Text Augmentation: Inserting markup into natural language text with PPM Models

A thesis
submitted in partial fulfilment
of the requirements for the degree
of
Doctor of Philosophy
at the
University of Waikato
by

Stuart A. Yeates

Department of Computer Science

The University of Waikato
Te Whare Wānanga o Waikato

Hamilton, New Zealand

July 9, 2006

Abstract

This thesis describes a new optimisation and new heuristics for automatically marking up XML documents. These are implemented in CEM, using PPM models. CEM is significantly more general than previous systems, marking up large numbers of hierarchical tags, using n-gram models for large n and a variety of escape methods.

Four corpora are discussed, including the bibliography corpus of 14682 bibliographies laid out in seven standard styles using the BIBTEX system and marked-up in XML with every field from the original BIBTEX. Other corpora include the ROCLING Chinese text segmentation corpus, the Computists' Communique corpus and the Reuters' corpus. A detailed examination is presented of the methods of evaluating mark up algorithms, including computation complexity measures and correctness measures from the fields of information retrieval, string processing, machine learning and information theory.

A new taxonomy of markup complexities is established and the properties of each taxon are examined in relation to the complexity of marked-up documents. The performance of the new heuristics and optimisation is examined using the four corpora.

Keywords: hidden Markov models, HMM, PPM, Viterbi search, part-of-speech tagging, XML, markup, metadata.

Dedication

To Jacqui,

my trapping state.

Acknowledgements

Thank you my family, for always being there.

Thank you David, Ian, Sally Jo and Matt for guidance, encouragement and technical help.

Thank you to the Royal Society of New Zealand for funding through the Marsden Fund.

Thank you to Reuters for the use of ‘Reuters Corpus, Volume 1, English language, 1996-08-20 to 1997-08-19’. Thank you to the ROCLING SIGIR for the use of the ROCLING corpus. Thank you to Kenneth I. Laws for the use of the Computists’ Communique.

Thank you to Pauline for handling the long-distance submission.

Thank you my fellow students Carl, Catherine, Dana, Dave, David, Geoff, Gordon, Hayley, Imene, Jack, John, Justin, Karl, Kathy, Lin-Yi, Mark, Mark, Shane, Stuart, Yingying, and everyone else in the New Zealand Digital Library research group. Thank you to the tutoring, secretarial and technical staff.

Thank you Aimee, Aliene, Amanda, Andraus, Andrew, Andrew, Andy, Anne, Anne, Barry, Belinda, Bill, Bob, Brent, Bret, Carolee, Caroline, Chris, Christine, Christine, Christine, Dale, Dave, Dave, David, David, Deborah, Dee, Des, Douglas, Erin, Erin, Gail Gayle, Gaylene, Georgina, Haylee, Ian, Jacqui, Jane, Janice, Jenny, Jenny, Kay, Kay, Kirsten, Kumar, Lee, Leigh, Leo, Linda, Lyn, Mandy, Matt, Maz, Micheal, Murray, Rachel, Rachel, Rachel, Rhonda, Roland, Rosie, Sam, Sam, Sara, Sarah, Shauna, Stuart, Sue, Terri, Terry, Terry, Toni, Tony, Wayne, Wendy and everyone else I’ve danced with in Christchurch, Hamilton, Auckland, London and Oxford during the course of my enrolment.

Thank you to OUCS at Oxford for the use of their resources to finish this thesis. Thank you to all the RTS crew for their encouragement. Thanks to Sebastian for the LaTeX and XML help.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Timely news is in great demand, and the value increases if the news is tightly focused on specific areas of interest to the readers. Often readers are interested in specific organisations, dates and sources, so the fragment:

> *1997 was a record spending year for computer-industry mergers and acquisitions, and companies such as Compaq, Dell, IBM, and Hewlett-Packard are still hot to buy what's left. [InfoWorld Electric, 24Dec97. EduP.]*

might be considerably more valuable to a reader if the organisations, dates and sources of information were marked up with *<o>* *<d>* and *<s>* tags, respectively:

> *<d>1997</d>was a record <d>spending year</d>for computer-industry mergers and acquisitions, and companies such as <o>Compaq</o>, <o>Dell</o>, <o>IBM</o>, and <o>Hewlett-Packard</o>are still hot to buy what's left. [<s>InfoWorld Electric</s>, <d>24Dec97</d>. <s>EduP</s>.]*

The extraction of references to company names in particular forms the backbone of systems such as `finance.yahoo.com`, which aggregate news from many hundreds of sources into thousands of tightly focused categories.

Languages such as Chinese and Japanese are usually written without whitespace segmenting the characters into words. One of the first operations that must be performed by many information systems dealing with such text is to augment it with segmentation information, for example: 小学校屋内運動場建設 is augmented to

<w>小学校</w><w>屋内</w><w>運動</w><w>場</w><w>建設</w>, ([elementary school][building interior][sports][area][construction], i.e. the construction of an elementary school indoor sports arena). Such segmented text can then be used in all the ways that words from a western language can be used [3]. The tags can then be discarded to display the text in the original form or used to process the text in the word-by-word fashion common to most western information systems, or some combination of the two.

There are many thousands, perhaps many millions, of peer-reviewed academic papers available on the Internet, each with bibliographic entries linking it to other papers and materials, for example:

> *Donald E. Knuth. Semantics of context-free languages. Mathematical System Theory, 1968, 2(2), 127–145.*

A competent researcher or librarian can readily separate this entry into all the parts necessary to find the document to which it refers. When there are collections of thousands of electronic documents, separating these manually is a huge, tedious and error-prone task. What would be useful would be a system that took the entry and automatically augmented it as:

```
<entry>
        <author>
                <forenames>Donald E.</forenames>
                <surname>Knuth </surname>.
        </author>
        <title>Semantics of context-free languages</title>.
        <journal>Mathematical System Theory</journal>,
        <year>1968</year>,
        <volume>2</volume>(<number>2</number>),
        <pages>127-145</pages>.
</entry>
```

Data in such an augmented format could then be used in a number of operations, including interloaning a copy of the document, reformatting the reference for inclusion in another bibliography, citation analysis and querying by date.

Digital library software is increasingly interacting with non-computer specialists on their own terms. This can be done using generic interfaces (witness the success of the slim-line Google interface) or interfaces tailored to the domain of the users or the content. In order to provide this, the digital library needs to know what those terms are and how they apply to the documents in the collections, whether they are organisations, dates and sources or authors, titles and dates of publication. Manual augmentation with this knowledge is typically expensive, slow and inconsistent.

This thesis describes a method for automating text augmentations for a large class of problems covering all of these examples. Such text augmentation is performed by building models from training text marked-up with XML tags, then using the models and searching to insert similar tags into testing text that does not yet contain any tags. Building effective models requires considerable volumes of training text with consistently used tags, and that the training text be representative of testing text. The text augmentation described in this thesis covers a broader range of information than preceding approaches, but is shallower than most information extraction systems in that all reasoning is fine-grained, with no higher-level or document-level reasoning, limiting the text augmentations that can be attempted.

The quality of text augmentation is evaluated by splitting a marked-up corpus into a set of training documents and a set of testing documents; training a model on the former; stripping the tags from the latter; augmenting the stripped testing documents using the model; and finally comparing the testing documents as augmented by the system with the original documents. Several different methods to compare the augmented document to the original are explored in this thesis.

1.1 Plan of the Thesis

Following this introduction, Chapter 2 gives the background to the current work, starting by examining the nature of text and an overview of methods of extracting information from text. Approaches to evaluating the correctness and efficiency of the such extractions are then examined, together with ways on encoding extracted information in XML. Chapter 3 introduces Markov models built from text, and algorithms for search using such models to extract information.

Chapter 4 discusses the architecture of the implemented system, and examines the rationale for some of the design choices. It then presents an optimisation and a number of heuristics, and examines the search spaces of different classes of problems with respect to these. Chapter 5 introduces the corpora used in this thesis.

Chapter 6 sets out the experimental results of the optimisation and heuristics on the corpora. Chapter 7 concludes the thesis with an overview of the research, a list of the original contributions, and a summary of unanswered questions. The appendix contains samples from each of the corpora used in this work.

1.2 Thesis Statement

Text augmentation is the automated insertion of XML tags into documents in the context of a digital library to make implicit textual information accessible to conventional processing.

> Text augmentation can be expanded to a larger class of problems than those previously studied. It can be partitioned into three classes of problem: segmentation, classification and entity extraction. Each class of problem has distinctive properties, computational complexity and types of failure, necessitating different evaluation methodologies.
>
> Markov models and searching can be used to solve these problems. Given the context of their application, there are a number of optimisa-

tions and heuristics which can be used to make these algorithms computationally tractable.

Text augmentation is a computational process by which natural language text is augmented by the addition of XML tags to elucidate the implicit structure. Three different classes of text augmentation are discussed. Each class has a structurally different schema which affects the performance and evaluation of text augmentation.

Text augmentation is performed using statistical modelling techniques, such as hidden Markov and PPM models, and using searching algorithms to find a good augmentation. In the past, text augmentation has been performed using Teahan search (see Section 4.5), but in this thesis a variety of algorithms is used. Viterbi search is computationally intractable in many interesting text augmentation situations, but an optimisation of it, and a number of heuristics to it, can be exploited, given the application, to make searching computationally feasible.

To these ends, this thesis aims to:

1. Examine text augmentation problems, in the large, to attempt to determine which are susceptible to automated text augmentation and whether some sets of problems are inherently easier than others.

2. Build a text augmentation system capable of solving at least as wide a range of problems as existing low-human-input systems, with an eye to eventual inclusion as part of a digital library system.

3. Locate and/or build corpora to test this system.

4. Find specific heuristics and optimisations which perform well in relation to a particular set of augmentation problems.

5. Evaluate both the text augmentation system and the heuristics and optimisations in the system.

These aims are reviewed in Section 7.1.

Chapter 2

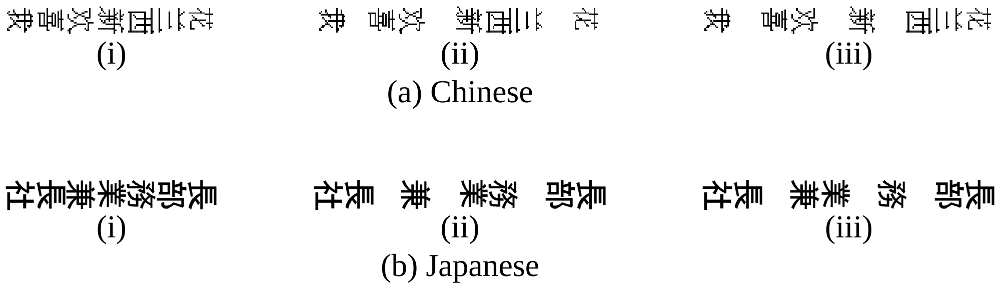
Background

This chapter examines the background to the current work. First it looks at the nature of text, various types of ambiguities in natural language text and then examines metadata, namely explicit information about text. Information extraction systems, whose purpose is to extract metadata from text, are then surveyed and various methods of evaluating such extraction systems are examined, together with methods of evaluating the correctness and efficiency of such systems. Finally, aspects of XML and Unicode relevant to text augmentation are surveyed.

2.1 The nature of text

One task in text augmentation is the Chinese text segmentation problem, the task of segmenting a stream of Chinese characters into words. The task is often the first step in Chinese information processing systems, since Chinese is normally written without explicit word delimiters. The task is made more challenging by the fact that line delimiters may occur anywhere, including between letters in a word or digits in a number [42].

The task is harder than it appears because Chinese text is ambiguous. The text shown in Figure 2.1(a)(i) (taken from [137]) can be segmented as shown in (ii) or as shown in (iii), meaning ‘I like New Zealand flowers’ and ‘I like fresh broccoli’ respectively. Similarly the Japanese title shown in Figure 2.1(b)(i) (taken from [3]) can be segmented as shown in (ii) or as shown in (iii) meaning ‘president both business and general manager’ and ‘president (of) subsidiary business (for) (the proper



(a) Chinese

(b) Japanese

Figure 2.1: Examples of segmentation ambiguity in east Asian languages.

name) Tsutomu, general manager' respectively. Since this last is four nouns and thus identical from the point of view of a part of speech system, it is a particularly ambiguous situation.

2.1.1 Ambiguity

Segmentation ambiguity is not confined to Asian languages. There is a widely circulated joke featuring sentence segmentation ambiguity in English:

> Dear John: I want a man who knows what love is all about. You are generous, kind, thoughtful. People who are not like you admit to being useless and inferior. You have ruined me for other men. I yearn for you. I have no feelings whatsoever when we're apart. I can be forever happy—will you let me be yours? Gloria

and

> Dear John: I want a man who knows what love is. All about you are generous, kind, thoughtful people, who are not like you. Admit to being useless and inferior. You have ruined me. For other men, I yearn. For you, I have no feelings whatsoever. When we're apart, I can be forever happy. Will you let me be? Yours, Gloria

There is an entire class of English expression, double entendre, which exploits ambiguity of meaning [128]. This ambiguity is resolved using context—the style and genre of a piece of text. A sentence with two possible meanings has the more risqué meaning if it appears in a Blackadder [38] script and has the less risqué of the two if it appears in a Reuters' dispatch. There are also forms of text in which resolving ambiguity of meaning is not possible, a well-known example of which is Lewis Carroll's poem 'Jabberwocky'.

Ambiguity resolution using context is an example of what is known in artificial intelligence as 'common sense reasoning'. It is known to be difficult for computers to resolve such ambiguity, with the difficulty lying in the wide range of world-knowledge and subtle reasoning that humans use to solve this class of problem [107].

Partly to reduce the need for ambiguity resolution, the overwhelming majority of text mining is performed on collections of text with uniform style and genre. Uniformity of linguistic style highlights the patterns and structures within the text and the uniformity of genre ensures that the patterns have the same meanings.

2.1.2 Metadata

Metadata means 'a set of data that describes and gives data about other data' [128]. Usually at the granularity of the document (the catalogue entry for a book or the title and author of a web page), metadata can be at the character level [5] or cover entire collections of documents (Table 2.1). In many systems and standards much of the metadata is stored at the document level, even though it may apply to the collection, section or even character level, because this is the level at which most processing, storage, licensing, retrieval and transmission operations take place. The RDF standard [156] is notable for granularity independence, addressing, individual tags (elements), documents or collections of documents.

This thesis centres on fine-grained metadata, at the character and word levels,

Granularity	Relevant metadata
Collection	Scope; purpose; coverage; copyright; maintenance status; maintainer contact details;
Document	Author; title; date of publication; subject classification;
Section	Topics; cross references;
Sentence	Semantic meanings;
Word	Part of speech; glossary links; dictionary links; collation order;
Character	Encoding; reading direction; case;

Table 2.1: Metadata at different granularities.

and how such metadata can be inferred from, and then annotated into, the text itself. This process of augmenting the text is referred to as text augmentation. It has been previously called 'tag insertion' [136, 135], but the author believes that *'text augmentation'* better portrays the action and intent of the process.

2.2 Extraction of Textual Information

A wide range of distinct approaches and many hybrid ones have been used to extract fine-grained information from text for various purposes. This section reviews several of them, including regular expressions, machine learning and information extraction. The following section examines how to measure the correctness of the extraction.

2.2.1 Regular Expressions

Regular expressions are compact representations of a set of strings which can be converted into a finite-state machine. The machine can efficiently recognise instances of the set of strings within a stream of text. Their close relationship to the well-studied field of formal language parsing has led to them being well understood [2].

Regular expressions are the tool of choice for extracting information with an exact and precise format, such as email addresses, post codes, dates and the like.

They are, however, fragile in the face of mistakes, ambiguity and stylistic variations in the text.

2.2.2 Handcrafted Rules

Handcrafted rules or templates can also be used to extract information from text. These typically involve searching for short fragments of text or regular expressions within text, with each rule processed in order of precedence. Unfortunately, systems of handcrafted rules can be complex and fragile in the face changing input data. They also scale poorly with the number classes of information being extracted, particularly when there is a requirement that rules do not overlap.

These systems typically can consider large windows and potentially have access to 'out of band' sources of information such as dictionaries and name lists [17, 1, 74].

2.2.3 Instance Based Machine Learning

Instance based machine learning is a field concerned primarily with classifying instances into classes. Machine learning can be applied to text [149], but requires that the text be pre-segmented into instances, potentially losing significant information and/or leading to large instances.

Machine learning handles noise and ambiguity significantly better than regular expressions. Mis-classified instances, once detected, can be added incrementally to the training instances, allowing an existing model to be refined and improved. The widely-used Brill tagger [28] uses this approach as a primary method.

2.2.4 Information Extraction

The field of information extraction typically involves multi-step systems that first extract atoms from text (using regular expressions, part-of-speech tagging, etc.) and then use higher-order reasoning to solve ‘real world’ problems. The Text REtrieval Conferences series (TREC) [53, 54, 142, 143] is built round text retrieval tasks and the Message Understanding Conferences (MUC) and Document Understanding Conferences (DUC) are built around competitions between systems. The intent is to focus research and systems development towards specific, known targets.

MUC Named Entity [35] problems centre on the extraction of proper nouns (e.g. company names), often with subsidiary information (e.g. market symbols or addresses) from stylised information sources, typically news articles such as the Reuters’ corpus. The problems set in the MUC tracks explicitly required the extraction of facts from the texts into a separate database and subsequent higher-order reasoning about those facts, in two separate systems. Many involve multiple steps, such as sentence and word segmentors, part-of-speech taggers, hypothesis generators, hypothesis evaluators and disambiguators [167].

The systems include many opportunities for encoding handcrafted or externally curated domain knowledge, from the notion of the word embedded in the word segmentors, to domain-specific word lists used in the part-of-speech tagger and handcrafted heuristics for template filling. Word lists include lists of first names, corporate names, colleges and universities, corporate suffixes, times and dates, world regions and state codes [23]. Many of the systems use trained models, either learnt rules or Markov models, but only for an individual step of solving the problem.

Many of these systems and corpora suffer from proper-noun ambiguity errors (see Sections 2.1.1 and 5.1). Methods employed to overcome the ambiguity include leveraging company and personal titles (*Mr*, *Ltd* and *Corp.*) [22]) and deeper parsing to detect structures such as standard formatting of place names.

The GATE system is a Java GUI framework for linguistic engineering. It incorporates a wide variety of tools for using hand- or tool-generated rules, and regular expressions and links to gazetteers of cities and organisations. Testing and evaluation tools are included for classification problems. GATE focuses on the inclusion of extra-textual information:gazetteers, word-lists, grammars and similar, and their interactive development to solve particular problems. It also has tools for higher-level reasoning about texts[1] [37, 22, 95]. GATE's choice to have a GUI enables it to allow display and input of multiple texts and scripts: 21 are supported.

Citeseer [80] uses a two-stage approach, with an edit distance metric to merge similar references across the entire collection and then a hand-crafted 'invariants first' heuristic that parses those parts of the reference with the fewest differences first and uses standard machine learning on them. The system was able to leverage two extra-document sources of information, tables of common western personal names and repetition of the same reference (often in slightly different form) in multiple documents. Citeseer does not parse the diversity of fields that occur in the bibliography corpus, instead focusing on the title and author fields which are also extracted from the start of documents and which link most easily to external sources in the bibliography at the end. The public interface of the Citeseer system allows end-users to correct the extracted fields and add the missing ones. It is not clear whether feedback from these corrections is applied to the internal algorithms.

2.2.5 Markov Modelling

A number of systems and approaches have used Markov models to extract information from, or add information to, text. The early Xerox tagger [40, 39] uses hidden Markov models and Viterbi search to good effect, but handles unseen words and novel contexts poorly.

Built using arithmetic encoder [102] models, one for 'good' text and one (called a 'confusion model' [36]) for errors, the TMT (Text Modelling Toolkit) and later SMI (Statistical Modelling Interface) systems [134, 36] can correct errors in text and classify textual fragments [133, 26]. With a large number of options and supporting a wide range of static and adaptive models, SMI is entirely capable of solving the

[1] `http://gate.ac.uk/`

news and Chinese examples given in the opening Chapter, but not the bibliographic example, because SMI models are not recursive; they cannot represent a hierarchy of textual fragments.

Arithmetic encoder models provide slightly more information than conventional Markov models, providing an ordering of symbols as well as probabilities represented using integer ratios. Integer ratios avoid using floating-point arithmetic to whose inaccuracies arithmetic encoding is particularly sensitive. These steps make SMI useful for both textual augmentation and full text compression.[2]

Freitag and McCallum [46, 96] report work on a bibliography corpus using hand-crafted, then automatically shrunk, Markov models, giving good results. Freitag and McCallum build models with increasingly complex structures in a similar manner to Dynamic Markov Compression (DMC) [151], which are then blended using linear combination.

Recently Besagni et al. [15] have had some success in marking up bibliographies using part of speech tagging, building chains of which parts of speech occur in which bibliographic fields and then correcting fields using a post-processing step. As with the post-processing performed in part of speech tagging, this includes super-adjacency. They use six tags and get a recall (see Section 2.3.1) of between 82% and 97% of the time for a corpus of 2500 references. Not all of the failures are complete failures, since sometimes part of a name is successfully returned. This may be useful, depending on the context.

2.2.6 Trained versus Handcrafted Models

The use of automatically trained models rather than handcrafted models lends itself to use in situations where training data is cheaper or more accessible than domain-knowledgeable humans. With the increasing volumes of data available at the cost of transfer on the Internet and the relatively stable cost of labour, using large amounts

[2] ‘Full text compression’ in this context means lossless compression, as opposed to the lossy compression often used for images which effectively destroys text [151].

of training data rather than people is likely to be an increasingly attractive choice.

While much of the freely available material for training models is of low or questionable quality, the existence and growth of curated repositories such as the Oxford Text Archive,[3] the Linguistic Data Consortium[4] and Project Gutenberg[5] suggest that the availability of curated textual and linguistic materials is increasing.

There are limits on what trained models can recognise, because of the finite training text available, their lack of 'common sense' reasoning and various theoretic limits [13]. For example, most model training and template building systems cannot recognise structures characterised by n *a*'s then n *b*'s followed by n *c*'s. While systems can be built to recognise these structures for a particular n, it is not possible to recognise these structures for unknown n's with a regular expression while rejecting structures with different numbers of *a*'s, *b*'s and *c*'s. These limits do not apply to handcrafted models. Handcrafted models run into the well-known difficulties of hand-building large, complex systems [83] and labour costs.

Building and maintaining a set of handcrafted rules or a handcrafted model may be more cost effective than building a corpus of documents with the concepts marked-up if the documents are sufficiently rare or sufficiently difficult to handle (for example they contain embedded private or confidential information). Handcrafting is also more attractive if the concept is well understood by non-specialists, meaning labour is relatively cheap.

Trained models also have the option of automated incremental improvement by using the Baum–Welch algorithm [10, 11] in production situations. Long-term use of Baum–Welch may result in divergence and poor performance. However, if the data seen in production is changing at a rate faster than this divergence, then using the Baum–Welch may be advantageous. This thesis focuses on trained models.

[3] `http://ota.ahds.ac.uk/`

[4] `http://www.ldc.upenn.edu/`

[5] `http://www.gutenberg.org/`

2.2.7 Single Step versus Multiple Step Systems

Multiple step text augmentation systems have an advantage over single step systems in allowing a different choice of algorithm for each step, providing the system builders with a wider range of options and making the intermediate forms accessible for ‘boosting’ using word lists and similar. A wider range of choices for systems builders enables them to hand-select algorithms that perform well on the expected input for the systems. Unfortunately, this often leads to poor performance on other input: other genre, other character encodings and other languages.

Multiple step text augmentation systems also encourage reuse of system components, such as the Brill part-of-speech tagger, which is widely used as a preprocessor [37]. Single step augmentation systems can be reused as a whole, but are not as amenable to the development of UNIX-style ‘pipelines’. Corpora used to train models and rules are amenable to incremental development, either by adding additional documents of the same type or by adding documents in additional languages, as is common in corpora used in comparative linguistics. Steps can also be arranged in a cascade or waterfall [68].

This thesis focuses on single-step markup processes using Markov models. There is no theoretical reason why the systems and approaches used here could not be used as individual steps within a multiple system, but training data for the intermediate stages appears to be rarer, except where the individual step has already been studied in isolation, as with part-of-speech tagging.

2.3 Correctness

The ultimate test of a computer system is in terms of interactions with users—does the system work correctly? Are any errors made, minor or catastrophic? Is it fast enough? Is it easy to use? Do the users like it? These questions, however, are hard to phrase in terms that allow the answers to be compared among systems, versions of the same system, and software packages across time in the face of changing requirements, user expectations, groups of users and operating environments. They are also hard to ask of sub-systems that provide a subset of functionality required by a full system.

There are, however, two features of overall performance which are widely used for comparing systems: correctness and efficiency. This section examines these and how they can be applied to text augmentation.

The approaches to measuring correctness examined here come from the fields of information retrieval, string processing, machine learning and information theory.

2.3.1 Recall and Precision

The information retrieval paradigm [122, 6] assumes that a query (single operation) retrieves a set of items, some of which are relevant to the query. Evaluation is based around the question 'Is item n relevant and was it returned?' The answer to this question puts each item into one of four distinct classes: true positive (relevant and retrieved), true negative (not relevant and not retrieved), false positive (not relevant and retrieved) and false negative (relevant and not retrieved).

Accumulating counts of each of these four classes over a large number of independent experiments allows the calculation of two higher-level measures. Recall [31] is the proportion of all relevant items that were retrieved:

$$Recall = \frac{number\,of\,relevant\,items\,retrieved}{total\,number\,of\,relevant\,items\,in\,collection} = \frac{true\,positives}{true\,positives + false\,negatives}$$

Precision is the proportion of retrieved items that are relevant:

$$Precision = \frac{number\,of\,relevant\,items\,retrieved}{total\,number\,of\,items\,retrieved} = \frac{true\,positives}{true\,positives + false\,positives}$$

Recall and precision represent a trade-off. A system could return many items (for high recall and low precision) or few items (for low recall and high precision) and so they are sometimes expressed as their harmonic mean:

$$F - measure = \frac{2 \times recall \times precision}{recall + precision}$$

Often the number of false negatives is unknown, such as when retrieving documents from the World Wide Web, whose exact size is unknown but large [81]. When the number of false negatives is known (or can be reliably estimated), another measure, called ‘Fallout’ [84], which is a measure of how good the result is as a result for the negated query, can be used:

$$Fallout = \frac{number\,of\,irrelevant\,items\,retrieved}{total\,number\,of\,irrelevant\,items\,in\,collection} = \frac{false\,positives}{false\,positives + true\,negatives}$$

Fallout measures how effectively irrelevant items are winnowed from the query results. Fallout is rarely used, as it is sensitive to the size of the collection and the addition of clearly-irrelevant items to the collection. Recall, precision, and their combination in the F-measure, are the primary means of evaluating correctness in information retrieval systems.

2.3.2 Edit Distance

Edit distance is a standard technique in the string processing field. It is a well-studied measure used in spelling correction [73, 89] (where transposes are common because of the mechanics of typing) and Optical Character Recognition (OCR) [73] (where swaps are common due to mis-recognition of one character for another). These research fields measure edit distance on data, whereas when used in text augmentation, edit distance is used on combined data and metadata with an expectation that errors be closely linked to the metadata.

Edit distance is performed in terms of individual tags rather than tag-pairs. False negatives (inserts) and false positives (deletes) are counted and then summed to get an edit distance.

Edit distance is solely concerned with mistakes made in text augmentation and neither true negatives nor true positives impact on edit distance. Edit distance explicitly recognises the sequential nature of text but, because true positives are ignored, the independence problems discussed in relation to recall and precision do not occur in edit distance calculation. Teahan [133] uses edit distance to evaluate text augmentation and Nahm et al. [106] uses edit distance as an input to a multi-stage text mining system. All edit distances used in the current work are normalised for document length to give edits per character.

2.3.3 Confusion Matrices

Whereas recall and precision assume an underlying binary classification, confusion matrices are a tool for evaluating many-class classification tasks, and are widely used in machine learning for evaluating such tasks [149]. The following is a confusion matrix for a classification problem with i classes:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,i} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,i} \\ \vdots & \vdots & \ddots & \vdots \\ a_{i,1} & a_{i,2} & \cdots & a_{i,i} \end{bmatrix}$$

The matrix is square, with a row and a column for each class. $a_{m,n}$, in column n and row m, is the number of symbols that should have been classified in class n that were actually classified in class m. Correct classification is indicated when $n = m$, on the leading diagonal of the matrix.

Any non-zero numbers off the leading diagonal, indicate misclassification and there is often symmetry about the diagonal. Non-zero numbers in both $a_{n,m}$ and $a_{m,n}$ indicate that if symbols of class m can be mistaken for symbols of class n, then symbols of class n are also likely to be mistaken for symbols of class m. This ability to highlight confusion between tags makes the confusion matrix an excellent tool for fine-tuning tagsets and finding markup errors. For example, Bray et al. [26] used a confusion matrix to find errors and demonstrate the strong correlation between name tags and place tags in the Computists' corpus. Confusion matrices are conventionally normalised by converting the rows into percentages.

2.3.4 Entropy

Entropy is a measure from information theory widely used in signal processing, error-correction and compression fields of computer science [102, 151]. It is inversely related to probability. A 'good' augmentation of text has a high probability and a low entropy (measured in bits per character) [13].

Unlike other measures of correctness, entropy does not measure results against a predefined answer, but rather measures how closely a set of results matches a model. This is effective in situations where perfect answers are either unobtainable or obtainable only at great expense.

For entropy measures to be an effective measurement of accuracy of an augmentation of text, the model used to measure entropy must be independent of both the testing and training data. This problem is closely related to the over-fitting problem in machine learning, and can be avoided by training two models on separate training data and using one to augment the text and one to measure entropy.

If an independently trained model is unavailable, an untrained model can be used with an adaptive algorithm. This is the standard methodology for measuring the strength of lossless compression algorithms [152, 103, 13].

An entropy measurement is relative to a model, and so conveys little clear knowledge about the absolute quality of an augmentation: the user of augmented text is unable to infer as much from an entropy measurement as from a recall/precision pair or an edit distance. It can, however, be used to compare the relative merit of different augmentations of the same text, provided the model captures pertinent details and the same model is used to calculate both entropy measurements.

2.3.5 Hybrid and Other Measures

Many reports of text augmentation use a combination of measures to report their results. For example Bray [26] decomposed tag insertion evaluation in the Computists' corpus into a pair of operations, firstly segmenting characters into tokens and, secondly, classifying the tokens into their respective types.

The segmentation operation was measured in terms of the error count (false-negatives + false-positives), and classification of the segments was measured using confusion matrices. Other systems use measures expressed in terms of their interaction with larger information systems, such as extraction of acronyms [165] and bibliographies [21].

2.4 Efficiency

Computer programs can be written in a wide variety of computer languages and run on a wide variety of platforms. Since the efficiency of these languages and platforms varies widely, it is useful to compare algorithms independent of their language and platform. One methodology which allows this is time complexity analysis using 'big O notation' [70]. The function is simplified to remove constant factors and is referred to as $\mathcal{O}$.

Time complexity analysis is defined in terms of a characteristic operation—in the case of tag insertion this is visiting a node in the search space—and counting how many times the operation is performed, and expressed as a function of the parameters and input size of the algorithm.

The size of the search space is normalised by the document length to give a measurement in terms of search space per character. There are special cases when searching at the start and end of documents, but for the corpora used in this thesis the initial and final characters in documents are low entropy, so they should not effect this normalisation.

2.5 XML Tags

EXtensible Markup Language (XML) [25] tags have a name (or type), span a (potentially empty) range of text and have a (potentially empty) set of attributes. The tags may be nested, but only strictly hierarchically. Thus, if a document has tags indicating pages from the physical document, it may also have tags indicating lines and, because each line is wholly within a page, the tags are hierarchical. A tag which contains only hierarchical tags, or no tags at all, is said to be well-balanced.

An XML document has an enclosing, top level, tag holding information about the document as a whole. An XML document that is well-balanced is said to be well-formed.

XML cannot directly represent overlapping hierarchies (such as the physical and logical document layout), unlike the preceding SGML [51] which had a feature, CONCUR, which permitted overlapping tags. XML can represent non-hierarchical tags using higher-order structures, using empty tags with attributes which associate them in pairs or in a sequence. The difficulties of tagging overlapping structures, and standard ways of overcoming them, are described in detail in [130].

There are several schema languages for describing which XML tags may occur within other XML tags. The W3C schema language includes an ANY tag to refer to any well-balanced tag [43]. Schemas which feature the ANY tag are flexible but challenging to model, because literally anything can be encoded, including structures equivalent to entire documents of the type being marked up.

2.5.1 Nested Tags

The XML standard largely attempts to avoid statements about the semantics of tags and the semantics of nested tags, other than their well-formedness. It is tempting to extend practice in XHTML to cover XML. In XHTML *<em><a href="…">…</a></em>* is typically considered semantically equivalent from *<a href="…"><em>…</em></a>* because most presentation engines (browsers) present these identically. Presentational customisation systems such as CSS [24] and XSLT [155], however, have no difficulty differentiating these two situations and the XML standard is silent on their semantic relationship. One can imagine a (fictional) programming language expressed in XML in which the semantics are clearly different. For example

```
<if cond="undefined(symbol)">
   <define name="symbol">
       <action/ >
   </define >
</if >
```

has different semantics to

```
<define name="symbol">
    <if cond ="undefined(symbol)">
       <action/ >
    </if >
</define >
```

The current work attempts to avoid making semantic assumptions such as this, except explicitly in the state-tying heuristic (see Section 4.3.7).

2.5.2 Attributes of Tags

The current work focuses exclusively on direct representations and does not consider attributes during training or testing (with the exception of attributes of the document-level node). All of the corpora used in this thesis have been created or transformed, as described above, to convert attributes into tags.

Attributes are syntactic sugar and any XML document with attributes can be transformed into one without attributes and back in a lossless fashion. For example, the tag <*word partofspeech*="*verb*">*jump*</*word*> can easily be transformed to <*word*><*verb*>*jump*</*verb*></*word*> but such transforms can lead to combinatorial explosion of tags if there are large number of attributes or the attributes contain large numbers of unique values. Real-valued attributes would lead to an infinite number of tags, one for each possible value. If the order of attributes of a tag is significant, the situation is significantly worse. The XML standard is silent on the question of whether the order of attributes is significant, but several subsidiary standards, including XSLT [155] and DOM [154] do not even permit discovery of the order of tags. The author knows of no use of an XML corpus in which the order of attributes is significant or of toolsets which support the processing of such XML.

2.5.3 Other Issues

A key feature XML shares with many other natural language processing approaches is the linearisation of language. While written language across a wide range of cultures is laid out in rectangular regions, whether read left-to-right and top-to-bottom, or bottom-to-top and right-to-left, digitised language—written or spoken—is almost always linear to the detriment of any secondary rectangular structure. For example, the limerick shown in Figure 2.2 is shown twice, first with the secondary

The limerick packs laughs anatomical
Into space that is quite economical.
But the good ones I've seen
So seldom are clean—
And the clean ones so seldom are comical.

(a)

The limerick packs laughs anatomical Into space that is quite economical. But the good ones I've seen So seldom are clean—And the clean ones so seldom are comical.

(b)

Figure 2.2: A limerick shown with and without secondary structure.

rectangular structure and then without. The second form of the limerick has the same rhymes and cadence as the first but loss of the explicit rectangular structure makes it harder to recognise. None of the data dealt with in this thesis has a strong secondary rectangular structure.

XML can be canonicalised [25], a process which, amongst other things, standardises whitespace. This is a lossy operation, whitespace can contain information, particularly about line and paragraph boundaries which is lost by canonicalisation. For this reason all operations preparing the corpora used in this thesis are performed without canonicalisation and preserve whitespace.

Standardisation for representing annotated linguistic data in XML [25] is currently underway, led by the Architecture and Tools for Linguistic Analysis Systems (ATLAS)[6] [78]. The standardisation work includes a content-independent method of specifying regions and anchors in linear linguistic signals, and a query language over those regions and anchors. Similar work, with greater implemented functionality, is being undertaken by the Linguistic Data Consortium[7] [20, 19]. As with the current work, these approaches embed the inferred information within the linguistic

[6] http://www.nist.gov/speech/atlas/

[7] http://www.ldc.upenn.edu/

data rather than removing it to the document header or an external data store as in most information extraction.

The current work is based on the Unicode and a subset of XML restricts the types of texts and annotations which can be easily worked with. With the exception of attributes, most of the important features of documents in modern information systems can be represented. By using Unicode and XML a range of data preparation and processing tools is available. A range of corpora is available for reuse in XML and, by using XML for the corpora produced in the current work, their potential for reuse is higher than if non-standard formats had been used.

Chapter 3

Models and Algorithms

This chapter examines Markov models and some of the searching algorithms that operate on them. Exhaustive treatment of many aspects touched on here can be found in the standard texts [63] and [13].

3.1 Markov Models

Markov models are Finite State Machines (FSMs) which consist of a finite number of states and the transitions between them. In a probabilistic FSM, each transition has an associated probability and generates (or predicts) a symbol from some alphabet of symbols. The FSM has a set of start states (often only one) and a set of end states (again, often only one). A stream of data is generated by a FSM by starting in one of the start states and moving through a succession of states (using the current state's probability density function to determine the next state) until it reaches an end state. An excellent review of the use of Markov models and similar statistical techniques as applied to language processing can be found in McMahon and Smith [99].

Markov models encapsulate the Markov assumption: that 'the value of the next state is only influenced by the value of the state that directly preceded it' [41]. The Markov assumption is useful because it gives a bound on how much system context needs to be modelled. Markov models produce probability density functions, which estimate the likelihood of each possible value for the next state.

Problem	Observable Sequence	Hidden Sequence	Observable Alphabet Size	Hidden Alphabet Size	Type	Ref.
Chinese word segmentation	Characters	Words	Large	2	Segmentation	[137]
English sentence segmentation	Words	Sentences	Large	2	Segmentation	[133]
Part-of-speech tagging	Words	Word classes	Large	≈ 50	Classification	[28]
Phone identification	Digitised, audio waveforms	Phones	Very large	Large	Entity extraction	[166, 33]

Table 3.1: Observable and hidden sequences for a variety of linguistic problems tackled with hidden Markov models.

3.2 Hidden Markov Models

Hidden Markov models (HMM) are composite models involving a number of hidden states each of which contains a complete Markov model. The hidden states typically represent the information the model is designed to infer, the words to be segmented or the parts of speech to be distinguished between.

Table 3.1 shows some of the wide variety of previous uses of hidden Markov models in linguistic problems. Chinese word segmentation and English sentence segmentation use simple models. Part-of-speech tagging, which has already been discussed, has a larger hidden alphabet and thus more hidden models.

Phone identification is a key step in voice recognition in which digitised audio waves are mapped to phones, speech sounds, which are later built into words [166]. HMMs are also widely used in computational biology [72, 9, 27].

A key property of hidden Markov models that makes them so widely used in these fields is that they handle noisy and ambiguous data well, unlike rule-based systems which are based on a series of binary decisions and are relatively brittle in the face of noise and ambiguity. Markov models are, however, much less convenient for the extraction of pertinent details. While rule-based systems have sets of rules, typically with clear means of identifying the most important, Markov models have matrices of hundreds, or even hundreds of thousands, of numbers, with none being clearly more important than others.

3.3 Higher Order Models

Higher order Markov models involve a relaxation of the Markov assumption, allowing multiple states to be taken into account [41]: 'the values of the next state are only influenced by the values of the n states that directly preceded it'. Each Markov model of order $k > 1$ is isomorphic with a family of Markov models of order $k-1, k-2, k-3, \cdots 3, 2, 1$.

Figure 3.1 shows this isomorphism for an FSM with a two-character alphabet. Figure 3.1(a) shows an order 3 Markov model, with a single state and eight (2^3) transitions, each starting and finishing in the single state, and transition probabilities dependent on the previous two characters. Figure 3.1(b) shows an isomorphic order 2 Markov model, in which the number of states has been multiplied by the size of the alphabet. The same eight transitions shown in Figure 3.1(a) appear in Figure 3.1(b), with all transitions generating an *a* leading to state *a* and a *b* leading to state *b*. Although the transition probabilities are still dependent on the previous two characters, the immediately previous character is implicit in the state and transitions are labelled with only the previous-but-one character.

Figure 3.1(c) shows an isomorphic order 1 Markov model: again the number of states has been multiplied by the size of the alphabet; and again the same 8 transitions appear. Generating a pair of '*a*'s leads to state *aa*, generating an *a* then a *b* leads to state *ab*, and so forth. In this case the proceeding two characters are implicit in the state. Such order 1 models can then be used in software and tools such as HTK [166].

Computational linguistics uses terms such as n-gram, uni-gram, bi-gram and tri-gram [73, 120, 3] to denote the order of models, while information sciences refer to the order of models [4]. Table 3.2 shows the relationship between these two terminologies.

Markov models are often represented using a table, with cells representing the transition probabilities between each pair of states and each symbol, but these grow

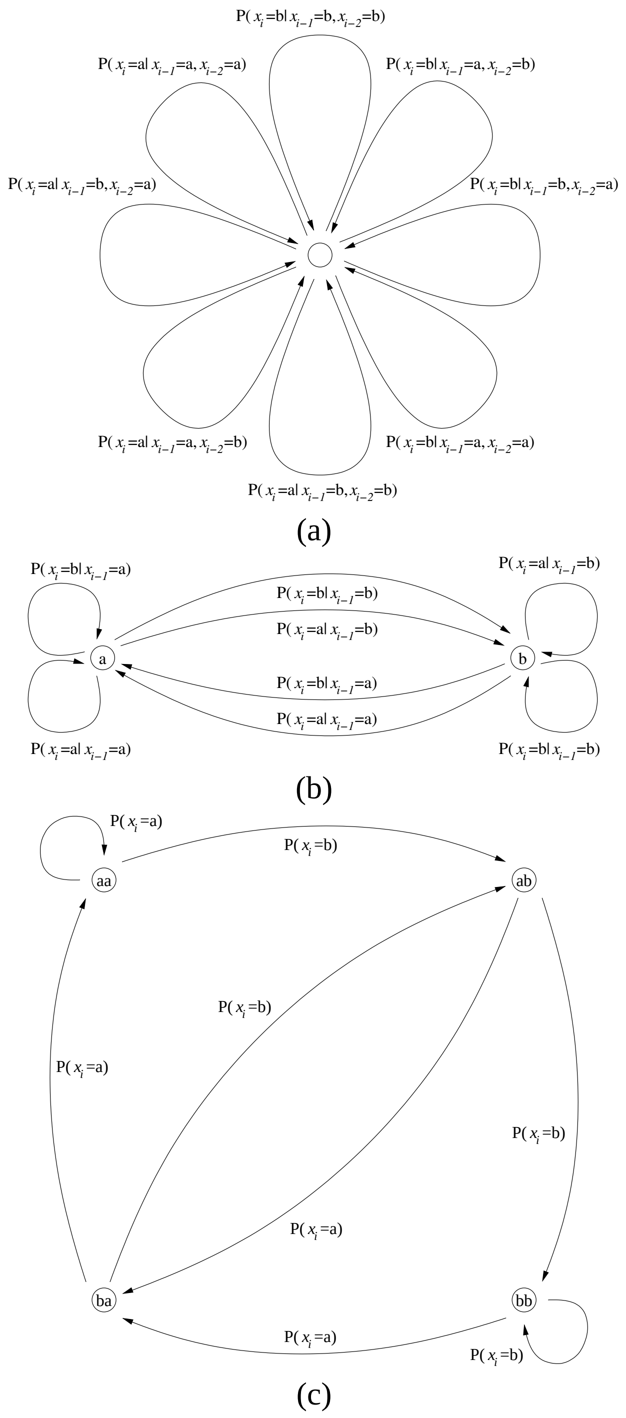


Figure 3.1: Isomorphism in Markov models. (a) an order 3 model, (b) an order 2 model isomorphic to (a), (c) an order 1 model isomorphic to (a) and (b).

n-gram	Order	Meaning
	-1	All symbols to be equal probability
Uni-gram	0	Symbol probability based on their frequency in training data
Bi-gram	1	Symbol probability based on their frequency in training data following the previous symbol
Tri-gram	2	Symbol probability based on their frequency in training data following the previous two symbols
Quad-gram	3	Symbol probability based on their frequency in training data following the previous three symbols
...	...	...
n-gram	$k-1$	Symbol probability based on their frequency in training data following the previous $k-1$ symbols
$n+1$-gram	k	Symbol probability based on their frequency in training data following the previous k symbols

Table 3.2: n-gram models and models of order k.

large for high-order models, as the size is s^k entries, where s is the alphabet of observable symbols and k is the order of the model. The isomorphism between higher- and lower-order models preserves the number of transitions, meaning that the table for a lower-order model has the same number of entries as the higher-order: it is not possible to reduce the table size by using the isomorphism demonstrated in Figure 3.1.

Even with large amounts of training data, it is unlikely that every state and transition of a high-order model is visited during training. The remaining untravelled transitions have zero probability, meaning that the model may generate zero probabilities for a sequence seen during testing. The problem, called the ‘zero-frequency problem’ [146], appears when no non-zero transition exists from the current state to the state that generates the next symbol in the observable sequence. (In hidden Markov models there can be more than one transition, each emitting a different symbol (or symbols) in the hidden sequence.) The zero-frequency problem is often solved by shrinkage (also known as backing off and smoothing [34]), namely the use of a simpler model to estimate probabilities for zero-frequency transitions in more complex models.

Many later systems use n-gram methods together with specialised handling of novel characters. Such systems are effective in tackling problems such as Chinese text segmentation partly because of the large character sets involved. Typically this involves the introduction of a special token (or character) to model the concept of an unseen character.

The differences between this approach and the normal n-gram models are highlighted by the handling of a known character between a pair of novel characters: $\ldots a\ b\ A\ d\ B\ f\ g \ldots$. In the current work the unknown characters *A* and *B* are modelled by escaping back to the order -1 model and the known character *b* is seen in a context which has never been seen before (an order 0 model). The introduction of a synthetic novel character *N* would enable a probability of encountering the sequence $\ldots a\ b\ N \ldots$ to be estimated, then $\ldots a\ b\ N\ d \ldots$ and $\ldots a\ b\ N\ d\ N \ldots$ etc., all without escaping back to the order -1 model. This effectively allows the concept of 'the character following a novel character' to be modelled, something conventional n-gram models cannot do. Part of the reason such techniques are so important is that novel characters in Chinese text, like novel words in English, are often nouns [133]: significant information can be inferred simply from novelty.

The zero-frequency problem can solved using escape methods [146], a recursive case of shrinkage in which unseen transition probabilities are estimated by reference to a lower-order model. Other cases are also common in information extraction systems, for example, Freitag et. al. [46] escape back to a more general class of tags rather than to a lower-order of model for the same tag.

There are several studies of the effectiveness of different smoothing strategies [34, 144], but there is no *a priori* reason why one should perform better than another in the absence of *a priori* knowledge about the symbol distribution within the model. An alternative approach to smoothing is to use Markov as a prescriptive model and reject outright any sequence containing a zero probability. This approach may be useful in closed systems or for carefully curated corpora, but is unlikely to result in robust systems in production environments.

Two aspects of Markov models can be trained: the topology (the number of states and transitions between them) and the weights of individual transitions. In theory the former aspect can be folded into the latter because: (a) a model with a transition of zero probability is indistinguishable from one lacking the transition and, (b) a model with a state which has only zero probability transitions to it is indistinguishable from one lacking that state. In real-world situations, with bounded training data, these are generally treated as separate problems. Model topology is commonly a fixed pattern, variable but selected or trained prior to training the transitions, or trained in parallel to training the transitions (as in DMC [151]). One fixed pattern of topology is used by PPM.

