Thesis, Department of Computer Science, University of Toronto, Toronto, Canada, 1993.

[Zhai and Lafferty, 2001] C. Zhai and J. Lafferty. A study of Smoothing Methods for Language Models Applied to Ad Hoc Information Retrieval. *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, ACM Press, pp. 334-342, (2001)

[Zhou and Su, 2002] G. Zhou , J. Su. Named entity recognition using an HMM-based chunk tagger. *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, July 07-12, Philadelphia, Pennsylvania, 2002.