3.4 Prediction by Partial Matching

A Prediction by Partial Matching (PPM) model of order n examines the previous n characters to calculate a probability density function for the next character. To calculate the function, PPM keeps a record of sequences of n characters already seen and the character that followed them. If a sequence of n characters is seen that has not been seen before, then PPM 'escapes' back to sequences of $n-1$ characters. If a match is still not found, PPM escapes back to sequences of $n-2$, and so on, eventually escaping back to the order -1, in which all characters in the observable alphabet have the same probability.

The PPM model keeps the sequences of characters in a suffix tree, with each node labelled with the number of times the sequence has been seen [13]. This suffix tree can be converted to a single state Markov model of order $n+1$. The suffix tree is an efficient representation of a sparse model (one for which many of the possible states have not been observed) because unused branches are not expanded. The equivalent Markov model is an array in which all leaves are present, with those not seen during training appearing as small probabilities. In the current work, suffix trees are used for all processing.

The PPM model is deterministic [75] (or subsequential [104]) in that it always has one transition for each output symbol. In this regard it differs from the work of Lafferty and McCallum which has built non-deterministic HMMs for similar tasks to those seen in this thesis, using non-deterministic conditional random fields [75].

An additional benefit of the suffix-tree based Markov models over the traditional table models is that they greatly reduce the cost of introducing extra symbols. Increasing the character set size from 8 bit ASCII to 32 bit Unicode incurs a cost only for those characters are actually used in the training set or when the -1 model is escaped to.

PPM models may seem far removed from the way that humans deal with natural language text. However, as the following story reveals, it may be closer to the way that humans deal with natural language text when they have no linguistic information about it [30]:

> [A] typesetter working on a Greek text at the Oxford University Press announced he'd found a mistake in the text. As the typesetter couldn't read Greek, his colleagues and then his superiors dismissed his claim. But the man insisted. So finally an editor came down to the compositing room. At first, she, too, dismissed the idea, but checking more closely, she found there was an error. Asked how he knew, the typesetter said he had been hand-picking letters for Greek texts for most of his professional life and was sure that he'd never made the physical move to pick the two letters in that order before.

This implies that the typesetter had built an implicit model of which characters followed which other characters and had sufficient confidence in the model to question the text.

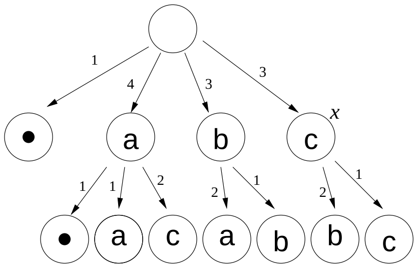
PPM is an incremental compression algorithm [151] with two widely-known variants, PPMC and PPMD [57]. PPMD is used in other text-augmentation work [133, 26]. PPMC and PPMD differ in the probabilities they put aside for unexpected events, seeing a character in a context in which they have not seen that character before. In a context in which C_t total characters and C_d distinct characters have been seen, PPMC sets aside $\frac{C_d}{C_t+C_d}$ and PPMD sets aside $\frac{C_d}{2C_t}$. Katz [67] takes a different approach and for an order n model uses $\frac{C_n}{N}$, where C_n is the count of the number of n grams that have been seen exactly once and N is the training text size.

PPMII is a PPM variant with special handling for the case in which only a singleton example of the current context has been seen during training. The occurrence of such contexts rises with the model order to 60–80% of all contexts. PPMII implementations typically also use adaptive models, and re-scale counts frequently to favour text seen recently over text seen at the start of training, to give good performance on compression corpora [127].

As implemented in this thesis, the PPM model does not store probabilities but rather counts of occurrences. These counts are converted into probabilities dynamically using an escape method which allocates the probability between seen and unseen symbols in the observable alphabet [152].

Figure 3.2 shows three representations of the adaptive order 1 PPMD model built from the string •*aabbccabca*.... The • represents the start of the string. Figure 3.2(a) is the suffix-tree representation. The tree is not complete, for example the *c*-labelled node marked x has no transition to an *a*-labelled node because the string •*aabbccabca*...contains no sub-string *ac*. Figure 3.2(b) shows the occurrence tables for order -1, order 0 and order 1, which correspond to the root node of the suffix tree, the first row of the suffix tree, and the leaves of the suffix tree respectively. Each non-zero entry in the order 1 table corresponds to a leaf in the tree above, while each zero entry thus corresponds to missing leaf.

Figure 3.2(c) shows the Markov models of order -1, order 0 and order 1. These have the same structure as the occurrence tables in Figure 3.2(b), but the occurrences have been converted to probabilities using escape method D. Each count in the



(a)

$k=-1$	
●	1
a	1
b	1
c	1

$k=0$	
●	1
a	4
b	3
c	3

$k=+1$	●	*a*	*b*	*c*
●	0	0	0	0
a	1	1	0	2
b	0	2	1	0
c	0	0	2	1

(b)

$k=-1$	
●	$\frac{1}{4}$
a	$\frac{1}{4}$
b	$\frac{1}{4}$
c	$\frac{1}{4}$

$k=0$	
●	$\frac{1}{11}$
a	$\frac{4}{11}$
b	$\frac{3}{11}$
c	$\frac{3}{11}$

$k=+1$	●	*a*	*b*	*c*
●	$\frac{1}{11}$	$\frac{4}{11}$	$\frac{3}{11}$	$\frac{3}{11}$
a	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{2}{5}$
b	$\frac{1}{16}$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{3}{16}$
c	$\frac{1}{20}$	$\frac{4}{20}$	$\frac{2}{4}$	$\frac{1}{4}$

(c)

Figure 3.2: Three representations of the PPMD model for ●*aabbccabca*. . . .

order -1 and 0 tables is divided by the total of counts in the table to obtain a probability. Each non-zero count in the order $+1$ table is divided by the total of counts in that row plus one. The probability corresponding to the extra (plus one) count is distributed among the zero counts.

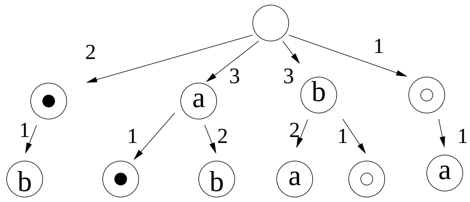
Each type of XML tag corresponds to a hidden state and has a separate model built for it. In the observable sequence the tags are mapped to single character symbols. Thus the string *aba<sometag>cbc</sometag>bab* is mapped to *aba*○*cbc*○*bab*, with a different symbol corresponding to each pair of tags, with the ●, seen earlier indicating the start of the string, being used for the entire string (what the XML standard refers to as the 'document element' [25]). Therefore if *aba*○*cbc*○*bab* is the entire string, it is represented as ●*aba*○*cbc*○*bab*●.

A distinct PPM model is built for each tag, in this case for ● and ○. The models for ● and ○ built from the string ●*aba*○*cbc*○*bab*● are shown in Figures 3.3 and 3.4, which have similar structures to Figure 3.2. The ● model is built from the strings ●, ●*a*, *ab*, *ba*, *a*○, ○*b*, *ba*, *ab* and *b*●. The ○ model is built from the sub-strings ○*c*, *cb*, *bc*'' and *c*○.

● occurs in the ○ model because it can be part of the alphabet in which the context. Even though it cannot be seen within the ○ model, it can appear in the context which is carried into the model, for example in the string ●○*c*○●.

When a ○ is seen in the ● model, a transition occurs from the ● model to the ○ model. When a ○ is seen in the ○ model, a transition occurs from the ○ model into the previous model, in this case the ● model.

Figures 3.3 and 3.4 show how we can use Viterbi search to find the most likely sequence of tags in the sequence ●*abbacbccbbab*..., the first step of which is shown in Figure 3.5, which has a lookahead of four. Between each two symbols in the observed sequence, the algorithm calculates the probability of there being a transition within the hidden state (the right branch from each node), and the probability of there being a transition to the other hidden state (the left branch from each node).



(a)

$k=-1$	
•	1
a	1
b	1
c	1
○	1

$k=0$	
•	2
a	3
b	3
c	0
○	1

$k=+1$	•	*a*	*b*	*c*	○
•	0	0	1	0	0
a	1	0	2	0	0
b	0	2	0	0	1
c	0	0	0	0	0
○	0	1	0	0	0

(b)

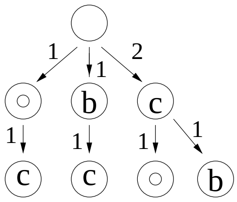
$k=-1$	
•	$\frac{1}{5}$
a	$\frac{1}{5}$
b	$\frac{1}{5}$
c	$\frac{1}{5}$
○	$\frac{1}{5}$

$k=0$	
•	$\frac{2}{10}$
a	$\frac{3}{10}$
b	$\frac{3}{10}$
c	$\frac{1}{10}$
○	$\frac{1}{10}$

$k=+1$	•	*a*	*b*	*c*	○
•	$\frac{2}{14}$	$\frac{3}{14}$	$\frac{1}{2}$	$\frac{1}{14}$	$\frac{1}{14}$
a	$\frac{1}{4}$	$\frac{3}{20}$	$\frac{1}{2}$	$\frac{1}{20}$	$\frac{1}{20}$
b	$\frac{2}{12}$	$\frac{1}{2}$	$\frac{2}{24}$	$\frac{1}{24}$	$\frac{1}{4}$
c	$\frac{2}{10}$	$\frac{3}{10}$	$\frac{3}{10}$	$\frac{1}{10}$	$\frac{1}{10}$
○	$\frac{2}{14}$	$\frac{1}{2}$	$\frac{3}{14}$	$\frac{1}{14}$	$\frac{1}{14}$

(c)

Figure 3.3: The • model built from •*aba*○*cbc*○*bab*•.



(a)

$k = -1$	
●	1
a	1
b	1
c	1
○	1

$k = 0$	
●	0
a	0
b	1
c	2
○	1

$k = +1$	●	a	b	c	○
●	0	0	0	0	0
a	0	0	0	0	0
b	0	0	0	1	0
c	0	0	1	0	1
○	0	0	0	1	0

(b)

$k = -1$	
●	$\frac{1}{5}$
a	$\frac{1}{5}$
b	$\frac{1}{5}$
c	$\frac{1}{5}$
○	$\frac{1}{5}$

$k = 0$	
●	$\frac{1}{10}$
a	$\frac{1}{10}$
b	$\frac{1}{5}$
c	$\frac{2}{5}$
○	$\frac{1}{5}$

$k = +1$	●	a	b	c	○
●	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{5}$	$\frac{2}{5}$	$\frac{1}{5}$
a	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{5}$	$\frac{2}{5}$	$\frac{1}{5}$
b	$\frac{1}{12}$	$\frac{1}{12}$	$\frac{2}{12}$	$\frac{1}{2}$	$\frac{2}{12}$
c	$\frac{1}{18}$	$\frac{1}{18}$	$\frac{1}{3}$	$\frac{2}{9}$	$\frac{1}{3}$
○	$\frac{1}{12}$	$\frac{1}{12}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$

(c)

Figure 3.4: The ○ model built from ●*aba*○*cbc*○*bab*●.

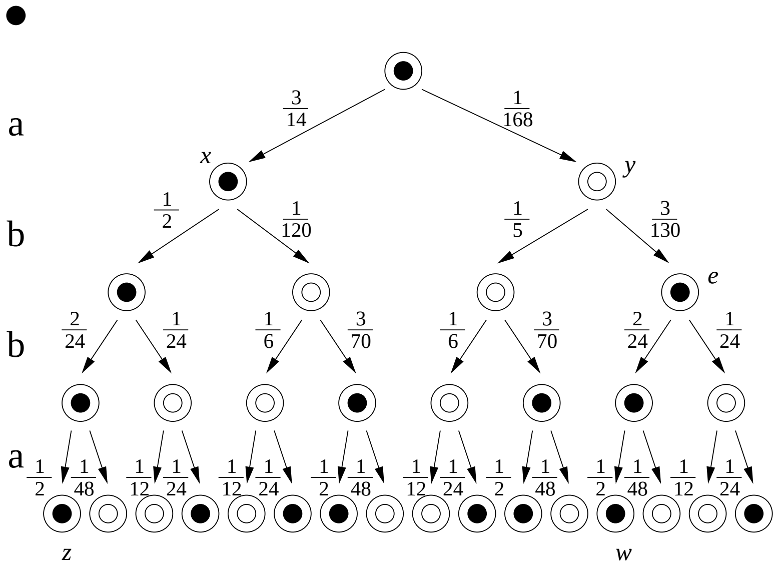


Figure 3.5: The expansion step in a Viterbi search of •*abbacbccbbab*. . . .

The probability for the left branch can be taken from the right hand tables in Figure 3.4(c) (for states in the ○ model) or Figure 3.3(c) (for states in the • model). The probability for a right branch is the product of two probabilities, that of the transition from one model into the other and of seeing the observed character.

Following the expansion step shown in Figure 3.5 is a pruning step. Either node *x* or node *y* must be pruned from the search tree, taking all descendants with it. Since node *z* is the leaf with the highest probability and a descendant of *x* rather than *y*, *y* must be pruned. Nodes *w* and *e* are discussed in Section 4.3.2.

Figure 3.6 shows the tree after pruning. Node *x* in Figure 3.5 has become x_{-1} and there are a new *x* and a new *y* based on the location of *z*, the lowest entropy leaf. Figure 3.7 shows the situation two steps later. For the first time the algorithm is about to prune the *x* branch rather than the *y* branch, and insert a ○ tag.

Viterbi search says that even for this demonstration example a lookahead of four is insufficient to guarantee an optimal tagging: the lookahead must be one more than the sum of the order of the model (1) and the longest tag length (3). Real examples typically have significantly longer tag lengths (see the samples in Appendix A) and

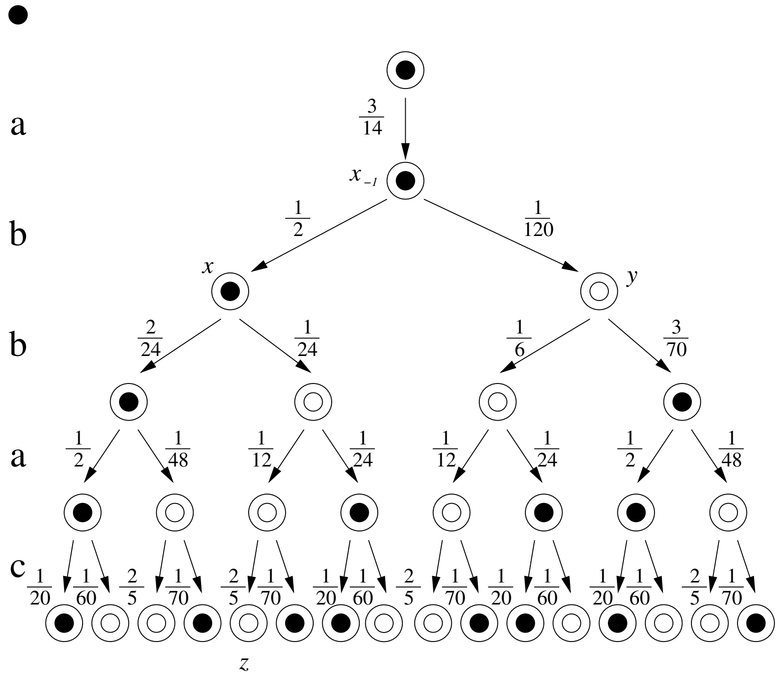


Figure 3.6: The next expansion step in a Viterbi search of •*abbacbccbbab*. . . .

often higher-order models, but for clarity a short lookahead has been used in this example.

3.5 Granularity of Models

Many published reports of text mining, information retrieval and other information systems model text as words [61]. This *a priori* assumption of segmentation into words leads to two separate problems:

1. In many contexts it is not clear what is and is not a word. In English two areas of ambiguity are contractions and abbreviations (for example ‘i.e.’ and ‘can’t’) and sometimes joined words (for example ‘real-time’ which is used variously as ‘realtime,’ ‘real time’ and ‘real-time’).

2. Words seen during testing (or practical application) that are not seen during training raise the ‘unknown-word problem’ [144]. This problem is a variant

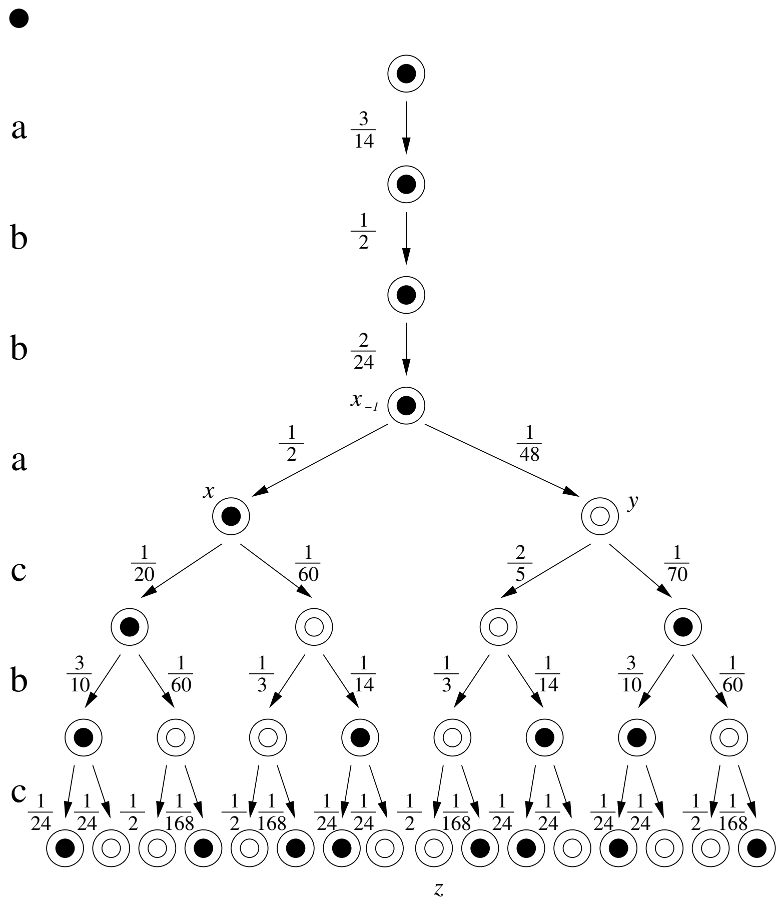


Figure 3.7: The fourth expansion step in a Viterbi search of •*abbacbccbbab*. . . .

of the zero-frequency problem (see Section 3.3). In many system-evaluation contexts, the problem is solved by leaking information from the testing set to the training set in the form of a ‘Perfect Lexicon’ containing every word in the system [17]. In production systems this approach is not possible because, unless a constraint is placed on the system vocabulary (so-called ‘controlled vocabularies’ [84, 105]), an unbounded number of words may be seen over the life of the system.

Approaches to solving the unknown-word problem include merging all unseen words into a single class and treating all unknown words the same, which works surprisingly well for news articles in which most unknown words are proper nouns, and escaping back to a character-level model, requiring two models, one at the word level and one at the character level.

An alternative to this is modelling text as a sequence of characters [133]. At first glance neither of the problems discussed above affects character-based models, but similar problems arise at a different level of granularity.

1. Unicode allows combining character sequences—characters built from a base character and combining characters, which add elements to it (i.e. accents or enclosing circles). All characters in most living natural languages (including English, Maori and Mandarin) are representable without combining characters, but should a system see them in input, handling them is problematic.

2. Though the Unicode character set is bounded, it is sufficiently large (many tens of thousands of characters) that if characters are hyper-geometrically distributed (as can be expected in natural languages [99, 169]), only rarely will a system see an instance of every character. Unicode is also expanding, with more characters being added; in theory a production system could see characters which were undefined when the system was built.

These character-level problems appear to be of a similar nature, if not a similar frequency, to the word-level problems. This suggests that the transition from word to character level has not actually solved the word level problems but rather transformed them to a lower level.

3.6 Searching in Models

Once built, the models can be used to find the most likely sequence of hidden states for a sequence of observed states. This is done using a search tree, in which each node is labelled with a state in the model. Each node is also labelled with the sum of all probabilities on the path between it and the root of the search tree. Entropy is inversely related to likelihood [126], and the most likely sequence corresponds to the leaf node with the lowest entropy.

```
$oldLeaves \leftarrow$ root;
while moreInputSymbols do
    $newLeaves \leftarrow \emptyset$;
    for leaf $\in$ oldLeaves do
        $newleaves \leftarrow$ ExpandLeaf(leaf) ;
        CalculateEntropy(newleaves);
    end
    $oldLeaves \leftarrow newLeaves$ ;
end
$result \leftarrow$ SelectLowestEntropyLeaf(oldLeaves);
```

Algorithm 1: The complete search algorithm.

An exhaustive, or complete, search for the most likely sequence involves a search space as deep as the sequence is long. This algorithm is shown in Algorithm 1. The function $ExpandLeaf$ takes a single leaf node in the search tree, examines the state in the model with which it is labelled and adds a new leaf to the search tree for each out-going transition from the state in the model. The function $CalculateEntropy$ calculates entropy of the each of these new leaves.

For many interesting sequences this search space is computationally infeasible, but the ‘Viterbi search’ [140] algorithm provides a computationally feasible searches in situations when only local information matters. The Viterbi proof [140] guarantees that Viterbi search will find the most likely sequence, provided the model determines the entropy for a node based on bounded local knowledge, rather than on global knowledge required by the exhaustive search. Fortunately Markov models, even high order Markov models, meet this criterion [90]. The length of the sequence that must be modelled for this local knowledge is called the ‘lookahead’.

```
$oldLeaves \leftarrow$ root;
while moreInputSymbols do
    $newLeaves \leftarrow \emptyset$;
    for leaf $\in$ oldLeaves do
        $leaves \leftarrow$ ExpandLeaf(leaf) ;
        CalculateEntropy(leaves);
        AddLeavesToSet(newLeaves,leaves);
    end
    $bestLeaf \leftarrow$ SelectLowestEntropyLeaf(oldLeaves);
    $oldLeaves \leftarrow$ PruneBranch(bestLeaf, newLeaves);
end
$result \leftarrow$ SelectLowestEntropyLeaf(oldLeaves);
```

Algorithm 2: The Viterbi search algorithm.

Viterbi is a beam search, as shown in Algorithm 2. This is expressed as a search tree which is built independently of the Markov model in use, but with pointers in every node to a state in the model. The operation $PruneBranch$ takes a $bestLeaf$ from a selection of $newLeaves$, traces parents of $bestLeaf$ up until it finds a node which is the parent of every leaf in $newLeaves$ and prunes all daughters from that node except the one which leads to $bestLeaf$.

There is an alternative representation, that of a search lattice, in which nodes from the search tree are not pruned but ‘merged’ with other nodes with identical state in the underlying models. Merged nodes have the lowest entropy of any of the nodes from which they were merged, this representing the minimum entropy path through the search tree (now a search lattice) to the node. The search lattice is either unified with the Markov model or has a similar structure. This representation is widely used in signal processing and reflects common low-level and hardware implementations in that field [63].

The stack algorithm, a variant of Viterbi search, uses a sorted list rather than an explicit search tree. The list is sorted by the entropy of the node and initially populated with the first symbol. The lowest entropy node is removed from the list and its children calculated and added to the list. The search ends when a leaf node is found.

The Fano algorithm, related to the stack algorithm, does not use a stack but moves incrementally though the search tree guided by entropy-based thresholds, revisiting many nodes, but using only tightly-bounded memory, thus making it suitable for implementation in hardware. The creeper algorithm is a hybrid of the stack and Fano algorithms, using complex tables. All three of these algorithms are described in detail in Johannesson and Zigangirov, Chapter 6 [63].

Viterbi search implemented as a lattice or tree, the stack algorithm, the Fano algorithm, and the creeper algorithm all represent different trade-offs between time and space, and between simple and complex algorithms. The search-tree representation is traditional in computer science, because it allows a more direct comparison with other forms of searching; it is used in this thesis for a more natural representation of the pruning explored in Section 4.3.

```
$oldLeaves \leftarrow$ root;
while moreInputSymbols do
    $newLeaves \leftarrow \emptyset$;
    for leaf $\in$ oldLeaves do
        $leaves \leftarrow$ ExpandLeaf(leaf) ;
        CalculateEntropy(leaves);
        AddLeavesToSet(newLeaves,leaves);
    end
    $oldLeaves \leftarrow$ SelectNLowestEntropyLeaves(newLeaves,N);
end
$result \leftarrow$ SelectLowestEntropyLeaf(oldLeaves);
```

Algorithm 3: The Teahan search algorithm.

Algorithm 3 shows the Teahan search, a ‘Viterbi-inspired’ [136] algorithm which has been found effective [133]. Rather than search a fixed distance ahead into the search space on each increment, it only expands the N lowest entropy nodes at each level in the tree (line 4).

The Teahan search algorithm is a heuristic: it is not guaranteed to find the lowest entropy tagging. The Viterbi proof cannot usefully be applied to Teahan search. This is because the only point at which Teahan search is guaranteed to search the local search space at every step in the search is when N is the number of leaves in the exhaustive search. At this point the Teahan search and exhaustive search become identical.

For many interesting problems, limited amounts of data with correlated hidden and observable sequences are available for training, but data with only observable sequences abound. An algorithm to utilise these un-correlated observable sequences was developed by Baum and is known as the Baum–Welch algorithm [10, 11, 118]. This (Algorithm 4) is similar to Viterbi search with the addition of a step (line 5) that updates the model after the most likely branch has been found [118, 90]. The *UpdateModel* function updates the hidden Markov model to include seeing *bestLeaf*.

```
  $oldLeaves \leftarrow$ root;
  while moreInputSymbols do
      $newLeaves \leftarrow \emptyset$;
      for leaf $\in$ oldLeaves do
          $leaves \leftarrow$ ExpandLeaf(leaf) ;
          CalculateEntropy(leaves);
          AddLeavesToSet(newLeaves,leaves);
      end
      $bestLeaf \leftarrow$ SelectLowestEntropyLeaf(oldLeaves);
5     UpdateModel(bestLeaf);
      $oldLeaves \leftarrow$ PruneBranch(bestLeaf, newLeaves);
  end
  $result \leftarrow$ SelectLowestEntropyLeaf(oldLeaves);
```

Algorithm 4: The Baum–Welch algorithm.

The Baum–Welch algorithm is a specialisation of Expectation Maximisation (EM) which is widely used in machine learning [149] and statistics [60]. McLachlan and Krishnan [98] describe EM and the relationship between it and the Baum–Welch algorithm in detail and [18] discusses this relationship mathematically.

The Baum–Welch algorithm is the primary training mechanism for several information-extraction systems, for learning either the transition probabilities [82, 125, 17] or the model structure [125], or both. In this thesis, the Baum–Welch algorithm is used only for learning the transition probabilities, the Markov model structure is imposed by the PPM algorithm and the hidden Markov model structure reflects the schema of the documents seen during training.

This thesis uses a variant of the Baum–Welch algorithm, in which an entire document, or group of documents, has tags inserted which are then used to update the model, rather than to perform tag insertion and model re-estimation in such a closely-linked manner. This approach precludes the possibility of intra-document learning (lowering the entropy of a sequence of symbols in a tag because they have already been seen) but allows the efficient use of non-adaptive models, and avoids the cost of 'unlearning' during searching. The effect of this is likely to be most significant for long, single-subject, documents which contain frequent occurrences of proper nouns and other features which are rare within, or absent from, the training corpus. Proper and rare nouns are typically introduced in stylised forms [160] which can then used to update the model for their less stylised subsequent use. Without the ability to update the model, subsequent uses of the features are likely to be ambiguous.

Much research on the Baum–Welch algorithm is performed in the context of voice recognition [11, 118], where is it used at the phone level for adapting a model to an individual's accent. In voice recognition, the observable sequence is a discretised representation of a continuous signal. The symbols in the discretised representation can be ordered, for example, it is possible to say that 50 dB $<$ 51 dB $<<$

1000 dB. Much of this research cannot be applied to text because the observable character set in text modelling (characters) has no useful implicit ordering.

The Baum–Welch algorithm is normally used during training. However, if the sequence being modelled is changing slowly over time, or if there is insufficient training data to characterise the sequence sufficiently it can be used during testing. Unfortunately, if a feature is mis-modelled when first seen, the reinforcement of the Baum–Welch algorithm makes it much more likely that it will be mis-modelled when seen subsequently, even in contexts which could have been clear if seen by a model without re-estimation.

3.7 XML and Unicode

This section examines some issues with Unicode and XML and their impact on the corpora and algorithms used in this thesis. These issues include the assumptions Unicode makes about text, the semantics of nested XML tags, and the order of XML attributes. These issues are important because they underpin much later work in this thesis.

XML is a standard [25] for encoding data and has emerged as the leading standard for encoding textual documents for archiving, academic study, interchange and corpus building. XML uses Unicode [138] by default, allowing a large number of languages and writing systems to be represented. Unicode makes various assumptions which make it significantly easier to reason about text, including:

- That characters are unique entities from a finite set.
- That each character falls into exactly one character class.
- That the character class of each character is known.

These assumptions do not hold universally, not even for all documents held in modern information systems. Handwritten texts or texts printed prior to the standardisation of book printing are particularly problematic because their digitisation commonly involves more semantic interpretation than the digitisation of later printed works with known conventions. The Early English Books Online project,[1] is an example of a real-world undertaking impacted by these issues. Unicode character classes are discussed in Section 4.3.3.

[1] http://www.lib.umich.edu/tcp/eebo

Chapter 4

The System

This chapter introduces the bulk of the new content in the thesis, starting with a new taxonomy for metadata markup problems. The architecture of the implementation is introduced, followed by a number of optimisations and heuristics implemented within it. The search space of these optimisations and heuristics for various metadata markup problems is then examined together with the impact of metadata problems on assessing experimental correctness.

4.1 Metadata

This thesis introduces a new taxonomy for fine granularities of metadata problems: in segmentation metadata, classification metadata, and entity metadata. The remainder of this section describes the taxa.

Metadata comprises encoded tags, in ranges of adjacent characters which share some property, and externalised as XML [25]. XML is a widely-used metadata format [156, 123, 147].

4.1.1 Segmentation

Segmentation problems involve finding the internal boundaries within text. The boundaries can be linguistic (e.g. in word or sentence boundaries), semantic (e.g. between topics) or both (e.g. between index or bibliography entries). Finding word boundaries in Chinese, Japanese or Thai text and finding suitable places to seg-

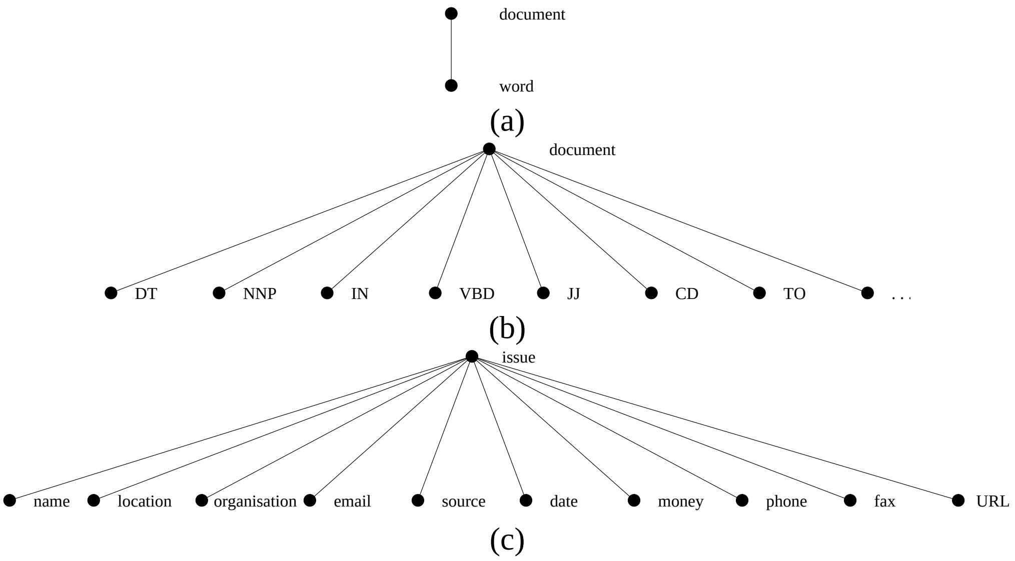


Figure 4.1: Schema structures for segmentation and classification problems. (a) The Chinese text segmentation problem. (b) The part of speech tagging classification problem. (c) The Computists' Communique classification problem. Details of these problems and corpora in which they are studied are given in Chapter 5.

ment English, German and French words for line-end hyphenation [77] and all well-known examples of segmentation problems.

As encoded in this thesis, all segmentation information is destroyed by tag merging. If adjacent tags are merged, all segmentation information is lost because information lies solely in where the tags start and end, rather than in which type of tag a piece of text falls.

Figure 4.1(a) is the schema for the Chinese text-segmentation problem. It has a single root-node and a single type of child-tag below it. There is an instance of the child-tag around each word. The schema for every segmentation problem has this shape, with a single type of child tag and all characters within instances of that tag type.

Various approaches have been used to segment text. Many early systems used simple lookup tables [157], which work surprisingly well on most text, except novel characters not seen in training. Most text segmentation systems use n-gram models or equivalent Markov models [137, 50, 117].

Recent segmentation research directions include conditional random fields [115], and using integrating segmentation with functionality such as part-of-speech tagging [58] and proper noun extraction [168]. Combining segmentation with higher-level processing allows leveraging segmentation to help solve other natural-language processing problems and the results of the higher-level processing to fine-tune the text segmentation.

4.1.2 Classification

Classification problems involve classifying textual elements (typically words or characters) into one of several classes. Many classification problems are referred to as tagging in the information extraction and document understanding communities, but this name has been avoided, because all of the problems discussed here involve inserting tags—literally 'tagging'. The term classification is used in machine learning to refer to problems which involve placing an instance into one of a set of classes, and it is used here in the same manner.

Classification metadata is immune to tag merging. If two adjacent tags of the same class are merged, no knowledge is lost, because the extracted information lies solely in which type of tag text falls. Similarly if a tag is split in two, no information is lost, provided the two new tags cover the same characters as the previous single tag.

Figures 4.1(b) and (c) show the schema structures of classification problems. The schemas have a single root node (representing the document), and each of the classes has a node directly connected to this root node. The schema for every classification problem has this shape, with a number of types of child tags and all characters within instances of these child tag types.

Much early work on classification problems was performed on part-of-speech taggers, drawing on traditional debates on the role of grammar in language. Several early systems were grounded in distinct schools of linguistic theory, but performed relatively poorly. Later approaches have used more generic statistical modelling techniques to better success.

The Brill tagger [28, 29] first trains a rule-based tagger and then learns transformation rules based on the errors of the rule-based tagger. The transformation rules allow for super-adjacency and higher-level reasoning, neither available to conventional Markov models. Super-adjacency, looking not at immediately adjacent words but at those several words away, allows wildcard-like effects. Applying rules is fast, so the whole system runs quickly, and it is widely used and well respected.

The MUC problems can be considered classification problems, but the focus is on information extraction: the inferred information is not embedded in the document text, but either included in the document header or completely separated from the document. Many problems contain higher-order reasoning outside the scope of text augmentation considered in this thesis. For example, the title *President* and the name *Bill Clinton* can be inferred to refer to the same individual combined as *President Bill Clinton*. Classification can identify title and name, both together and separately, but not perform the higher-order reasoning to link the instances or to present the separate components combined into a single sequence.

4.1.3 Entity Extraction

A superset of segmentation and classification, entity extraction, finds bounded sections of text that belong to a particular class. If adjacent tags are merged, some information may be lost, since information lies both in which symbols are in which class of tag and in where the individual tags start and finish.

Because entities have both a range and a depth, it is possible for entities to be nested, introducing extra complexity. Nesting of a tag within another of the same type is a technique used relatively widely in grammar-based linguistics. It is not inherently more complex than nesting a tag within a different type of tag.[1]

[1] However, the current work does not handle such cases gracefully, as explained in section 7.4

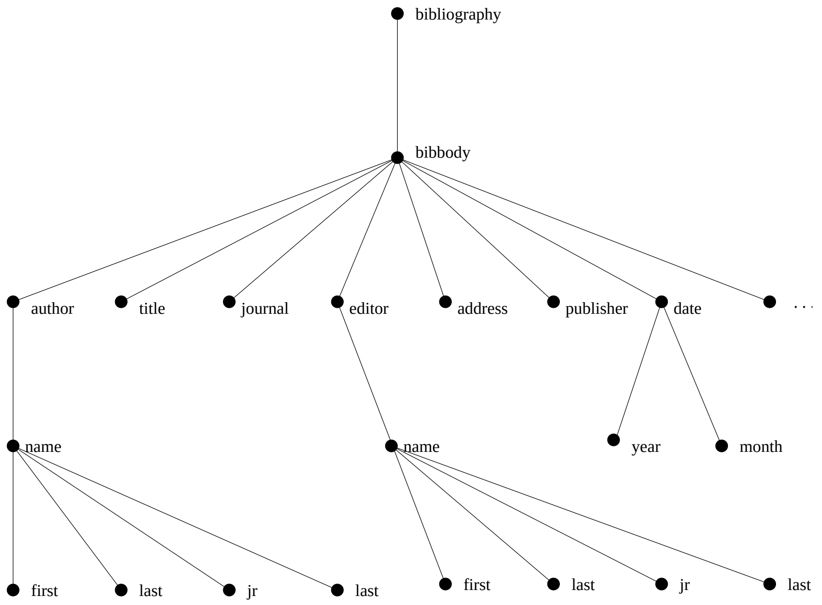


Figure 4.2: Schema structure for the bibliography entity extraction problem. Details of this problem and corpus in which it is studied are given in Chapter 5.

Figure 4.2 shows the schema structure for the bibliography corpus, an example of entity extraction in which the entities such as author names, article, titles and conference names are marked up. The schema for entity extraction problems allows arbitrary nesting of tags.

Bray [26] showed that, on a small sample, hierarchical tagging of personal names into first and last parts hindered the overall identification of names, but the hierarchical tagging of email addresses into username and host parts aided the identification of email addresses. The failure of hierarchical tagging of names in this case appears to be at least in part caused by the small number of names used. Wen [144] used eight tags from an early version of the bibliography corpus (see Section 5.2) and achieved an F-measure of 76%.

4.1.4 Limitations and Constraints

Text augmentation is not a universal method of inferring metadata. There is a range of text-augmentation problems that fall outside this taxonomy, including those with overlapping structures, those with attributes that are continuous numeric values, and those with escapes to the XML Schema ANY tag. The taxonomy is unsuitable for the coarser-grained metadata, such as document level or collection level information.

There are certain constraints derived from the XML tagging used (see Section 2.5):

1. Half the tags are opening tags $t_{tagname}$ and half are closing tags $t_{/tagname}$.
2. Only the most recently opened unclosed tag may be closed next.
3. Each opening tag must be separated from the corresponding closing tag by at least one data point from the underlying sequence.
4. No two tags of the same type are opened between any two characters.
5. Tags do not have attributes.

Constraints 1 and 2 are a restatement of the well-balancedness constraint of XML. Constraint 3 is not present in XML, but is present in the current representation to rule out the proliferation of arbitrary numbers of empty tags.

Constraint 4 is also not present in XML but is introduced here in order to make the sets of tags enumerable, both a consequence of implementation choices and a prerequisite for calculating the size of search spaces. The lack of attributes has been discussed in Section 2.5.2.

4.2 Architecture

The implementation used in this thesis is called ‘Colloquial Entropy Markup’ or CEM. CEM is built in pure Java [52], no platform-dependent library bring used. All input and output of data is performed using the Apache / Xerces implementation of the standard Java XML Document Object Model (DOM) [154]. In this thesis a

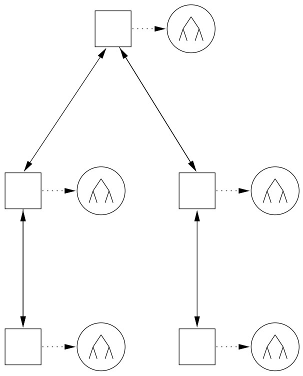


Figure 4.3: The structure of a CEM model, hidden states (square boxes) with associated PPM models (circles).

deliberately standards-based approach was taken largely in response to difficulties encountered Teahan's [133] implementation.

CEM uses Unicode throughout and recursive modelling of tags, the latter enabling it to tackle the more challenging entity-extraction tasks, as well as those of segmentation and classification. There are two main internal data-structures, the model and the search tree. DOM is not used in the internal data-structures, because when the software was first designed, the DOM was immature and it was not clear that it would prove as stable and effective as it has done.

4.2.1 The Model

The structure of the hidden Markov models implemented in CEM is shown in Figure 4.3. Each of the circles is a PPM model in the form of a suffix tree, as shown in Figure 3.2. Each of the squares is a hidden state in the hidden Markov model; the associated PPM model is the Markov model for that hidden state.

The presence of two characters without a tag between them is represented as a transition between two states within the same PPM model. The presence of two characters with one or more tags between them is represented as a series of one or more transitions between states in different PPM models (or between states in the same PPM model in the case of closing tags immediately followed by opening tags). Closing tags indicate transitions up, towards the root of the hidden Markov model and opening tags indicate transitions down, towards the leaves of the model. XML well-formedness is enforced by starting in the root of the hidden model at the start of the sequence and by forcing a return to the root by inserting close tags at the end of the sequence.

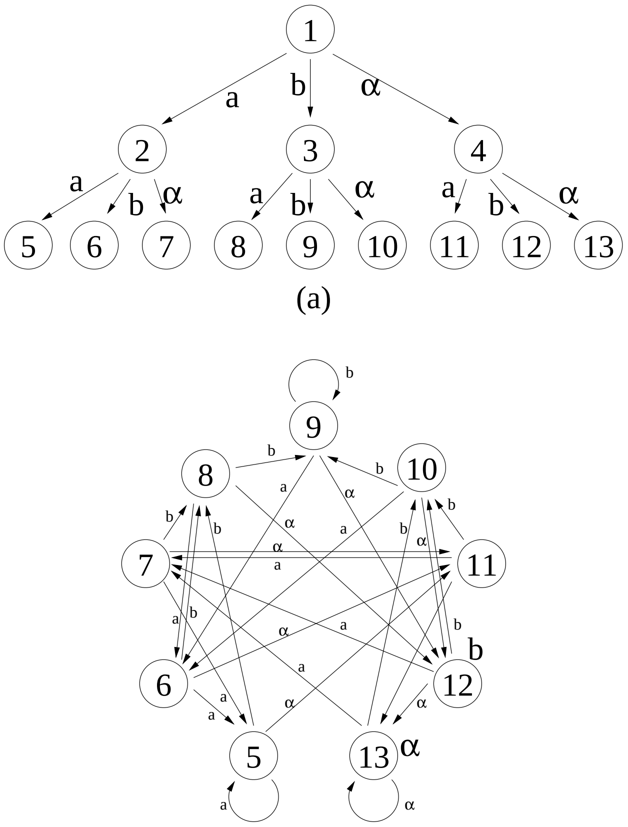
Figure 4.4 shows the relationship between the suffix tree representation of Markov models used in CEM and a more traditional representation. Nodes are numbered for identification. The implementation uses only the suffix tree during training and testing, although it can output low-order Markov models for manual validation. Figure 4.4(b) is directly convertible to a tabular format.

Each state is adjacent to an end state, because each state has an α transition from it. When building PPM models, α is treated as just another letter in the alphabet: α represents one third of the alphabet in Figure 4.4(b). Having multiple start and end states is unusual for a Markov model used in an HMM, but is natural and efficient to implement when suffix trees are used, because the suffixes can have the extra character added for hidden state transition prepended (α in this case), and be carried from one hidden state to the next.

The CEM model is implemented as shown in Figure 4.4(a): a simple tree, with each node labelled with a character and a number. The tree representation allows branches to be expanded as and when they are first seen during training, saving memory on unseen branches.

Transition probabilities are computed dynamically from counts, using escape methods, in the manner of adaptive text compressors [146]. Counts rather than probabilities are stored, so the escape method can be changed after training. This feature is desirable during experimentation, but unlikely to be important in production environments.

CEM models are serialisable: they can be streamed to a file using standard Java serialisation and later streamed back into memory intact. Models are streamed



(a)

(b)

Figure 4.4: The structure of a PPM model, (a) as a suffix tree, in which leaf nodes (5–13) are reached by navigating from the top of the tree each time an entropy is calculated, using the suffix of recently seen characters, and (b) as a finite state machine using traditional Markov model notation, in which a pointer to a node is used for state rather than a suffix and the next node is found by traversing the transition labelled with the current character.

through a gzip [88] stream reducing their size by approximately 90%, primarily because Java serialisation focuses on issues such as portability and flexibility rather than output size. No experiment was undertaken relating the size of training texts to the size of streamed or in-memory models. Streaming models to and from disk allows the reuse of models across testing sessions.

4.2.2 Differences between CEM and other systems

There are two key architectural differences between CEM Markov models and comparable systems: the handling of context between models and the symmetric, recursive structure of the hidden states. This section examines these differences in more detail.

Systems such as HTK and SMI have Markov models with a single start state, so that no matter how much context is taken into account within the models, each transition between hidden states results in a complete loss of context. HTK partly overcomes this by having a large number of hidden states in a complex structure. When moving between hidden states, CEM prepends a single character to the context for each transition (and thus each tag that is opened or closed). This is seen, for example, in the α symbol in Figure 4.4. For tagging problems with many fine-grained, deeply-nested tags this can represent a considerable loss of context, but for lightly-tagged text with a PPM model of non-trivial order the loss is less significant.

This retention of context allows for the efficient modelling of the situation in which tags are marked by a distinctive characters. For example, consider the fragments:

. . . <x> [a] </x> b

and

. . . [<x> a </x>] b

When CEM calculates the entropy for *b* with an order 3 model, in each case it has a full context to use for the calculation, and avoids the need to escape to a lower-order model. This is not true for most other Markov model implementations.

CEM hidden models have a symmetric, recursive structure, reflecting the well-formedness requirement of the XML from which it is automatically generated. This differs from the flat (non recursive) model of SMI and generic finite-state machine model of HTK and other voice-recognition systems. The flat model is sufficient for segmentation and classification problems, but not for entity extraction problems. The added complexity of a generic finite-state machine model is used in voice recognition to represent models of sentence-level structure, based on separate analysis and testing. While there are certainly areas of text augmentation which might benefit from such generic models, it is hard to imagine how they would be readily incorporated into CEM's low human-input approach.

4.2.3 The Search Tree

The search tree is the second of the two main data structures in CEM. Each node in the search tree is labelled with:

- the current character from the input stream;
- any XML tags inserted immediately before the current character;
- the current states in the hidden Markov and PPM models; and
- the cumulative entropy of traversing from the root of the search tree to this node.

There are two types of search tree implemented in CEM: Teahan search (see Algorithm 3 on page 46) and maximum lookahead search. When the maximum lookahead is used with a sufficiently long lookahead, it is a true Viterbi search. Except where explicitly stated, the maximum lookahead search (see Algorithm 2 on page 45) is used.

4.2.4 Full Exclusion

The PPM escape methods, as implemented in this thesis, differ from the standard escape methods because they do not use full exclusion. That is, when an order n model is escaped from back to an $n-1$ model, the $n-1$ model is not modified by removing characters which appear in the order n model. Removal of these characters from the $n-1$ model is safe because they have already been considered in the n model. This variant has been dubbed PPM-SY after the initials of the author, to differentiate it from other forms of PPM.

The effect of not using full exclusion is to modify slightly the action of the escape methods used. As noted on page 32, there is no *a priori* reason either to think that one escape method should model a sequence better than another, or when using PPM for text augmentation to suggest that PPMD should give better results than PPM-SY.

When using PPM to drive an arithmetic encoder, using PPM-SY would squander a small amount of probability whenever a model is escaped from, resulting in a longer coded text, and would thus be undesirable. In text-augmentation applications, the absolute entropy values are not important, only the relative values: the coded text is never used or produced so the length is irrelevant.

The choice not to use full exclusion was made for reasons of efficiency: performing set operations on large character sets in the inner loop of a computation is understandably expensive. It is expected that the cost of full exclusion will be substantially higher for larger character sets than for small ones. A version of PPM with full exclusion is tested in Section 6.1.

The implementation of full exclusion calculates the exclusion dynamically as it occurs. An alternative implementation was considered in which exclusions were calculated the first time they were used, and then cached for reuse thereafter. This would have consumed considerable extra memory, particularly for the large character-set segmentation corpus (see Section 5.3), for which the size of the model was an issue.

4.3 Optimisations and Heuristics

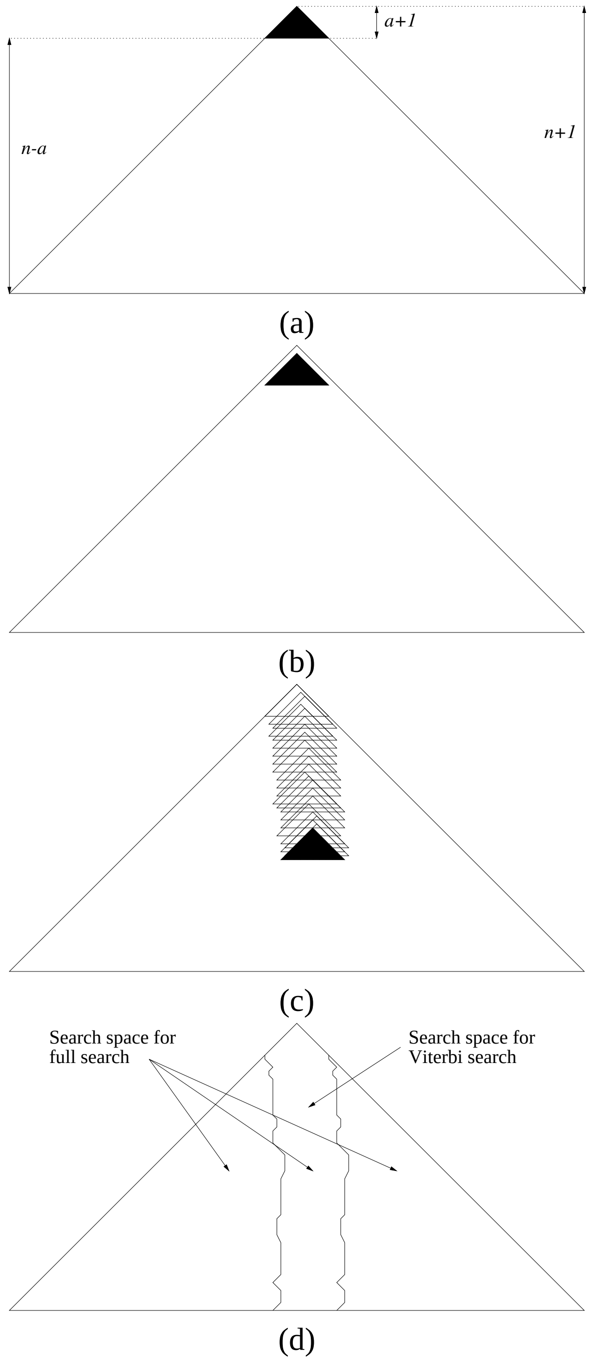
The pruning of search trees using optimisations and heuristics to enable them to be searched as efficiently as possible has a long history in computer science [71]. This section applies this tradition to the search space of text augmentation. Optimisations are techniques that improve the efficiency of problem solving without altering correctness. Heuristics are techniques that improve the efficiency of problem solving but may potentially reduce correctness. This section looks first at techniques and then at how some of them affect the search spaces in three different classes of text augmentation.

4.3.1 Viterbi Optimisation

Viterbi search [140, 141] (Algorithm 2, page 45) is an optimisation of complete search (Algorithm 1, page 44), which Viterbi proved [140] has no impact on correctness provided the lookahead a is large enough and the encoding scheme has the right properties. For text-augmentation problems 'large enough' is the maximum possible length of a tag, plus the order of the PPM model in use, plus one.

Relating search-space size to the maximum length of the tags being inserted means that some tags require smaller search spaces than others. Inserting short tags, such as personal names or parts of speech, gains more advantage from the Viterbi search than do large tags such as the <*html*> or <*body*> tags in XHTML [114] which contain an entire document.

Figure 4.5 shows an example of Viterbi search space, with each small black triangle being the search space for the current increment, page 45) and the large triangle being the full search space (respectively the for and the *while* loops in Algorithm 2, page 45). Figure 4.5(a) shows the initial search space of depth $a + 1$,



(a)

(b)

(c)

(d)

Figure 4.5: Viterbi search of a large search space.

before the first pruning of the search space, and the full search space of depth $n+1$. Figure 4.5(b) shows the second search space of depth a after the first pruning. Figure 4.5(c) shows the search half-way though, and Figure 4.5(d) shows the completed search.

aaa({)(}{)···(}{)···(}{)(}{)(}{x

(a)

aaaa({)(}{)(}{)(}{)(}{)}aa

(b)

○	a	{	}	(	)	○	⋆	◇
a	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{y}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$
{	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{6}{7}$	$\frac{1}{49}$
}	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$
(	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{6}{7}$
)	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$
○	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$
⋆	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{6}{7}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$
◇	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{x}{49}$	$\frac{1}{49}$	$\frac{6}{7}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$

(c)

⋆	a	{	}	(	)	○	⋆	◇
a	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	0	$\frac{1}{6}$	0
{	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	0	$\frac{1}{6}$	0
}	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	0	$\frac{1}{6}$	0
(	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	0	$\frac{1}{6}$	0
)	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	0	$\frac{1}{6}$	0
○	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	0	$\frac{1}{6}$	0
⋆	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	0	$\frac{1}{6}$	0
◇	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	0	$\frac{1}{6}$	0

(d)

◇	a	{	}	(	)	○	⋆	◇
a	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	0	0	$\frac{1}{6}$
{	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	0	0	$\frac{1}{6}$
}	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	0	0	$\frac{1}{6}$
(	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	0	0	$\frac{1}{6}$
)	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	0	0	$\frac{1}{6}$
○	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	0	0	$\frac{1}{6}$
⋆	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	0	0	$\frac{1}{6}$
◇	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	0	0	$\frac{1}{6}$

(e)

Figure 4.6: A set of models and sequences for which the Viterbi assumption does not hold and Viterbi search fails. (a) a class of difficult sequences (b) a single sequence (c) top-level Markov model (d) model for $\star$ (e) model for $\diamond$. x=1 and y=1.

As the following contrived example illustrates, it is not obvious that the Markov assumption, and with it Viterbi proof, in any form holds for natural language text. Figure 4.6 shows a Markov model with three hidden states and an alphabet of eight symbols. Hidden model $\star$ models the contents of matched { } braces (d). Hidden model $\diamond$ models the contents of matched *( )* parentheses (d). The columns of zeros in the $\star$ and $\diamond$ models indicate that no direct transitions between them are possible, and that transitions must be via the top-level Markov model for the $\circ$ hidden state.

Figure 4.6(a) shows a class of sequences which is problematic with respect to this model: parentheses and brackets used in ways that do not match. Furthermore, a repeating chain of parentheses and brackets can extend the ambiguity indefinitely until some other symbols, such as an *x* are seen. Figure 4.6(b) shows a string for which Viterbi search will yield two equally likely hidden sequences. The model may be changed to prefer one over the other by changing x in $\circ$ (and adjusting the other probabilities so that the sum is 1). However, such a solution still requires that the search sees the end of the chain before pruning the search tree at the start of the chain.

Fortunately such situations are rare, none of the datasets presented in this thesis appears to contain such sequences, and none has been reported in the literature. Experience [136, 144, 145, 135, 26, 163] has shown that in practice Viterbi search does work on natural language text.

Figure 4.6 shows a situation in which Teahan Search (Algorithm 3 on page 46) performs admirably. Teahan Search expands a fixed number of nodes at each level in the search tree so it is capable of exploring equal entropy branches of the search tree to an arbitrary depth, providing at each level one node from each branch is expanded. However, if a branch has higher entropy (for example, y in Figure 4.6(c) is raised), then it will probably get pruned, even if the lowest global entropy lies down that branch of the search tree.

4.3.2 Best First Optimisation

The best first optimisation is based on the observation that once a candidate augmentation has been found and the entropy calculated, all nodes within the search space with higher entropy can be pruned immediately. If a likely candidate augmentation can be found computationally cheaply, and the probability distribution function is steep (i.e. the model has high discrimination), the search space can be reduced considerably. In Figure 3.5, e has an log probability of $\frac{3}{168\times130} = \frac{3}{21840}$, and node z has an log probability of $\frac{6}{14\times2\times24\times2} = \frac{6}{1344} = \frac{1}{224}$: neither w nor any other child of e can have a lower log probability (and thus entropy) than that of node z, so node e need not be expanded.

The savings made from best first are difficult to calculate, because they depend on the probability distribution function for each state in the model and the exact sequence of symbols seen. In general, however, the savings are larger for probability density functions that are highly discriminative. Discrimination generally increases as models are better trained.

The CEM implementation finds a best first candidate by calculating the entropy of the left most leaf (the only leaf reachable without inserting any tags). This is the computationally cheapest leaf to find and in many situations it is a low-entropy leaf, if not the lowest. Hardware and Field Programmable Gate Array (FPGA) implementations of Viterbi search may avoid the need for the best first optimisation by performing this part of the algorithm in parallel [140, 141, 121]. Such treatment is not possible with text augmentation because of the significantly larger lookahead.

4.3.3 Automatic Tokenisation Heuristic

The automatic tokenisation heuristic is based on the observation that in many problems there are classes of characters between which no tag ever occurs. For example, in the Computists' and bibliography corpora, no tag ever occurs between a pair of lower-case letters or between a pair of whitespace characters. If no tag is ever seen in a situation during training, and a sufficient amount of training data has been seen, it is reasonable not to consider inserting tags in such positions during testing. This assumption may prove false, which is why automatic tokenisation is a heuristic not an optimisation.

The saving in search space depends on the structure of the text. However, if text were uniform words of four letters starting with a capital letter and separated from the next by a space ($\ldots␣ Abcd␣ Efgh␣ \ldots$) and automatic tokenisation meant the search did not have to consider inserting tags between pairs of lower-case letters, two of every five nodes in the search space would not need to be expanded. This $\frac{2}{5}$ approximation is assumed throughout this chapter.

Some types of contraction and abbreviation have a direct impact on automatic tokenisation. For example, the string *John Anthony Smith* may have the same search space as *J. A. Smith*, even though they differ markedly in length.

The CEM implementation keeps an occurrence table of possible pairs of Unicode character classes [138], and counts how many tags are seen between each pair. During augmentation, each node in the search tree is checked to see whether more than a threshold number of tags has been seen between the current pair of character classes, before considering whether to expand the search tree. Common threshold values include -1, 0, 1 and the default 5.

Unicode characters are divided into a set of 28 classes. The most common classes seen in the corpora used in this thesis are lowercase letter, uppercase letter, other letter (common in the segmentation corpus), space separator, line separator, decimal digit number, and various classes of punctuation. The classes are particularly convenient in Java, which uses Unicode throughout [52]. The ANSI C [59] functions *isspace(), isupper(), isdigit()*, etc. have a long history in parsing applications [2] and would almost certainly have performed well in this role for the English language corpora. There are been proposals [5] for much more sophisticated character-level metadata systems in Unicode, but these are not considered here.

One Unicode character class, the private use class, is reserved for ‘use by software developers and end users who need a special set of characters for their applications. [These characters] are reserved for private use and do not have defined, interpretable semantics except by private agreement’ [138]. CEM uses these to represent tags in character-level models, assigning a character to each tag to enable it to be modelled as just another character within the PPM models: the α in Figure 4.4 and the $\diamond$, $\star$ and $\circ$ in Figure 4.6. These characters are used by CEM only internally, and always mapped to or from full XML representations of the tags when externalised.

4.3.4 Alphabet Reduction

Alphabet reduction is a heuristic based on the same character classes as automatic tokenisation. In the bibliography corpus, repeating patterns of punctuation and capitalisation involving names in bibliographies were noticed. Names, which are commonly unique strings, remain a problem for the PPM model which sees limited context.

Alphabet reduction merges a class of characters into a single character in the model. For example, merging all upper case letters to *A* and all lower case letters to *a* means that *John A. Smith and, Jill K. Jones and* and *Yong X. Xiong and* all merge to *Aaaa A. Aaaaa aaa*. Throwing away this information homogenises these names. Considerably less memory and training data are needed to produce high-order models because alphabet reduction reduces the size of the alphabet so drastically. Empirically, alphabet reduction has raised the maximum order of the model to between 15 and 25. The performance of alphabet reduction in practice is examined in detail in Section 6.4.3.

This method is related to methods used elsewhere for finding acronyms [32, 160] using capitalisation patterns for generating candidate acronyms, which are then winnowed using other techniques. The benefits of alphabet reduction are hard to model, as they depend on the gains from modelling at a higher order compared with the loss of information about each character.

4.3.5 Maximum Lookahead Heuristic

The lookahead a required by the Viterbi proof is not always needed in practice, and previous work [133] suggests that the results of tag insertion commonly converge at lookaheads much lower than a. The maximum lookahead heuristic is to select a lower lookahead that represents a trade-off between correctness and efficiency. The lower lookahead is denoted a'. If a' is too low, the lowest entropy tagging may not be found; this may be detectable during evaluation (see Section 2.3.4). If a' is too high, the search space is unnecessarily large.

The CEM implementation collects statistics on the maximum size of every tag, but leaves the selection of a lower lookahead to the user. The performance of maximum lookahead in practice is examined in detail in Section 6.4.4. Various methods for limiting the depth of Viterbi search are discussed in [118].

4.3.6 TagC Heuristic

As presented so far, CEM considers every possible combination of tags whenever it considers inserting any tags. In real documents, however, only limited ranges of permutations of tags are found. The TagC heuristic involves tracking during training the set of all tag permutations seen. For example, the training text *<entry> <author><forenames> Donald E.</forenames> <surname>Knuth. </surname></author>*... would add { (*<entry> <author> <forenames>*), (*</forenames><surname>*) and (*</surname> </author>*)} to the set of permutations. When tags are inserted, only the permutations seen in training are considered for insertion (plus closing tags at the end of the file to guarantee that all tags are closed).

The TagC heuristic has no effect on segmentation problems (since there are only two states) and only limited effect on classification, because only one tag can be closed and one opened, limiting the number of permutations. The significantly more complex schemas involved in entity extraction (see Figure 4.2) give considerable scope for savings to be made. The savings will be greater for complex schemas when a relatively small set of permutations is seen during training. The performance of the TagC heuristic is discussed in Section 6.4.5.

4.3.7 State Tying

State tying is a widely-used heuristic in speech recognition [60], which appears not to have been used before in text modelling. The insight on which state tying is

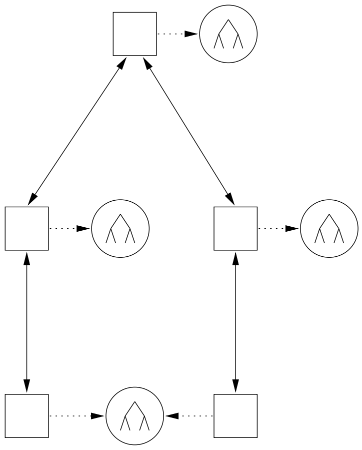


Figure 4.7: The structure of a hidden Markov model, with state tying. The squares are hidden states, linked by the solid arrows of the model structure and by dotted arrows to their associated models.

built is that some states in a large model are similar not by chance but because they model similar concepts. Thus in a speech-recognition system, the models for the second half of the words ‘hair’ and ‘pair’ are similar (or at least they are for certain dialects) even though the words themselves are different and they may represent different parts of speech. State tying uses a single underlying Markov model to model several hidden states. The hidden states are not merged—at a higher level the model tracks the difference between them—but they share a PPM model and should require significantly less training data. Figure 4.7 shows the hidden Markov model shown in Figure 4.3 with two leaf states tied.

The key benefit of state tying is the ability to share training data between relatively common and relatively rare tags so as to achieve better performance from the same amount of training data. State tying only works on entity extraction problems, because it requires at least two levels below the document root to tie together. Tying two states in a classification problem would leave two indistinguishable states. In a segmentation problem there is one (non-root) state, which cannot be tied to itself.

By default CEM performs state tying on all states with the same tag *name*. The effect of *not* tying the *name* tag is examined in Section 6.4.6.

4.4 Search Space

As discussed in Section 2.4, the efficiency of abstract computer operations is expressed by complexity, using the $\mathcal{O}(x)$. In the case of the tag insertion methodology presented here, the parameters are the numbers of tags (t), the lookahead (a) and the size of the input is the length (n) of the text. This complexity is a reflection of tagging action, rather than the complexity of the underlying intellectual or syntactic complexity [16].

If u is a constant and x and y are unbounded positive variables, $\mathcal{O}(u) \ll \mathcal{O}(x) \ll \mathcal{O}(x^u) \ll \mathcal{O}(u^y) \ll \mathcal{O}(x^y)$. Algorithms with $\mathcal{O}(u^y)$ or greater are referred to as intractable and run in non-polynomial time on conventional computer equipment.

A line of investigation in the MUC conferences (see Section 2.2.4) was measuring the inherent complexity in the web of atoms in the named entity tasks [7]. This approach relied on a uniform model of textual atoms extracted into a relational database and a network of inferred relations between them, not readily adaptable to the approach under consideration in this thesis. It was discovered was that tasks considered in MUC-5, MUC-6 and MUC-7 had surprisingly similar complexity, suggesting that the underlying complexity of textual understanding tasks may not be as great as that of the solutions presented here. This approach is not applicable to the present work because no web of atoms or equivalent structure is constructed by systems such as CEM.

This thesis examines only the efficiency of text augmentation by tag insertion, rather than the building of models which is a prerequisite to this activity. There is other work in the area of efficiently building models [97, 133], but it is outside the scope of this thesis. CEM builds the suffix tree with a hash table from the standard Java libraries. The hash key is the character leading to the node stored in the hash value. Character counts are stored in the child node. Character counts are stored as Java longs and never rescaled (none of the corpora dealt with in this thesis are sufficiently large to overflow a long).

This analysis of search space is dependent on the constraints introduced in Section 4.1.4. Removing Constraint 3 would add an infinite number of empty tags into the search space, and removing Constraint 4 would add an infinite number of non-empty tags. Therefore analysis includes recursive tags, but only when there is at least one character between each two open tags of each type.

If a document contains a single character, it could potentially have tags inserted either before or after that character. By Constraint 3, which forbids empty tags, any tags inserted into such a document must open before the character and close after it. By constraint 4, each tag can only open once. If the document is being marked up using a set of t tags, then $0, 1, 2, 3, \ldots$ or t tags could occur before the character, with the tags chosen being a permutation of the t tags. Thus, the number of combinations of tags that might be inserted prior to the first character is:

$$\sum_{i=0}^{t} {}_tP_i \qquad = \qquad \sum_{i=0}^{t} \frac{t!}{(t-i)!}$$

Constraint 3, which prevents the opening of tags that would be empty, and Constraint 2 which requires that all open tags must be closed, means the only tags following the final character in any document are close tags matching those tags remaining unclosed. Thus the number of taggings of the entire document is the same as the combinations of tags that might be inserted prior to the first character.

If a document with the single character ‘a’ is tagged with the two tags, ‘<x>’ and ‘<y>’, then there are $\sum_{i=0}^{2} {}_2P_i = 1 + 2 + 2 = 5$ possible taggings.

In a document of two characters, the same tags might be inserted prior to the first character as in the case of a one-character document. More tags may occur between the first and second characters: tags may be closed as well as opened. The maximum number of tags that may be opened is directly related to the number of tags previously opened:

$$\sum_{j=0}^{i}(\sum_{k=0}^{t} {}_tP_k)$$

where i is the number of tags opened before the first character.

As before, the tags following the last character can only be the closing tags of already open tags. This gives the total number of taggings for a two character document as:

$$\sum_{i=0}^{t}\left({}_tP_i \times \sum_{j=0}^{i}\sum_{k=0}^{t} {}_tP_k\right) = \sum_{i=0}^{t}((i+1) \times {}_tP_i) \times \sum_{k=0}^{t} {}_tP_k$$

Thus, if a document with the two characters ‘ab’ is tagged with the two tags the ‘<x>’ and ‘<y>’, then there are $1 \times 1 \times 5 + 2 \times 2 \times 5 + 2 \times 3 \times 5 = 55$ possible taggings. The formula on the right can be considerably simplified, but the $\sum_{j_x=0}^{t\ldots} \sum_{k_x=0}^{t} {}_tP_{k_x}$ factor can be factored out.

The number of taggings for a three-character document follows from this:

$$\sum_{i=0}^{t}\left({}_tP_i \times \sum_{j_1=0}^{i}\sum_{k_1=0}^{t}\left({}_tP_{k_1} \times \sum_{j_2=0}^{i-j_1+k_1}\sum_{k_2=0}^{t} {}_tP_{k_2}\right)\right)$$

$$= \sum_{i=0}^{t} {}_tP_i \times \sum_{j_1=0}^{i}\sum_{k_1=0}^{t}(k_1 - j_1 + i + 1) \times \sum_{k_2=0}^{t} {}_tP_{k_2}$$

and each additional character in the document adds a $\sum_{j_x=0}^{t\ldots} \sum_{k_x=0}^{t} {}_tP_{k_x}$ term to the number of taggings, which is $\mathcal{O}(t^2 t!) = \mathcal{O}(t!) = \mathcal{O}(t^t)$.

Classification is significantly simpler, because each character can be put into only one of t classes, giving ${}_tP_1$ or t options, which is $\mathcal{O}(t)$. Segmentation is even simpler: either a tag is inserted or no tag is inserted, a binary decision, giving $\mathcal{O}(c)$ where c is a constant.

Table 4.1 gives the number of nodes in search spaces, first for inserting tags between two characters in a document and then for inserting tags into an entire document for each variant.

4.4.1 The Semantics of Nested Tags

Permutation is a significant contributor to the search space, particularly when t is large. If the semantics of nested tags (see Section 2.5.1) were changed so that opening tags occurring between two adjacent characters are semantically equivalent, independent of order (i.e. widely expected HTML / XHTML semantics), this would change the permutation to a combination, substantially reducing the search space for entity extraction. Changing the semantics of nested tags also drastically reduces the maximum number of Markov models which would be needed in the case where tags are not used consistently, increasing the usefulness of state tying (see Section 4.3.7).

Segmentation and classification do not involve nested tags, so their semantics are irrelevant.

4.5 Teahan Search

Not all of the optimisations and heuristics described above can be applied to the Teahan search algorithm. In particular, those that relate to pruning the depth of the search space (the Viterbi and best-first optimisations, and the maximum lookahead heuristic) cannot be used because the Teahan search does not consider depth of search. Automatic tokenisation, which applies to the nodes at which the search tree can branch, can be used with Teahan search, as can the TagC heuristic, which relates to the width of the branching.

Algorithm	Segmentation	Classification	Entity Extraction
per Character	$\mathcal{O}(c)$	$\mathcal{O}(t)$	$\mathcal{O}(t^t)$
Complete	$\mathcal{O}(c^n)$	$\mathcal{O}(t^n)$	$\mathcal{O}(t^{tn})$
Viterbi	$\mathcal{O}(c^a)$	$\mathcal{O}(t^a)$	$\mathcal{O}(t^{ta})$
Maximum Lookahead	$\mathcal{O}(c^{a'})$	$\mathcal{O}(t^{a'})$	$\mathcal{O}(t^{ta'})$

Table 4.1: Search space size. t is the number of tags, t is the document length, a is the lookahead for Viterbi search, a' is the lookahead for maximum lookahead search and c is a constant.

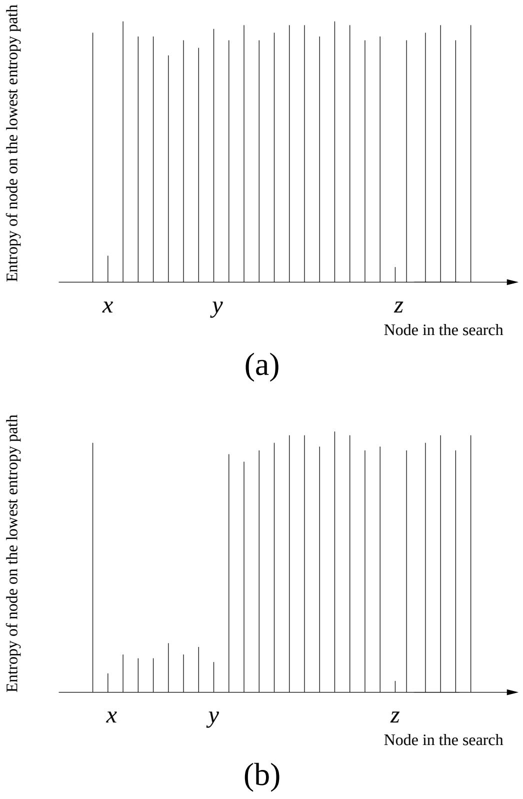


Figure 4.8: Scenarios in which Teahan search and Viterbi search can be expected to perform differently, (a) Teahan search performs well and (b) Viterbi search performs well.

Both Teahan search and Viterbi search with maximum lookahead are heuristics and it makes sense to ask which can be expected to perform better, or *might* perform better, than the other. There is no *a priori* reason to believe that one will perform better in the general case, but in specific cases they perform differently. Viterbi search can be expected to perform well in situations in which there is a great deal of ambiguity (a small entropy difference between a large number of nodes at the same level) in the search tree, because it focuses on searching the current, immediate context. Teahan search will perform better when the search contains long sequences of low ambiguity interspersed with short sequences of high ambiguity because, by counting only the leaves, it is able to look effectively past the long sequences of low ambiguity.

Figure 4.8 shows two scenarios which illustrate such situations. It shows the entropy implications of inserting a single tag at various points in a sequence. In Figure 4.8(a) all the points are high-entropy, except *x* and *z* which are low entropy. Viterbi search with maximum lookahead is only capable of determining whether *x* or *z* is the better place to insert the tag if the difference between them is a' or less. Teahan search is capable of making the differentiation no matter what the separation, provided there are no (or relatively few) other low entropy branching options between *x* and *z*. Figure 4.8(b) still has *x* and *z* but also has a range of relatively low-entropy branching options between *x* and *y*. In such a situation Teahan search is likely to prune prematurely at *x*, whereas Viterbi search with maximum lookahead is guaranteed to find the best option within the a' maximum lookahead.

4.6 Evaluation

This section examines how the measures of correctness first introduced in Section 2.3 can be used in conjunction with the metadata taxonomy introduced in Section 4.1. For each of the measures, each of the three taxa is examined. A new correctness measure, *type confusion matrices*, is introduced.

4.6.1 Recall and Precision

Recall, precision, and their combination in the F-measure, are the primary means of evaluating correctness in information-retrieval systems, but the definition of what constitutes a document varies for each type of text-augmentation problem.

Segmentation

For segmentation problems the evaluation question is ‘Does a segment end between one symbol and the next and was that segment end found?’ Recall and precision are good measures for evaluating segmentation problems because both operate on *to␣be␣or␣not␣to␣be*

(a)

<to>to</to>␣<be>be</be>␣<cc>or</cc>␣<xnot>not</xnot>
␣<to>to</to>␣<be>be</be>

(b)

Figure 4.9: A short quote from Hamlet. (a) without and (b) with part of speech tags.

a binary distinction. Recall and precision are the standard methodology for measuring correctness in the fields of Chinese text segmentation [137, 145, 12, 50] and Japanese text segmentation [3], both widely-studied segmentation problems.

Classification

For classification problems, the evaluation question is ‘Is the class predicted for symbol n correct?’, where symbols are the characters, words, sentences or documents being placed into classes. Recall and precision are standard methodology for measuring correctness in the fields of part-of-speech tagging [28, 76, 94] and genre classification [66], which are probably the most widely-studied textual classification problems.

Figure 4.9(a) shows a short quote from Hamlet and Figure 4.9(b) the same quote marked up using the tags of the Lancaster Oslo/Bergen part-of-speech corpus [64]. Teahan’s work (from which this example is taken) [133] is a word-based approach and uses word-based evaluation mechanisms: there are 6 words in the sample and they are all correctly tagged, giving 6 true-positives. Character-based approaches see only characters not words: there are 18 characters, including 5 spaces, all correctly tagged, giving 18 true-positives. Evaluation of the output from a character-based system using a word-based evaluation might be considered. However, this works for mistakes such as misclassification of an entire word, but fails when only part of a word or a non-word character is misclassified. There are similar problems in evaluating Optical Character Recognition (OCR) at a word level when word boundaries can be incorrectly identified [73].

The core problem is that character-based approaches are more expressive and can be wrong in ways that cannot be represented in conventional word-based approaches. The reverse is not the case, however, and the output of a word-based system can be compared to that of a character-based system at the character level.

The expressiveness of character-based approaches definitely has advantages in some corpora. For example, dates in the Computists' corpus (Section 5.1) are expressed as a single word in the form *19Jan98* which word-based approaches see as a single word (unless they have customised word boundaries heuristics) and are unable to do better that identifying it as a date (*<date>19Jan98</date>*). Character-based approaches are capable of breaking the date into component parts (*<date><day>19</day><month>Jan</month><year>98</year></date>*).

The difference in expressiveness applies to all three types of text augmentation problem if the standard measurement technique is word-based, but is most obvious in classification problems such as part of speech tagging.

Entity Extraction

Measuring entity extraction as an information retrieval problem is challenging. The four basic classes (true positives, false positives, false negatives and true negatives) are accumulated over successive independent trials, but the XML well-balancedness constraint (see page 56) introduces inter-dependencies between trials.

Figure 4.10 shows inter-dependencies in a small entity extraction problem. The untagged input text is shown in Figure 4.10(a). The task is to insert *<name>* and *<title>* tags into the text, as shown in Figure 4.10(b). Figure 4.10(c) shows an error: the boundary between the first two names has been inserted in an incorrect place: the tag *<name>Smolensky, P., Fox, </name>* is a false positive. The independence criterion is broken because seeing this false positive does not just preclude the possibility of seeing the tag *<name>Smolensky, P., </name>*. It also precludes the

Smolensky, P., Fox, B., King, R., and Lewis, C. Computer-aided reasoned discourse...

(a)

<name>Smolensky, P., </name><name>Fox, B., </name><name>King, R. </name>, and <name>Lewis, C. </name><title>Computer-aided reasoned discourse... </title>

(b)

<name> Smolensky, P., Fox, </name> B., <name> King, R. </name> , and <name> Lewis, C. </name> <title> Computer-aided reasoned discourse... </title>

(c)

Figure 4.10: Inter-dependencies in a small entity extraction problem.

possibility of seeing the tag *<name>Fox, B., </name>*.

The possibility of *<name>Smolensky, P., Fox, </name>*, *<name>Smolensky, P., </name>* and *<name>Fox, B., </name>* as names is not precluded if the data is segmented into a relation before processing. However, such segmented results could not be merged back into XML using tags such as we are using if these three names are included.

It is unclear whether breaking of the independence criterion matters. Certainly it means that recall and precision results from entity-extraction problems are in some way different from segmentation and classification results, and not directly comparable. Recall and precision are the primary means of comparison in the TREC, MUC and DUC conferences (see Section 2.2.4).

4.6.2 Edit Distance

The correctness of all kinds of metadata used in text augmentation can be measured using edit distance.

4.6.3 Confusion Matrices

As with recall and precision, the effectiveness of confusion matrices on different kinds of text augmentation problems varies.

Segmentation

Confusion matrices of segmentation problems represent a degenerate case in which there are only two classes. The matrix contains the four basic measures from the information retrieval paradigm and is a contingency table:

$$\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} = \begin{bmatrix} true\,positives & false\,positives \\ false\,negatives & true\,negatives \end{bmatrix}$$

For this reason evaluating segmentation using a confusion matrix or the information retrieval metrics produce the same results, but the information retrieval metrics have higher level metrics (recall and precision) built upon them.

Classification

Confusion matrices are the standard method of evaluating classification tasks [149]. Their only disadvantage is that they are somewhat verbose, especially for problems (such as part-of-speech tagging) which have a large number of classes.

Entity Extraction

Confusion matrices have identical independence problems to recall and precision when used to evaluation entity extraction from text. Confusion matrices assume an underlying many-class classification task, but entity extraction in the most general form is more general than this; it is a *hierarchical* many-class classification task. If the hierarchy depth is bounded in some way, it is possible to re-define the problem such that every possible state in the hierarchy is a new class. This approach suffers from problems of combinatoric explosion, leading to large, sparse, matrices which cannot be normalised, since this leads to division by zero.

4.6.4 Type Confusion Matrices

Type confusion matrices are a new extension of confusion matrices suitable for application to hierarchical many-class classification tasks. Every node in the hierarchy is assigned a type, which is the most recently opened tag. The type confusion matrix for a hierarchical classification problem with i classes is:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,i} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,i} \\ \vdots & \vdots & \ddots & \vdots \\ a_{i,1} & a_{i,2} & \cdots & a_{i,i} \end{bmatrix}$$

$a_{m,n}$ in column n and row m is the number of symbols that should have been classified in a node of class n that were actually classified in a node of class m.

Type confusion matrices can be used similarly to confusion matrices, but it should be noted that information has been thrown away. For example, if the sequence …*S. Kraus, and V. Subrahmanian*… is marked up as:

…*<editor><name><first>S.</first><last>Kraus,</last></name>and <name><first>V.</first><last>Subrahmanian</last></name></editor>*…

rather than as:

…*<author><name><first>S.</first><last>Kraus,</last></name>and* - *<name><first>V.</first><last>Subrahmanian</last></name></author>*…

the *author* / *editor* confusion would only be apparent in the *and* sub-sequence. Other sub-sequences such as *Kraus,* do not have the erroneous tag as an immediately enclosing tag. This situation is much worse when dealing with classes whose only content is other classes such as the *bibbody* tag which always contains a single other tag.

Type confusion matrices are applicable to any tag insertion problems. However, when applied to a classification problem, they degenerate to a confusion matrix because the immediately enclosing tag is the only tag. When applied to segmentation problems, type confusion matrices degenerate to a contingency table (see page 81).

4.6.5 Entropy

All types of text augmentation can be evaluated using entropy. Care does need to be taken to avoid using the same model or a model built from the same data for both augmentation and evaluation. If entropy is being used for evaluation, it is normal to either use an empty adaptive model or a model built from data which is distinct from the training, re-estimation or testing data.

When a tag insertion using a Viterbi algorithm, produces an incorrect result, entropy measurements can be used to determine whether the fault lies with the model or the searching algorithm. If the result produced by tag insertion has lower entropy than the baseline (or ground truth) text, the model is flawed (i.e. has not seen enough training data, is not of sufficient order, or is attempting to linguistically model non-linguistic features). If the experimental result has higher entropy than the baseline (or ground truth), the searching algorithm is flawed (i.e. one of the heuristics is making an assumption that does not hold for this text). This technique is used in Section 6.4.3 to examine the effectiveness of the alphabet-reduction heuristic.

Chapter 5

The Text

In this chapter the four corpora used in this thesis are introduced, the problems posed by the corpora are described and previous work solving these, or similar, problems is discussed.

In the information-retrieval paradigm, a collection of documents is called a 'corpus' and is assumed to have some commonality: the documents are either from the same source, cover the same topic, or are a representative sample of a larger population of documents. Building corpora, especially those with rich metadata about and within the documents, can be expensive and time-consuming.

In the research community, corpora serve as pools of data for exploratory research [91, 92] and as benchmarks for comparative research [65, 64]. This thesis uses them for both these purposes. The corpora used here are referred to as: the Computists' corpus, the bibliography corpus, the Chinese text-segmentation corpus and the Reuters' corpus. Each of these is discussed in the following sections. Short samples of each can be found in Appendix 1.

5.1 Computists' Corpus

The Computists' corpus [136, 135, 148, 26, 144] is composed of issues of a magazine called 'The Computists' Communique' converted from ASCII text to XML. Each of the 38 issues is approximately 1200 words in length and consists of a number of short articles usually followed by a list of job openings. Previous workers marked up ten features (*name*, *location*, *organisation*, *email*, *source*, *date*, *money*, *phone*, *fax* and *url*) by hand, and then made corrections based on the results of the Teahan's TMT [135].

```
(937) 255-2902. <http://web.fie.com/htdoc/fed/afr/wri/any
/proc/any/07209802.htm>. [CBD, 20Jul98.]
```

(a)

```
<p>(937) 255-2902</p>. <<<u> http://web.fie.com/htdoc/fed/afr/wri/any</u>
/proc/any/07209802.htm>. [<s> CBD</s>, <d> 20Jul98</d>.]
```

(b)

```
<p>(937) 255-2902</p>. &lt;<u> http://web.fie.com/htdoc/fed/afr/wri/any
/proc/any/07209802.htm</u>&gt;. [<s> CBD</s>, <d>20Jul98</d>.]
```

(c)

Figure 5.1: Corrections in the Computists' Communique. (a) the original text (b) the text as received (c) the text used in this thesis.

For this thesis the data was converted from the XML-like format used by TMT into well-formed XML and a number of systemic errors corrected. Figure 5.1(a) shows two lines from corpus as it appears in the original text, notice that a URL has been broken across a line break. Figure 5.1(b) shows the text as used by Teahan, Bray and Wen [135, 26, 144]. Four tags have been added: phone number, URL, source and date. Only the first part of the URL has been marked-up as a URL. The insertion of the URL and email address (not shown) tags was done automatically, inserted extra '<' and the URL detection failed when the URL had been line-wrapped. The text also has un-escaped '<', '>' and '&' (not shown) characters, which are non-well-formed XML. Figure 5.1(c) shows the same text with these deficiencies corrected. This is the version used in this thesis.

The corpus has a number of endemic ambiguity issues: (a) mailing-list names are listed as sources when derived from the mailing list but not when creation of the mailing list is announced; (b) many of the organisation names (particularly *Apple*) were marked up intermittently and (c) many words are marked up coincidentally.

For example, in a discussion about computers from IBM and Sun Microsystems, *Sun* is marked as an organisation even when used as a class of computers. *PC* is never marked as an organisation. These issues, and the fact that organisations and sources are named after places and people, and that place names are often coined from personal names, account for the many of errors previously reported [26, 144].

Several corrections to the corpus are made in this thesis to attempt to resolve (b) and (c). Two passes were made over the corpus, marking-up organisations (and to a lesser extent sources) which had not been marked-up in previous work. This revised corpus is used everywhere in this thesis except Section 6.2.2, where results are compared with previous work and therefore the uncorrected data must be used. To the author's knowledge the corpus is in the public domain. Copies are available from the author.

Inserting the ten features into the Computists' corpus is a classification problem. Figure 4.1(c) shows the schema structure for the problem. The MUC named entity problems from the MUC conferences have strong correspondences to the *name*, *location*, *organisation*, *source*, *date* and *money* tags.

5.2 Bibliography Corpus

The bibliography corpus was created specifically for this thesis from bibliography records. It was designed to resemble the bibliographies found in the computer science technical report collection at the New Zealand Digital Library [153, 109]. The corpus consists of a large number of bibliographies generated by the LaTeX / BibTeX tool-chain which is widely used throughout technically-oriented scientific disciplines. It is anticipated that a model trained on the bibliography corpus may be adaptable for academic fields which use humanities citation conventions by using the Baum–Welch algorithm (see Section 3.6), but this is not explored in this thesis. Marking up bibliographies is a first step for several activities, including document linking, bibliometrics [111] and a range of possible integrated reading list, bibliography and citation systems, making it a desirable feature for a digital library.

A collection of publicly available bibliographic databases[1] has been maintained and expanded by other workers for a number of years. Samples of bibliographic entries were taken from the same sources as this collection, split into 14682 bibliographies with up to 25 entries and formatted using the BibTEX and LATEX [77] text-formatting systems. Seven of the standard bibliography styles (*abbrv*, *alpha*, *apalike*, *ieeetr*, *plain*, *siam* and *unsrt*) and several different page layout techniques (*article*, *book* and *report*) were used to mitigate secondary effects due to line, column and page wrapping.

Addition of metadata tags into the bibliographies changed the layout of entries; line breaks and hyphenation, in particular, were radically changed. To avoid this, each bibliography was processed twice, once using the standard style file and once using a modified style file which inserted metadata tags around parts of the entries. This process is shown in Figure 5.2. The upper half of the figure shows the processing of the bibliography (.bib) using the unmodified style file (.sty) to produce the laid-out bibliography (.bbl) using BibTEX. This laid-out bibliography was then processed to a PostScript (.ps) document using LATEX and dvips, and then the PostScript document processed to a text file (.txt) using ps2txt. The lower half of the figure shows the processing of the bibliography using the modified style file to insert escaped XML tags. The resulting two text files were then merged into a single XML document, taking the layout, whitespace and punctuation from the text derived from the unmodified style file and un-escaping the escaped XML tags from the text derived from the modified style file. The resulting bibliographies were processed using the XML ‘preserve-space’ style to preserve whitespace.

There are several peculiarities in the corpus, largely because of how it was constructed.

[1] `http://liinwww.ira.uka.de/bibliography/index.html`

1. All first names are marked-up in a single tag rather than each first name in a separate tag. The BST language[2] in which the style files are written has primitives for laying out names. Marking-up individual first names separately would have required a modified BST interpreter rather than modified BST programs.

2. There are inconsistencies in the relative location of punctuation and close tags at the end of words. The period following an initial is an indication of contraction, semantically part of the initial, whereas the period at the end of a sentence is semantically separated from the word it follows. The tagging attempts to reflect this, but there are some deeply ambiguous cases, particularly where an initial falls at the end of a sentence and the period fills both roles. In such cases the punctuation has been included within the tag.

3. Splitting a large bibliography into many smaller ones breaks cross-references between entries unless both referrer and referent happen to appear in the same smaller bibliography. Broken cross references appear as '*[?]*'.

LaTeX commands to generate non-ASCII characters in the text are escaped to Unicode characters. The conversion is based upon the commands observed in the corpus rather than a comprehensive list of commands, but includes many common mathematical symbols and letters from a wide variety of Western European languages (Portuguese, Spanish, German, Polish, Swedish, etc.). Most of the letters appear in names, either in the name field or as references to people in titles. A few of the entries were entirely in French. Many bibliography entries with non-ASCII characters also occur in a Romanised form, with the non-ASCII characters converted to ASCII characters by bibliography creators.

[2] The author knows of no comprehensive description of the BST language; the implementation is part of BibTeX. It is a stack-based language in which sets of non-recursive macros (called 'style files') are used to format convert entries in a standard format (for which again, a canonical description appears to be lacking) into bibliography entries conforming to the stylistic conventions of a particular publication.

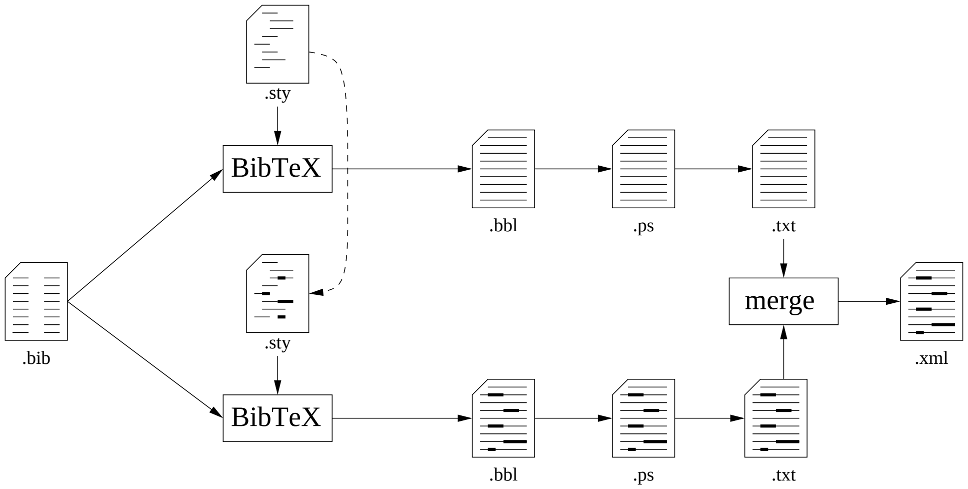


Figure 5.2: Data-flow diagram for creating the bibliography collection.

Escaping non-ASCII characters rather then dropping them out of the corpus made the corpus significantly less close to the computer science technical report collection, but significantly closer to bibliographies as they appear in the majority of electronic documents, and closer to how they were intended to appear. Other researchers have discarded such bibliographies, at the rate of 6.5% [125].

Many of the discarded bibliographies contain LaTeX macros which could never be processed by standard LaTeXṠome appear to be mis-typed macros, but there is no way to distinguish these from macros which individual researchers have defined locally. There are also many sets of macros circulating in subject- and language-specific communities to represent features of interest within those communities. The lack of namespaces in LaTeX means that there is no easy way to differentiate these, and because macro files are imported into the document rather than the bibliography, isolated bibliographies contain no reference to the file name which defines (or redefines) macros.

The structure of the schema is shown in Figure 5.3. The tags at levels B and C indicate bibliographies marked-up according to certain bibliography and document styles respectively. All combinations of these were used when creating the corpus. Tags at level E correspond to tags of different types of documents being referenced.

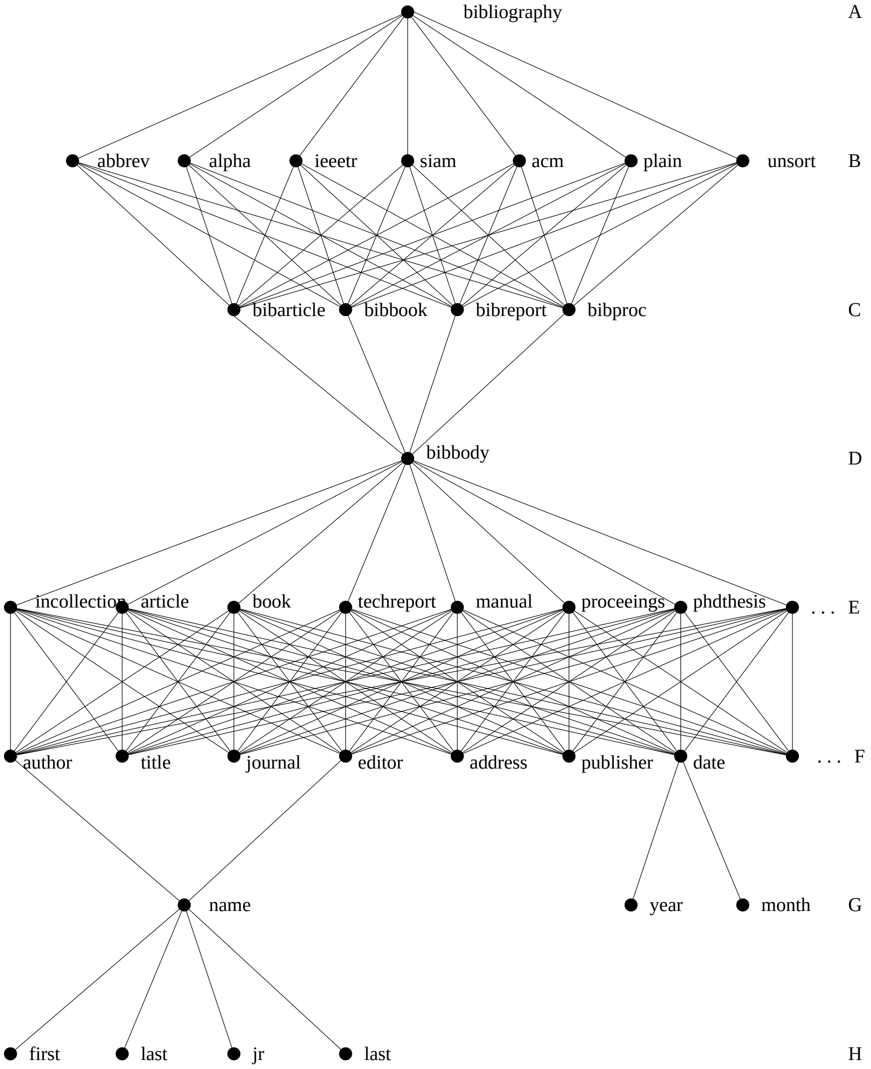


Figure 5.3: Schema for the bibliography corpus with all tags.

Tags at level F correspond to the fields in bibliographic records.

The structure of names in the BIBTEX format is somewhat unusual. With four parts (first, last, van and jr), the structure reflects American English names as conceptualised in the 1980s, but handles rather poorly a number of features of names as used internationally, particularly double-barrelled surnames, von parts[3] starting with a capital and names in which the given name follows the surname. One of the causes is systematic confusion between the portion of the name which is written first and the given (as opposed to inherited, parental) portion. These issues are compounded by the difficulties representing non-ASCII characters in LATEX, for example the need to encode '*Céline*' as '`C{\'{e}}line`,' and the use of a simplistic sorting algorithm for ordering the entries.

A number of different workarounds have been developed to force BIBTEX and LATEX to 'do the right thing' in sorting, formatting and hyphenating particular names. A collection of these can be found in the archives of the `comp.lang.tex` newsgroup. Other name formats, such as the Library of Congress authority lists [112] used in the MARC [108, 48] format are actively curated, enabling such issues to be handled systematically, if not optimally. In this thesis, the original BIBTEX terminology is used because it is precise and clear to workers and tool builders in the field [77, 101].

Not all the tags shown in Figure 5.3 are used in this thesis. Figure 5.4 shows only those tags in the corpus which are used in experiments in this thesis. Note, in particular, that the tags at levels B, C and E in Figure 5.3 are missing in Figure 5.4. The variant schema structure shown in Figure 5.5, and explained in Section 4.3.7, is used in experiments with state tying.

Freitag and McCallum [46, 96] report work on a similar, although non-hierarchical, corpus initially hand-crafted, then incrementally improved using Markov models. Citeseer [80] (see Section 2.2.4) also involves bibliographic data,

[3] In the BIBTEX model of names, fragments such as '*von*' and '*van der*' are referred to as the 'von part'.

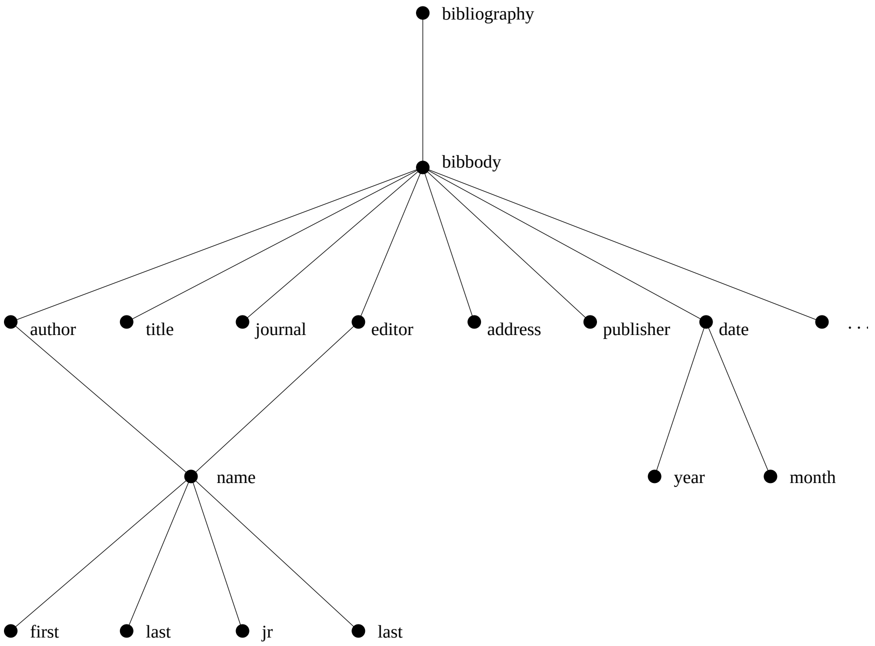


Figure 5.4: Schema for bibliography corpus with tags used in this thesis (with state tying).

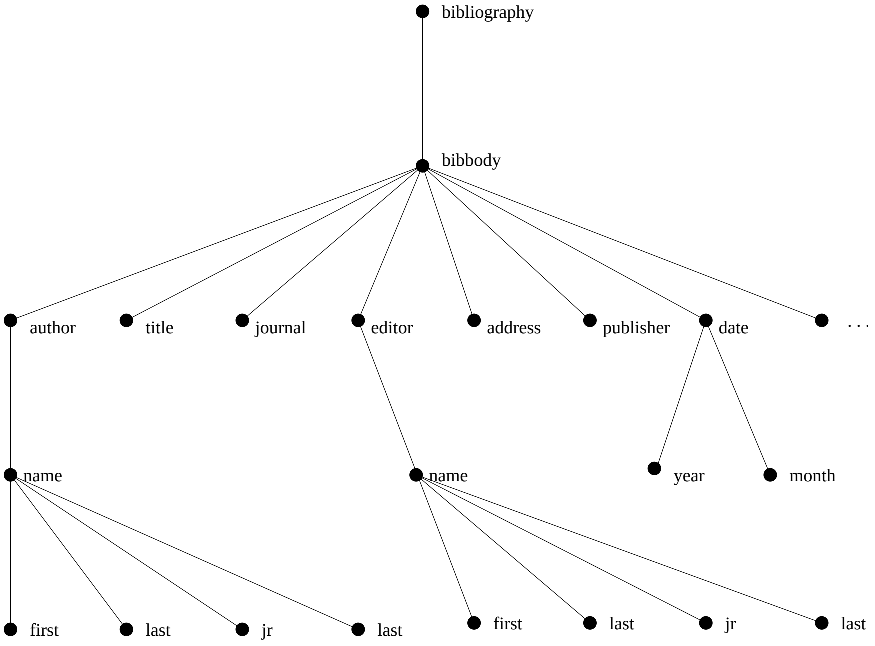


Figure 5.5: Schema for the bibliography corpus without state tying.

using a handcrafted multi-step algorithm.

5.3 Segmentation Corpus

The segmentation corpus was derived from the ROCLING segmentation corpus. which contains about two million pre-segmented words, represented in the Big5 coding scheme. The corpus was converted from Big5 encoding to GB (Guojia Biaozhun) by Wen [137].

The corpus was further converted from GB encoding to Unicode. After inserting *word* tags, whitespace (but not punctuation) was removed and the text split on sentence boundaries into 1000 documents of approximately the same size. The XML was output as ASCII to force all non-8-bit clean characters to be converted into Unicode escapes to reduce the chance of handling errors.

In the resulting corpus, a two character word looks like: *<word>&#x065f6;-&#x05019;</word>*. The corpus also includes western terms (for example, proper nouns and currency symbols). A thorough review of Chinese text segmentation is given in Teahan and Wen [137]. As the author neither reads nor speak Chinese, he is unable to give a detailed analysis. The results of previous workers are shown in Table 6.7.

The segmentation corpus appears to suffer from the overly 'optimistic segmentation' described by Wu and Fung [157]. This phenomenon is caused by the tendency for many segmentation algorithms to be biased towards smaller segments when faced with even genuine ambiguity.

Inserting *word* tags into the segmentation corpus is a segmentation problem. Figure 4.1(a) shows the schema structure for the problem.

5.4 Reuters' Corpus

The Reuters' corpus is a collection of news articles taken from the Reuters' news wire and referred to by Reuters as 'Reuters Corpus, Volume 1, English language, 1996-08-20 to 1997-08-19'. The articles range from two-paragraph summaries of financial information to in-depth articles on political or literary topics. The corpus has been widely studied for a number of purposes, including text categorisation and clustering [62, 55], information extraction [45, 46, 119], authorship [68], and part of speech tagging [46].

This is the sort of news discussed on page 1: automatically inserting tags, either as a first step in a more sophisticated information-extraction process, or simply to tag articles as being connected to the organisations and locations. This process, or one similar to it, is performed ubiquitously in the field of news aggregation.

The corpus was prepared for this thesis by taking the first 7471 articles from the full Reuters' corpus, removing the document level metadata (title, author, topic and copyright information) and passing it through the Brill tagger [28], a widely used part-of-speech tagger that tags every word with a label that indicates the role it plays in speech. The tagger's notion of what constitutes a word is sometimes unusual—*Don't* is regarded as two words and *dollar/yen* as one word—but the tagger was used 'out of the box' according to accepted practice [46, 119]. 11 documents containing URLs, which confused the tagger's parser, were removed. The full Reuters' corpus contains many duplicates [69], but as with other corpora and information systems, the presence or absence of duplicates is not as important as whether the corpus is a representative sample of the larger population of documents. Given that identical or similar news articles commonly appear in a number of publication outlets, having duplicates and near-duplicates in the Reuters' corpus is a sign of correlation with 'real-world' news sources, rather than a sign of a flaw.

The full Reuters' corpus is large (over 800,000 articles), but only the first block of articles is used here, since the behaviour of text augmentation on large bodies of text is not the primary interest of this thesis and has been studied elsewhere [133]. A complete explanation of the meanings of each of the 38 tags is contained in [94]. The text of the Reuters' corpus is copyright Reuters and not for redistribution. Copies of the corpus are, however, available from Reuters.

Inserting part of speech tags into the Reuters' corpus is a classification problem. Figure 4.1(b) on page 52 shows the schema structure for the problem.

Chapter 6

Results

In this chapter the effects of applying the earlier discussed optimisations and heuristics to the four corpora discussed in the previous chapter are examined. The correctness results are then given and, where possible, compared against experimental results given in the literature. The effects of Baum–Welch re-estimation are examined and, finally, the effectiveness of individual optimisations and heuristics are examined.

6.1 PPM-SY versus PPMD

CEM normally uses PPM-SY, and in this section it is compared with PPMD . Figure 6.1 shows the search time per node of the search in the Computists' corpus, for a range of orders of model and a lookahead of six. The search time increases less than linearly for PPM-SY and more than linearly for PPMD.

Despite the use of leave-one-out cross-validation, the correctness of PPM-SY and PPMD was identical in all cases except for the case of the location *Capitol Hill*, which was correctly identified as a location by PPMD using models of order three and four when PPM-SY incorrectly identified it as an organisation. Using an order-five model correctly identified it as a location.

Figure 6.2 shows the search time per node of the search in the Chinese segmentation corpus, for a range of orders of model and a lookahead of four. The time increases less than linearly for PPM-SY and more than linearly for PPMD. This increase in the cost is substantially larger than in the Computists' corpus, probably

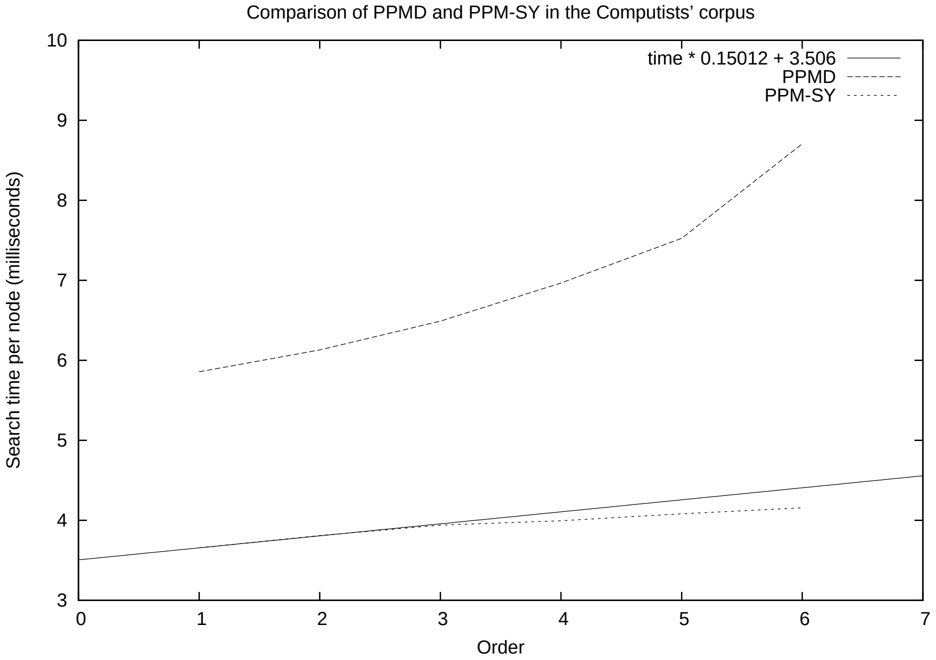


Figure 6.1: Graph showing the speed of searching in the Computists' corpus for PPMD and PPM-SY. A reference line is included to show that the speed for PPM-SY is growing less than linearly with respect to model order. Timings are averaged over leave-one-out cross-validation.

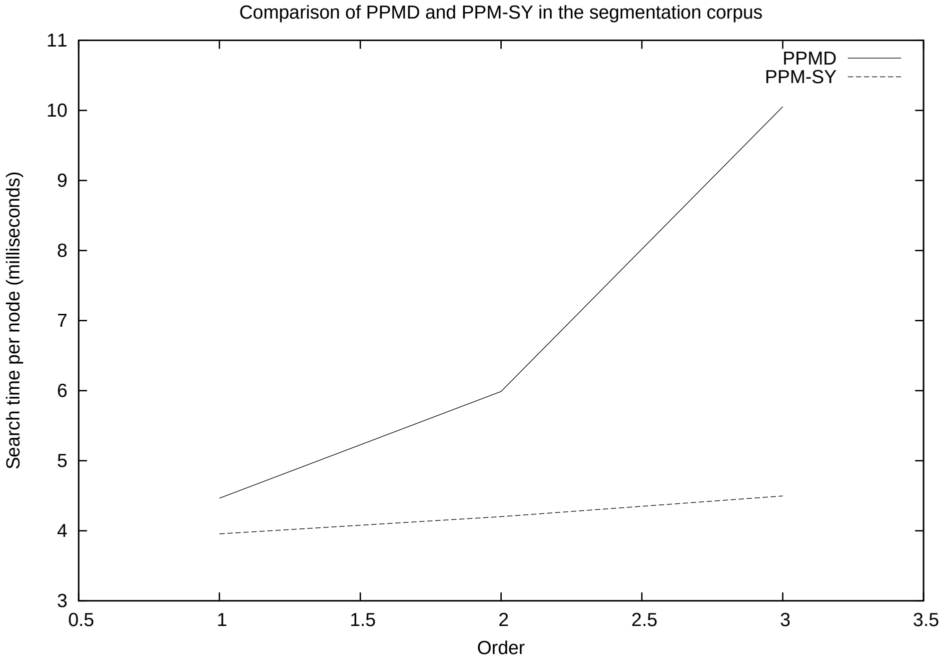


Figure 6.2: Graph showing the speed of searching in the segmentation corpus for PPMD and PPM-SY. All runs use 900 training documents and a single testing document. Results shown are averages over 100 runs.

because of the significantly larger character set involved. PPMD gave better results, on average, than PPM-SY, with a difference in F-measure of $+0.03$%, $+0.02$% and $+0.04$% for orders one, two and three respectively.

6.2 Correctness

Correctness (see Section 2.3) is studied on a corpus-by-corpus basis. Leave-one-out cross-validation is used only for the Computists' corpus, because that corpus is so small. In all other experiments, no cross validation is used except where specifically stated.

6.2.1 Granularity and Heterogeneity

Unfortunately text-mining systems of the type being examined in this thesis make the assumption that text seen during training is the same as the text seen during testing. In this sense they are not general-purpose systems in the way that PPM [133], bzip [88] or gzip [124] are. How well this assumption hold varies from corpus to corpus depending in the internal granularity and heterogeneity. For the corpora described in Chapter 5:

- The Chinese text-segmentation corpus was built from pre-homogenised data, no variation among the 1000 documents is apparent to the author.
- The Computists' corpus contains documents which all have the same structure, but with considerable variation on subject matter.
- The Bibliography corpus contains relatively homogeneous documents with two exceptions: (a) those documents generated from personal bibliographies containing all publications by an individual, and (b) those documents generated from forum bibliographies containing all publications appearing in a journal, conference or book series. These documents are entirely an artifact of the way the data was prepared—an insignificant number of peer reviewed articles are published in computer science which contain references to only a single author or source.
- The Reuters' corpus, by contrast, contains genuinely heterogeneous articles, ranging from short market-report articles, with columns of numeric figures, to long in-depth articles of political commentary.

Only the Reuters' corpus is evaluated both at a corpus level and at a document level (see Section 6.2.5). The other corpora are evaluated at the corpus level.

6.2.2 Computists’ Corpus

The Computists’ corpus has been previously studied by Bray [26], using TMT, and Wen [144]. Bray evaluated extraction based upon a confusion matrix (see Section 2.3.3) and this is reproduced in Table 6.1(a). Tables 6.1(b) and (c) show the confusion matrices for CEM on the corrected data using maximum lookahead search and Teahan search respectively. The values in (a) are measured in words, the values in (b), (c) and (d) are in characters. The *issue* tag is the background: both TMT and CEM build Markov models for the *issue* tag but Bray does not report the full results for this, so the CEM results in (b), (c) and (d) have an extra row.

For most of the tags the CEM results were comparable to, but slightly worse than the results given in Bray. Because the Bray results are percentages of words correctly classified and the CEM results are percentages of characters correctly classified, direct comparison between these results is difficult. Many of the mistakes shown in Table 6.1 for both systems appear be connected to inconsistencies, as described in Section 5.1.

Three of the tags with the best performance (*url*, *email* and *money*), deserve close attention. The first two can be described using a regular expression and the last is uniquely and exclusively identified by a single character (*$*). These properties make tag insertion much more consistent; they also make modelling such tags easier for certain kinds of models. Unfortunately it also makes marking-up using Markov models pointless: except in extreme cases marking up by regular expression is always more efficient than marking-up using Markov models and searching.

The systemic confusion between *name*, *source*, *location* and *organisation*, as discussed in Chapter 5, is clear in all three confusion tables, with greater confusion for CEM than for TMT.

Another situation in which CEM performs much worse than the Bray analysis is the *fax* tag. The most common type of error with *fax* and *phone* tags in both systems is where the fax numbers are mistaken for phone numbers: *<p>617-373-*

	d	n	s	l	o	u	e	p	f	m	i
[d]ate	**93.46**		+								6.40
[n]ame		**89.35**	+	1.31	1.50						7.48
[s]ource		+	**60.09**		2.85						36.62
[l]ocaton		+		**81.64**	4.69						12.89
[o]rg		2.56	2.56	1.63	**69.23**						24.01
[u]rl						**100.00**					
[e]mail							**97.34**				2.66
[p]hone								**82.29**	10.71		
[f]ax									**100.00**		
[m]oney										**100.00**	

(a)

	d	n	s	l	o	u	e	p	f	m	i	#
[d]ate	**91.18**	+	+	+	+			+	+		8.12	10070
[n]ame	+	**85.49**	2.75	1.78	2.02		+		+		7.27	10494
[s]ource		1.13	**51.97**	+	3.02		+		+		41.71	9983
[l]ocation	+	2.66	1.96	**72,38**	4.79		+		+		17.65	5155
[o]rg	+	3.48	2.99	3.66	**27.50**	+	+		+		60.99	5688
[u]rl	+	+	+	+	+	**95.23**	+	+			3.11	20023
[e]mail	+	+	1.14	+	+	+	**93.60**	+	+		3.30	12164
[p]hone								**88.69**	9.95		1.36	955
[f]ax								27.86	**69.14**		3.01	499
[m]oney									+	**99.47**	+	1133
[i]ssue	+	+	1.14	+	1.47	+	+	+	+		**95.92**	317169

(b)

	d	n	s	l	o	u	e	p	f	m	i	#
[d]ate	**91.04**		+	+	+		+				8.22	10098
[n]ame		**87.92**	2.19	1.26	3.92		+				4.64	11167
[s]source	+	1.27	**64.02**	+	5.99		+				27.81	14229
[l]ocation	+	2.46	1.26	**75.82**	11.38	+					8.75	5534
[o]rg		2.57	2.13	4.27	**58.48**		+				32.40	12212
[u]rl		+	+		+	**95.90**	+		+		2.88	20089
[e]mail	+	+	+	+	+	1.08	**94.70**				3.38	12186
[p]hone								**75.03**	8.88		16.10	969
[f]ax								16.43	**57.11**		26.45	499
[m]oney			+					+	+	**90.35**	7.98	1140
[i]ssue	+	+	+	+	1.28	+	+	+	+	+	**96.94**	303100

(c)

	d	n	s	l	o	u	e	p	f	m	i	#
[d]ate	**92.23**	+	+	+			+				6.99	10075
[n]ame		**92.46**	+	+	2.65						3.49	11135
[s]source	+	+	**68.11**	+	5.12		+				25.57	13881
[l]ocation		1.62	+	**84.53**	7.87						5.86	5619
[o]rg	+	2.03	2.47	2.39	**66.47**		+				26.53	12169
[u]rl		+	+		+	**96.48**	1.00				2.25	19668
[e]mail	+	+	+	+		+	**96.55**				2.23	12436
[p]hone								**72.55**	6.60		20.85	969
[f]ax		1.20						4.21	**70.14**		24.45	499
[m]oney			+					+	+	**88.80**	9.58	1107
[i]ssue	+	+	+	+	1.09	+	+	+	+	+	**97.48**	301326

(d)

Table 6.1: Confusion matrices for the Computists' corpus (a) from Bray using TMT [26] page 70, (b) from CEM/maximum lookahead using the same data as Bray, (c) from CEM/maximum lookahead using corrected data, (d) from CEM/Teahan search using corrected data. Character counts (#) are in characters, all other values are in percent, '+' indicates a figure lower than 0.99%. A lookahead of 6 was used.

Author	Recall	Precision	F-measure
Wen	65.29	73.35	69.09
CEM/maximum lookahead (Wen's data)	49.17	63.38	55.38
CEM/maximum lookahead (corrected)	71.06	61.21	66.13
CEM/Teahan (corrected)	74.65	67.71	71.18

Table 6.2: Accuracy for the Computists' corpus, from Wen [144] page 75 and from the current work. A lookahead of 6 was used.

5358</p>, <p>617-373-5121</p><f>Fax</f>. In CEM, because of the small number of *fax* tags seen (28 at most), the model for the *fax* tag is the closest to an untrained model: it is the least biased against apparently random sequences. The range of characters seen in the *fax* tag is narrow, but not significantly narrower than *phone* tag. This results in errors such as: *<f>REAL</f>basic*, *<f>pp. 43-45</f>*, and *Unix <f>ht://Di</f>g search*.

As predicted in Section 4.5, CEM with Viterbi search performed differently from CEM with Teahan search. With the ability of Teahan search to 'see' long distances it might have been expected to correctly classify phone and fax numbers, which commonly have the differentiator at the end. Unfortunately the numeric content of these tags, being effectively random digits, has high entropy which limited the gains made here. The clearest improvements were situations such as *(703) 306-0599 Fax* which maximum lookahead search broke in two as: *<p>(703) 306-0599</p><f>Fax</f>*, whereas Teahan search correctly marked-up as *<f>(703) 306-0599 Fax</f>*.

Wen [144] expresses accuracy in terms of recall, precision and error rates for each type of tag, as shown in Table 6.2. The Wen model is trained on 25 documents, whereas this thesis uses leave-one-out cross-validation for the Computists' corpus. The apparent reason for the better performance of Teahan search in this case is that many of the ambiguities are of type (a) rather than type (b), as shown in Figure 4.8. The values in Table 6.1 bear no direct relationship with those in Table 6.2 because the former are at the word (or character) level, whereas the latter is the recall and precision of whole tags (excluding the issue tag).

name	pages	date	volume	number	title	journal	booktitle	publisher	address	bibliography	count character
97.80					1.46	+	+	+	+	+	626,917
	98.79		+		+		+			+	140,260
+	+	**97.32**	+		1.07	+	+	+	+	+	161,128
+	+	+	**95.32**	+	1.05	+	+	+	+	2.99	31,182
	+	+	1.32	**96.54**	+		+		+	1.03	11,321
1.47		+			**92.58**	+	3.84	+	+	1.53	1,510,757
+		+	+		1.31	**94.55**	+	+	+	3.17	276,285
+		+	+		7.54	+	**88.61**	+	+	2.83	272,458
+	+	+	+		3.45	1.72	+	**82.71**	3.82	6.93	98,670
+	+	+	+		2.32	+	1.62	3.92	**86.77**	3.99	101,187
0.61	+	+	+	+	2.31	6.55	2.56	1.37	1.01	**84.68**	937,928

Table 6.3: Confusion matrix for the bibliography corpus without *note*. Counts are in characters, all other values are in percent, a '+' indicates a figure lower than 0.99%. Order 6 models trained on 6000 documents and tested on 1000 documents with a lookahead of 5.

6.2.3 Bibliography Corpus

Because the bibliography corpus was developed in the present study, there is not a wide range of results from other systems to compare the results from CEM against. Wen [144] gives some results on three tags (*publisher*, *date* and *pages*) from an early version of the corpus, but these results are not sufficiently detailed for comparison.

Table 6.3 shows the confusion matrix for a large number of tags in the bibliography corpus. A significant number of the errors were caused by use of the note field in BIBTEX. This field allows arbitrary text to be inserted at the end of an entry. Often this extra text is an abbreviated reference (for example: *(Published version of*

UWCS Tech. Report No. 226., 1974)), information which should ideally be in other fields of the reference (such as *Lecture Notes in Computer Science 866* should be in the series and number fields) or a citation (such as *Erratum in it JPL 25:5, 2000, pp. 541–542.*). In Table 6.3 the note tags were stripped prior to tagging, the text previously included in them appeared at the document level, polluting the trained model by adding noise.

The root of these errors is that the generation of the corpus (and all BIBTEX processing) assumes that the BIBTEX file format is prescriptive, when in fact it is descriptive: users will put whatever they need to into a BIBTEX file to get the entry to look 'right' in the style they are using. This leads to a situation in which the meaning of bibliographic entries (when formatted for publication) is clear to researchers and librarians passingly familiar with the field, but the content of the BIBTEX fields does not correspond to field definitions. No increase in lookahead, training data or model order can remedy such a problem.

A different kind of error is seen at the boundary between the author list and the document title because of the wide variation in layout of the author list and the tendency of titles to start with lengthy proper nouns which are easily mistaken for author names. The first word or two of the title are sometimes tagged as author names, either as part of the last genuine author name or as a separate name. This kind of error is strongly linked to the lookahead (see section 6.4.4): as more context is taken into account these errors diminish.

Table 6.4 shows a confusion matrix with the *note* tag added. The overall performance is not substantially different, but that for the *number* tag drops considerably. This appears to be because many of the *note* tags contained numeric sequences (see examples above) and separating *note* tags out from the background model enables it to effectively model numbers.

Table 6.5 shows the type confusion matrix for the bibliography corpus. The *bibliography* tag is still the document tag, but almost all the content is now with

name	pages	date	volume	number	title	journal	booktitle	publisher	address	note	bibliography	count character
97.75			+		1.14	+	+	+		+	+	67807
	98.32				+					+	+	13245
+		**96.40**		+	1.41	0.05	1.16	+		+	+	15275
		+	**96.09**	+	1.17	0.09		+		+	1.63	3426
			1.22	**87.13**					1.82	2.84	6.99	987
2.20		+	+	+	**91.89**	+	3.84	+	+	+	+	150900
			+		1.08	**94.30**	+	1.18		1.22	2.05	29738
+	+	+			5.42	+	**90.90**	+	+	+	1.49	27407
1.14					4.96	1.89	1.12	**83.20**	2.11	1.90	3.69	11178
+		+	+		1.78	+	1.39	6.82	**85.14**	1.38	1.95	12715
+	+	+			2.43	+		1.55	+	**93.40**	1.17	16476
+	+	+	+		1.57	3.12	3.54	1.49	+	+	**87.49**	82518

Table 6.4: Confusion matrix for the bibliography corpus with *note*. Counts are in characters, all other values are in percentage, a '+' indicates a figure lower than 0.99%. Order 6 models trained on 6000 documents and tested on 100 documents with a lookahead of 5.

bibbody tags which contain the bodies of the references (but not the leading reference key in bibliography styles which use one).

Many of the characters mistakenly marked-up as *bibbody* are punctuation (and the *note* tag as explained above), whereas the errors in the *title* column mainly represent the first few words of the *title* confused with the end of the preceding *author* tag. As in Tables 6.3 and 6.4, there is confusion between *title* and *booktitle* because *booktitle* is used in the place of *title* when there are two titles to a document (i.e. a chapter title and a book title, or an article title and a collection title).

There is confusion between the *publisher* and *address* tags because many *publisher* tags have the address of the publisher included within them, especially in entries for *proceedings* and *inproceedings* in which the *address* tag is reserved for the address of the conference rather than the publisher.

In Table 6.4, the *name* from Table 6.3 has been split into five separate tags: *editor*, *author*, *name*, *first* and *last*. There is considerable confusion among the various tags, but surprisingly little difference between the *editor* and name *tags*, because the *name* is almost always immediately following a *bibbody* start tag while an *editor* tag is in the middle of the *bibbody* tag.

Table 6.6 shows the effect of increasing model order—as the model order increases, the experimental result converges with the expected results, the number of defects falling. Placing name tags is particularly challenging because of the diversity in the way names are laid out in the training text.

The results given here appear much better than the figures given for other systems, such as [46]. However, such a direct comparison is at best an approximation because of the different granularity at which the results are measured and the different number of tags. Informal comparison of these results to uncorrected results[1] listed on the Citeseer website[2] suggest that a significantly better determination of

[1] The Citeseer system allows for users to correct or complete bibliographic information. These corrected entries are not considered here.

[2] `http://citeseer.nj.nec.com/cs`

bibbody	editor	author	name	first	last	pages	date	volume	number	title	journal	booktitle	publisher	address	bibliography	count character
79.51	+	+	+	+	+	+	+	+	+	+	7.47	5.17	1.78	1.55		255,064
3.15	**85.11**	2.98	1.94	+	3.84					2.57		+	+	+		12,432
+	+	**92.58**	2.47	+	+					3.86	+		+			26,446
1.46	+	4.17	**89.51**	+	1.82					2.11	+	+	+			20,261
+		+	+	**94.68**	2.87					1.90	+	+	+	+		79,359
+		+	+	1.33	**94.61**	+				3.32	+	+	+	+		149,929
+	+			+	+	**99.09**	+	+		+			+	+		56,241
+				+		+	**97.17**	+	+	1.01	+	1.07	+	+		64,928
2.56						+	+	**95.31**	+	+	+	+	+	+		12,805
+						+	+	+	**97.73**	+		+				4,321
1.25		+	+	+	1.25		+	+		**93.73**	+	2.61	+	0.16		602,944
3.19				+	+	+		+		1.01	**95.00**	+	+	+		118,350
2.99			+	+	+		+			10.17	+	**85.37**	+	+		114,846
9.29	+		+	+	+	+	+	+		4.65	1.53	+	**80.04**	3.20		40,286
3.87	+		+	+	+	+	+	+	+	2.41	+	2.16	2.33	**87.94**		48,225
+				+						+		+			**99.84**	77,247

Table 6.5: Type confusion matrix for the bibliography corpus for many tags. Character counts (#) are in characters, all other values are in percent, a '+' indicates a figure lower than 0.99%. Order 6 models trained on 6000 documents and tested on 300 documents with a lookahead of 5.

Order	Text
0	[5] <name><first>T. </first><last>Matsui,</last> ◇ ◇ <first>T. </first><last>Matsuoka,</last> </name>and <name><first>S.</first> <last>Furui,</last></name> <title>/Smoothed N-best-based speaker adaptation for speech recognition,"◇ in ◇Proc. ICASSP◇*</title>* ’*<pages>*97,*</pages>* (*<journal>*Munich, Germany*</journal>*), pp. <pages>1015–1018,</pages> Apr. *<date>*1997</date>.
1	[5] <name> <first> T. </first> <last> Matsui,</last> <first> T. </first> <last> Matsuoka,</last> </name> and <name> <first> S.</first> <last> Furui,</last> </name> <title> /Smoothed N-best-based speaker adaptation for speech recognition,"◇ in ◇Proc. ICASSP*</title>* ’*<pages>* 97,*</pages>* (*<journal>* Munich, Germany*</journal>*), pp. <pages> 1015–1018,</pages> Apr. <date>1997</date>.
2	[5] <name> <first> T.</first> <last> Matsui,</last> </name> <name> <first> T.</first> <last> Matsuoka,</last> </name> and <name> <first> S.</first><last> Furui,</last> </name> <title> /Smoothed N-best-based speaker adaptation for speech recognition,"</title> in <booktitle> Proc. ICASSP ’97◇, (◇Munich, Germany◇),*</booktitle>* pp. <pages> 1015–1018,</pages> <date>Apr. 1997</date>.
3	[5] <name> <first> T.</first>◇ ◇ <last> Matsui,</last> <first> T.</first><last> Matsuoka,</last> </name> and <name> <first> S.</first> <last>Furui,</last> </name> <title> /Smoothed N-best-based speaker adaptation for speech recognition,"</title> in <booktitle> Proc. ICASSP ’97,</booktitle>(<address> Munich, Germany</address>), pp. <pages> 1015–1018,</pages><date> Apr. 1997</date>.
4	[5] <name> <first> T.</first> ◇ ◇<last> Matsui,</last> <first> T.</first><last> Matsuoka,</last> </name> and <name> <first> S.</first> <last>Furui,</last> </name> <title> /Smoothed N-best-based speaker adaptation for speech recognition,"</title> in <booktitle> Proc. ICASSP ’97,</booktitle>(<address> Munich, Germany</address>), pp. <pages> 1015–1018,</pages><date> Apr. 1997</date>.
5	[5] <name> <first> T.</first> <last> Matsui,</last> </name> <name> <first> T.</first> <last> Matsuoka,</last> </name> and <name> <first> S.</first><last> Furui,</last> </name> <title> /Smoothed N-best-based speaker adaptation for speech recognition,"</title> in <booktitle> Proc. ICASSP ’97,</booktitle> (<address> Munich, Germany</address>), pp. <pages> 1015–1018,</pages><date> Apr. 1997</date>.
Expected	[5] <name> <first> T.</first> <last> Matsui,</last> </name> <name> <first> T.</first> <last> Matsuoka,</last> </name> and <name> <first> S.</first><last> Furui,</last> </name> <title> /Smoothed N-best-based speaker adaptation for speech recognition,"</title> in <booktitle> Proc. ICASSP ’97,</booktitle>(<address> Munich, Germany</address>), pp. <pages> 1015–1018</pages> , <date> Apr. 1997</date>.

Table 6.6: Example of effect of model size on defects, using models trained on 4000 documents and a lookahead of 5. Tags in *italics* are incorrectly placed. ◇ indicates a missing tag.

Author	Corpus	Recall	Precision	F-measure	Perfect	Ref.
Peng	People’s Daily & Treebank	74.0	75.1	74.2	Yes	[116]
Ponte & Croft	People’s Daily & Xinhua	93.6	96.0	94.8	Yes	[117]
Ponte & Croft	People’s Daily & Xinhua	89.8	84.4	87.0	No	[117]
Palmer	TREC-5	—	—	82.7	Yes	[113]
Teahan	Xinhua	93.4	89.6	91.5	No	[137]
CEM/Teahan	ROCLING	97.8	98.1	97.9	No	
CEM/Viterbi	ROCLING	98.2	98.0	98.1	No	

Table 6.7: Performance of Chinese text segmentors. Perfect indicates that the system uses a perfect lexicon.

non-name structures by CEM and similar determination of names by CEM and McCallum’s system described in [47, 75].

6.2.4 Segmentation Corpus

Segmentation of Chinese text is an archetypical segmentation task and there are many published recall and precision figures for this task. Table 6.7 shows a selection of these, together with the best-case results obtained in the present study for CEM on the segmentation corpus described in Section 5.3. Many systems use a perfect lexicon: a list of all words which may be seen during testing and effectively solves the zero frequency problem [146] but prevents the results from being transferred to many real-world problems. The difference between the two Ponte and Croft results[117] in Table 6.7 shows the drop in performance of a system used with and without a perfect lexicon. Production systems typically cannot assume access to a perfect lexicon. There is a relationship between the perfect lexicon and the order -1 (or 0-gram) model in PPM, which includes all characters representable in the character set,

The results from CEM using maximum lookahead search and CEM using Teahan search are similar, with the maximum lookahead search performing marginally better. The Teahan search used 2000 leaves and averaged 5983 nodes per character. The maximum lookahead search used a lookahead of 6 and averaged 4081 nodes per character. Both used an order 3 model trained on 900 documents and 10 testing documents.

Taken at face value, the results for CEM are clearly better than those for the other segmentation systems. However, most of the other systems appear to be assessing recall and precision on the number of whole words rather than on word boundaries, which can double the perceived number of false positives and false negatives for isolated errors. This is because a single segmentation error can cause the words on either side of a boundary both to become false negatives. Another issue is that the data used in the present work was sorted at the sentence level, and it is not clear that this was the case for the other reported results. Data was used in the form it was obtained in, and with no notes on the sorting or otherwise in the literature, no extra processing was performed.

CEM differs from Teahan's TMT system in internal character handling. TMT uses ASCII internally, breaking Unicode characters into multiple characters. Because of the way in which Unicode characters are laid out in the available 32 bits (in 'code pages') there are a number of artifacts, the primary one being that novel Unicode characters are always mapped to novel characters within CEM, escaping back to the order -1 model, but within TMT they may not escape back only as far as the code page. As noted earlier, there is no *a priori* reason for preferring one escape method over another (see Section 3.4) and these results are unlikely to be generalisable beyond Chinese text segmentation.

Because of the large alphabet used in Chinese, the models for even modest orders are large, making the problem significantly more difficult than it would be in a smaller alphabet language such as English. No attempt has been made to optimise the memory usage by CEM models, meaning that it cannot be used to build such large models as Teahan's TMT.

6.2.5 Reuters' Corpus

Figures 6.3 and 6.4 show both recall and precision curves for the entity extraction task in the Reuters' corpus, with training on 7100 documents and testing on 100

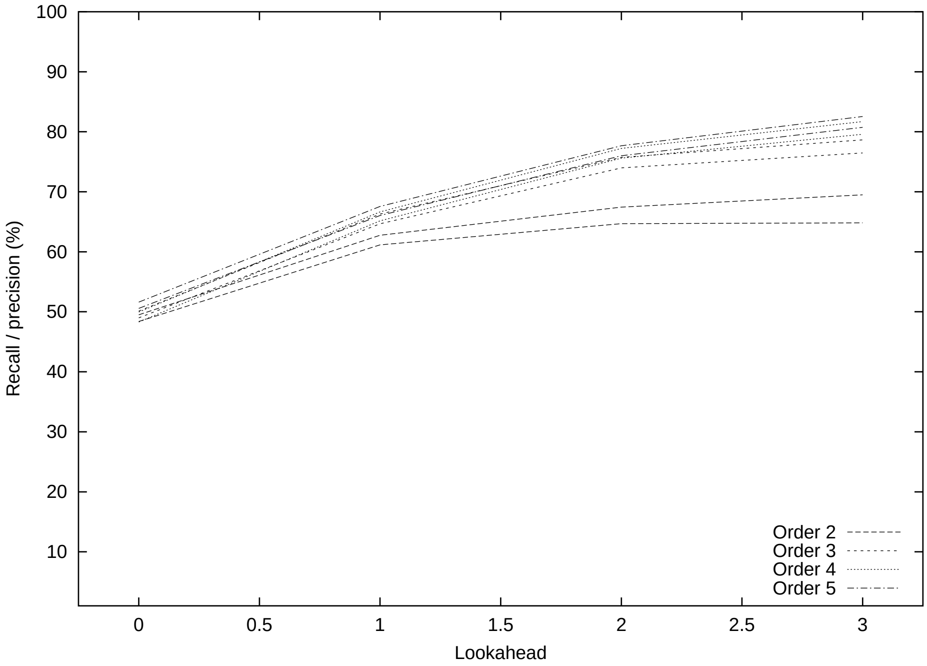


Figure 6.3: Graph of recall and precision against lookahead for various orders of models for documents in the Reuters' corpus.

documents. The difference between Figures 6.3 and 6.4 is granularity, as explained in Section 6.2.1. Figure 6.3 shows recall and precision calculated for each document and then averaged over the testing set. Figure 6.4 shows the recall and precision calculated over the entire testing set. In every case shown, recall and precision are highly correlated and similar.

The difference between Figures 6.3 and 6.4, up to six percent and greatest at low lookaheads, is caused by a number of shorter market-report articles with columns of figures which are easier to tag than are longer articles of a more literary nature. Fortunately, while the results are different, the trends are still clearly the same: incremental gains as the lookahead is increased. Unfortunately the prohibitive size of large models prevented the creation of higher order models.

Overall, the performance of CEM was poor, as state-of-the-art taggers routinely have recall and precision measures in the 90% range [28]. The results are particularly disappointing since the baseline data was generated using a finite-state based

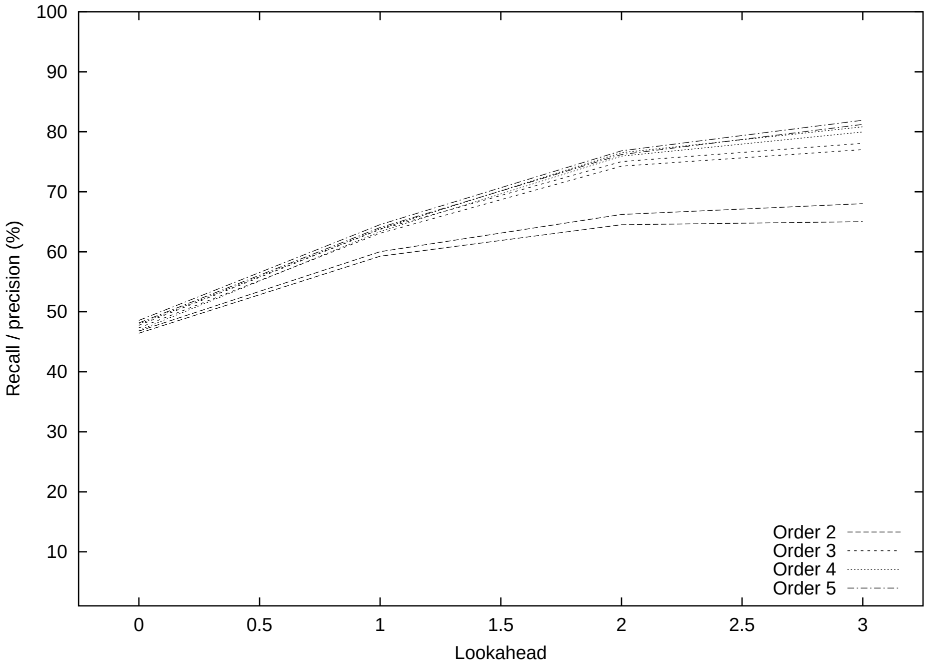


Figure 6.4: Graph of recall and precision against lookahead for various orders of models for the Reuters' corpus taken as a whole.

system (the Brill tagger) which word-level taggers have been able to emulate relatively easily. There are two possible causes. Firstly, whereas the Brill tagger uses a model and search context of a handful of words, CEM uses a model and search context of a handful of characters. Secondly, CEM's linear context and lack of super-adjacency handicapped it against the Brill tagger which uses rule-based post-processing which can examine not just immediate words, but more remote words. Small-scale investigations suggested that increasing model order and lookahead had little effect.

6.3 Baum–Welch Re-estimation

The Baum–Welch algorithm (see Section 3.6) allows untagged data to be used to boost models' performance. This section looks at the application of Baum–Welch re-estimation in the bibliography corpus. This is pertinent, because, as has been

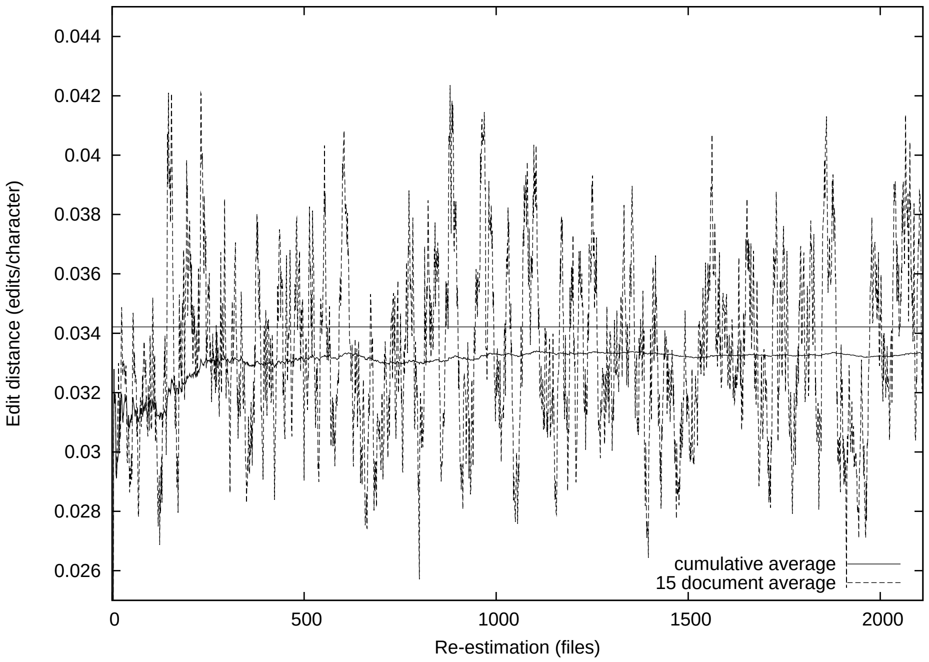


Figure 6.5: Graph of edit distance with increasing re-estimation. Trained with 2110 *abbrv* documents, re-estimated with up to 2111 *acm* documents, using the *first* and *last* tags only, order 4 and lookahead of 3.

pointed out in Sections 5.2 and 6.2.3, the bibliography corpus is significantly less diverse than an uncurated bibliography collection in a digital library and it would be beneficial to be able to generalise the models built on the bibliography corpus to these more diverse collections.

Figure 6.5 shows an attempt to generalise from the *abbrv* bibliography format to the *acm* bibliography format. The *abbrv* format is an abbreviated form with author forenames initialised, while the *acm* format is more standard style which includes the full author forenames, if known. Only the *first* and *last* tags are considered.

As might be expected, a model built on the *abbrv* format and tested on the *acm* format makes many errors. The line across Figure 6.5 at 0.0342 edits per character is the average number of edits over the entire 2111 *acm* documents without any re-estimation. The most common error is the tagging of a *first* tag as a *last* tag, which is seen by the edit distance metric as four separate errors: removing one opening and one closing tag, and adding one opening and one closing tag. A novel error is the misidentification of *eds* (the token indicating the start of an editor list in the *acm* format) as last name.

The 15-document average is a running average of the previous 15 points. It shows a great deal of noise and no obvious pattern of increase or decrease. The cumulative average reaches 0.0323 edits per character after all 1269 documents, a significant drop from the 0.0342 edits per character without re-estimation. Re-estimation clearly reduces the edit distance in this case, lowering the average edit distance for the *acm* documents. EM theory [60] predicts this is not a true convergence (as an increasing proportion of the data is estimated rather than true data, the fidelity of the model slowly falls) but there is insufficient re-estimation data in this example for this to become apparent.

The documents are processed here in random order, but these figures are particularly sensitive to the order in which the documents are processed. The first handful of documents used in the re-estimation appear to be important. It may be worth exploring whether documents should be used ordered in some manner, perhaps those with the lowest mutual-entropy first.

6.4 Effectiveness of Optimisations and Heuristics

The bibliography corpus is a useful dataset for evaluating the effectiveness of optimisations and heuristics because the wide variety of tags in the corpus allows a selection of tags to be examined. The segmentation corpus is also used because it represents a widely-studied problem and a sharp contrast to the bibliography corpus.

6.4.1 Best First

Best first (Section 4.3.2) is an optimisation that exploits the nature of the maximum lookahead search, linking the discrimination of the models to the search space

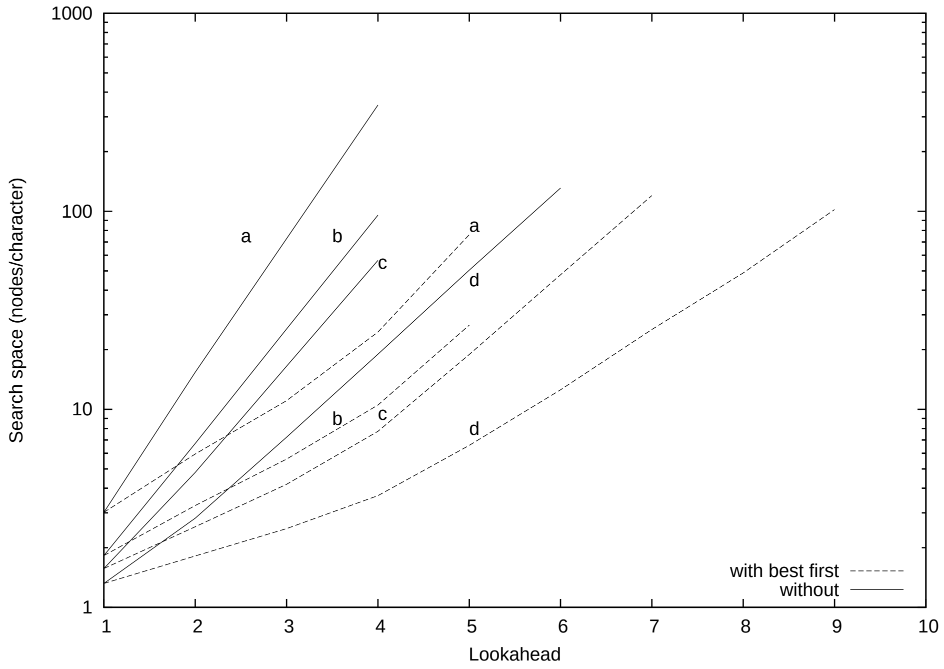


Figure 6.6: Best first optimisation in hierarchical tag insertion. The lines are: a *author*, *editor*, *name*, *first* and *last*; b *name*, *first* and *last*; c *name* and *last*; d *name*. All runs used an order 3 model with 200 training documents and a single testing document.

required to find the lowest entropy tagging of a sequence with respect to that model.

Figure 6.6 shows the effect of the best first optimisation on the hierarchical (nested) tags *author*, *editor*, *name*, *first* and *last* in the bibliography corpus. In all cases where the lookahead is > 1, the search space was significantly reduced. The effect was greatest with the largest number of tags, because as the number of tags increases, the chance that an observed sequence will have low entropy relative to a particular model increases.

Figure 6.7 shows the effect of the best first optimisation on the non-hierarchical tags *name*, *pages*, *date*, *volume* and *number* in the bibliography corpus.

Figure 6.8 shows the effect of the best first optimisation on the *word* tag in the segmentation corpus. Without best first, the order of the model has no impact on the search space. Best first reduces the search space (a versus b), with the effect increasing as the order increases the discrimination of the model (b, c, d, and e).

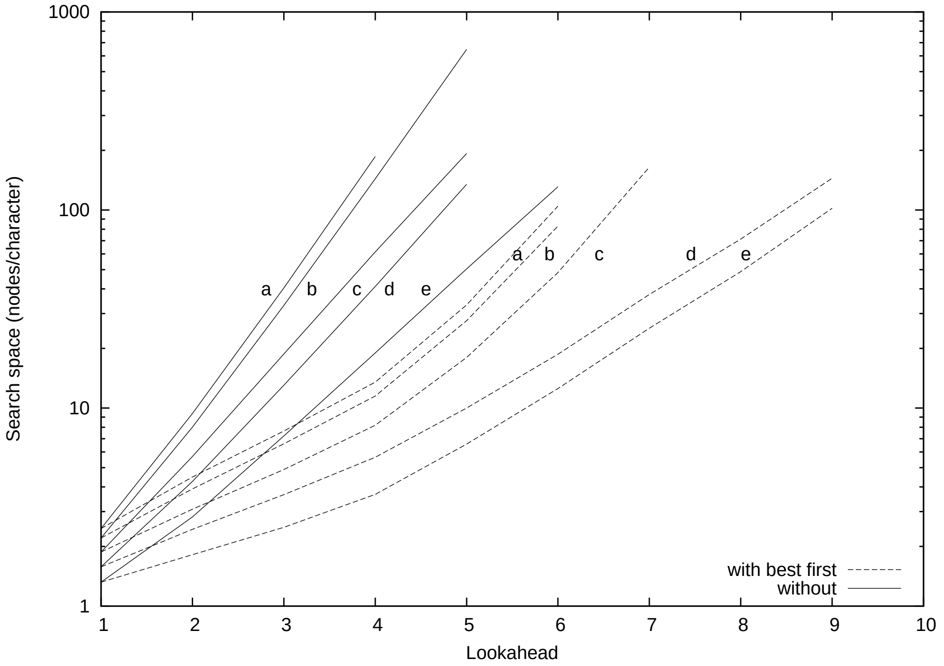


Figure 6.7: Best first optimisation in non-hierarchical tag insertion. The lines are: a *name*, *pages*, *date*, *volume* and *number*; b *name*, *pages*, *date* and *volume*; c *name*, *pages* and *date*; d *name* and *pages*; e *name*. All runs used an order 3 model with 200 training documents and a single testing document.

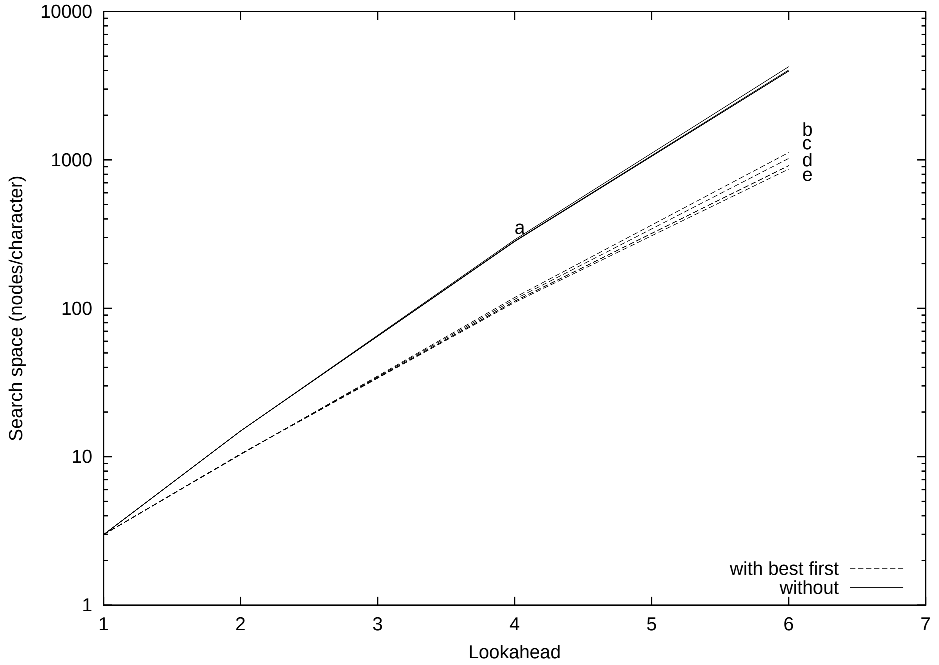


Figure 6.8: The effect of best first on *word* for varying model orders. a labels nearly co-incident quadruple lines representing the search spaces for orders 1, 2, 3 and 4 without best first; b is order 1 with best first; c is order 2 with best first; d is order 3 with best first; e is order 4 with best first. All runs used 900 training documents and a single testing document.

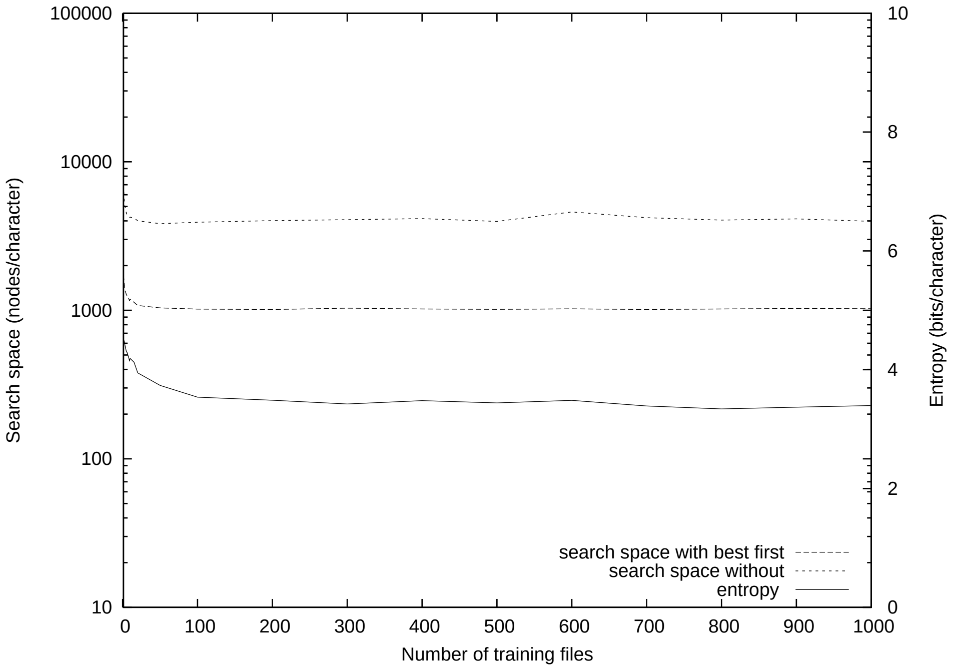


Figure 6.9: Effect of best first when the number of training documents is varied. All runs use order 3 models with a lookahead of 6 and a single testing document from the segmentation corpus. Entropy is the entropy of the entire entire document with respect to the model using for text augmentation, normalised for document length.

The documents in the segmentation corpus are significantly more homogeneous than those in the bibliography corpus, resulting in less noise in their respective graphs.

Figure 6.9 shows how little the effectiveness of the best first increases with the amount of training in the segmentation corpus. Without best first, the search space is independent of the number of documents trained on, but with best first the search space drops. Most of the drop occurred over the first 200 training documents, with relatively little drop over the remaining 799 documents (one document was always withheld for testing).

Figures 6.7, 6.8 and 6.9 each show the results for a single document. This is because while the trends are the same (in all cases best first improves performance and that improvement increases with model order) the size of the improvement varies considerably depending on the problem, and indeed the document, being tackled. In all cases the results are representative of larger-scale experimentation, but averaged results are naturally smoother.

These findings are consistent with the expectations from Section 4.3.2. Well-trained, high-order models allow the probability distribution function to distinguish accurately between likely and unlikely branches, and models with many tags have many more unlikely branches to prune. Given the good performance, the relatively simple implementation and fact that no extra state is required in the model, the best first optimisation is valuable in these tag insertion problems.

6.4.2 Automatic Tokenisation

Automatic tokenisation (see Section 4.3.3) is explored using occurrence tables for illustrative purposes. Table 6.8 shows an occurrence table for the Reuters' corpus after the start and end tags have been converted to special-use characters. In Table 6.8(a) each row contains counts of characters appearing in the corpus belonging to each Unicode character class. Each column contains counts of the character class of the characters immediately following them. In Table 6.8(b) each row contains counts of characters in a Unicode character class that occur immediately prior to a tag (either a start tag or an end tag). Each column contains counts of the class of the character immediately following a tag. An empty cell in Table 6.8(b) indicates that a pair of classes between which a tag has not been seen and which it is reasonable to assume need not be considered for inserting tags. Cells that are empty in Table 6.8(b) but occupied in Table 6.8(a) represent a genuine saving, particularly if the number in the cell in Table 6.8(a) is high, as these are pairs of characters between which the search is not considered inserting tags.

The distinctive cross-shape in Table 6.8(b) is due to the fact that opening tags usually follow a space character and are followed by almost anything, while closing tags can be preceded by almost anything but are followed by a space or '\n' character. This effect is reinforced by the uniform formatting of the corpus. The

First Character		Second Character															Sum
Symbol	#	0	1	2	9	12	15	18	20	21	22	23	24	25	26	27	
UNASSIGNED	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
UPPERCASE LETTER	1	-	85k	223k	1k	-	-	30k	1k	57	2k	2	13k	73	1k	1	360k
LOWERCASE LETTER	2	-	2k	5m	254	-	-	1m	13k	35	4k	-	134k	24	6	-	6m
DECIMAL DIGIT NUMBER	9	-	1k	1k	145k	-	-	66k	5k	36	2k	-	53k	267	52	-	275k
SPACE SEPARATOR	12	-	-	-	-	-	62k	1m	-	-	-	-	-	-	-	-	1m
CONTROL	15	-	-	-	-	-	54	69k	-	-	-	-	-	-	-	-	69k
PRIVATE USE	18	7k	248k	1m	65k	1m	7k	7k	8k	8k	17	-	14k	2k	6k	4	2m
DASH PUNCTUATION	20	-	3k	13k	5k	-	-	6k	4k	5	9	-	27	-	322	-	33k
START PUNCTUATION	21	-	4k	1k	1k	-	-	206	131	-	-	-	92	99	713	-	8k
END PUNCTUATION	22	-	27	6	20	-	-	7k	8	4	1	-	1k	-	2	-	8k
CONNECTOR PUNCTUATION	23	-	-	-	-	-	-	3	-	-	-	119	-	-	-	-	122
OTHER PUNCTUATION	24	-	15k	18k	44k	-	-	139k	141	78	303	1	20k	8	18	-	238k
MATH SYMBOL	25	-	50	19	2k	-	-	475	8	4	1	-	39	16	44	-	2k
CURRENCY SYMBOL	26	-	13	36	9k	-	-	344	-	10	6	-	32	18	-	-	9k
MODIFIER SYMBOL	27	-	-	5	-	-	-	-	-	-	-	-	-	-	-	-	5
Sum		7k	360k	6m	275k	1m	69k	2m	33k	8k	8k	122	238k	2k	9k	5	11m

(a)

First Character		Second Character															Sum
Symbol	#	0	1	2	9	12	15	18	20	21	22	23	24	25	26	27	
UNASSIGNED	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
UPPERCASE LETTER	1	-	-	-	-	30k	-	-	-	-	-	-	-	-	-	-	30k
LOWERCASE LETTER	2	-	-	-	-	1m	-	-	-	-	-	-	-	-	-	-	1m
DECIMAL DIGIT NUMBER	9	-	-	-	-	66k	-	-	-	-	-	-	-	-	-	-	66k
SPACE SEPARATOR	12	-	201k	1m	62k	-	-	-	5k	8k	17	-	6k	2k	6k	4	1m
CONTROL	15	7k	46k	302	3k	3	-	-	3k	406	-	-	7k	89	63	-	69k
PRIVATE USE	18	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DASH PUNCTUATION	20	-	-	-	-	6k	-	-	-	-	-	-	-	-	-	-	6k
START PUNCTUATION	21	-	-	-	-	206	-	-	-	-	-	-	-	-	-	-	206
END PUNCTUATION	22	-	-	-	-	7k	-	-	-	-	-	-	-	-	-	-	7k
CONNECTOR PUNCTUATION	23	-	-	-	-	3	-	-	-	-	-	-	-	-	-	-	3
OTHER PUNCTUATION	24	-	-	-	-	139k	-	-	-	-	-	-	-	-	-	-	139k
MATH SYMBOL	25	-	-	-	-	475	-	-	-	-	-	-	-	-	-	-	475
CURRENCY SYMBOL	26	-	-	-	-	344	-	-	-	-	-	-	-	-	-	-	344
MODIFIER SYMBOL	27	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Sum		7k	248k	1m	65k	1m	-	-	8k	8k	17	-	14k	2k	6k	4	2m

(b)

Table 6.8: Occurrence tables for the Reuters' corpus. (a) Table of all pairs of characters. (b) Table of pairs of characters either side of a tag. 'k' and 'm' indicate units of a thousand and a million respectively.

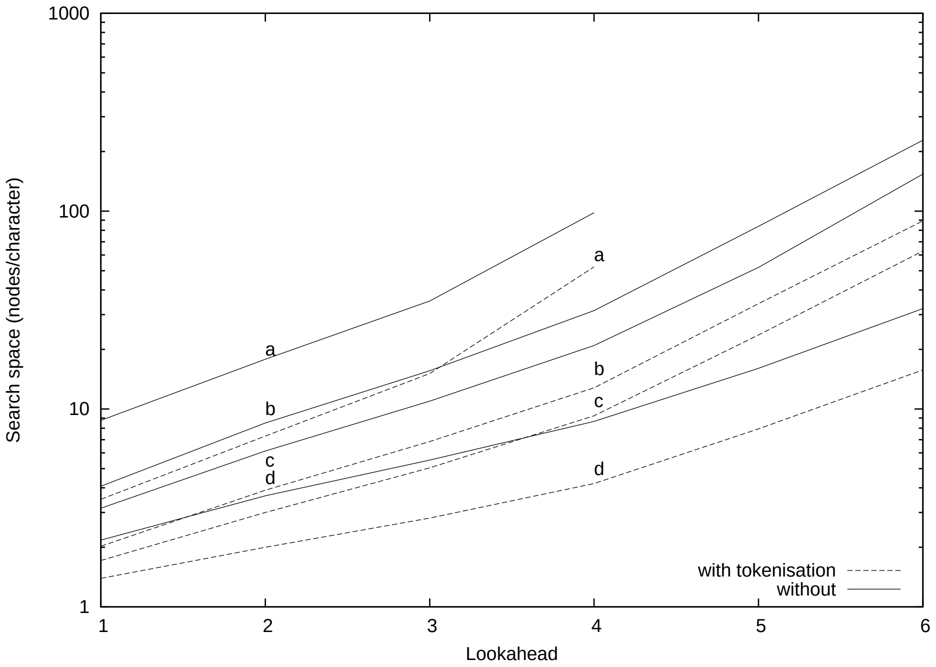


Figure 6.10: Effect of tokenisation on a group of hierarchical tags. The lines are: a *name*, *last*, *first*, *editor* and *author*; b *name*, *last* and *first*; c *name* and *last*; d *name*. Each run was performed with 2000 training documents, one testing document and order 3 models.

CONTROL[3] character class includes '\n', '\r' and EOF.

Figures 6.10 and 6.11 show the effect of tokenisation of hierarchical and non-hierarchical tags in the bibliography corpus. The reason for the differences between hierarchical and non-hierarchical tags is shown in Table 6.9. Table 6.9(a) shows all pairs of characters; Table 6.9(b) shows those either side of the *name* tag, the sparseness of the latter indicating that a procedure such as tokenisation has the potential to make an improvement. The hierarchical tags shown in Table 6.9(c) are similar to the non-hierarchical tags shown in Table 6.9(b), not because they are hierarchical but because they are sequences of case-sensitive characters delimited with spaces, commas and full-stops. The non-hierarchical tags shown in Table 6.9(d) by comparison have a significantly more diverse context. The *date* tag is a sequence of digits and case-sensitive characters and *volume* and *number* tags are strings of dig-

[3] The standard method of writing the names of Unicode characters and character classes is in capitals.

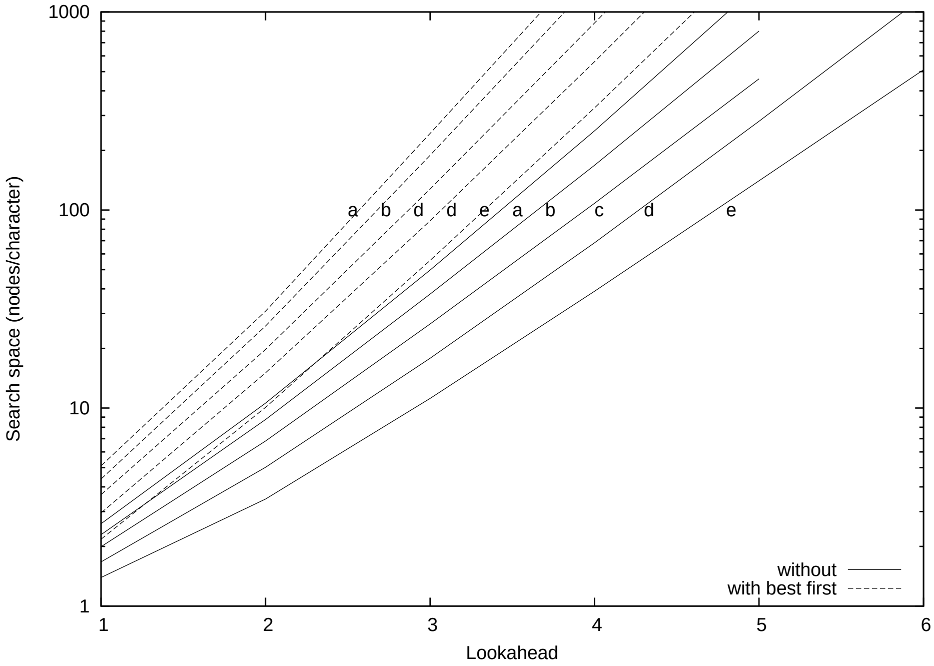


Figure 6.11: Effect of tokenisation on a group of non-hierarchical tags. The lines are: a *name*, *pages*, *date*, *volume* and *number*; b *name*, *pages*, *date* and *volume*; c *name*, *pages* and *date*; d *name* and *pages*; e *name*. Each run was performed with 2000 training documents, one testing document and order 3 models.

its commonly delimited by brackets and semicolons. The resultant occurrence table is much less sparse than the previous table.

Tokenisation potentially interacts with other errors. For example, in Table 6.10 some errors on the bibliography corpus result from problems finding the boundary between the author list and the title tag. In this example, *Athena*, the first word of the article title, has been split in two. The string *Athen* has a slightly lower entropy in the last tag than in the title tag, but *a:* has never been seen in the last tag. The *a:* has not been seen when the decision is taken whether or not to start the tag name tag, so the word is split in two.

Whether the first or the second error is preferable will probably depend on the application. As lookahead gets longer, such errors are greatly reduced, but the proper nouns commonly found at the start of titles are often long words (particularly corporate, place and personal names transliterated into English) and remain problematic even at long lookaheads.

Of 100 differences in correctness examined in the bibliography corpus, using the experimental scenario from Figure 6.10 but using 500 testing documents, 98 were errors of the type shown by Table 6.10. Both the tokenisation and non-tokenisation results were incorrect but the non-tokenisation results recovered more quickly. The remaining were situations in which every tag occurred between rare pairs of character classes.

The appearance of tags between novel or rare pairs of character classes could be guarded against by also inserting tags between character classes seen fewer times than a separate threshold (of the order of 25). In all cases examined this would have solved the problem. If the training corpora is representative, this should have little effect on the search space.

Table 6.11(a) and (b) show the occurrence tables for the Computists' corpus and all the tags within it. Table 6.11(b) is significantly less sparse than Table 6.8(a). However, the frequently-occurring alpha-numeric pairs in the upper left corner are

First Character		Second Character															Sum
Symbol	#	0	1	2	9	12	15	18	20	21	22	23	24	25	26	27	
UNASSIGNED	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
UPPERCASE LETTER	1	-	100k	375k	5k	26k	2k	127	2k	763	1k	-	132k	362	2	19	649k
LOWERCASE LETTER	2	-	2k	3m	4k	479k	60k	10k	16k	1k	5k	1	233k	1k	-	57	4m
DECIMAL DIGIT NUMBER	9	243	288	3k	384k	13k	4k	2	1k	56k	81k	-	86k	88	-	1	632k
SPACE SEPARATOR	12	-	382k	367k	98k	37k	-	84k	225	20k	756	9	6k	825	1	103	1m
CONTROL	15	1k	34k	43k	17k	2	13k	1k	9	50k	3	-	359	47	1	14	162k
PRIVATE USE	18	-	87k	469	6	83k	4k	3	-	1	-	1	70	17	-	1	176k
DASH PUNCTUATION	20	-	7k	11k	2k	277	556	-	-	2	14	-	49	-	1	-	22k
START PUNCTUATION	21	-	26k	2k	96k	128	2k	62	5	9	4	-	1k	15	-	1	130k
END PUNCTUATION	22	-	9	75	17	49k	143	3	112	26	142	-	40k	24	1	-	90k
CONNECTOR PUNCTUATION	23	-	6	5	-	-	-	-	-	-	-	-	-	-	-	-	11
OTHER PUNCTUATION	24	-	6k	8k	23k	309k	76k	75k	1k	202	1k	-	27k	32	-	1	529k
MATH SYMBOL	25	-	622	589	282	747	152	-	36	6	24	-	81	579	-	2	3k
CURRENCY SYMBOL	26	-	-	1	4	-	-	-	-	-	-	-	1	-	-	-	6
MODIFIER SYMBOL	27	-	66	104	13	5	-	-	2	-	1	-	8	-	-	-	199
Sum		2k	649k	4m	632k	1m	164k	172k	22k	130k	90k	11	529k	3k	6	199	7m

(a)

First Character		Second Character															Sum
Symbol	#	0	1	2	9	12	15	18	20	21	22	23	24	25	26	27	
UNASSIGNED	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
UPPERCASE LETTER	1	-	-	-	-	124	-	-	-	-	-	-	3	-	-	-	127
LOWERCASE LETTER	2	-	-	2	-	10k	44	-	-	-	-	-	5	-	-	-	10k
DECIMAL DIGIT NUMBER	9	-	-	-	-	2	-	-	-	-	-	-	-	-	-	-	2
SPACE SEPARATOR	12	-	84k	454	2	-	-	-	-	1	-	1	59	17	-	1	84k
CONTROL	15	-	1k	7	2	-	-	-	-	-	-	-	2	-	-	-	1k
PRIVATE USE	18	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DASH PUNCTUATION	20	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
START PUNCTUATION	21	-	62	-	-	-	-	-	-	-	-	-	-	-	-	-	62
END PUNCTUATION	22	-	-	-	-	3	-	-	-	-	-	-	-	-	-	-	3
CONNECTOR PUNCTUATION	23	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
OTHER PUNCTUATION	24	-	1	-	-	73k	2k	-	-	-	-	-	-	-	-	-	75k
MATH SYMBOL	25	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
CURRENCY SYMBOL	26	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
MODIFIER SYMBOL	27	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Sum		-	85k	463	4	83k	2k	-	-	1	-	1	69	17	-	1	172k

(b)

First Character		Second Character															Sum
Symbol	#	0	1	2	9	12	15	18	20	21	22	23	24	25	26	27	
UNASSIGNED	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
UPPERCASE_LETTER	1	-	-	-	-	192	2	-	-	-	-	-	4	-	-	-	198
LOWERCASE_LETTER	2	-	2	2	-	16k	208	-	-	-	-	-	168	-	-	-	17k
DECIMAL_DIGIT_NUMBER	9	-	-	-	-	3	-	-	-	-	-	-	-	-	-	-	3
SPACE_SEPARATOR	12	-	168k	659	2	-	-	-	-	5	-	4	93	37	-	-	169k
CONTROL	15	-	2k	6	2	-	-	-	-	1	-	-	1	-	-	-	2k
PRIVATE_USE	18	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DASH_PUNCTUATION	20	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
START_PUNCTUATION	21	-	56	-	-	-	-	-	-	-	-	-	-	-	-	-	56
END_PUNCTUATION	22	-	-	-	-	32	1	-	-	-	-	-	-	-	-	-	33
CONNECTOR_PUNCTUATION	23	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
OTHER_PUNCTUATION	24	-	1	-	-	152k	3k	-	-	-	56	-	-	-	-	-	156k
MATH_SYMBOL	25	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	1
CURRENCY_SYMBOL	26	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
MODIFIER_SYMBOL	27	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Sum		-	171k	667	4	169k	3k	-	-	6	56	4	266	37	-	-	345k

(c)

First Character		Second Character															Sum
Symbol	#	0	1	2	9	12	15	18	20	21	22	23	24	25	26	27	
UNASSIGNED	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
UPPERCASE_LETTER	1	-	-	-	-	143	1	-	-	4	5	-	35	-	-	-	188
LOWERCASE_LETTER	2	-	2	3	-	10k	55	-	-	-	619	-	290	-	-	-	11k
DECIMAL_DIGIT_NUMBER	9	-	-	-	-	8k	784	-	-	11k	33k	-	44k	-	-	-	99k
SPACE_SEPARATOR	12	-	97k	518	57k	-	-	-	-	3	-	-	708	20	-	2	156k
CONTROL	15	-	2k	23	6k	-	-	-	-	-	-	-	55	-	-	-	9k
PRIVATE_USE	18	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DASH_PUNCTUATION	20	-	-	-	-	-	-	-	-	-	1	-	2	-	-	-	3
START_PUNCTUATION	21	-	4k	6	29k	-	-	-	1	1	-	-	247	-	-	-	33k
END_PUNCTUATION	22	-	-	-	-	42	5	-	-	-	72	-	37	-	-	-	156
CONNECTOR_PUNCTUATION	23	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
OTHER_PUNCTUATION	24	-	58	11	20k	101k	5k	-	-	8	31	-	38	-	-	-	127k
MATH_SYMBOL	25	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
CURRENCY_SYMBOL	26	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
MODIFIER_SYMBOL	27	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

(d)

Table 6.9: Occurrence tables for the bibliography corpus. (a) Table of all pairs of characters. (b) Table of pairs of characters either side of a *name* tag. (c) Table of pairs of characters either side of *name*, *last*, *first*, *editor* and *author* tags. (d) Table of pairs of characters either side of *name*, *pages*, *date*, *volume* and *number* tags.

Case	Text
0	[Son] D. Song. Athena: A new efficient automatic checker for security protocol analysis.
1	[Son] <name> <first> D.</first> <last> Song.</last> </name>-<name> <last> Athena:</last> </name> <title> A new efficient automatic checker for security protocol analysis.</title>
2	[Son] <name> <first> D.</first> <last> Song.</last> </name>-<name> <last> Athen</last> </name> <title> a: A new efficient automatic checker for security protocol analysis.</title>

Table 6.10: Interaction between errors. The unmarked-up text (0), the text with a markup error (1) and with the first error confounded by a second error which splits a word in two (2).

First Character		Second Character														Sum
Symbol	#	0	1	2	9	12	15	18	20	21	22	23	24	25	26	
UNASSIGNED	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
UPPERCASE LETTER	1	-	5k	10k	55	1k	39	1k	174	-	110	4	971	27	-	20k
LOWERCASE LETTER	2	-	451	198k	974	33k	1k	2k	896	-	116	40	10k	9	-	249k
DECIMAL DIGIT NUMBER	9	-	1k	52	4k	339	50	1k	234	-	88	1	746	359	-	8k
SPACE SEPARATOR	12	-	8k	29k	1k	8k	2k	2k	163	1k	-	1	418	609	-	54k
CONTROL	15	-	897	2k	42	1k	2k	679	73	441	-	71	178	317	-	8k
PRIVATE USE	18	36	2k	1k	904	1k	179	36	31	51	356	-	2k	1k	275	10k
DASH PUNCTUATION	20	-	215	864	201	225	23	44	233	-	5	-	3	-	-	1k
START PUNCTUATION	21	-	616	171	67	1	-	611	-	-	-	-	31	129	-	1k
END PUNCTUATION	22	-	-	-	-	352	828	14	-	-	10	-	435	-	-	1k
CONNECTOR PUNCTUATION	23	-	10	32	2	-	72	-	-	-	-	4k	1	-	-	4k
OTHER PUNCTUATION	24	-	607	4k	350	7k	1k	303	8	-	948	-	1k	35	-	18k
MATH SYMBOL	25	-	3	41	18	427	30	1k	1	-	6	-	966	190	-	2k
CURRENCY SYMBOL	26	-	-	-	275	-	-	-	-	-	-	-	-	-	-	275
Sum		36	20k	249k	8k	54k	8k	10k	1k	1k	1k	4k	18k	2k	275	383k

(a)

First Character		Second Character														Sum
Symbol	#	0	1	2	9	12	15	18	20	21	22	23	24	25	26	
UNASSIGNED	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
UPPERCASE LETTER	1	-	-	-	-	308	48	-	16	-	179	-	603	6	-	1k
LOWERCASE LETTER	2	-	-	35	-	750	48	-	7	-	164	-	880	908	-	2k
DECIMAL DIGIT NUMBER	9	-	-	-	-	135	42	-	8	-	13	-	924	18	-	1k
SPACE SEPARATOR	12	-	1k	397	804	-	-	-	-	38	-	-	4	8	230	2k
CONTROL	15	36	478	61	62	-	-	-	-	13	-	-	5	3	21	679
PRIVATE USE	18	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DASH PUNCTUATION	20	-	3	4	19	-	-	-	-	-	-	-	-	-	18	44
START PUNCTUATION	21	-	543	45	17	-	-	-	-	-	-	-	-	-	6	611
END PUNCTUATION	22	-	-	-	-	9	-	-	-	-	-	-	5	-	-	14
CONNECTOR PUNCTUATION	23	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
OTHER PUNCTUATION	24	-	57	26	-	75	5	-	-	-	-	-	21	119	-	303
MATH SYMBOL	25	-	24	1k	2	-	-	-	-	-	-	-	2	-	-	1k
CURRENCY SYMBOL	26	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Sum		36	2k	1k	904	1k	143	-	31	51	356	-	2k	1k	275	10k

(b)

Table 6.11: Occurrence tables for the Computists' corpus. (a) Table of all pairs of characters. (b) Table of pairs of characters either side of a tag.

First Character		Second Character														Sum
Symbol	#	0	1	2	5	9	11	18	20	21	22	23	24	25	26	
UNASSIGNED	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
UPPERCASE LETTER	1	-	3k	2k	470	61	-	1k	1	1	-	18	1	-	1	8k
LOWERCASE LETTER	2	-	146	17k	696	4	-	4k	1	1	1	-	-	-	7	22k
OTHER LETTER	5	-	45	259	1m	524	-	1m	-	56	389	225	-	-	572	3m
DECIMAL DIGIT NUMBER	9	-	18	346	1k	4k	-	1k	2	-	383	262	-	-	62	7k
OTHER NUMBER	11	-	-	-	-	-	-	3	-	-	-	-	-	-	-	3
PRIVATE USE	18	999	3k	1k	1m	2k	1	2m	-	18k	20k	314k	28	9	1k	4m
DASH PUNCTUATION	20	-	-	2	-	2	-	-	-	-	-	-	-	-	-	4
START PUNCTUATION	21	-	-	-	393	383	-	26k	-	1	23	20	-	-	-	27k
END PUNCTUATION	22	-	2	-	49	1	-	21k	-	3	-	4	-	-	1	21k
OTHER PUNCTUATION	24	-	601	487	276k	54	2	28k	-	8k	19	1	5	-	348	315k
MATH SYMBOL	25	-	8	13	-	-	-	13	-	-	-	-	-	-	-	34
CURRENCY SYMBOL	26	-	-	-	-	-	-	9	-	-	-	-	-	-	-	9
OTHER SYMBOL	28	-	2	72	604	107	-	1k	-	13	-	-	-	-	9	2k
Sum		999	8k	22k	3m	7k	3	4m	4	27k	21k	315k	34	9	2k	7m

(a)

First Character		Second Character														Sum
Symbol	#	0	1	2	5	9	11	18	20	21	22	23	24	25	26	
UNASSIGNED	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
UPPERCASE LETTER	1	-	126	15	875	34	-	-	-	46	259	482	4	-	45	1k
LOWERCASE LETTER	2	-	721	616	1k	30	-	-	-	33	1k	509	-	-	98	4k
OTHER LETTER	5	-	1k	347	1m	2k	1	-	-	17k	18k	304k	-	8	1k	1m
DECIMAL DIGIT NUMBER	9	-	29	58	878	22	-	-	-	12	110	246	24	-	26	1k
OTHER NUMBER	11	-	-	-	3	-	-	-	-	-	-	-	-	-	-	3
PRIVATE USE	18	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DASH PUNCTUATION	20	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
START PUNCTUATION	21	-	1k	343	24k	130	-	-	-	92	6	55	-	-	21	26k
END PUNCTUATION	22	-	15	2	12k	44	-	-	-	405	150	8k	-	-	22	21k
OTHER PUNCTUATION	24	998	255	13	26k	67	-	-	-	821	30	221	-	1	21	28k
MATH SYMBOL	25	-	-	-	-	-	-	-	-	-	5	8	-	-	-	13
CURRENCY SYMBOL	26	-	-	-	-	9	-	-	-	-	-	-	-	-	-	9
OTHER SYMBOL	28	1	70	83	1k	25	-	-	-	51	16	318	-	-	87	1k
Sum		999	3k	1k	1m	2k	1	-	-	18k	20k	314k	28	9	1k	2m

(b)

Table 6.12: Occurrence tables for the segmentation corpus. (a) Table of all pairs of characters. (b) Table of pairs of characters either side of a tag.

mainly zero, so the heuristic is of some benefit.

Table 6.12 is the occurrence table for the segmentation corpus and indicates that the OTHER LETTER is by far the most common character class, which is to be expected since most Chinese characters fall into this class. The nature of the corpus means that all of the frequently-occurring pairs in Table 6.12(a) also appear in Table 6.12(b) (as non-zeros), indicating that automatic tokenisation is going to have little effect on the search space in this corpus.

Figure 6.12 shows the interaction between best first and tokenisation for the *name* tag. The addition of tokenisation to best first always reduces the search space, but the effect is most noticeable at low lookaheads when best first is less effective. This is because automatic tokenisation prunes branches of the search tree without having to expand the first node in the branch to calculate the entropy.

Consistent with the expectations from Section 4.3.3, these results show that automatic tokenisation improves performance on some datasets. However, it does not

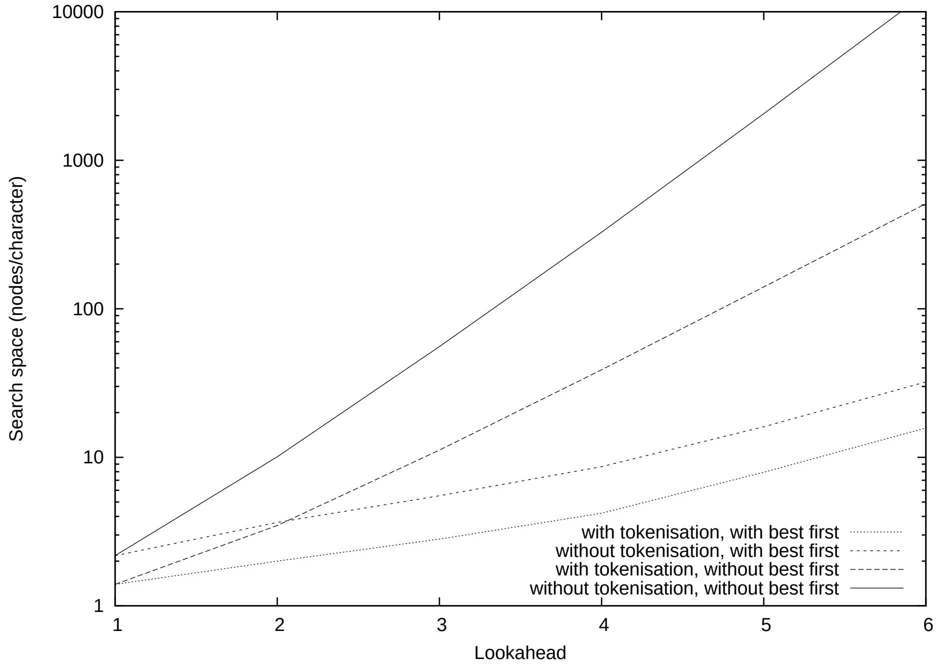


Figure 6.12: Effect of best first and automatic tokenisation on *name* tag. Each run was performed with 2000 training documents, one testing document and order 3 models.

perform consistently well across all datasets, and a number of the corpora have noise in the occurrence tables. Such noise is likely to be significantly greater in digital library collections of heterogeneous documents of diverse origin than in the curated corpora used here. Anecdotal evidence of HTML and XHTML documents from the Internet suggest that tags do occur in a significantly wider variety of places than in the corpora examined here. Automatic tokenisation requires a small and tightly-bounded amount of extra state per model in the form of an occurrence table.

Unlike best first, automatic tokenisation is not linked to the discrimination of the models. This means it can perform well even for a poorly trained model. The reason that automatic tokenisation does not perform as well as the occurrence table method is that the PPM model already discriminates between these situations and that best first ensures that the branches that get pruned by automatic tokenisation are not explored anyway.

Name	Symbol	Example
Null folder	N	*Jones,␣Jill␣K.␣and*
Capitals folder	c	*JONES,␣JILL␣K␣AND*
Case folder	C	*Aaaaa,␣Aaaa␣A.␣aaa*
Unicode folder	u	*AaaaaPSAaaaSAPSaaa*
Vowel folder	V	*nvnvn,␣nvnn␣n.␣vnn*
Vowel & case folder	VC	*Nvnvn,␣Nvnn␣N.␣vnn*

Table 6.13: Folders used in alphabet reduction.

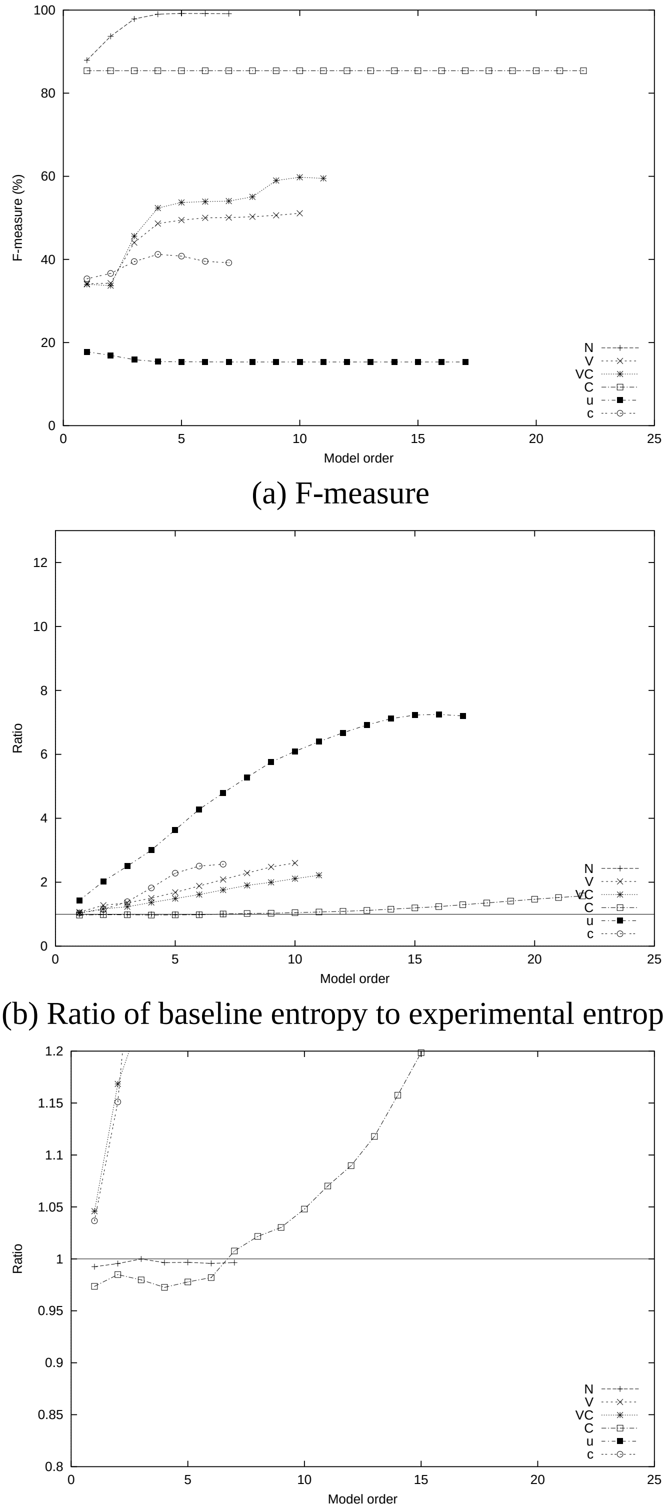
6.4.3 Alphabet Reduction

Table 6.13 shows the six ‘folders’ used in the alphabet reduction experiment. They ‘fold’ the alphabet used in the model, as their effects on a sample string show.

The Null folder does not change the alphabet at all. The Capitals folder removes the distinction between upper and lower case. The Case folder folds all uppercase letters to a single letter and all lowercase letters to a single letter. The Unicode folder folds each of the Unicode character classes (see Section 4.3.3) to a single character per class. The Vowel folder folds all vowels to a single letter and all non-vowels to a single letter. The Vowel and Case folder folds uppercase vowels to a single letter, lowercase vowels to a single letter, uppercase non-vowels to a single letter and lowercase non-vowels to a single letter.

Figure 6.13 shows the results of these six folders on *name* in the bibliography corpus. Figure 6.13(a) shows the F-measure against the order of the model for each of the folders. The experiment was performed in 750 megabytes of heap memory, and the data is shown only for those models and lookaheads which could be built and used in that memory.

The N folder performed best, but N models could only be built to order seven, because of the large alphabet. The C models also performed well and could be built to order 23. However, increasing order did not increase the performance because useful information was thrown away by the folder. The c, V and VC models all performed similarly poorly and could be built to orders between seven and ten. The



(a) F-measure

(b) Ratio of baseline entropy to experimental entropy

(c) Ratio of baseline entropy to experimental entropy (detail)

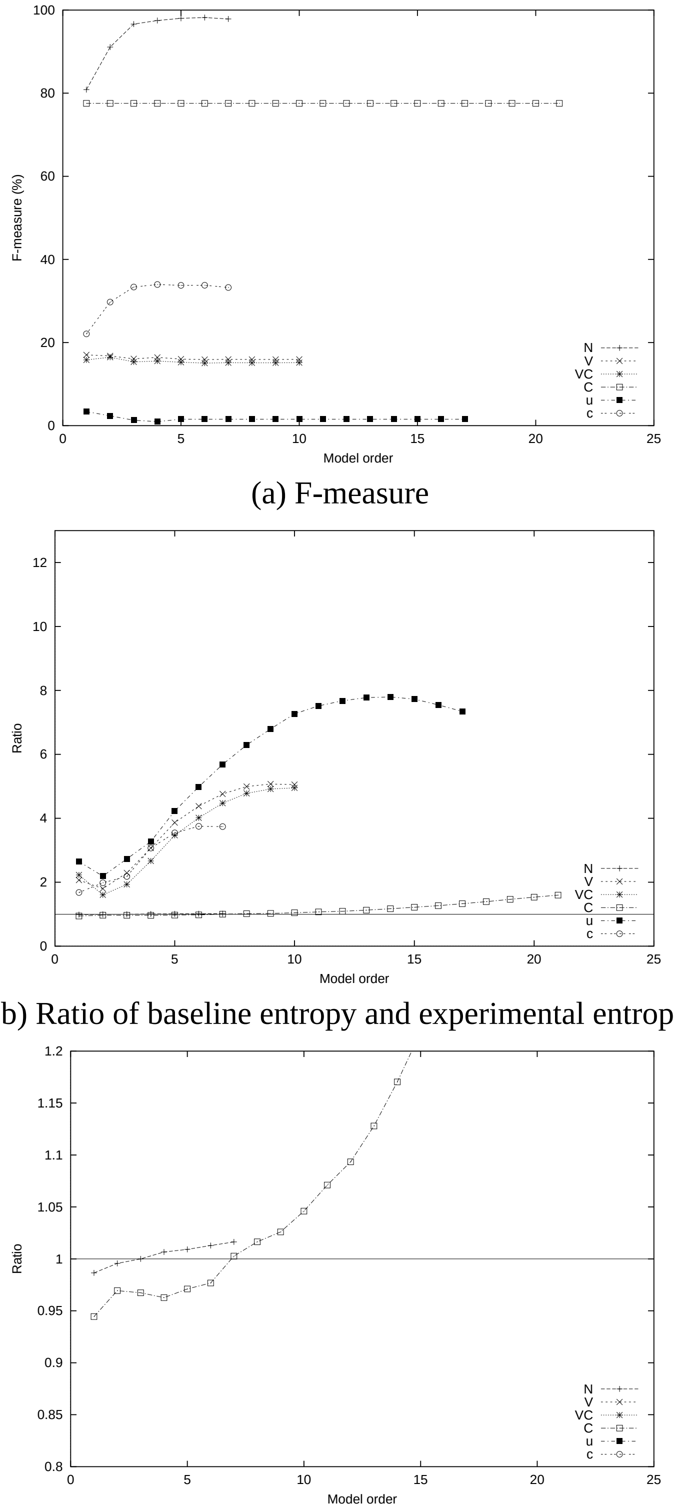
Figure 6.13: The effects of alphabet reduction on finding the *name* tag of bibliography corpus. Lookahead of 8, trained on 2000 documents and tested on 20 documents.

u models performed badly with an F-measure less than twenty despite being able to be built to order 18. This is particularly surprising given that the u folder has a close relationship with the C folder, which performed well. The reason for this difference appears to be that '.' and ',' are important in delimiting names and other features in bibliographies and the u models were unable to distinguish between these characters.

Figure 6.13(b) shows the ratio of baseline to experimental entropy for the same experiments while Figure 6.13(c) shows detail of the same relationship where the ratio approaches one. As discussed previously (see page 83), the entropy can be used to determine whether the model or the search is responsible for a mis-tagging. All data points with a ratio less than one indicate that the search was deficient (i.e. the lookahead could be increased for greater correctness). All data points where the ratio is greater than one indicate that the model is deficient in some regard; in the ideal situation the ratio is 1:1. There are three likely ways in which the model can be deficient: it may have seen insufficient training data, it may be of insufficient order, or it may be failing to capture important features of the data. 2000 training bibliographies (approximately 45,000 bibliographic entries) would appear to be sufficient training data: models with smaller alphabets generally require less training data. Increasing the order of the u, V, c and VC models clearly moves the ratio further from 1:1. Thus the problem is likely to be that these models are not capturing important features of the data.

The upward trend in the entropy ratio for the C models of order higher than 6 (Figure 6.13(c)) is consistent with the behaviour of PPM models when the order is increased beyond optimal. This species of over-fitting is caused by the building of a higher order model than there is training data available to train effectively, leading to many common states having their probabilities generated via the escape method.

The increase in noise for the ratio of entropies (particularly for the u model) as order increases is due to sampling effects.



(a) F-measure

(b) Ratio of baseline entropy and experimental entropy

(c) Ratio of baseline entropy and experimental entropy (detail)

Figure 6.14: The effects of alphabet reduction on finding multiple tags in the bibliography corpus. Lookahead of 4, trained on 2000 documents and tested on 20 documents.

Figure 6.14 shows the same details as Figure 6.13 for tags *name*, *pages*, *date*, *volume* and *number* at a lower lookahead (necessary because of the greatly increased search space caused by the additional tags). Performance in Figure 6.14 was consistently poorer than that in Figure 6.13, but the relative performance of the folders was similar. The one deviation from this is the c folder, whose F-measure is similar to the VC and V folders in Figure 6.13, but clearly superior in Figure 6.14. This is because the *pages*, *date*, *volume* and *number* tags in Figure 6.14 are number-centric rather than text-centric, so the loss of capitalisation does not effect them as badly.

The large reductions in correctness shown in Figures 6.13 and 6.14 strongly suggest that, with the possible exception of C, alphabet reduction is unlikely to be useful in production systems for such corpora.

6.4.4 Maximum Lookahead Heuristic

For the majority of tag-insertion problems, maximum lookahead is problematic because the lookahead at which the accuracy becomes asymptotic is computationally infeasible. For problems with a small number of tags, maximum lookahead is obtainable. Table 6.14 shows the effect of various lookahead values on a single bibliographic entry. The result converges on the expected text within a lookahead of 5, much shorter than the maximum tag length of ~ 60 which Viterbi search suggests would be required.

The defects displayed in Table 6.14 are mainly of types already discussed in Section 6.2.3: confusion caused by the wide variety of name formats and confusion between article titles and book titles. Similar defects were also seen in Table 6.6, in which the same reference was used to examine the performance with varying model orders. However, as shown in Figure 6.15, there is often a great deal of noise, and it may not be clear whether the asymptote has been reached or whether the lookahead must be increased.

The primary sources of errors when inserting the *pages* tag were four-digit page

lookahead	text
1	[5] <name> <first> T.</first> <last> Matsui,</last> </name> ⋄ ⋄*<title>* T.⋄ ⋄ *</title>* *<journal>*⋄ ⋄ Matsuoka,*</journal>* and <name> <first> S.</first> <last> Furui,</last> </name> <title> /N-best-based speaker adaptation for speech recognition,"</title> in <booktitle> Proc. ICASSP ’97,</booktitle> (⋄*<name>* *<first>* Munich,*</first>* *</name>* *<title>* Germany ⋄),*</title>* pp. <pages> 1015–1018,</pages> <date> Apr. 1997</date>.
2	[5] <name> <first> T.</first><last> Matsui,</last> ⋄ ⋄ <first> T.</first> <last> Matsuoka,</last> </name> and <name> <first> S.</first> <last> Furui,</last> </name> <title> /N-best-based speaker adaptation for speech recognition,"</title> in <booktitle> Proc. ICASSP ’97,</booktitle> (<address> Munich, Germany</address>), pp. <pages> 1015–1018,</pages> <date> Apr. 1997</date>.
3	[5] <name> <first> T.</first><last> Matsui,</last>⋄ ⋄ <first> T.</first> <last> Matsuoka,</last> </name> and <name> <first> S.</first> <last> Furui,</last> </name> <title> /N-best-based speaker adaptation for speech recognition,"</title> in <booktitle> Proc. ICASSP ’97,</booktitle> (<address> Munich, Germany</address>), pp. <pages> 1015–1018,</pages> <date> Apr. 1997</date>.
4	[5] <name> <first> T.</first> <last> Matsui,</last>⋄ ⋄<first> T.</first> <last> Matsuoka,</last> </name> and <name> <first> S.</first> <last> Furui,</last> </name> <title> /N-best-based speaker adaptation for speech recognition,"</title> in <booktitle> Proc. ICASSP ’97,</booktitle> (<address> Munich, Germany</address>), pp. <pages> 1015–1018,</pages> <date> Apr. 1997</date>.
5	[5] <name> <first> T.</first> <last> Matsui,</last> </name> <name> <first> T.</first> <last> Matsuoka,</last> </name> and <name> <first> S.</first> <last> Furui,</last> </name> <title> /N-best-based speaker adaptation for speech recognition,"</title> in <booktitle> Proc. ICASSP ’97,</booktitle> (<address> Munich, Germany</address>), pp. <pages> 1015–1018,</pages> <date> Apr. 1997</date>.
baseline	[5] <name> <first> T.</first> <last> Matsui,</last> </name> <name> <first> T.</first> <last> Matsuoka,</last> </name> and <name> <first> S.</first> <last> Furui,</last> </name> <title> /N-best-based speaker adaptation for speech recognition,"</title> in <booktitle> Proc. ICASSP ’97,</booktitle> (<address> Munich, Germany</address>), pp. <pages> 1015–1018,</pages> <date> Apr. 1997</date>.

Table 6.14: Example of effect of lookahead on defects, using order 4 models trained on 4000 documents. Tags in *italics* are incorrectly placed. ⋄ indicates a missing tag.

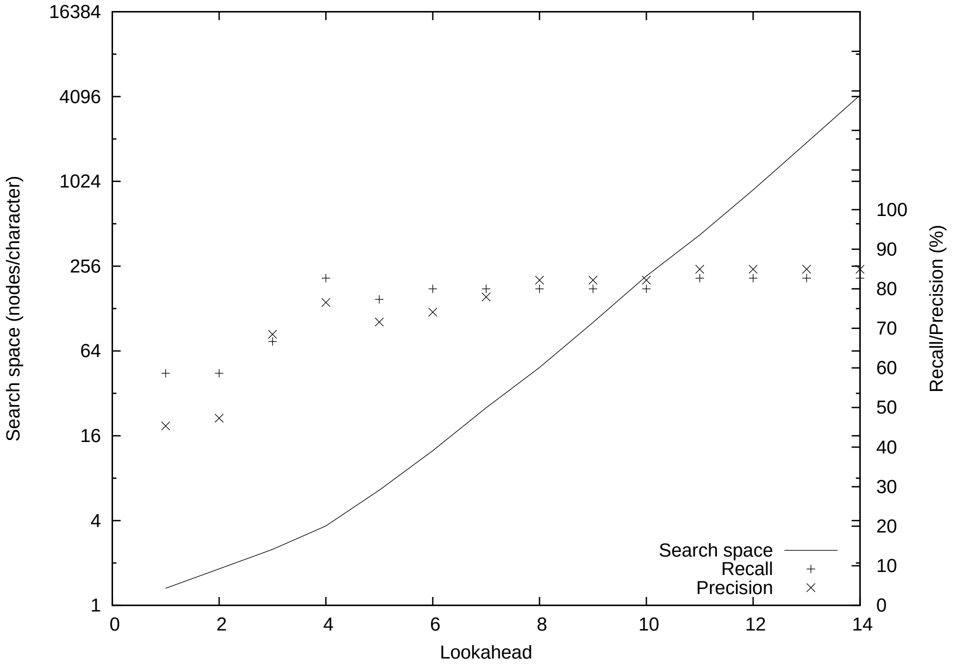


Figure 6.15: Graph of recall, precision and search space against lookahead for the single *name* tag. Models trained on 2000 documents and tested on one document.

numbers that looked like years such as *1993–2002* and features such as *n–n+4*, which is a common format when the citation is taken from an electronic copy and the document length is known but not the location within the larger journal or collection. These sources of noise are compounded by variability in the length of bibliographies, which may be as short as a single entry with only one *pages* tag and only one *name*. These problems are not resolved by increasing the lookahead.

Figure 6.16 shows the same analysis for the *word* tag in the segmentation corpus. The data from this graph (Table 6.15) show that while the search space increased by five orders of magnitude, the recall and precision increased by less than one percent. It is not clear why recall and precision cross-over in Figures 6.15 and 6.16 as lookahead increases, but the levelling-off of increase in recall and precision, indicative and representative of larger samples, suggests that the model does not contain all the information needed to make the underlying relevancy decisions.

These results show that the maximum lookahead heuristic can be effective. In-

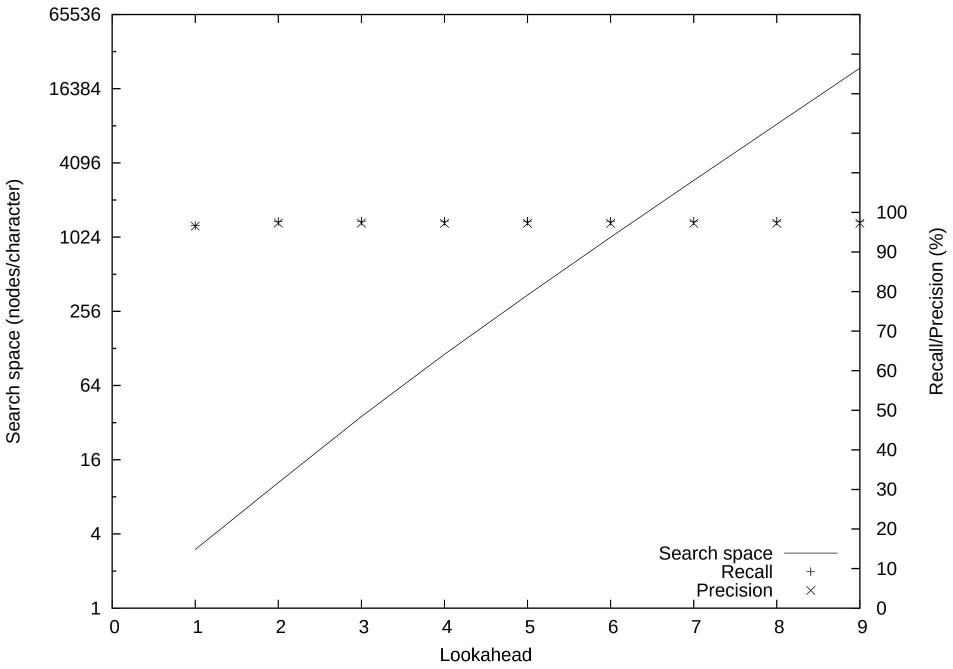


Figure 6.16: Graph of recall, precision and search space against lookahead for the *word* tag. Models trained on 2000 documents and tested on one document.

Lookahead	Search space (nodes per character)	Recall (%)	Precision (%)
1	6.00	97.10	97.37
2	27.26	97.83	97.79
3	86.22	97.82	97.53
4	241.07	97.73	98.21
5	633.54	97.74	98.21
6	1598.50	98.30	98.06
7	3976.08	97.72	97.59
8	9801.47	97.61	98.16
9	23457.08	97.77	97.87
10	58153.64	97.84	98.09
11	139079.05	97.71	98.02

Table 6.15: Table of recall, precision and search space against lookahead for the *word* tag. The data is plotted in Figure 6.16.

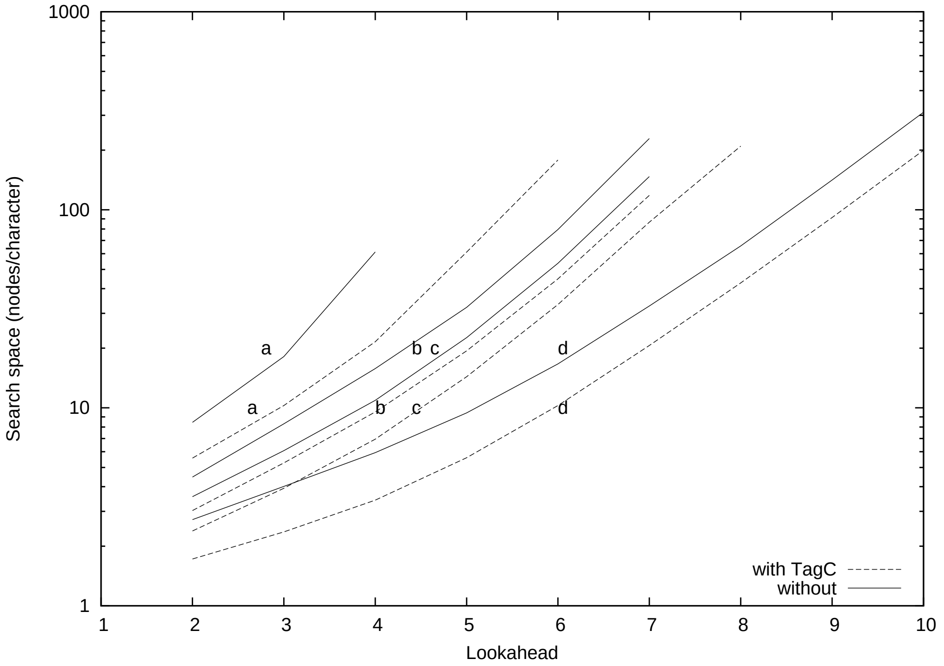


Figure 6.17: TagC heuristic in hierarchical tag insertion. From steepest to shallowest the lines are: (a) *author*, *editor*, *name*, *first* and *last*; (b) *name*, *first* and *last*; (c) *name* and *last*; (d) *name*. All runs used an order 3 model with 200 training documents and a single testing document.

creasing the lookahead beyond six has, in this case, no obvious benefit to recall and precision but is of great detriment to the search space.

6.4.5 TagC Heuristic

The TagC heuristic (Section 4.3.6) limits the number of tags to be considered for insertion between two characters in a document. Figure 6.17 shows the effect of the TagC heuristic on the hierarchical tags *author*, *editor*, *name*, *first* and *last* in the bibliography corpus. In all cases the search space was reduced. Figure 6.18 shows the effect of the TagC heuristic on the non-hierarchical tags *name*, *pages*, *date*, *volume* and *number* in the bibliography corpus.

Results show the TagC heuristic to be consistent and significant. Much of the pruning of the TagC heuristic is similar to that of the best first optimisation. A tag that is ruled out by the TagC heuristic has not been seen in this model before,

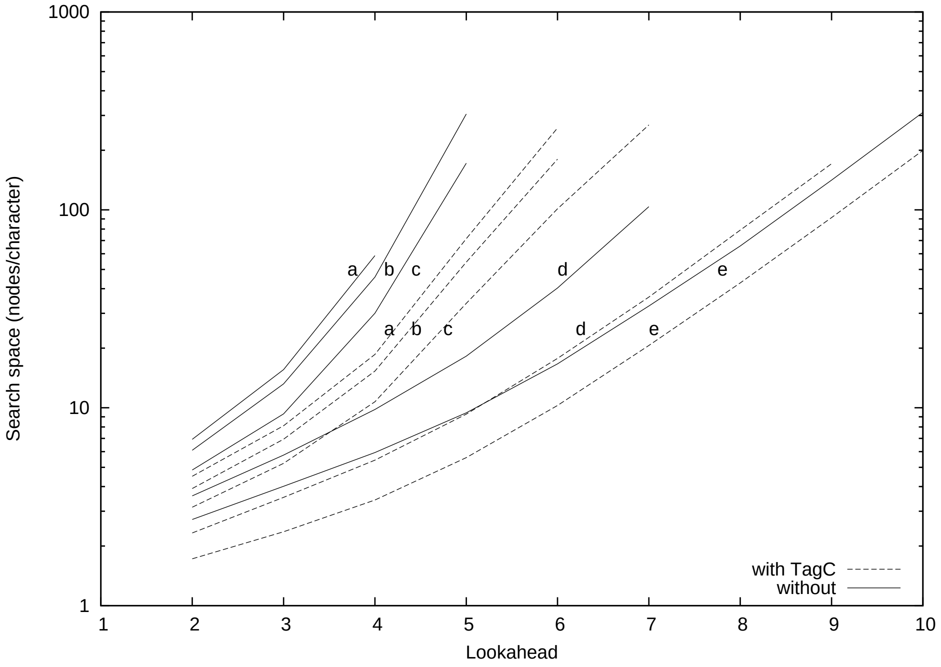


Figure 6.18: TagC optimisation in non-hierarchical tag insertion. From steepest to shallowest the lines are: (a) *name*, *pages*, *date*, *volume* and *number*; (b) *name*, *pages*, *date* and *volume*; (c) *name*, *pages* and *date*; (d) *name* and *pages*; (e) *name*. All runs used an order 3 model with 200 training documents and a single testing document.

meaning the PPM model must escape back to order -1 (see Section 3.4), and implying high entropy transitions. The structure of PPM models means an order a transition can be followed, at most, by an order $a+1$ transition (except for the start of sequence symbol), so an order -1 transition can be penalised over an order n transition for $n+1$ transitions. Many of the tags and tag sequences ruled out by the TagC heuristic would mean three or four order -1 transitions and can be rapidly pruned by the best first under normal circumstances.

The set of observed tag combinations is smaller in the bibliography corpus than it may be in real-world corpora because, when integrating the tagged and untagged bibliographies (see Figure 5.2), placement of tags with respect to inter-word whitespace was performed automatically and therefore consistently. Diverse, real-world, uncurated sources are unlikely to display this degree of consistency.

6.4.6 State Tying

The opportunity to apply the state tying heuristic (see Section 4.3.7) occurred only once in the corpora studied, on the *name* tag which may occur within the *editor* or the *author* tag in the bibliography corpus. The schema for the bibliography dataset with and without state tying are shown in Figure 5.4 and Figure 5.5 respectively. Figure 5.4 differs from Figure 5.5 in that the *name* subtree has been cloned and a copy appears for each parent. This section examines the effect this duplication has on the performance of the model.

Table 6.16 shows the type confusion matrices, with and without state tying, for the bibliography corpus. Perhaps surprisingly, the two key leaf tags *first* and *last* perform similarly in the two models. This is evidence that good models were built for these tags both with and without tying. At a slightly higher level, the tying performed noticeably better (more than 1%) at identifying *name* tags, while without tying performed noticeably better (more than 1%) at identifying *editor* tags. This later improvement appears to be because that proceedings editors often only have

bibliography	bibbody	editor	author	name	first	last	pages	date	volume	number	title	journal	booktitle	publisher	address
99.88	+	+		+	+	+					+	+	+		+
+	**80.06**	+	+	+	+	+	+	+	+	+	3.02	7.09	4.64	1.80	1.54
	3.01	**83.93**	3.76	2.21	+	3.49					2.85		+	+	+
	+	+	**92.59**	2.43	+	+					3.91	+	+	+	+
+	1.52	+	4.27	**89.50**	+	1.87					1.94	+	+	+	+
	+	+	+	+	**94.58**	2.92					1.87	+	+	+	+
	+	+	+	+	1.28	**94.78**	+		+		3.21	+	+	+	+
	+	+		+	+	+	**99.04**	+	+		+			+	+
	+	+		+	+	+	+	**97.27**	+	+	1.01	+	+	+	+
	3.32					+	+	+	**94.79**	+	+	+	+	+	+
	1.46						+	+	+	**96.97**	+		+		
+	1.28	+	+	+	+	1.35	+	+	+	+	**93.91**	+	2.41	+	+
+	3.28	+		+	+	+	+	+	+	+	1.27	**94.52**	+	+	+
	3.13	+	+	+	+	+		+	+	+	7.94	+	**87.29**	+	+
	8.01	+		+	+	+	+	+	+	+	3.61	1.57	+	**81.70**	3.98
	4.19	+		+	+	+	+	+	+	+	2.60	+	1.85	2.68	**87.30**

bibliography	bibbody	editor	author	name	first	last	pages	date	volume	number	title	journal	booktitle	publisher	address
99.87	+	+				+	+				+		+	+	+
+	**77.14**	+	+	+	+	+	+	+	+	+	3.01	8.25	6.52	1.82	1.39
	3.31	**85.88**	3.33	1.39	+	2.82					2.62		+	+	+
	+	+	**92.78**	2.37	+	+					3.56		+	+	+
	1.59	+	5.30	**87.70**	+	2.49					1.91	+	+	+	+
	+	+	+	+	**94.35**	3.31					1.55	+	+	+	+
	+	+	+	+	1.45	**94.49**			+		3.10	+	+	+	+
	+						**98.95**	+	+	+	+			+	+
	+	+		+	+	+	+	**96.87**	+		1.06		1.18	+	+
	2.09			+	+	+	+	+	**95.99**	+	1.11	+	+	+	+
	+						+	+	+	**96.09**	1.53		+		
+	1.17	+	+	+	+	1.70	+	+	+	+	**93.08**	+	3.05	+	+
	2.63	+	+	+	+	+	+	+	+		1.11	**95.54**	+	+	+
+	3.71	+	+	+	+	+	+	+	+		10.23	+	**84.59**	+	+
	7.25	+	+	+	+	+	+	+	+		4.57	1.85	+	**81.47**	3.77
	3.54	+		+	+	1.18	+	+	+		3.02	+	2.35	2.86	**85.82**

Table 6.16: Type confusion matrices for the bibliography corpus. The matrix on the left is with state tying and the matrix on the right is without state tying. All values are in percent, a ‘+’ indicates a figure lower than 0.99%. Order 6 models trained on 6000 documents and tested on 500 documents with a lookahead of 5.

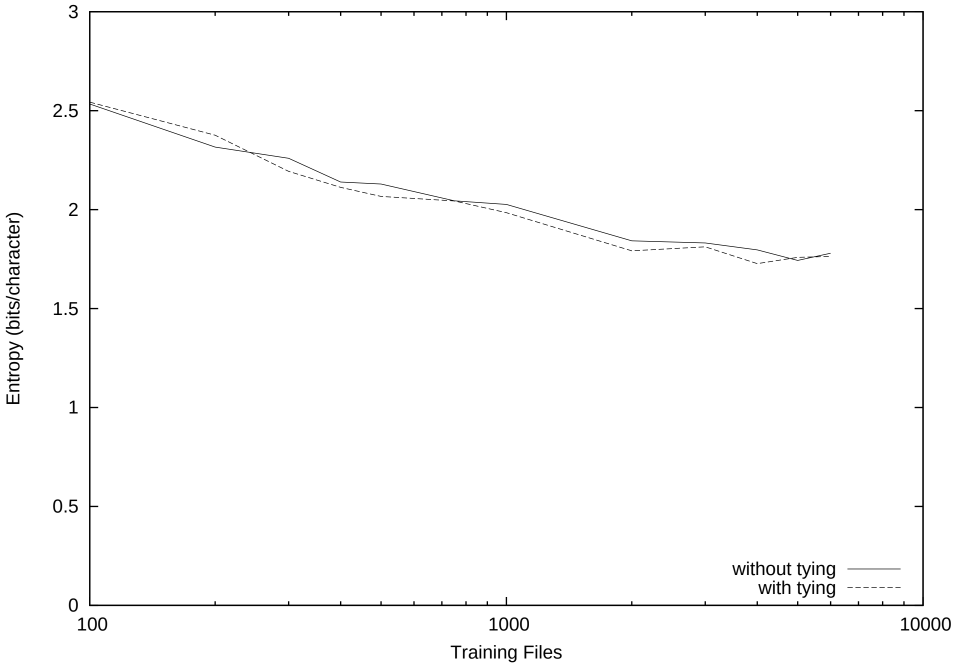


Figure 6.19: Entropy dropping with increased training data, with and without state tying. Order 6 models tested on 500 documents with a lookahead of 5.

their last name given in bibliography entries and modelling *editor* tags separately from *author* tags allowed this information to be captured.

In the tags not directly related to names, the state tying results are slightly better than the without state tying results, having a higher number of results on the leading diagonal in nine of eleven cases. This is, perhaps, because the state tying presented a more consistent model of the concepts of names to the rest of the model. Other features of type confusion matrices for the bibliography corpus are explained in Section 6.2.3.

Figure 6.19 shows how entropy drops with increased training data, with and without state tying, for the tags shown in Table 6.16. Entropy with state tying appears to be slightly less, but not consistently less, than entropy without state tying. This is somewhat surprising since the motivation for state tying was to achieve better performance from the same amount of training data (Section 4.3.7), and this appears not to be happening consistently. This is probably because the effect which

+	2.79	2.20	+	3.85	**95.64**	last	1.29	1.74	1.78	+	2.67	**95.56**
+	+	+	+	**94.73**	1.49	first	+	1.01	+	+	**95.96**	1.81
+	4.86	6.36	**93.38**	+	+	author	+	5.45	5.90	**91.78**	+	+
+	+	**81.03**	1.68	+	+	editor	+	+	**81.36**	2.29	+	
+	**88.50**	+	1.88	+	+	name	+	**89.04**	+	2.44	+	+
98.98	2.85	9.91	2.86	1.10	2.68	bibliography	**98.53**	2.57	10.52	3.41	1.16	2.48

Table 6.17: Type confusion matrices for the bibliography corpus. The matrix on the left is with state tying and the matrix on the right is without state tying. All values are in percent, a ‘+’ indicates a figure lower than 0.99%. Order 6 models trained on 6000 documents and tested on 100 documents with a lookahead of 5.

is noticeable in Table 6.16 is too small to be detected over the sampling error.

Table 6.17 shows the type confusion matrices, with and without state tying, for the bibliography corpus for a greatly reduced set of tags compared with Table 6.16. The results do not show a clear pattern of similarity with those shown in Table 6.16 for the larger set of tags, suggesting that the results are not generally applicable.

An unanticipated benefit of state tying is that the combined models are significantly smaller than the separate models. The memory consumption of models increases linearly with extra tags but less than linearly with extra training data: if two tags are tied together to use the same PPM model, memory can be saved. The CEM implementation uses memory näively, no experimentation or tuning has been used to reduce the memory consumption.

The state tying optimisation gives at best a marginal improvement in results, but can be expected to lead to smaller models. Occam’s Razor (also called the ‘principle of parsimony’ or the ‘principle of simplicity’) asserts that a simpler or smaller model of a phenomenon is to be preferred over a more complex or larger one.

Chapter 7

Conclusions

This thesis extended text augmentation to cover entity extraction problems. It investigated three classes of text augmentation: segmentation, classification and entity extraction, and described how they are connected to data mining, text mining and related fields.

Segmentation, the computationally simplest class, involves segmenting the text. Information is encoded in where one segment ends and the next starts. Tasks such as Chinese text segmentation were evaluated using recall and precision on the segment boundaries.

Classification, which is more computationally expensive than segmentation, involves classifying textual elements into one of several classes. Information is encoded in the class an element falls into. Classification tasks, such as part of speech tagging, have close ties to machine learning, and share with it the confusion matrix evaluation method.

Entity extraction is the most computationally expensive class of text augmentation. It marks-up textual fragments with a nested hierarchy of classes and information is encoded both in where fragments start and finish and in their type. Inserting attribute-free XML into text is an entity-extraction task. Entity extraction was evaluated using type confusion matrixes and using edit distances.

7.1 Review of Aims

In Section 1.2 various aims were introduced; in this section they are examined to determine whether they have been met.

1. *Examine text-augmentation problems, in the large, to attempt to determine which are susceptible to automated text augmentation and whether some sets of problems are inherently easier than other.*

   Section 4.1 built a taxonomy of three taxa of text-augmentation problems: segmentation, classification and entity extraction. Collection and document level metadata are poorly catered for. Section 4.1.4 covers a number of forms of fine-grained metadata which does not sit within the taxonomy. Sections 4.4 and 6.4 examines the different static and dynamic performance of various searches over the different problems. Segmentation is computationally easier than classification, which is computationally easier than entity extraction. This aim has been met.

2. *Build a text-augmentation system capable of solving at least as wide a range of problems as existing low-human-input systems, with an eye to eventual inclusion as part of a digital library system.*

   Section 4.2 describes CEM, a system capable of solving a wider range of text-augmentations problems than the immediately previous systems TMT and SMI, which did not solve entity-extraction problems. CEM has low-human-input and has a number of design characteristics such as using Unicode throughout and using standard XML documents. This aim has been met.

3. *Locate and/or build corpora to test this system.*

   The four corpora used in this thesis are described in Chapter 5. The Computists' corpus was developed from an earlier corpus; the Chinese text segmentation and Reuters' corpora were existing corpora adapted for use. The bibliography corpus was built as a model entity-extraction corpus. This aim has been met.

4. Use specific heuristics and optimisations which perform well in relation to a particular set of augmentation problems.

   The best first optimisation and automatic tokenisation, alphabet reduction, maximum lookahead, TagC and state tying heuristics are described in Chapter 5 and used with particular types of augmentation problems. State tying is effective only on entity extraction problems (Section 4.3.7) and TagC only works on entity extraction and classification problems (Section 4.3.6). This aim has been met.

5. *Evaluate both the text-augmentation system and the heuristics and optimisations in the system.*

   Chapter 6 contains a systematic evaluation of both the system as a whole and individual heuristics and optimisations. This aim his been met.

7.2 Performance of CEM and the New Techniques

The implementation, CEM, created for this thesis uses a substantially different form of model from that used by previous workers. The model not only allows fully recursive modelling to deeply tagged XML, it also carries context between hidden states, which avoids prejudicing entry to these states by avoiding escaping back to low-order models. CEM also uses a significantly more efficient variation on the PPMD escape method avoiding full exclusion. Non-full exclusion is a substantial performance improvement over full-exclusion with marginal less of correctness.

The best first optimisation leads to substantial gain. It could be argued that the best first optimisation was an implementation detail rather than a true optimisation. It is, however, absent from the immediately preceding system, Teahan's TMT.

Hardware implementations of Viterbi search usually avoid the need for the best first optimisation by performing this step in parallel.

The maximum lookahead heuristic is used elsewhere and was shown to work in CEM to good effect. Unfortunately there is no apparent *a priori* method for selecting a maximum lookahead, other than by splitting a known-good corpus into a training corpus and a testing corpus. This technique is less effective once the Baum–Walsh algorithm has been used to adapt the model to a superset of the original corpus.

CEM also implements two novel heuristics, TagC and automatic tokenisation, to some advantage. Both are reliant on the consistency of the training data and are unlikely to be widely useful on uncurated diverse corpora. They also largely prune the search tree in ways that the best first optimisation also prunes effectively.

The state tying heuristic, which is widely used in voice-recognition systems, was found to have little effect on the search space, but reduced the size of the hidden Markov model by merging some of the underlying Markov models. If the semantics of tag nesting are changed, state tying is likely to be more effective. In either case, it reduces the number of Markov models, and proportionally reduces the required volume of training data. The use of state tying in this way, however, hampers the convergence towards consistent tagging in the marked up text, by making the Markov model that best matches a fragment accessible at multiple hidden states. This is likely to be a significant barrier to the incremental development of corpora using the system to improve the quality of the training text. It may be possible to enable state tying during training, and disable it during testing and re-estimation to restrict access to each Markov model to a single hidden state, thus standardising the tagging.

Four corpora were used in this thesis. Marking-up the Chinese text segmentation corpus was a task on which CEM achieved an F-measure of 98%, in the same range as other systems and better than TMT. The Reuters' corpus was used in conjunction with the Brill part of speech tagger, but CEM performed poorly on this classification task, because the PPM models in CEM have a linear context and lack super-adjacency, a key aspect of the Brill tagger and other part-of-speech taggers.

A detailed comparison of the performance of CEM and the similar TMT system on the Computists' corpus showed that TMT performed consistently better. The differences were shown to be related to both the modelling characters rather than words, and the search algorithm.

The fourth corpus was the bibliography corpus, which was used for entity extraction. CEM appeared to perform well, but the lack of a standard test corpus made comparison with other systems difficult.

CEM includes the Baum–Welch algorithm: this was successfully used to help adapt a model trained on one style of bibliography to markup a different style. In this thesis the Baum–Welch algorithm was evaluated using the edit-distance metric.

CEM can be applied to solve a significantly wider range of problems than the immediately preceding system (TMT), which could solve segmentation and classification problems but not entity extraction. CEM performed well at both the simple and complex ends of the computational spectrum. It was, however, not so well optimised for speed or memory consumption as TMT.

7.3 Impact of Unicode and Document Orientation

Use of Unicode solves many internationalisation issues, but not the unknown-character problem: the character level equivalent of the unknown word problem. It also provides a set of cross-language character classes on which word-level rules and models can be built. The character classes are similar in approach to the character classes from the C programming language, which have a long history of use in parsers.

Encoding metadata, as a CEM does, in a single hierarchical insertion of attribute-free XML tags, limits the classes of metadata that can be represented, in particular, overlapping structures and alternative interpretations of the same passage. There are interesting sets of metadata that fall into the excluded category, in particular: overlapping hierarchies such as physical and logical document structure, and metadata constructed from fragments scattered throughout the document text.

The view of the data and metadata as an annotated document rather than a collection of facts has a number of impacts on further use, even though metadata held in an external database could be processed to embed it in the document and *vice versa*. Firstly it makes the document more amenable to presentation as a metadata-enhanced document, such as in a digital library or an XML-based document repository. Secondly it makes the kinds of higher-level processing used in the later stages of many of the MUC systems harder, because these perform operations such as relational joins which have no direct equivalent in an annotated document. Thirdly it makes the metadata significantly less amenable to export for use in external systems, many of which expect relations of data. Fourthly document-centric, XML-native, databases allow queries on the annotated XML documents, including aspects of the documents which the querier might consider important which the metadata extractor might not. The best representation for inferred metadata is thus likely to be determined by the larger context and the intended uses of the metadata.

7.4 Limitations of CEM

CEM has two broad sets of limitations, those imposed by modelling and search techniques, and those due to the implementation of those techniques.

Attribute data CEM does not capture attribute data. For enumerable attributes, this can be mitigated by XML transformations which transform each possible combination of attributes in each tag to a separate tag. For continuous attributes this technique leads to an infinite number of tags. It is not clear how many continuous attributes occur in linguistic corpora, the author has seen continuous attributes in spoken linguistic corpora (particularly in the time dimension) but not in written linguistic corpora.

Differentiable tags Tags that do not have different character distributions, or whose character distributions PPM is unable to model, cannot be inserted. An extreme case of this might be the task of marking-up the prime-numbered digits in a decimal representation of π. While automating such a marking up is possible, doing it with Viterbi search and learnt PPM models is not. The author is aware of no linguistic corpora for which this is an issue.

Consistency Tags are assumed to be used consistently. This does not hold for many real-world situations, but curated textual corpora are becoming more common. There are also various tools such as jtidy[1] which regularise some aspects of HTML/XHTML.

These three limitations are shared with all directly comparable applications of searching using Markov models, including TMT and HTK. The second set of limitations are implementation-based, caused by choices made when building CEM.

Number of tags CEM has an upper bound on the number of Markov models and thus of tags modelled. The implementation represents tags using Unicode characters from the private use range \uE000–\uF8FF, of which 3 are reserved as special markers. While an order of magnitude greater than the number of tags appearing in commonly used markup such as XHTML, MathML and those appearing in this thesis, this limits the use of tag transformations as work-arounds for other limitations.

Nesting of tags CEM cannot represent tags nested directly within tags of the same type. This is currently impossible because in the search nodes only the tag is

[1] `http://jtidy.sourceforge.net/`

noted and not whether it is opening or closing. None of the corpora examined here displays such nesting and while it would be relatively easy to fix, it would involve an extra test in the inner loop of the search operation, slowing searching. An alternative to changing the implementation is to transform the text so that every odd-depth tag has a different name, and then use state-tying to tie the odd and even tags together. HTK supports models such as these, TMT does not.

Adaptive Models The PPM models implemented in CEM are not adaptive. This means that the Baum–Welch algorithm cannot be applied any finer than the document level, for example to allow intra-document learning. This is likely to be a problem when the re-estimation text contains relatively few but unusually large documents, allowing few re-estimation cycles. If the documents are internally homogeneous, it may be possible to overcome this by splitting them to increase the number of inter-document re-estimation cycles. Both HTK and TMT can be adaptive.

Streaming documents Documents are held entirely in memory rather than being streamed. Holding documents in memory consumes extra memory. While this was not a problem for corpora used in this work, which have reasonably short documents, it would prevent processing of large documents. Documents as large as 6MB (unmarked up size) have been successfully marked up. Document length is linearly related to this aspect of memory consumption. HTK allows documents to be streamed, TMT does not.

Document-at-once processing An entire XML document, rather than an XML fragment, must be marked up at once. The command line to interface CEM requires documents be read from the file system, one document per file. A Java interface allowing arbitrary XML nodes to be marked-up exists but is not used in the experiments presented here. Marking-up document fragments is important in interfacing CEM with other systems. Both HTK and TMT have interfaces allowing partial documents to be processed.

Integer overflow The PPM models implemented in CEM implicitly assume that none of their counters rolls over. This assumption holds unless more than $2^{31} - 1$ characters of training data (or combined training and re-estimation data) are seen. HTK overcomes this limit by encoding probabilities as floating-point numbers rather than as ratios of integers. TMT overcomes this limit using integers that are scaled prior to overflow. The latter could be worked into CEM.

CEM does not have a mode of operation which calculates the entropy of entire documents in each of the Markov models. This is used effectively by TMT for calculation of whole document metadata such as language and genre. Of these implementation limitations, only making the PPM models adaptive and removing the upper bound on the number of tags would require extensive redesign of CEM.

7.5 Problems Suitable for CEM and Text Augmentation

There are several broad indicators that metadata will be marked up well by CEM: it should be relatively fine-grained, at the character, word or phrase level; it should be discriminatable from the immediately surrounding text; there should be a training corpus which matches the testing text sufficiently well to build a model from (or text available to build such corpus from); if the testing text is changing with time, it should be changing sufficiently slowly that the model can be re-estimated to track the changes.

Segmentation problems that meet these requirements include the segmentation of languages written without spaces between words (i.e. Chinese, Japanese and

Thai) and locating potential hyphenation points in European languages (i.e. English, German and French). Classification problems that meet these requirements include part-of-speech tagging, finding proper nouns, email addresses, URLs, stock, cross-references and similar classes of textual entities. Entity-extraction problems that meet these requirements include marking-up bibliographies, title and frontis pages, email headers, standard forms and other highly-structured sections of text.

Parsing of many computer programming languages, including Scheme, Java and C, into an XML representation is an entity-extraction problem, although not one CEM is ideal for, because of the length of structures involved. Parsing of the Python language is not, and CEM is not capable of this task: the concept ‘the same indentation as the previous line’ cannot be learnt using PPM.

In all cases, higher-order reasoning based on the inferred metadata is beyond the ability of CEM. For example, while it can find proper nouns in English text, but it cannot be used to find equivalences between different nouns used for the same subject, because this requires reasoning about on non-adjacent values. Since this higher-order reasoning is an integral part of many systems used in the wild, CEM is unlikely to be a suitable drop-in replacement for many systems.

7.6 Training Corpora Sizes

The relative success of text augmentation on the Computists’ corpus, with only 38 issues of 1200 words, shows that augmentation can be useful even when trained on relatively small volumes of text. Certainly this augmentation is of high enough quality to be used for transforming the document for presentation to end users. With F-measures as low as 55%, however, the augmented text should be used with care. In particular, the compilation of indexes and of extracted terms, in which recurring terms contribute less than singly-occurring terms should be avoided, as this emphasises errors, which tend to be unique, singly-occurring items.

Estimating the quantity of training text needed to produce results of a certain quality is challenging because of the many factors that influence this, but it seems apparent, supported by the experimental results in Chapter 6, that model discrimination is key. For example in the Computists' corpus, the easily-discriminated URL and email tags were augmented reliably, whereas the poorly-discriminated name, organisation and location tags were augmented poorly, despite considerably more examples being seen in training.

The incremental development of the Computists' corpus, together with an examination of the errors of text augmentation systems leading the correction of the training text, is likely to be particularly scalable, since it allows leveraging of work already completed to converge on a consistently marked up corpus. Unfortunately, incremental development may reveal flaws in the initial assumptions, which are unlikely to be rectifiable without considerable work.

The automated conversion of existing data and metadata into a corpus, as for the bibliography corpus, has the advantage that the metadata in existing data is presumably present for a reason, reflecting the use or meaning of the data. The conversion is automated, so if the conversion reveals issues it can be re-performed completely.

Automatic conversion is limited to those corpora for which a suitable data source can be found with suitable metadata, and those found are unlikely to be structured to allow for control of arbitrary variables of interest. The growth of curated repositories may increase the likelihood that a corpus already exists that can be converted, extended or developed to be suitable.

7.7 Original Contributions

A number of original contributions are made in this thesis. A system called 'Colloquial Entropy Markup' or CEM was designed and implemented. CEM builds a hidden Markov model from a corpus of marked-up XML documents and uses variants of Viterbi search to augment unmarked-up XML documents with tags in the marked-up XML documents.

Four corpora were used. The Reuters' and segmentation corpora required relatively little data preparation. The Computists' corpus was systematically re-marked-up. The bibliography corpus is a new corpus.

The following are the key novel aspects of the work presented in this thesis.

- Partitioning of tag insertion problems into a coherent taxonomy with three taxa (Section 2.1.2).
- Exploration of the relationship between PPM (Prediction by Partial Matching) models and Markov models (Section 3.3). Previously published as [164].
- Expansion of text augmentation to include nested tags (Chapter 4).
- The best first (Section 4.3.2) optimisation, the automatic tokenisation (Section 4.3.3), alphabet reduction (Section 4.3.4) and TagC (Section 4.3.6) heuristics.
- Detailed analysis of the search space size of tag insertion (Section 4.4). Earlier versions of this work were published as [162].
- Detailed analysis of the correctness measures for different types of tag insertion problems and research methodology (Section 2.3).
- Development of an entropy-based technique to determine whether tag-insertion errors are the result of a PPM modelling failure or of a searching failure (Section 2.3.4).
- A new extension of confusion matrices suitable for evaluating hierarchical many-class classification problems (Section 4.6.4).

7.8 Open Questions

There are a number of open questions not examined in this thesis:

1. Whether the conceptualisation of context used here (and elsewhere) is optimal. There is an alternative method for computing the context of the current character in a character stream. This was discovered during the experimental work for this thesis, but not explored. The context for *e* in …*<a>ab</a><b>cd</b><c>e*… can be 'collapsed' to …*<a/><b/>e*. … This could be achieved by substituting the character representing the transition into the tag for the entire tag. This approach is likely to be most successful where tag densities are highest, such as in part-of-speech tagging, where state-of-the-art systems take advantage of super-adjacency.

2. Whether adding a default tag with an uninitialised (untrained) model accessible from every context would remove the tendency to place high-entropy sequences in the model with the least training data.

3. Whether different escape methods would reduce the tendency to place high entropy sequences in the model with the least training data.

4. Whether a more universal similarity metric such as Kolmogorov complexity [85, 86] might be an appropriate measure for comparing sequences. This would move evaluation to a theoretical framework independent of any particular approach to solving the problem and resolve some of the complexities of evaluating performance.

5. Whether certain textual strings (such as *References* on a line by itself) can be used as synchronisation points in a finite automata sense. This is likely to form part of the infrastructure integrating CEM into a possible digital library structure, which will need ways of detecting when it is appropriate to use various tools such as CEM.

6. Whether Teahan search or Viterbi search will perform better on certain classes of text-augmentation tasks.

All of these seem useful avenues of investigation, 1 and 4 being significantly more novel than 2 and 3. Issues 5 and 6 are likely to be directly and immediately relevant to a practical production system.

Bibliography

[1] Eugene Agichtein and Luis Gravano. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the Fifth ACM conference on Digital libraries*, pages 85–94, San Antonio, Texas, United States, 2000.

[2] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques and Tools*. Addison-Wesley, 1986.

[3] Rie Kubota Ando and Lillian Lee. Mostly unsupervised statistical segmentation of Japanese Kanji sequences. *Journal of Natural Language Engineering*, 9(2):127–149, August 2003.

[4] J. Anigbogu and A. Belaid. Hidden Markov models in text recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 9(6):925–958, 1995.

[5] Steven Atkin and Ryan Stansifer. A generalized mechanism for Unicode metadata. In *Proceedings of the Nineteenth International Unicode Conference*, San Jose, California, USA, 10–14 September 2001.

[6] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. ACM / Addison–Wesley, Massachusetts, USA, 1999.

[7] Amit Bagga. Analyzing the complexity of a domain with respect to an Information Extraction task. In *Proceedings of the Fifth Workshop on Very Large Corpora*, 1997.

[8] David Bainbridge, Dana McKay, Ian H. Witten, and Stefan Boddie. *Greenstone Digital Library Developer's Guide*. Digital Library Laboratory, University of Waikato, March 2004.

[9] Alex Bateman, Ewan Birney, Lorenzo Cerruti, Richard Durbin, Laurence Etwiller, Sean R. Eddy, Sam Griffiths-Jones, Kevin L. Howe, Mhairi Marshall, and Erik L. L. Sonnhammer. The Pfam protein families database. *Nucleic Acids Research*, 28:263–266, 2000.

[10] Leonard E. Baum. An inequality and associated maximisation technique in statistical estimation for probabilistic functions of a Markov process. *Inequalities*, 3:1–8, 1972. Not sighted.

[11] Leonard E. Baum, Ted Petrie, George Soules, and Norman Weiss. A maximisation techniques occurring in the statistical analysis of probabilistic functions of markov chains. *The Annals of Mathematical Statistics*, 41(1):164–171, 1970.

[12] Doug Beeferman, Adam Berger, and John D. Lafferty. Statistical models for text segmentation. *Machine Learning*, 34(1-3):177–210, 1999.

[13] Timothy C. Bell, John G. Cleary, and Ian H. Witten. *Text Compression*. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1990.

[14] Timothy C. Bell, Ian H. Witten, and John G. Cleary. Modeling for text compression. *ACM Computing Surveys*, 21(4):557–591, December 1989.

[15] Dominique Besagni, Abdel Belaïd, and Nelly Benet. A segmentation method for bibliographic references by contextual tagging of fields. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition*, page 384, Edinburgh, Scotland, August 2003.

[16] Alan W. Biermann and Amit Bagga. Analyzing the complexity of a domain with respect to an Information Extraction task. In *Proceedings of the Fifth Workshop on Very Large Corpora*, 1997.

[17] Daniel M. Bikel, Richard L. Schwartz, and Ralph M. Weischedel. An algorithm that learns what's in a name. *Machine Learning*, 34(1-3):211–231, 1999.

[18] Jeff A. Bilmes. A gentle tutorial on the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models. Technical Report ICSI-TR-97-021, University of Berkeley, Massachusetts, USA, 1997.

[19] Steven Bird, Peter Buneman, and Wang-Chiew Tan. Towards a query language for annotation graphs. In *Proceedings of the Second International Conference on Language Resources and Evaluation*, pages 807–814, Paris, France, 2000. European Language Resources Association.

[20] Steven Bird, Kazuaki Maeda, Xiaoyi Ma, Haejoong Lee, Beth Randall, and Salim Zayat. TableTrans, Multitrans, InterTrans and TreeTrans: Diverse tools built on the annotation graph toolkit. In *Proceedings of the Third International Conference on Language Resources and Evaluation*, Paris, France, 2002.

[21] Kurt Bollacker, Steve Lawrence, and C. Lee Giles. CiteSeer: An autonomous web agent for automatic retrieval and identification of interesting publications. In Katia P. Sycara and Michael Wooldridge, editors, *Proceedings of the Second International Conference on Autonomous Agents*, pages 116–123, New York, New York, USA, 1998. ACM Press.

[22] Kalina Bontcheva, Marin Dimitrov, Diana Maybard, Valentin Tablin, and Hamish Cunningham. Shallow methods for named entity coreference resolution. In *Conference annuelle sur le Traitement Automatique des Langues Naturelles*, Nancy, France, 24–27 June 2002.

[23] A. Borthwick, J. Sterling, E. Agichtein, and R. Grishman. Nyu: Description of the mene named entity system as used in MUC. In *Proceedings of the Seventh Message Understanding Conference (MUC-7)*, 1998.

[24] Bert Bos, Tantek Çelik, Ian Hickson, and Håkon Wium Lie. Cascading style sheets 2.1 specification. Technical report, World Wide Web Consortium (W3C), 25 February 2004.

[25] Tim Bray, Jean Paoli, and C. Michael Sperberg-McQueen. EXtensible Markup Language 1.0. Recommendation, The World Wide Web Consortium, 10 February 1998.

[26] Zane C. Bray. Using compression models for text mining. Master's thesis, Department of Computer Science, University of Waikato, Hamilton, New Zealand, July 1999.

[27] Alvis Brazma, Inge Jonassen, Ingvar Eidhammer, and David Gilbert. Approaches to the automatic discovery of patterns in biosequences. *Journal of Computational Biology*, 5(2):277–304, 1998.

[28] Eric Brill. A simple rule-based part-of-speech tagger. In *Proceedings of the Third Conference on Applied Natural Language Processing*, pages 152–155, Trento, Italy, 1992.

[29] Eric Brill. Some advances in transformation-based part of speech tagging. In *Proceedings of the Twelth National Conference on Artificial Intelligence*, pages 722–727, 1994.

[30] John Seely Brown and Paul Duguid. *The Social Life of Information*. Harvard Business School Press, Boston, Massachusetts, USA, March 2000. Excerpt published in First Monday 5:4 April 2000.
http://www.firstmonday.org/issues/issue5_4/brown_contents.html.

[31] M. Buckland and F. Gey. The relationship between recall and precision. *Journal of the American Society for Information Science*, 45(1):12–19, 1994.

[32] Jeffrey T. Chang, Hinrich Schütze, and Russ B. Altman. Creating an online dictionary of abbreviations from MEDLINE. *Journal of the American Medical Informatics Association*, 23 July 2002.

[33] Eugene Charniak. *Statistical Language Learning*. MIT Press, Cambridge, Massachusetts, USA, 1993.

[34] Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. In Arivind Joshi and Martha Palmer, editors, *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pages 310–318. Morgan Kaufmann, 1996.

[35] Nancy A. Chinchor. Overview of MUC-7/MET-2. In *Proceedings of the Seventh Message Understanding Conference*, April 1998.

[36] John G. Cleary and William J. Teahan. An open interface for probabilistic models of text. In James A. Storer and Martin Cohn, editors, *Proceedings of the Data Compression Conference*, 1999.

[37] Hamish Cunningham, Yorick Wilks, and Robert J. Gaizauskas. Gate—a general architecture for text engineering. In *Sixteenth International Conference on Computational Linguistics*, volume 2, pages 1057–1060, Denmark, August 1996.

[38] Richard Curtis, Ben Elton, and John Lloyd. *Blackadder: the Whole Damn Dynasty*. Michael Joseph, London, England, 1998.

[39] Doug Cutting, Julian Kupiec, Jan Pedersen, and Penelope Sibun. A practical part-of-speech tagger. In Oliviero Stock, editor, *Proceedings of the Third Conference on Applied Natural Language Processing*, pages 133–140, Trento, Italy, April 1992.

[40] Doug Cutting and Jan Pedersen. *The Xerox Part-of-Speech Tagger*. Xerox Palo Alto Research Center, Palo Alto, California, USA, v1.0 edition, 12 April 1993.

[41] E. B. Dynkin. *Markov Processes*. Springer, New York, 1965. Translated into English by J. Fabius, V. Greenberg, A. Maitra and G. Majone.

[42] Tom Emerson. Segmentation of Chinese text. *MultiLingual Computing & Technology*, 12(38), February 2003.

[43] David C. Fallside. *XML Schema*. The World Wide Web Consortium (W3C), 2 May 2001.

[44] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. RFC 2068: HyperText Transfer Protocol — HTTP 1.1, January 1997.

[45] Dayne Freitag. Multistrategy learning for information extraction. In *Proceedings of the Fifthteenth International Conference on Machine Learning*, pages 161–169, San Francisco, California, USA, 1998. Morgan Kaufmann.

[46] Dayne Freitag and Andrew McCallum. Information extraction with HMM structures learned by stochastic optimization. In *Proceedings of the American Association for Artificial Intelligence Conference*, pages 584–589, 2000.

[47] Dayne Freitag and Andrew Kachites McCallum. Information extraction with HMMs and shrinkage. In *Proceedings of the American Association for Artificial Intelligence Workshop on Machine Learning for Information Extraction*, 1999.

[48] Betty Furrie. *Understanding MARC Bibliographic: Machine-Readable Cataloging*. Cataloging Distribution Service, Library of Congress, fifth edition, 1990.

[49] R. Gaizauskas and Y. Wilks. Information extraction: Beyond document retrieval. *Journal of Documentation*, 54(1):70–105, 1998.

[50] Xianping Ge, Wanda Pratt, and Padhraic Smyth. Discovering Chinese words from unsegmented text. In *Research and Development in Information Retrieval*, pages 271–272, 1999.

[51] Charles F. Goldfarb. *The SGML Handbook*. Oxford, 1990.

[52] James Gosling, Bill Joy, and Guy Steele. *The Java Language Specification*. Addison–Wesley, Massachusetts, USA, 1996.

[53] Donna K. Harman. Overview of the first text retrieval conference. In *Proceedings of the First Text REtrieval Conference*, pages 1–20, Gaithersburg, Maryland, USA, 4–6 November 1992. National Institute of Standards and Technology.

[54] Donna K. Harman. Overview of the fourth text retrieval conference. In *Proceedings of the Fourth Text REtrieval Conference*, pages 1–24, Gaithersburg, Maryland, USA, 1–3 November 1995. National Institute of Standards and Technology.

[55] M. Hearst. Untangling text data mining. In *Proceedings of the Thirty seventh Annual Meeting of the Association for Computational Linguistics*, 1999.

[56] Chris Heegard and Stephen B. Wicker. *Turbo Coding*. Kluwer Academic Publishers, 1999.

[57] Paul Glor Howard. *The Design and Analysis of Efficient Lossless Data Compression Systems*. PhD thesis, Department of Computer Science, Brown University, June 1993.

[58] Akira Ishikawa. A functional operator-based morphological analysis of Japanese. In *14th International Conference of Applications of Prolog*, Tokyo, Japan, 20–22 October 2001.

[59] ISO-9899—Harmonized standard for the C programming language, 1990.

[60] Frederick Jelinek. *Statistical Methods for Speech Recognition*. MIT Press, Boston, Massachusetts, USA, 1998.

[61] Thorsten Joachims. A probabilistic analysis of the Rocchio algorithm with TF/IDF for text categorization. In Douglas H. Fisher, editor, *Proceedings of Fourteenth International Conference on Machine Learning*, pages 143–151, Nashville, USA, 1997. Morgan Kaufmann.

[62] Thorsten Joachims. Text categorization with support vector machines: learning with many relevant features. In Claire Nédellec and Céline Rouveirol, editors, *Proceedings of Tenth European Conference on Machine Learning*, pages 137–142, Chemnitz, Germany, 1998. Springer.

[63] Rolf Johannesson and Kamil Sh. Zigangirov. *Fundamentals of Convolutional Coding*. Wiley, 1999.

[64] S. Johansson, G. Leech, and H. Goodluck. Manual of information to accompany the Lancaster-Oslo-Bergen corpus of british english for use with digital computers. Technical report, Bergen: Norwegian Computing Center for the Humanities, 1978.

[65] Karen Sparck Jones and C. J. van Rijshergen. Report on the need for and provision of an “ideal” information retrieval test collection. Technical report, Cambridge University Computer Laboratory, December 1975.

[66] Jussi Karlgren and Douglass Cutting. Recognizing text genres with simple metrics using discriminant analysis. In *Proceedings of the Fifteenth International Conference on Computational Linguistics*, volume II, pages 1071–1075, Kyoto, Japan, 1994.

[67] Slava M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics Speech and Signal Processing*, 35(3):400–401, March 1997.

[68] D. Khmelev and William J. Teahan. A repetition-based measure for verification of text collections and for text categorization. In *Proceedings of the 26th Annual International ACM SIGIR Conference (SIGIR),* Toronto, Canada, 28 July–1 August 2003.

[69] Y Khmelev and William J. Teahan. A repetition-based measure for verification of text collections and for text categorization. In *26th Annual International ACM Special Interest Group on Information Retrieval Conference (SIGIR),* Toronto, Canada, 28 July–1 August 2003.

[70] Jeffrey H. Kingston. *Algorithms and Data Structures—Design, Correctness, Analysis*. Addison–Wesley, Massachusetts, USA, 1990.

[71] Donald E. Knuth. *The Art of Computer Programming*, volume 1, Fundamental Algorithms. Addison–Wesley, first edition, 1968.

[72] Anders Krogh, Michael Brown, I. Saira Mian, Kimmen Sjolander, and David Haussler. Hidden Markov models in computational biology: Applications to protein modeling. Technical Report UCSC-CRL-93-32, University of California at Santa Cruz, 1993.

[73] Karen Kukich. Techniques for automatically correcting words in text. *ACM Computing Surveys*, 24(4):377–439, December 1992.

[74] Nicholas Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118:15–68, 1999.

[75] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence

data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289. Morgan Kaufmann, 2001.

[76] Corrin Lakeland. Part of speech tagging in statistical parsing. In M. Jason-Smith, A. Renaud, and T. Wright, editors, *Proceedings of the New Zealand Computer Science Research Students' Conference*, pages 138–139, April 2001.

[77] Leslie Lamport. *LaTeX: A Document Preparation System: User's Guide and Reference Manual*. Addison–Wesley, Massachusetts, USA, 1986.

[78] Christophe Laprun, Jonathan G. Fiscus, Sylvain Pajot, and John Garofolo. A practical introduction to ATLAS. In *Human Language Technology*, San Diego, California, USA, 24–27 March 2002.

[79] Steve Lawrence and C. Lee Giles. Searching the World Wide Web. *Science*, 280(5360):98–100, 1998.

[80] Steve Lawrence, C. Lee Giles, and Kurt Bollacker. Digital libraries and autonomous citation indexing. *IEEE Computer*, 32(6):67–71, 1999.

[81] Steve Lawrence and Lee Giles. Accessibility of information on the web. *Nature*, 400:107–109, 1999.

[82] T. R. Leek. Information extraction using hidden Markov models. Master's thesis, University of California, San Diego, USA, 1997.

[83] Douglas B. Lenat. CYC: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, 1995.

[84] D. D. Lewis. Evaluating Text Categorization. In *Proceedings of Speech and Natural Language Workshop*, pages 312–318. Morgan Kaufmann, 1991.

[85] Ming Li, Xin Chen, Xin Li, Bin Ma, and Paul Vitányi. The similarity metric. In *Proceedings of the Fourteenth Annual Symposium on Discrete Algorithms*, pages 863–872, Baltimore, MD, 12–14 January 2003. ACM/SIAM.

[86] Ming Li and Paul M. B. Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, Berlin, Germany, 1993.

[87] Hokon Wium Lie and Bert Bos. *Cascading Style Sheets, level 1*. The World Wide Web Consortium (W3C), 17 December 1996.

[88] Jean loup Gailly. *gzip v1.0*, 1993. UNIX Manual page.

[89] Cláudio L. Lucchesi and Tomasz Kowaltowski. Applications of finite automata representing large vocabularies. *Software—Practice and Experience*, 23(1):15–300, January 1993.

[90] Helmut Lucke. Which stochastic models allow Baum-Welch training. *IEEE Transactions on Signal Processing*, 44(11):2746–2756, November 1996.

[91] Hans Peter Luhn. A statistical approach to mechanised encoding and searching of literary information. *IBM Journal*, pages 309–317, October 1957. Presented at the American Chemical Society meeting in Miami, April 8 1957.

[92] Hans Peter Luhn. *Modern Trends in Documentation*, chapter Auto-Encoding of Documents for Information Retrieval Systems, pages 45–58. Pergamon Press, London, England, 1959.

[93] Clifford Lynch. The battle to define the future of the book in the digital world. *First Monday*, 6(6), June 2001.

[94] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1994.

[95] Diana Maynard, Kalina Bontcheva, Horacio Saggion, Hamish Cunningham, and Oana Hamza. Using a text engineering framework to build an extendable and portable IE-based summarisation system. In *Proceedings of the ACL Workshop on Text Summarisation*, 2002.

[96] Andrew McCallum, Dayne Freitag, and Fernando Pereira. Maximum entropy Markov models for information extraction and segmentation. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 591–598. Morgan Kaufmann, 2000.

[97] Katherine J. McGowan. Efficient phrase hierarchy inference. Master's thesis, University of Waikato, Hamilton, New Zealand, 2002.

[98] Geoffrey J. McLachlan and Thriyambakam Krishnan. *The EM Algorithm and Extensions*. Probability and Statistics. John Wiley & Sons, Indianapolis, Indiana, USA, 1996.

[99] John G. McMahon and F. Jack Smith. A review of statistical language processing techniques. *Artificial Intelligence Review*, 12(5):347–391, 1998.

[100] Rodger J. McNab, Ian H. Witten, and S. J. Boddie. A distributed digital library architecture incorporating different index styles. In *Forum on Research and Technology Advances in Digital Libraries*, pages 36–45, Santa Barbara, California, 1998. IEEE Computer Society Press, Los Alamitos.

[101] F. Mittelbach, M. Goossens, Braams, Carlisle, and Rowley. *The LaTeX Companion*. Addison–Wesley, second edition, 2004.

[102] Alistair Moffat, Timothy C. Bell, and Ian H. Witten. Lossless compression for text and images. *International Journal of High-Speed Electronics*, 8(1):179–231, 1997. Special issue on Signal Compression.

[103] Alistar Moffat. Lossless compression. *The Computer Journal*, 40(2/3):65–66, 1997. Special Issue on Lossless Compression, Editorial.

[104] Mehryar Mohri and Michael Riley. Weighted determinization and minimization for large vocabulary speech recognition. In *Proceedings of the Fifth European Conference on Speech and Communication Technology*, pages 131–134, Rhodes, Greece, 1997.

[105] Manijya Rao Muddamalle. Natural language versus controlled vocabulary in information retrieval: A case study in soil mechanics. *Journal of the American Society for Information Science*, 49(10):881–887, 1998.

[106] Un Yong Nahm, Mikhail Bilenko, and Raymond J. Mooney. Two approaches to handling noisy variation in text mining. In *International Conference on Machine Learning Text Learning Workshop*, pages 18–27, Sydney, Australia, July 2002.

[107] Theodor H. Nelson. *Computer Lib*. Microsoft Press, Redmond, Washington, 1987. 'Dream Machines' by the same author in the same volume.

[108] Network Development and MARC Standards Office, Library of Congress. *MARC 21 Format for Bibliographic Data: Field List*, 1999 english edition, 1999.

[109] Craig G. Nevill-Manning, T. Reed, and Ian H. Witten. Extracting text from PostScript. Working Paper 97/10, Department of Computer Science, University of Waikato, April 1998.

[110] David M. Nichols, Kirsten Thomson, and Stuart A. Yeates. Usability and open-source software development. In Elizabeth Kemp, Chris Phillips, Kinshuk, and John Haynes, editors, *Symposium on Computer Human Interaction*, pages 49–54, Palmerston North, New Zealand, 6 July 2001. ACM SIGCHI NZ.

[111] Joseph Z. Nitecki. *Metalibrarianship: A Model for Intellectual Foundations of Library Information Science*. Published by Joanne Twining Williams at the Texas Woman's University, 1993. Volume 1 of the Nitecki Trilogy.

[112] Library of Congress. *Library of Congress Classification Outline*. Library of Congress, 1990.

[113] David D. Palmer and John D. Burger. Chinese word segmentation and information retrieval. In *Proceedings of the Symposium on Cross-Language Text and Speech Retrieval*. American Association for Artificial Intelligence Conference, 1997.

[114] Steven Pemberton, Daniel Austin, Jonny Axelsson, Tantek Celik, Doug Dominiak, Herman Elenbaas, Beth Epperson, Masayasu Ishikawa, Shin'ichi Matsui, Shane McCarron, Ann Navarro, Subramanian Peruvemba, Rob Relyea, Sebastian Schnitzenbaumer, and Peter Stark. *XHTML—The Extensible HyperText Markup Language—A Reformulation of HTML 4 in XML 1.0*. The World Wide Web Consortium, 1.0 edition, August 2002.

[115] Fuchun Peng, Fangfang Feng, and Andrew McCallum. Chinese segmentation and new word detection using conditional random fields. In *Proceedings of The 20th International Conference on Computational Linguistics*, Geneva, Switzerland, 23–27 August 2004.

[116] Fuchun Peng and Dale Schuurmans. Self-supervised Chinese word segmentation. *Lecture Notes in Computer Science*, 2189:238–248, 2001.

[117] Jay M. Ponte and W. Bruce Croft. USeg: a retargetable word segmentation procedure for information retrieval. In *Symposium on Document Analysis and Information Retrieval*, 1996.

[118] Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, February 1989.

[119] Martin Rajman and Romaric Besanon. Text mining knowledge extraction from unstructured textual data. In *Proceedings of the Sixth Conference of International Federation of Classification Societies*, Rome, Italy, 1998.

[120] R. Rosenfeld. A maximum entropy approach to adaptive statistical language modeling. *Computer, Speech and Language*, 10:187– 228, 1996.

[121] M. S. Ryan and G. R. Nudd. The Viterbi algorithm. Warwick Research Report RR238, Computer Science, University of Warwick, Coventry, England, 12 February 1993.

[122] Gerard Salton. *Automatic Text Processing: The transformation, Analysis, and Retrieval of Information by Computer*. Addison–Wesley, Massachusetts, USA, 1989.

[123] Dieter Scheffner and Johann-Christoph Freytag. The xml query execution engine (xee). Technical Report hub-ib-157, Humboldt University Berlin, Berlin, Germany 2001.

[124] Julian Seward. *bzip2 v1.0*. UNIX Manual page.

[125] Kristie Seymore, Andrew McCallum, and Ronald Rosenfeld. Learning hidden Markov model structure for information extraction. In *Proceedings of the Sixteenth National Conference on Articial Intelligence: Workshop on Machine Learning for Information Extraction*, pages 37–42, Orlando, FL, 1999.

[126] Claude Elwood Shannon and Warren Weaver. *The Mathematical Theory of Communication*. The University of Illinois Press, Urbana, Illinois, USA, 1964.

[127] Dmitry Shkarin. Ppm: One step to practicality. In James A. Storer and Martin Cohn, editors, *Proceedings of the 12th Data Compression Conference*, pages 202–210. IEEE Press, 2002.

[128] John Simpson, editor. *Oxford English Dictionary (Online edition)*. Oxford University Press, Oxford, England, 2002.

[129] Tony C. Smith. *N-gram Models of Agreement in Language*. PhD thesis, Computer Science Department, University of Waikato, Hamilton, New Zealand, 2001.

[130] C.M. Sperberg-McQueen and Lou Burnard. *Guidelines for Electronic Text Encoding and Interchange*. Association for Computers and the Humanities Association for Computational Linguistics and Association for Literary and Linguistic Computing, Chicago, USA and Oxford, England.

[131] James A. Storer and Martin Cohn, editors. *Data Compression Conference*, Snowbird, Utah, USA, 28–30 March 2000. IEEE.

[132] James A. Storer and Martin Cohn, editors. *Data Compression Conference*, Snowbird, Utah, USA, 27–29 March 2001. IEEE.

[133] William J. Teahan. *Modelling English Text*. PhD thesis, Department of Computer Science, University of Waikato, Hamilton, New Zealand, May 1998.

[134] William J. Teahan. An improved interface for probabilistic models of text. Technical report, School of Computer and Mathematical Sciences, The Robert Gordon University, 2000.

[135] William J. Teahan and John G. Cleary. Tag based models of English text. In Storer and Cohn [132], page 582.

[136] William J. Teahan and D. J. Harper. Combining PPM models using a text mining approach. In Storer and Cohn [131], pages 153–162.

[137] William J. Teahan, Yingying Wen, Roger McNab, and Ian H. Witten. A compression-based algorithm for Chinese word segmentation. *Computational Linguistics*, 26(3):375–393, September 2000.

[138] The Unicode Consortium. *The Unicode Standard—Worldwide Character Encoding*. Addison-Wesley, 1992.

[139] Michael B. Twidale, David M. Nichols, and Chris D. Paice. Browsing is a collaborative process. *Information Processing and Management*, 33(6):761–783, 1997.

[140] Andrew J. Viterbi. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Transactions on Information Theory*, 13:260–269, 1967.

[141] Andrew J. Viterbi and J. K. Omura. *Principles of Digital Communication and Coding*. McGraw–Hill, 1979.

[142] Ellen M. Voorhees and Donna K. Harman. Overview of the fifth text retrieval conference. In *Proceedings of the Fifth Text REtrieval Conference*, pages 1–28, Gaithersburg, Maryland, USA, 20–22 November 1996. National Institute of Standards and Technology.

[143] Ellen M. Voorhees and Donna K. Harman. Overview of the tenth text retrieval conference. In *Proceedings of the Tenth Text REtrieval Conference*, pages 1–17, Gaithersburg, Maryland, USA, 13–16 November 2001. National Institute of Standards and Technology.

[144] Yingying Wen. Text mining using HMM and PPM. Master's thesis, Computer Science, University of Waikato, Hamilton, New Zealand, July 2001.

[145] Ian H. Witten. Applications of lossless compression in adaptive text mining. In *Proc 2000 Conference on Information Sciences and Systems 2*, pages 13–18, Princeton, USA, March 2000.

[146] Ian H. Witten and Timothy C. Bell. The zero-frequency problem: Estimating the probablitiies of novel events on adaptive text compression. *IEEE Transaction of Information Theory*, 37(4):1085–1094, 1991.

[147] Ian H. Witten and Stefan Boddie. *Greenstone Digital Library Users Guide*. Digital Library Laboratory, University of Waikato, Hamilton, New Zealand, 2003.

[148] Ian H. Witten, Zane Bray, Malika Mahoui, and William J. Teahan. Using language models for generic entity extraction. In *Proceedings of the International Conference on Machine Learning Workshop on Text Mining*, 1999.

[149] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, San Francisco, 1999.

[150] Ian H. Witten, Rodger J. McNab, Steve Jones, Mark Apperley, David Bainbridge, and Sally Jo Cunningham. Managing complexity in a distibuted digital library. *IEEE Computer*, 32(2):6, Feburary 1999.

[151] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes — Compressing and Indexing Documents and Images*. Morgan Kaufmann, 2nd edition, 1999.

[152] Ian H. Witten, R. M. Neal, and John G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, June 1987.

[153] Ian H. Witten, Craig G. Nevill-Manning, Rodger J. McNab, and Sally Jo Cunningham. A public library based on full-text retrieval. *Communications of the Association for Computing Machinery*, 41(4):71–75, April 1998.

[154] Lauren Wood. Document object model (dom) level 1 specification. Technical report, World Wide Web Consortium (W3C), 1 October 1998.

[155] World Wide Web Consortium. *XSL Transformations (XSLT)*, version 1.0 edition, 16 November 1999.

[156] The World Wide Web Consortium (W3C). *RDF/XML Syntax Specification*, 10 February 2004.

[157] Dekai Wu and Pascale Fung. Improving chinese tokenization with linguistic filters on statistical lexical acquisition. In *Proceedings of the Fourth ACL Conference on Applied Natural Language Processing*, Stuttgart, Germany, 13–15 October 1994.

[158] Stuart Yeates. Automatic extraction of acronyms from text. In Stuart Yeates, editor, *Third New Zealand Computer Science Research Students' Conference*, pages 117–124, Hamilton, New Zealand, April 1999. University of Waikato.

[159] Stuart Yeates. *Colloquial Entropy Markup (CEM) Documentation*. University of Waikato, javadoc package edition, 2005.

[160] Stuart Yeates, David Bainbridge, and Ian H. Witten. Using compression to identify acronyms in text. In Storer and Cohn [131], page 582. A longer version of this appears as [161].

[161] Stuart Yeates, David Bainbridge, and Ian H. Witten. Using compression to identify acronyms in text. Working Paper 00/01, Department of Computer Science, University of Waikato, Hamilton, New Zealand, January 2000. A short version of this appears as [160].

[162] Stuart Yeates and Ian H. Witten. On tag insertion and its complexity. In Ah-Hwee Tan and Philip Yu, editors, *International Workshop on Text and Web Mining: Pacific Rim International Conference on Artificial Intelligence 2000*, pages 52–63, Melbourne, Australia, 28 August 2000.

[163] Stuart Yeates, Ian H. Witten, and David Bainbridge. Tag insertion complexity. In Storer and Cohn [132], pages 243–252.

[164] Stuart A. Yeates. The relationship between hidden markov models and prediction by partial matching models. In *Eighth Annual New Zealand Engineering and Technology Postgraduate Conference*, Hamilton, New Zealand, 30–31 August 2001. University of Waikato.

[165] Jeonghee Yi and Neel Sundaresan. Mining the web for acronyms using the duality of patterns and relations. In *Proceedings of the Second International Workshop on on Web Information and Data Management*, pages 48–52, Kansas City, Missouri, USA, November 2–6 1999. ACM.

[166] Steve J. Young. The HTK Hidden Markov model ToolKit: Design and philosophy. Technical Report CUED/F-INFENG/TR.152, Department of Engineering, Cambridge University, September 1994.

[167] Shilong Yu, Shuanhu Bai, and Paul Wu. Description of the Kent Ridge digital labs system used for muc-7. In *Proceedings of the Fifth Workshop on Very Large Corpora*, 1997.

[168] Xiaodan Zhu, Mu Li, Jianfeng Gao, and Chang-Ning Huang. Single character chinese named entity recognition. In *Second SIGHAN Workshop on Chinese Language Processing*, Sapporo, Japan, 7–12 July 2003.

[169] George Kingsley Zipf. *Human behavior and the principle of least effort; An introduction to human ecology*. Addison–Wesley, Massachusetts, USA, 1949. Republished 1965.

Appendix A
Corpora Samples

This appendix contains samples from each of the corpora discussed in Chapter 5 and used throughout this thesis. For reasons of space, documents have been abbreviated, an ellipsis marks a point at which content has been removed. All documents are presented after preparation rather than in the state in which they were received.

A.1 Computists' Corpus

The following is an issue from the Computists' corpus. The corpus is described in Section 5.1.

```
<issue>____________________________________________________________
AI Vol. 8, No. 1.1
IS <d>January 6, 1998</d>
CS <s>THE COMPUTISTS' COMMUNIQUE</s>
"Careers beyond programming."
1&gt;&gt; <o>NSF</o>news.
2&gt;&gt; Other funding.
3&gt;&gt; Career jobs.
____________________________________________________________
In the beginning the Universe was created. This has made
a lot of people very angry and been widely regarded as a bad move.
– <n>Douglas Adams</n>. [<s>QotD</s>, <d>16Oct97</d>.]
Greetings, Computists!
The <s>Computists' Communique</s>will now arrive three times
<d>per week</d>, on Tuesdays, Wednesdays, and Thursdays. Issues
will be shorter, for easy reading, and may vary a bit in length.
Part of each Wednesday issue will be a table of contents for
<d>that day</d>'s CAJ jobs digest. (You can request the digest issue
if you see an interesting opportunity.) I'm reducing the number
of <d>publication weeks</d>to 40 (or 120 issues!), to give me more time
for Web work and other activities. That means there will be
about <d>one week</d><d>each month</d>with no <s>Communique</s>s, usually
with
a holiday or at the end of <d>the month</d>. All to serve you better,
of course, but do get in touch with me if you have suggestions
about the changes.
Membership fees will hold steady at <d>last year</d>'s level,
but with a new "departmental rate" for groups of up to five
```

participants. At <m>$195</m>per year (or half of that outside the <l>US</l>),
it should be attractive to lab directors and other group leaders.
(Please circulate copies of the <s>Communique</s>to the appropriate
people. They can write to me or visit &lt;<u>http://www.computists.com</u>&gt;
to check out the service.) Members may offer <d>two-month</d>
free trials to friends, or <d>three-month</d>free trials
(excluding their own dues) for groups.
My wife <n>Lily</n>will be taking over some of the renewal
billing communication, and will be getting in touch with you.
The captain is on holiday, but his "cool job of <d>the week</d>"
should return in <d>a week</d>or two. (Sometimes he just doesn’t
find a cool enough job.) We’re taking care of business,
so have a fun and prosperous <d>new year</d>!
1&gt;&gt; <o>NSF</o>news:
<o>NSF</o>’s Awards for the Integration of Research and Education
at Baccalaureate Institutions program will make 10-20 awards of
up to <m>$500K</m>. Eligibility is restricted to Carnegie Classification
Baccalaureate I and II institutions and Specialized Technical
institutions that award only baccalaureate degrees. Deadlines
are <d>04Feb98</d>for letters of intent, <d>17Mar98</d>for preliminary
applications, and <d>17Jun98</d>for full applications.
&lt;<u>http://www.nsf.gov/od/osti</u>&gt;. [<s>grants</s>, <d>23Dec97</d>.]
<o>NSF</o>’s CISE and ENG directorates have a Combined
Research-Curriculum Development (CRCD) Program to support
dynamic, relevant engineering and CS/IS education.
<d>31Mar98</d>deadline.
&lt;<u>http://www.nsf.gov/cgi-bin/getpub?nsf9838</u>&gt;.
[<n>Maria Zemankova</n>&lt;<e>mzemanko@nsf.gov</e>&gt;, <s>dbworld</s>,
<d>30Dec97</d>.]
...
I have been poor and I have been rich. Rich is better.
– <n>Sophie Tucker</n>, American singer. [<s>DailyQuote</s>,
<d>02Jan98</d>.]
</issue>

A.2 Bibliography Corpus

The following is a bibliography from the bibliography corpus. The corpus is described in Section 5.2.

```
<bibliography xsi:schemaLocation="http://www.greenstone.org"
filename="/research/say1/bib/tmp_bib_files/graphics/2748.bib"><plain><bibproc>
<p></p>References
<p></p>[1] <bibbody><article><author><name><first>Till
B.</first><last>Anders</last></name>and
<name><first>Wolfgang</first><last>Jachmann.</last></name></author><title>Cross
sections with polarized spin-1over2 particles in terms of helicity
amplitudes.</title><journal><emphasis>Journal of Mathematical
Physics,</emphasis></journal><volume>24</volume>(<number>12</number>):<pages>2847-
2854,</pages><date><month>December</month><year>1983</year></date>.</article></bibbody>
<p></p>[2] <bibbody><article><author><name><first>V.
G.</first><last>Bagrov,</last></name><name><first>V.
V.</first><last>Belov,</last></name>and <name><first>I.
M.</first><last>Ternov.</last></name></author><title>Quasiclassical
trajectory-coherent states of a particle in an arbitrary electromagnetic
field.</title><journal><emphasis>Journal of Mathematical
Physics,</emphasis></journal><volume>24</volume>(<number>12</number>):<pages>2855-

2859,</pages><date><month>December</month><year>1983</year></date>.</article></bibbody>
...
<p></p>[25] <bibbody><article><author><name><first>W.
M.</first><last>Zheng.</last></name></author><title>The Darboux
transformation and solvable double-well potential models for Schrodinger
equations.</title><journal><emphasis>Journal of Mathematical
Physics,</emphasis></journal><volume>25</volume>(<number>1</number>):<pages>88-
90,</pages><date><month>January</month><year>1984</year></date>.</article></bibbody>
<p></p>Page <pagematter>2 </pagematter>
</bibproc></plain></bibliography>
```

If the output is indented to show the full structure, it appears as:

```
<bibliography xsi:schemaLocation="http://www.greenstone.org" filename="/research/say1/bib/tmp_bib_files/graphics/2748.bib">

 <plain>
  <bibproc>

    <p> </p> References

    <p> </p> [1]
    <bibbody>
      <article>
        <author>
```

```
            <name>
              <first> Till B.</first>
              <last> Anders</last>
            </name>  and
            <name>
              <first> Wolfgang</first>
              <last> Jachmann.</last>
            </name>
          </author>
          <title> Cross
sections with polarized spin-1over2 particles in
terms of helicity amplitudes.</title>
          <journal>
            <emphasis> Journal of Mathematical
Physics,</emphasis>
          </journal>
          <volume> 24</volume> (
          <number> 12</number> ):
          <pages> 2847-2854,</pages>
          <date>
            <month> December</month>

            <year> 1983</year>
          </date> .</article>
      </bibbody>

      <p> </p> [2]
      <bibbody>
        <article>
          <author>
            <name>
              <first> V. G.</first>
              <last> Bagrov,</last>
            </name>
            <name>
              <first> V. V.</first>
              <last> Belov,</last>
            </name>  and
            <name>
              <first> I. M.</first>
              <last> Ternov.</last>
            </name>
          </author>

          <title> Quasiclassical trajectory-coherent states of
a particle in an arbitrary electromagnetic field.</title>

          <journal>
            <emphasis> Journal of Mathematical Physics,</emphasis>
          </journal>
          <volume> 24</volume> (
          <number> 12</number> ):
          <pages> 2855-
2859,</pages>
          <date>
            <month> December</month>
            <year> 1983</year>
          </date> .</article>
      </bibbody>



...

      <p> </p> [25]
      <bibbody>
        <article>
          <author>
            <name>
              <first> W. M.</first>
              <last> Zheng.</last>
            </name>
          </author>
          <title> The Darboux transformation
and solvable double-well potential models for
Schrodinger equations.</title>
          <journal>
            <emphasis> Journal of Mathematical
Physics,</emphasis>
          </journal>
          <volume> 25</volume> (
          <number> 1</number> ):
          <pages> 88-90,</pages>
          <date>
            <month> January</month>
            <year> 1984</year>
          </date> .</article>
      </bibbody>

      <p> </p> Page
      <pagematter> 2

</pagematter>
</bibproc>
  </plain>
</bibliography>
```

A.3 Segmentation Corpus

The following is a single document from the segmentation corpus. Whitespace appearing here is a side-effect of layout, the only whitespace in the original file is a terminal EOL. The corpus is described in Section 5.3.

```
<document>
<word> &#20551;</word> <word> &#26230;&#21326;</word> <word>
&#39277;&#24215;</word> <word> &#20030;&#34892;</word> <word>
&#39041;&#22870;</word> <word> &#20856;&#31036;</word> <word>
&#65292;&#21040;&#24213;</word> <word> &#30495;&#30456;</word> <word>
&#22914;&#20309;</word> <word> &#21602;</word> <word>
&#65311;&#19968;</word> <word> &#12289;</word> <word>
&#36164;&#26684;</word> <word>
&#65306;&#19969;&#32903;&#20013;</word> <word>
&#38498;&#22763;</word> <word> &#21363;</word> <word> &#22240;</word>
<word> &#39318;&#20808;</word>
...
</document>
```

A.4 Reuters' Corpus

The following is a single document from the Reuters' corpus. The corpus itself is described in Section 5.4.

<document> <NNP> PDCP</NNP> <NNP> Development</NNP> <NNP> Bank</NNP> <VBD> said</VBD> <IN> on</IN> <NNP> Thursday</NNP> <PRPSTRING> its</PRPSTRING> <NN> board</NN> <VBD> approved</VBD> <DT> the</DT> <NN> issue</NN> <IN> of</IN> <CD> one</CD> <CD> billion</CD> <NN> pesos'</NN> <JJ> worth</JJ> <IN> of</IN> <JJ> convertible</JJ> <JJ> preferred</JJ> <CD> shares.</CD>

<DT> The</DT> <NNS> proceeds</NNS> <IN> of</IN> <DT> the</DT> <NN> issue</NN> <MD> will</MD> <VB> fund</VB> <NN> lending</NN> <NN> operations,</NN> <NN> computerisation,</NN> <CC> and</CC> <VBG> refurbishing</VBG> <IN> of</IN> <NN> branch</NN> <NN> offices,</NN> <PRP> it</PRP> <JJ> said.</JJ>

...

</document>

Text Augmentation: Inserting markup into natural language text with PPM Models

A thesis
submitted in partial fulfilment
of the requirements for the degree
of
Doctor of Philosophy
at the
University of Waikato
by

Stuart A. Yeates

Department of Computer Science
The University of Waikato
Te Whare Wānanga o Waikato
Hamilton, New Zealand

July 9, 2006

Abstract

This thesis describes a new optimisation and new heuristics for automatically marking up XML documents. These are implemented in CEM, using PPM models. CEM is significantly more general than previous systems, marking up large numbers of hierarchical tags, using n-gram models for large n and a variety of escape methods.

Four corpora are discussed, including the bibliography corpus of 14682 bibliographies laid out in seven standard styles using the BIBTEX system and marked-up in XML with every field from the original BIBTEX. Other corpora include the ROCLING Chinese text segmentation corpus, the Computists' Communique corpus and the Reuters' corpus. A detailed examination is presented of the methods of evaluating mark up algorithms, including computation complexity measures and correctness measures from the fields of information retrieval, string processing, machine learning and information theory.

A new taxonomy of markup complexities is established and the properties of each taxon are examined in relation to the complexity of marked-up documents. The performance of the new heuristics and optimisation is examined using the four corpora.

Keywords: hidden Markov models, HMM, PPM, Viterbi search, part-of-speech tagging, XML, markup, metadata.

Dedication

To Jacqui,

my trapping state.

Acknowledgements

Thank you my family, for always being there.

Thank you David, Ian, Sally Jo and Matt for guidance, encouragement and technical help.

Thank you to the Royal Society of New Zealand for funding through the Marsden Fund.

Thank you to Reuters for the use of ‘Reuters Corpus, Volume 1, English language, 1996-08-20 to 1997-08-19’. Thank you to the ROCLING SIGIR for the use of the ROCLING corpus. Thank you to Kenneth I. Laws for the use of the Computists’ Communique.

Thank you to Pauline for handling the long-distance submission.

Thank you my fellow students Carl, Catherine, Dana, Dave, David, Geoff, Gordon, Hayley, Imene, Jack, John, Justin, Karl, Kathy, Lin-Yi, Mark, Mark, Shane, Stuart, Yingying, and everyone else in the New Zealand Digital Library research group. Thank you to the tutoring, secretarial and technical staff.

Thank you Aimee, Aliene, Amanda, Andraus, Andrew, Andrew, Andy, Anne, Anne, Barry, Belinda, Bill, Bob, Brent, Bret, Carolee, Caroline, Chris, Christine, Christine, Christine, Dale, Dave, Dave, David, David, Deborah, Dee, Des, Douglas, Erin, Erin, Gail Gayle, Gaylene, Georgina, Haylee, Ian, Jacqui, Jane, Janice, Jenny, Jenny, Kay, Kay, Kirsten, Kumar, Lee, Leigh, Leo, Linda, Lyn, Mandy, Matt, Maz, Micheal, Murray, Rachel, Rachel, Rachel, Rhonda, Roland, Rosie, Sam, Sam, Sara, Sarah, Shauna, Stuart, Sue, Terri, Terry, Terry, Toni, Tony, Wayne, Wendy and everyone else I’ve danced with in Christchurch, Hamilton, Auckland, London and Oxford during the course of my enrolment.

Thank you to OUCS at Oxford for the use of their resources to finish this thesis. Thank you to all the RTS crew for their encouragement. Thanks to Sebastian for the LaTeX and XML help.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1
Introduction

Timely news is in great demand, and the value increases if the news is tightly focused on specific areas of interest to the readers. Often readers are interested in specific organisations, dates and sources, so the fragment:

> *1997 was a record spending year for computer-industry mergers and acquisitions, and companies such as Compaq, Dell, IBM, and Hewlett-Packard are still hot to buy what's left. [InfoWorld Electric, 24Dec97. EduP.]*

might be considerably more valuable to a reader if the organisations, dates and sources of information were marked up with <*o*> <*d*> and <*s*> tags, respectively:

> <*d*>*1997*</*d*>*was a record* <*d*>*spending year*</*d*>*for computer-industry mergers and acquisitions, and companies such as* <*o*>*Compaq*</*o*>*,* <*o*>*Dell*</*o*>*,* <*o*>*IBM*</*o*>*, and* <*o*>*Hewlett-Packard*</*o*>*are still hot to buy what's left. [*<*s*>*InfoWorld Electric*</*s*>*,* <*d*>*24Dec97*</*d*>*.* <*s*>*EduP*</*s*>*.]*

The extraction of references to company names in particular forms the backbone of systems such as `finance.yahoo.com`, which aggregate news from many hundreds of sources into thousands of tightly focused categories.

Languages such as Chinese and Japanese are usually written without whitespace segmenting the characters into words. One of the first operations that must be performed by many information systems dealing with such text is to augment it with segmentation information, for example: 小学校屋内運動場建設 is augmented to

<w>小学校</w><w>屋内</w><w>運動</w><w>場</w><w>建設</w>, ([elementary school][building interior][sports][area][construction], i.e. the construction of an elementary school indoor sports arena). Such segmented text can then be used in all the ways that words from a western language can be used [3]. The tags can then be discarded to display the text in the original form or used to process the text in the word-by-word fashion common to most western information systems, or some combination of the two.

There are many thousands, perhaps many millions, of peer-reviewed academic papers available on the Internet, each with bibliographic entries linking it to other papers and materials, for example:

> *Donald E. Knuth. Semantics of context-free languages. Mathematical System Theory, 1968, 2(2), 127–145.*

A competent researcher or librarian can readily separate this entry into all the parts necessary to find the document to which it refers. When there are collections of thousands of electronic documents, separating these manually is a huge, tedious and error-prone task. What would be useful would be a system that took the entry and automatically augmented it as:

```
<entry>
        <author>
                <forenames>Donald E.</forenames>
                <surname>Knuth </surname>.
        </author>
        <title>Semantics of context-free languages</title>.
        <journal>Mathematical System Theory</journal>,
        <year>1968</year>,
        <volume>2</volume>(<number>2</number>),
        <pages>127-145</pages>.
</entry>
```

Data in such an augmented format could then be used in a number of operations, including interloaning a copy of the document, reformatting the reference for inclusion in another bibliography, citation analysis and querying by date.

Digital library software is increasingly interacting with non-computer specialists on their own terms. This can be done using generic interfaces (witness the success of the slim-line Google interface) or interfaces tailored to the domain of the users or the content. In order to provide this, the digital library needs to know what those terms are and how they apply to the documents in the collections, whether they are organisations, dates and sources or authors, titles and dates of publication. Manual augmentation with this knowledge is typically expensive, slow and inconsistent.

This thesis describes a method for automating text augmentations for a large class of problems covering all of these examples. Such text augmentation is performed by building models from training text marked-up with XML tags, then using the models and searching to insert similar tags into testing text that does not yet contain any tags. Building effective models requires considerable volumes of training text with consistently used tags, and that the training text be representative of testing text. The text augmentation described in this thesis covers a broader range of information than preceding approaches, but is shallower than most information extraction systems in that all reasoning is fine-grained, with no higher-level or document-level reasoning, limiting the text augmentations that can be attempted.

The quality of text augmentation is evaluated by splitting a marked-up corpus into a set of training documents and a set of testing documents; training a model on the former; stripping the tags from the latter; augmenting the stripped testing documents using the model; and finally comparing the testing documents as augmented by the system with the original documents. Several different methods to compare the augmented document to the original are explored in this thesis.

1.1 Plan of the Thesis

Following this introduction, Chapter 2 gives the background to the current work, starting by examining the nature of text and an overview of methods of extracting information from text. Approaches to evaluating the correctness and efficiency of the such extractions are then examined, together with ways on encoding extracted information in XML. Chapter 3 introduces Markov models built from text, and algorithms for search using such models to extract information.

Chapter 4 discusses the architecture of the implemented system, and examines the rationale for some of the design choices. It then presents an optimisation and a number of heuristics, and examines the search spaces of different classes of problems with respect to these. Chapter 5 introduces the corpora used in this thesis.

Chapter 6 sets out the experimental results of the optimisation and heuristics on the corpora. Chapter 7 concludes the thesis with an overview of the research, a list of the original contributions, and a summary of unanswered questions. The appendix contains samples from each of the corpora used in this work.

1.2 Thesis Statement

Text augmentation is the automated insertion of XML tags into documents in the context of a digital library to make implicit textual information accessible to conventional processing.

> Text augmentation can be expanded to a larger class of problems than those previously studied. It can be partitioned into three classes of problem: segmentation, classification and entity extraction. Each class of problem has distinctive properties, computational complexity and types of failure, necessitating different evaluation methodologies.
>
> Markov models and searching can be used to solve these problems. Given the context of their application, there are a number of optimisa-

tions and heuristics which can be used to make these algorithms computationally tractable.

Text augmentation is a computational process by which natural language text is augmented by the addition of XML tags to elucidate the implicit structure. Three different classes of text augmentation are discussed. Each class has a structurally different schema which affects the performance and evaluation of text augmentation.

Text augmentation is performed using statistical modelling techniques, such as hidden Markov and PPM models, and using searching algorithms to find a good augmentation. In the past, text augmentation has been performed using Teahan search (see Section 4.5), but in this thesis a variety of algorithms is used. Viterbi search is computationally intractable in many interesting text augmentation situations, but an optimisation of it, and a number of heuristics to it, can be exploited, given the application, to make searching computationally feasible.

To these ends, this thesis aims to:

1. Examine text augmentation problems, in the large, to attempt to determine which are susceptible to automated text augmentation and whether some sets of problems are inherently easier than others.

2. Build a text augmentation system capable of solving at least as wide a range of problems as existing low-human-input systems, with an eye to eventual inclusion as part of a digital library system.

3. Locate and/or build corpora to test this system.

4. Find specific heuristics and optimisations which perform well in relation to a particular set of augmentation problems.

5. Evaluate both the text augmentation system and the heuristics and optimisations in the system.

These aims are reviewed in Section 7.1.

Chapter 2

Background

This chapter examines the background to the current work. First it looks at the nature of text, various types of ambiguities in natural language text and then examines metadata, namely explicit information about text. Information extraction systems, whose purpose is to extract metadata from text, are then surveyed and various methods of evaluating such extraction systems are examined, together with methods of evaluating the correctness and efficiency of such systems. Finally, aspects of XML and Unicode relevant to text augmentation are surveyed.

2.1 The nature of text

One task in text augmentation is the Chinese text segmentation problem, the task of segmenting a stream of Chinese characters into words. The task is often the first step in Chinese information processing systems, since Chinese is normally written without explicit word delimiters. The task is made more challenging by the fact that line delimiters may occur anywhere, including between letters in a word or digits in a number [42].

The task is harder than it appears because Chinese text is ambiguous. The text shown in Figure 2.1(a)(i) (taken from [137]) can be segmented as shown in (ii) or as shown in (iii), meaning ‘I like New Zealand flowers’ and ‘I like fresh broccoli’ respectively. Similarly the Japanese title shown in Figure 2.1(b)(i) (taken from [3]) can be segmented as shown in (ii) or as shown in (iii) meaning ‘president both business and general manager’ and ‘president (of) subsidiary business (for) (the proper

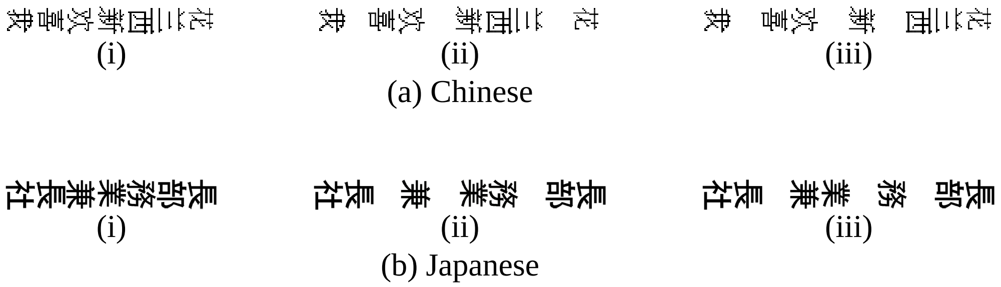


Figure 2.1: Examples of segmentation ambiguity in east Asian languages.

name) Tsutomu, general manager' respectively. Since this last is four nouns and thus identical from the point of view of a part of speech system, it is a particularly ambiguous situation.

2.1.1 Ambiguity

Segmentation ambiguity is not confined to Asian languages. There is a widely circulated joke featuring sentence segmentation ambiguity in English:

> Dear John: I want a man who knows what love is all about. You are generous, kind, thoughtful. People who are not like you admit to being useless and inferior. You have ruined me for other men. I yearn for you. I have no feelings whatsoever when we're apart. I can be forever happy—will you let me be yours? Gloria

and

> Dear John: I want a man who knows what love is. All about you are generous, kind, thoughtful people, who are not like you. Admit to being useless and inferior. You have ruined me. For other men, I yearn. For you, I have no feelings whatsoever. When we're apart, I can be forever happy. Will you let me be? Yours, Gloria

There is an entire class of English expression, double entendre, which exploits ambiguity of meaning [128]. This ambiguity is resolved using context—the style and genre of a piece of text. A sentence with two possible meanings has the more risqué meaning if it appears in a Blackadder [38] script and has the less risqué of the two if it appears in a Reuters' dispatch. There are also forms of text in which resolving ambiguity of meaning is not possible, a well-known example of which is Lewis Carroll's poem 'Jabberwocky'.

Ambiguity resolution using context is an example of what is known in artificial intelligence as 'common sense reasoning'. It is known to be difficult for computers to resolve such ambiguity, with the difficulty lying in the wide range of world-knowledge and subtle reasoning that humans use to solve this class of problem [107].

Partly to reduce the need for ambiguity resolution, the overwhelming majority of text mining is performed on collections of text with uniform style and genre. Uniformity of linguistic style highlights the patterns and structures within the text and the uniformity of genre ensures that the patterns have the same meanings.

2.1.2 Metadata

Metadata means 'a set of data that describes and gives data about other data' [128]. Usually at the granularity of the document (the catalogue entry for a book or the title and author of a web page), metadata can be at the character level [5] or cover entire collections of documents (Table 2.1). In many systems and standards much of the metadata is stored at the document level, even though it may apply to the collection, section or even character level, because this is the level at which most processing, storage, licensing, retrieval and transmission operations take place. The RDF standard [156] is notable for granularity independence, addressing, individual tags (elements), documents or collections of documents.

This thesis centres on fine-grained metadata, at the character and word levels,

Granularity	Relevant metadata
Collection	Scope; purpose; coverage; copyright; maintenance status; maintainer contact details;
Document	Author; title; date of publication; subject classification;
Section	Topics; cross references;
Sentence	Semantic meanings;
Word	Part of speech; glossary links; dictionary links; collation order;
Character	Encoding; reading direction; case;

Table 2.1: Metadata at different granularities.

and how such metadata can be inferred from, and then annotated into, the text itself. This process of augmenting the text is referred to as text augmentation. It has been previously called 'tag insertion' [136, 135], but the author believes that *'text augmentation'* better portrays the action and intent of the process.

2.2 Extraction of Textual Information

A wide range of distinct approaches and many hybrid ones have been used to extract fine-grained information from text for various purposes. This section reviews several of them, including regular expressions, machine learning and information extraction. The following section examines how to measure the correctness of the extraction.

2.2.1 Regular Expressions

Regular expressions are compact representations of a set of strings which can be converted into a finite-state machine. The machine can efficiently recognise instances of the set of strings within a stream of text. Their close relationship to the well-studied field of formal language parsing has led to them being well understood [2].

Regular expressions are the tool of choice for extracting information with an exact and precise format, such as email addresses, post codes, dates and the like.

They are, however, fragile in the face of mistakes, ambiguity and stylistic variations in the text.

2.2.2 Handcrafted Rules

Handcrafted rules or templates can also be used to extract information from text. These typically involve searching for short fragments of text or regular expressions within text, with each rule processed in order of precedence. Unfortunately, systems of handcrafted rules can be complex and fragile in the face changing input data. They also scale poorly with the number classes of information being extracted, particularly when there is a requirement that rules do not overlap.

These systems typically can consider large windows and potentially have access to 'out of band' sources of information such as dictionaries and name lists [17, 1, 74].

2.2.3 Instance Based Machine Learning

Instance based machine learning is a field concerned primarily with classifying instances into classes. Machine learning can be applied to text [149], but requires that the text be pre-segmented into instances, potentially losing significant information and/or leading to large instances.

Machine learning handles noise and ambiguity significantly better than regular expressions. Mis-classified instances, once detected, can be added incrementally to the training instances, allowing an existing model to be refined and improved. The widely-used Brill tagger [28] uses this approach as a primary method.

2.2.4 Information Extraction

The field of information extraction typically involves multi-step systems that first extract atoms from text (using regular expressions, part-of-speech tagging, etc.) and then use higher-order reasoning to solve ‘real world’ problems. The Text REtrieval Conferences series (TREC) [53, 54, 142, 143] is built round text retrieval tasks and the Message Understanding Conferences (MUC) and Document Understanding Conferences (DUC) are built around competitions between systems. The intent is to focus research and systems development towards specific, known targets.

MUC Named Entity [35] problems centre on the extraction of proper nouns (e.g. company names), often with subsidiary information (e.g. market symbols or addresses) from stylised information sources, typically news articles such as the Reuters’ corpus. The problems set in the MUC tracks explicitly required the extraction of facts from the texts into a separate database and subsequent higher-order reasoning about those facts, in two separate systems. Many involve multiple steps, such as sentence and word segmentors, part-of-speech taggers, hypothesis generators, hypothesis evaluators and disambiguators [167].

The systems include many opportunities for encoding handcrafted or externally curated domain knowledge, from the notion of the word embedded in the word segmentors, to domain-specific word lists used in the part-of-speech tagger and handcrafted heuristics for template filling. Word lists include lists of first names, corporate names, colleges and universities, corporate suffixes, times and dates, world regions and state codes [23]. Many of the systems use trained models, either learnt rules or Markov models, but only for an individual step of solving the problem.

Many of these systems and corpora suffer from proper-noun ambiguity errors (see Sections 2.1.1 and 5.1). Methods employed to overcome the ambiguity include leveraging company and personal titles (*Mr*, *Ltd* and *Corp.*) [22]) and deeper parsing to detect structures such as standard formatting of place names.

The GATE system is a Java GUI framework for linguistic engineering. It incorporates a wide variety of tools for using hand- or tool-generated rules, and regular expressions and links to gazetteers of cities and organisations. Testing and evaluation tools are included for classification problems. GATE focuses on the inclusion of extra-textual information:gazetteers, word-lists, grammars and similar, and their interactive development to solve particular problems. It also has tools for higher-level reasoning about texts[1] [37, 22, 95]. GATE's choice to have a GUI enables it to allow display and input of multiple texts and scripts: 21 are supported.

Citeseer [80] uses a two-stage approach, with an edit distance metric to merge similar references across the entire collection and then a hand-crafted 'invariants first' heuristic that parses those parts of the reference with the fewest differences first and uses standard machine learning on them. The system was able to leverage two extra-document sources of information, tables of common western personal names and repetition of the same reference (often in slightly different form) in multiple documents. Citeseer does not parse the diversity of fields that occur in the bibliography corpus, instead focusing on the title and author fields which are also extracted from the start of documents and which link most easily to external sources in the bibliography at the end. The public interface of the Citeseer system allows end-users to correct the extracted fields and add the missing ones. It is not clear whether feedback from these corrections is applied to the internal algorithms.

2.2.5 Markov Modelling

A number of systems and approaches have used Markov models to extract information from, or add information to, text. The early Xerox tagger [40, 39] uses hidden Markov models and Viterbi search to good effect, but handles unseen words and novel contexts poorly.

Built using arithmetic encoder [102] models, one for 'good' text and one (called a 'confusion model' [36]) for errors, the TMT (Text Modelling Toolkit) and later SMI (Statistical Modelling Interface) systems [134, 36] can correct errors in text and classify textual fragments [133, 26]. With a large number of options and supporting a wide range of static and adaptive models, SMI is entirely capable of solving the

[1] `http://gate.ac.uk/`

news and Chinese examples given in the opening Chapter, but not the bibliographic example, because SMI models are not recursive; they cannot represent a hierarchy of textual fragments.

Arithmetic encoder models provide slightly more information than conventional Markov models, providing an ordering of symbols as well as probabilities represented using integer ratios. Integer ratios avoid using floating-point arithmetic to whose inaccuracies arithmetic encoding is particularly sensitive. These steps make SMI useful for both textual augmentation and full text compression.[2]

Freitag and McCallum [46, 96] report work on a bibliography corpus using hand-crafted, then automatically shrunk, Markov models, giving good results. Freitag and McCallum build models with increasingly complex structures in a similar manner to Dynamic Markov Compression (DMC) [151], which are then blended using linear combination.

Recently Besagni et al. [15] have had some success in marking up bibliographies using part of speech tagging, building chains of which parts of speech occur in which bibliographic fields and then correcting fields using a post-processing step. As with the post-processing performed in part of speech tagging, this includes super-adjacency. They use six tags and get a recall (see Section 2.3.1) of between 82% and 97% of the time for a corpus of 2500 references. Not all of the failures are complete failures, since sometimes part of a name is successfully returned. This may be useful, depending on the context.

2.2.6 Trained versus Handcrafted Models

The use of automatically trained models rather than handcrafted models lends itself to use in situations where training data is cheaper or more accessible than domain-knowledgeable humans. With the increasing volumes of data available at the cost of transfer on the Internet and the relatively stable cost of labour, using large amounts

[2] 'Full text compression' in this context means lossless compression, as opposed to the lossy compression often used for images which effectively destroys text [151].

of training data rather than people is likely to be an increasingly attractive choice.

While much of the freely available material for training models is of low or questionable quality, the existence and growth of curated repositories such as the Oxford Text Archive,[3] the Linguistic Data Consortium[4] and Project Gutenberg[5] suggest that the availability of curated textual and linguistic materials is increasing.

There are limits on what trained models can recognise, because of the finite training text available, their lack of 'common sense' reasoning and various theoretic limits [13]. For example, most model training and template building systems cannot recognise structures characterised by n *a*'s then n *b*'s followed by n *c*'s. While systems can be built to recognise these structures for a particular n, it is not possible to recognise these structures for unknown n's with a regular expression while rejecting structures with different numbers of *a*'s, *b*'s and *c*'s. These limits do not apply to handcrafted models. Handcrafted models run into the well-known difficulties of hand-building large, complex systems [83] and labour costs.

Building and maintaining a set of handcrafted rules or a handcrafted model may be more cost effective than building a corpus of documents with the concepts marked-up if the documents are sufficiently rare or sufficiently difficult to handle (for example they contain embedded private or confidential information). Handcrafting is also more attractive if the concept is well understood by non-specialists, meaning labour is relatively cheap.

Trained models also have the option of automated incremental improvement by using the Baum–Welch algorithm [10, 11] in production situations. Long-term use of Baum–Welch may result in divergence and poor performance. However, if the data seen in production is changing at a rate faster than this divergence, then using the Baum–Welch may be advantageous. This thesis focuses on trained models.

[3] `http://ota.ahds.ac.uk/`

[4] `http://www.ldc.upenn.edu/`

[5] `http://www.gutenberg.org/`

2.2.7 Single Step versus Multiple Step Systems

Multiple step text augmentation systems have an advantage over single step systems in allowing a different choice of algorithm for each step, providing the system builders with a wider range of options and making the intermediate forms accessible for ‘boosting’ using word lists and similar. A wider range of choices for systems builders enables them to hand-select algorithms that perform well on the expected input for the systems. Unfortunately, this often leads to poor performance on other input: other genre, other character encodings and other languages.

Multiple step text augmentation systems also encourage reuse of system components, such as the Brill part-of-speech tagger, which is widely used as a preprocessor [37]. Single step augmentation systems can be reused as a whole, but are not as amenable to the development of UNIX-style ‘pipelines’. Corpora used to train models and rules are amenable to incremental development, either by adding additional documents of the same type or by adding documents in additional languages, as is common in corpora used in comparative linguistics. Steps can also be arranged in a cascade or waterfall [68].

This thesis focuses on single-step markup processes using Markov models. There is no theoretical reason why the systems and approaches used here could not be used as individual steps within a multiple system, but training data for the intermediate stages appears to be rarer, except where the individual step has already been studied in isolation, as with part-of-speech tagging.

2.3 Correctness

The ultimate test of a computer system is in terms of interactions with users—does the system work correctly? Are any errors made, minor or catastrophic? Is it fast enough? Is it easy to use? Do the users like it? These questions, however, are hard to phrase in terms that allow the answers to be compared among systems, versions of the same system, and software packages across time in the face of changing requirements, user expectations, groups of users and operating environments. They are also hard to ask of sub-systems that provide a subset of functionality required by a full system.

There are, however, two features of overall performance which are widely used for comparing systems: correctness and efficiency. This section examines these and how they can be applied to text augmentation.

The approaches to measuring correctness examined here come from the fields of information retrieval, string processing, machine learning and information theory.

2.3.1 Recall and Precision

The information retrieval paradigm [122, 6] assumes that a query (single operation) retrieves a set of items, some of which are relevant to the query. Evaluation is based around the question 'Is item n relevant and was it returned?' The answer to this question puts each item into one of four distinct classes: true positive (relevant and retrieved), true negative (not relevant and not retrieved), false positive (not relevant and retrieved) and false negative (relevant and not retrieved).

Accumulating counts of each of these four classes over a large number of independent experiments allows the calculation of two higher-level measures. Recall [31] is the proportion of all relevant items that were retrieved:

$$Recall = \frac{number\,of\,relevant\,items\,retrieved}{total\,number\,of\,relevant\,items\,in\,collection} = \frac{true\,positives}{true\,positives + false\,negatives}$$

Precision is the proportion of retrieved items that are relevant:

$$Precision = \frac{number\,of\,relevant\,items\,retrieved}{total\,number\,of\,items\,retrieved} = \frac{true\,positives}{true\,positives + false\,positives}$$

Recall and precision represent a trade-off. A system could return many items (for high recall and low precision) or few items (for low recall and high precision) and so they are sometimes expressed as their harmonic mean:

$$F - measure = \frac{2 \times recall \times precision}{recall + precision}$$

Often the number of false negatives is unknown, such as when retrieving documents from the World Wide Web, whose exact size is unknown but large [81]. When the number of false negatives is known (or can be reliably estimated), another measure, called 'Fallout' [84], which is a measure of how good the result is as a result for the negated query, can be used:

$$Fallout = \frac{number\, of\, irrelevant\, items\, retrieved}{total\, number\, of\, irrelevant\, items\, in\, collection} = \frac{false\, positives}{false\, positives + true\, negatives}$$

Fallout measures how effectively irrelevant items are winnowed from the query results. Fallout is rarely used, as it is sensitive to the size of the collection and the addition of clearly-irrelevant items to the collection. Recall, precision, and their combination in the F-measure, are the primary means of evaluating correctness in information retrieval systems.

2.3.2 Edit Distance

Edit distance is a standard technique in the string processing field. It is a well-studied measure used in spelling correction [73, 89] (where transposes are common because of the mechanics of typing) and Optical Character Recognition (OCR) [73] (where swaps are common due to mis-recognition of one character for another). These research fields measure edit distance on data, whereas when used in text augmentation, edit distance is used on combined data and metadata with an expectation that errors be closely linked to the metadata.

Edit distance is performed in terms of individual tags rather than tag-pairs. False negatives (inserts) and false positives (deletes) are counted and then summed to get an edit distance.

Edit distance is solely concerned with mistakes made in text augmentation and neither true negatives nor true positives impact on edit distance. Edit distance explicitly recognises the sequential nature of text but, because true positives are ignored, the independence problems discussed in relation to recall and precision do not occur in edit distance calculation. Teahan [133] uses edit distance to evaluate text augmentation and Nahm et al. [106] uses edit distance as an input to a multi-stage text mining system. All edit distances used in the current work are normalised for document length to give edits per character.

2.3.3 Confusion Matrices

Whereas recall and precision assume an underlying binary classification, confusion matrices are a tool for evaluating many-class classification tasks, and are widely used in machine learning for evaluating such tasks [149]. The following is a confusion matrix for a classification problem with i classes:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,i} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,i} \\ \vdots & \vdots & \ddots & \vdots \\ a_{i,1} & a_{i,2} & \cdots & a_{i,i} \end{bmatrix}$$

The matrix is square, with a row and a column for each class. $a_{m,n}$, in column n and row m, is the number of symbols that should have been classified in class n that were actually classified in class m. Correct classification is indicated when $n = m$, on the leading diagonal of the matrix.

Any non-zero numbers off the leading diagonal, indicate misclassification and there is often symmetry about the diagonal. Non-zero numbers in both $a_{n,m}$ and $a_{m,n}$ indicate that if symbols of class m can be mistaken for symbols of class n, then symbols of class n are also likely to be mistaken for symbols of class m. This ability to highlight confusion between tags makes the confusion matrix an excellent tool for fine-tuning tagsets and finding markup errors. For example, Bray et al. [26] used a confusion matrix to find errors and demonstrate the strong correlation between name tags and place tags in the Computists' corpus. Confusion matrices are conventionally normalised by converting the rows into percentages.

2.3.4 Entropy

Entropy is a measure from information theory widely used in signal processing, error-correction and compression fields of computer science [102, 151]. It is inversely related to probability. A 'good' augmentation of text has a high probability and a low entropy (measured in bits per character) [13].

Unlike other measures of correctness, entropy does not measure results against a predefined answer, but rather measures how closely a set of results matches a model. This is effective in situations where perfect answers are either unobtainable or obtainable only at great expense.

For entropy measures to be an effective measurement of accuracy of an augmentation of text, the model used to measure entropy must be independent of both the testing and training data. This problem is closely related to the over-fitting problem in machine learning, and can be avoided by training two models on separate training data and using one to augment the text and one to measure entropy.

If an independently trained model is unavailable, an untrained model can be used with an adaptive algorithm. This is the standard methodology for measuring the strength of lossless compression algorithms [152, 103, 13].

An entropy measurement is relative to a model, and so conveys little clear knowledge about the absolute quality of an augmentation: the user of augmented text is unable to infer as much from an entropy measurement as from a recall/precision pair or an edit distance. It can, however, be used to compare the relative merit of different augmentations of the same text, provided the model captures pertinent details and the same model is used to calculate both entropy measurements.

2.3.5 Hybrid and Other Measures

Many reports of text augmentation use a combination of measures to report their results. For example Bray [26] decomposed tag insertion evaluation in the Computists' corpus into a pair of operations, firstly segmenting characters into tokens and, secondly, classifying the tokens into their respective types.

The segmentation operation was measured in terms of the error count (false-negatives + false-positives), and classification of the segments was measured using confusion matrices. Other systems use measures expressed in terms of their interaction with larger information systems, such as extraction of acronyms [165] and bibliographies [21].

2.4 Efficiency

Computer programs can be written in a wide variety of computer languages and run on a wide variety of platforms. Since the efficiency of these languages and platforms varies widely, it is useful to compare algorithms independent of their language and platform. One methodology which allows this is time complexity analysis using 'big O notation' [70]. The function is simplified to remove constant factors and is referred to as $\mathcal{O}$.

Time complexity analysis is defined in terms of a characteristic operation—in the case of tag insertion this is visiting a node in the search space—and counting how many times the operation is performed, and expressed as a function of the parameters and input size of the algorithm.

The size of the search space is normalised by the document length to give a measurement in terms of search space per character. There are special cases when searching at the start and end of documents, but for the corpora used in this thesis the initial and final characters in documents are low entropy, so they should not effect this normalisation.

2.5 XML Tags

EXtensible Markup Language (XML) [25] tags have a name (or type), span a (potentially empty) range of text and have a (potentially empty) set of attributes. The tags may be nested, but only strictly hierarchically. Thus, if a document has tags indicating pages from the physical document, it may also have tags indicating lines and, because each line is wholly within a page, the tags are hierarchical. A tag which contains only hierarchical tags, or no tags at all, is said to be well-balanced.

An XML document has an enclosing, top level, tag holding information about the document as a whole. An XML document that is well-balanced is said to be well-formed.

XML cannot directly represent overlapping hierarchies (such as the physical and logical document layout), unlike the preceding SGML [51] which had a feature, CONCUR, which permitted overlapping tags. XML can represent non-hierarchical tags using higher-order structures, using empty tags with attributes which associate them in pairs or in a sequence. The difficulties of tagging overlapping structures, and standard ways of overcoming them, are described in detail in [130].

There are several schema languages for describing which XML tags may occur within other XML tags. The W3C schema language includes an ANY tag to refer to any well-balanced tag [43]. Schemas which feature the ANY tag are flexible but challenging to model, because literally anything can be encoded, including structures equivalent to entire documents of the type being marked up.

2.5.1 Nested Tags

The XML standard largely attempts to avoid statements about the semantics of tags and the semantics of nested tags, other than their well-formedness. It is tempting to extend practice in XHTML to cover XML. In XHTML *<em><a href="…">…</a></em>* is typically considered semantically equivalent from *<a href="…"><em>…</em></a>* because most presentation engines (browsers) present these identically. Presentational customisation systems such as CSS [24] and XSLT [155], however, have no difficulty differentiating these two situations and the XML standard is silent on their semantic relationship. One can imagine a (fictional) programming language expressed in XML in which the semantics are clearly different. For example

```
<if cond="undefined(symbol)">
   <define name="symbol">
       <action/ >
   </define >
</if >
```

has different semantics to

```
<define name="symbol">
    <if cond ="undefined(symbol)">
       <action/ >
    </if >
</define >
```

The current work attempts to avoid making semantic assumptions such as this, except explicitly in the state-tying heuristic (see Section 4.3.7).

2.5.2 Attributes of Tags

The current work focuses exclusively on direct representations and does not consider attributes during training or testing (with the exception of attributes of the document-level node). All of the corpora used in this thesis have been created or transformed, as described above, to convert attributes into tags.

Attributes are syntactic sugar and any XML document with attributes can be transformed into one without attributes and back in a lossless fashion. For example, the tag *<word partofspeech="verb">jump</word>* can easily be transformed to *<word><verb>jump</verb></word>* but such transforms can lead to combinatorial explosion of tags if there are large number of attributes or the attributes contain large numbers of unique values. Real-valued attributes would lead to an infinite number of tags, one for each possible value. If the order of attributes of a tag is significant, the situation is significantly worse. The XML standard is silent on the question of whether the order of attributes is significant, but several subsidiary standards, including XSLT [155] and DOM [154] do not even permit discovery of the order of tags. The author knows of no use of an XML corpus in which the order of attributes is significant or of toolsets which support the processing of such XML.

2.5.3 Other Issues

A key feature XML shares with many other natural language processing approaches is the linearisation of language. While written language across a wide range of cultures is laid out in rectangular regions, whether read left-to-right and top-to-bottom, or bottom-to-top and right-to-left, digitised language—written or spoken—is almost always linear to the detriment of any secondary rectangular structure. For example, the limerick shown in Figure 2.2 is shown twice, first with the secondary

The limerick packs laughs anatomical
Into space that is quite economical.
But the good ones I've seen
So seldom are clean—
And the clean ones so seldom are comical.

(a)

The limerick packs laughs anatomical Into space that is quite economical. But the good ones I've seen So seldom are clean—And the clean ones so seldom are comical.

(b)

Figure 2.2: A limerick shown with and without secondary structure.

rectangular structure and then without. The second form of the limerick has the same rhymes and cadence as the first but loss of the explicit rectangular structure makes it harder to recognise. None of the data dealt with in this thesis has a strong secondary rectangular structure.

XML can be canonicalised [25], a process which, amongst other things, standardises whitespace. This is a lossy operation, whitespace can contain information, particularly about line and paragraph boundaries which is lost by canonicalisation. For this reason all operations preparing the corpora used in this thesis are performed without canonicalisation and preserve whitespace.

Standardisation for representing annotated linguistic data in XML [25] is currently underway, led by the Architecture and Tools for Linguistic Analysis Systems (ATLAS)[6] [78]. The standardisation work includes a content-independent method of specifying regions and anchors in linear linguistic signals, and a query language over those regions and anchors. Similar work, with greater implemented functionality, is being undertaken by the Linguistic Data Consortium[7] [20, 19]. As with the current work, these approaches embed the inferred information within the linguistic

[6] `http://www.nist.gov/speech/atlas/`

[7] `http://www.ldc.upenn.edu/`

data rather than removing it to the document header or an external data store as in most information extraction.

The current work is based on the Unicode and a subset of XML restricts the types of texts and annotations which can be easily worked with. With the exception of attributes, most of the important features of documents in modern information systems can be represented. By using Unicode and XML a range of data preparation and processing tools is available. A range of corpora is available for reuse in XML and, by using XML for the corpora produced in the current work, their potential for reuse is higher than if non-standard formats had been used.

Chapter 3

Models and Algorithms

This chapter examines Markov models and some of the searching algorithms that operate on them. Exhaustive treatment of many aspects touched on here can be found in the standard texts [63] and [13].

3.1 Markov Models

Markov models are Finite State Machines (FSMs) which consist of a finite number of states and the transitions between them. In a probabilistic FSM, each transition has an associated probability and generates (or predicts) a symbol from some alphabet of symbols. The FSM has a set of start states (often only one) and a set of end states (again, often only one). A stream of data is generated by a FSM by starting in one of the start states and moving through a succession of states (using the current state's probability density function to determine the next state) until it reaches an end state. An excellent review of the use of Markov models and similar statistical techniques as applied to language processing can be found in McMahon and Smith [99].

Markov models encapsulate the Markov assumption: that 'the value of the next state is only influenced by the value of the state that directly preceded it' [41]. The Markov assumption is useful because it gives a bound on how much system context needs to be modelled. Markov models produce probability density functions, which estimate the likelihood of each possible value for the next state.

Problem	Observable Sequence	Hidden Sequence	Observable Alphabet Size	Hidden Alphabet Size	Type	Ref.
Chinese word segmentation	Characters	Words	Large	2	Segmentation	[137]
English sentence segmentation	Words	Sentences	Large	2	Segmentation	[133]
Part-of-speech tagging	Words	Word classes	Large	≈ 50	Classification	[28]
Phone identification	Digitised, audio waveforms	Phones	Very large	Large	Entity extraction	[166, 33]

Table 3.1: Observable and hidden sequences for a variety of linguistic problems tackled with hidden Markov models.

3.2 Hidden Markov Models

Hidden Markov models (HMM) are composite models involving a number of hidden states each of which contains a complete Markov model. The hidden states typically represent the information the model is designed to infer, the words to be segmented or the parts of speech to be distinguished between.

Table 3.1 shows some of the wide variety of previous uses of hidden Markov models in linguistic problems. Chinese word segmentation and English sentence segmentation use simple models. Part-of-speech tagging, which has already been discussed, has a larger hidden alphabet and thus more hidden models.

Phone identification is a key step in voice recognition in which digitised audio waves are mapped to phones, speech sounds, which are later built into words [166]. HMMs are also widely used in computational biology [72, 9, 27].

A key property of hidden Markov models that makes them so widely used in these fields is that they handle noisy and ambiguous data well, unlike rule-based systems which are based on a series of binary decisions and are relatively brittle in the face of noise and ambiguity. Markov models are, however, much less convenient for the extraction of pertinent details. While rule-based systems have sets of rules, typically with clear means of identifying the most important, Markov models have matrices of hundreds, or even hundreds of thousands, of numbers, with none being clearly more important than others.

3.3 Higher Order Models

Higher order Markov models involve a relaxation of the Markov assumption, allowing multiple states to be taken into account [41]: 'the values of the next state are only influenced by the values of the n states that directly preceded it'. Each Markov model of order $k > 1$ is isomorphic with a family of Markov models of order $k-1, k-2, k-3, \cdots 3, 2, 1$.

Figure 3.1 shows this isomorphism for an FSM with a two-character alphabet. Figure 3.1(a) shows an order 3 Markov model, with a single state and eight (2^3) transitions, each starting and finishing in the single state, and transition probabilities dependent on the previous two characters. Figure 3.1(b) shows an isomorphic order 2 Markov model, in which the number of states has been multiplied by the size of the alphabet. The same eight transitions shown in Figure 3.1(a) appear in Figure 3.1(b), with all transitions generating an *a* leading to state *a* and a *b* leading to state *b*. Although the transition probabilities are still dependent on the previous two characters, the immediately previous character is implicit in the state and transitions are labelled with only the previous-but-one character.

Figure 3.1(c) shows an isomorphic order 1 Markov model: again the number of states has been multiplied by the size of the alphabet; and again the same 8 transitions appear. Generating a pair of '*a*'s leads to state *aa*, generating an *a* then a *b* leads to state *ab*, and so forth. In this case the proceeding two characters are implicit in the state. Such order 1 models can then be used in software and tools such as HTK [166].

Computational linguistics uses terms such as n-gram, uni-gram, bi-gram and tri-gram [73, 120, 3] to denote the order of models, while information sciences refer to the order of models [4]. Table 3.2 shows the relationship between these two terminologies.

Markov models are often represented using a table, with cells representing the transition probabilities between each pair of states and each symbol, but these grow

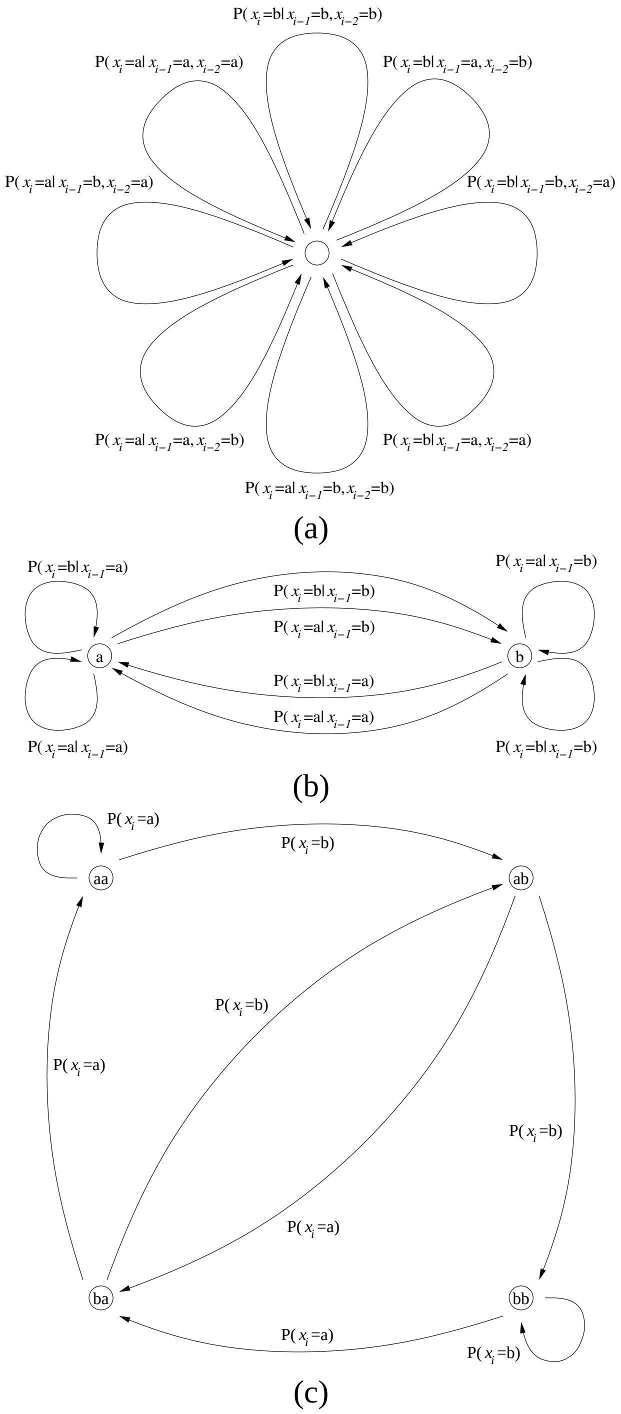


Figure 3.1: Isomorphism in Markov models. (a) an order 3 model, (b) an order 2 model isomorphic to (a), (c) an order 1 model isomorphic to (a) and (b).

n-gram	Order	Meaning
	-1	All symbols to be equal probability
Uni-gram	0	Symbol probability based on their frequency in training data
Bi-gram	1	Symbol probability based on their frequency in training data following the previous symbol
Tri-gram	2	Symbol probability based on their frequency in training data following the previous two symbols
Quad-gram	3	Symbol probability based on their frequency in training data following the previous three symbols
...	...	...
n-gram	$k-1$	Symbol probability based on their frequency in training data following the previous $k-1$ symbols
$n+1$-gram	k	Symbol probability based on their frequency in training data following the previous k symbols

Table 3.2: n-gram models and models of order k.

large for high-order models, as the size is s^k entries, where s is the alphabet of observable symbols and k is the order of the model. The isomorphism between higher- and lower-order models preserves the number of transitions, meaning that the table for a lower-order model has the same number of entries as the higher-order: it is not possible to reduce the table size by using the isomorphism demonstrated in Figure 3.1.

Even with large amounts of training data, it is unlikely that every state and transition of a high-order model is visited during training. The remaining untravelled transitions have zero probability, meaning that the model may generate zero probabilities for a sequence seen during testing. The problem, called the ‘zero-frequency problem’ [146], appears when no non-zero transition exists from the current state to the state that generates the next symbol in the observable sequence. (In hidden Markov models there can be more than one transition, each emitting a different symbol (or symbols) in the hidden sequence.) The zero-frequency problem is often solved by shrinkage (also known as backing off and smoothing [34]), namely the use of a simpler model to estimate probabilities for zero-frequency transitions in more complex models.

Many later systems use n-gram methods together with specialised handling of novel characters. Such systems are effective in tackling problems such as Chinese text segmentation partly because of the large character sets involved. Typically this involves the introduction of a special token (or character) to model the concept of an unseen character.

The differences between this approach and the normal n-gram models are highlighted by the handling of a known character between a pair of novel characters: $\ldots a\,b\,A\,d\,B\,f\,g\ldots$. In the current work the unknown characters *A* and *B* are modelled by escaping back to the order -1 model and the known character *b* is seen in a context which has never been seen before (an order 0 model). The introduction of a synthetic novel character *N* would enable a probability of encountering the sequence $\ldots a\ b\ N\ldots$ to be estimated, then $\ldots a\ b\ N\ d\ldots$ and $\ldots a\ b\ N\ d\ N\ldots$ etc., all without escaping back to the order -1 model. This effectively allows the concept of 'the character following a novel character' to be modelled, something conventional n-gram models cannot do. Part of the reason such techniques are so important is that novel characters in Chinese text, like novel words in English, are often nouns [133]: significant information can be inferred simply from novelty.

The zero-frequency problem can solved using escape methods [146], a recursive case of shrinkage in which unseen transition probabilities are estimated by reference to a lower-order model. Other cases are also common in information extraction systems, for example, Freitag et. al. [46] escape back to a more general class of tags rather than to a lower-order of model for the same tag.

There are several studies of the effectiveness of different smoothing strategies [34, 144], but there is no *a priori* reason why one should perform better than another in the absence of *a priori* knowledge about the symbol distribution within the model. An alternative approach to smoothing is to use Markov as a prescriptive model and reject outright any sequence containing a zero probability. This approach may be useful in closed systems or for carefully curated corpora, but is unlikely to result in robust systems in production environments.

Two aspects of Markov models can be trained: the topology (the number of states and transitions between them) and the weights of individual transitions. In theory the former aspect can be folded into the latter because: (a) a model with a transition of zero probability is indistinguishable from one lacking the transition and, (b) a model with a state which has only zero probability transitions to it is indistinguishable from one lacking that state. In real-world situations, with bounded training data, these are generally treated as separate problems. Model topology is commonly a fixed pattern, variable but selected or trained prior to training the transitions, or trained in parallel to training the transitions (as in DMC [151]). One fixed pattern of topology is used by PPM.

3.4 Prediction by Partial Matching

A Prediction by Partial Matching (PPM) model of order n examines the previous n characters to calculate a probability density function for the next character. To calculate the function, PPM keeps a record of sequences of n characters already seen and the character that followed them. If a sequence of n characters is seen that has not been seen before, then PPM 'escapes' back to sequences of $n-1$ characters. If a match is still not found, PPM escapes back to sequences of $n - 2$, and so on, eventually escaping back to the order -1, in which all characters in the observable alphabet have the same probability.

The PPM model keeps the sequences of characters in a suffix tree, with each node labelled with the number of times the sequence has been seen [13]. This suffix tree can be converted to a single state Markov model of order $n + 1$. The suffix tree is an efficient representation of a sparse model (one for which many of the possible states have not been observed) because unused branches are not expanded. The equivalent Markov model is an array in which all leaves are present, with those not seen during training appearing as small probabilities. In the current work, suffix trees are used for all processing.

The PPM model is deterministic [75] (or subsequential [104]) in that it always has one transition for each output symbol. In this regard it differs from the work of Lafferty and McCallum which has built non-deterministic HMMs for similar tasks to those seen in this thesis, using non-deterministic conditional random fields [75].

An additional benefit of the suffix-tree based Markov models over the traditional table models is that they greatly reduce the cost of introducing extra symbols. Increasing the character set size from 8 bit ASCII to 32 bit Unicode incurs a cost only for those characters are actually used in the training set or when the -1 model is escaped to.

PPM models may seem far removed from the way that humans deal with natural language text. However, as the following story reveals, it may be closer to the way that humans deal with natural language text when they have no linguistic information about it [30]:

> [A] typesetter working on a Greek text at the Oxford University Press announced he'd found a mistake in the text. As the typesetter couldn't read Greek, his colleagues and then his superiors dismissed his claim. But the man insisted. So finally an editor came down to the compositing room. At first, she, too, dismissed the idea, but checking more closely, she found there was an error. Asked how he knew, the typesetter said he had been hand-picking letters for Greek texts for most of his professional life and was sure that he'd never made the physical move to pick the two letters in that order before.

This implies that the typesetter had built an implicit model of which characters followed which other characters and had sufficient confidence in the model to question the text.

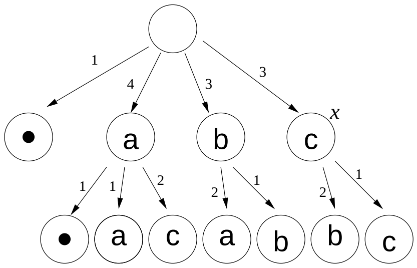
PPM is an incremental compression algorithm [151] with two widely-known variants, PPMC and PPMD [57]. PPMD is used in other text-augmentation work [133, 26]. PPMC and PPMD differ in the probabilities they put aside for unexpected events, seeing a character in a context in which they have not seen that character before. In a context in which C_t total characters and C_d distinct characters have been seen, PPMC sets aside $\frac{C_d}{C_t+C_d}$ and PPMD sets aside $\frac{C_d}{2C_t}$. Katz [67] takes a different approach and for an order n model uses $\frac{C_n}{N}$, where C_n is the count of the number of n grams that have been seen exactly once and N is the training text size.

PPMII is a PPM variant with special handling for the case in which only a singleton example of the current context has been seen during training. The occurrence of such contexts rises with the model order to 60–80% of all contexts. PPMII implementations typically also use adaptive models, and re-scale counts frequently to favour text seen recently over text seen at the start of training, to give good performance on compression corpora [127].

As implemented in this thesis, the PPM model does not store probabilities but rather counts of occurrences. These counts are converted into probabilities dynamically using an escape method which allocates the probability between seen and unseen symbols in the observable alphabet [152].

Figure 3.2 shows three representations of the adaptive order 1 PPMD model built from the string •*aabbccabca*.... The • represents the start of the string. Figure 3.2(a) is the suffix-tree representation. The tree is not complete, for example the *c*-labelled node marked x has no transition to an *a*-labelled node because the string •*aabbccabca*...contains no sub-string *ac*. Figure 3.2(b) shows the occurrence tables for order -1, order 0 and order 1, which correspond to the root node of the suffix tree, the first row of the suffix tree, and the leaves of the suffix tree respectively. Each non-zero entry in the order 1 table corresponds to a leaf in the tree above, while each zero entry thus corresponds to missing leaf.

Figure 3.2(c) shows the Markov models of order -1, order 0 and order 1. These have the same structure as the occurrence tables in Figure 3.2(b), but the occurrences have been converted to probabilities using escape method D. Each count in the



(a)

$k=-1$	
•	1
a	1
b	1
c	1

$k=0$	
•	1
a	4
b	3
c	3

$k=+1$	•	*a*	*b*	*c*
•	0	0	0	0
a	1	1	0	2
b	0	2	1	0
c	0	0	2	1

(b)

$k=-1$	
•	$\frac{1}{4}$
a	$\frac{1}{4}$
b	$\frac{1}{4}$
c	$\frac{1}{4}$

$k=0$	
•	$\frac{1}{11}$
a	$\frac{4}{11}$
b	$\frac{3}{11}$
c	$\frac{3}{11}$

$k=+1$	•	*a*	*b*	*c*
•	$\frac{1}{11}$	$\frac{4}{11}$	$\frac{3}{11}$	$\frac{3}{11}$
a	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{2}{5}$
b	$\frac{1}{16}$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{3}{16}$
c	$\frac{1}{20}$	$\frac{4}{20}$	$\frac{2}{4}$	$\frac{1}{4}$

(c)

Figure 3.2: Three representations of the PPMD model for •*aabbccabca*. . . .

order -1 and 0 tables is divided by the total of counts in the table to obtain a probability. Each non-zero count in the order $+1$ table is divided by the total of counts in that row plus one. The probability corresponding to the extra (plus one) count is distributed among the zero counts.

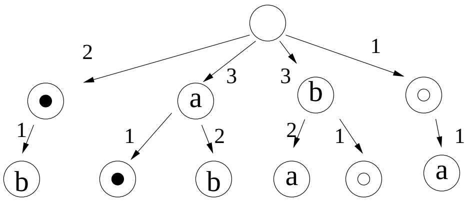
Each type of XML tag corresponds to a hidden state and has a separate model built for it. In the observable sequence the tags are mapped to single character symbols. Thus the string *aba<sometag>cbc</sometag>bab* is mapped to *aba○cbc○bab*, with a different symbol corresponding to each pair of tags, with the ●, seen earlier indicating the start of the string, being used for the entire string (what the XML standard refers to as the 'document element' [25]). Therefore if *aba○cbc○bab* is the entire string, it is represented as ●*aba○cbc○bab*●.

A distinct PPM model is built for each tag, in this case for ● and ○. The models for ● and ○ built from the string ●*aba○cbc○bab*● are shown in Figures 3.3 and 3.4, which have similar structures to Figure 3.2. The ● model is built from the strings ●, ●*a*, *ab*, *ba*, *a*○, ○*b*, *ba*, *ab* and *b*●. The ○ model is built from the sub-strings ○*c*, *cb*, *bc*" and *c*○.

● occurs in the ○ model because it can be part of the alphabet in which the context. Even though it cannot be seen within the ○ model, it can appear in the context which is carried into the model, for example in the string ●○*c*○●.

When a ○ is seen in the ● model, a transition occurs from the ● model to the ○ model. When a ○ is seen in the ○ model, a transition occurs from the ○ model into the previous model, in this case the ● model.

Figures 3.3 and 3.4 show how we can use Viterbi search to find the most likely sequence of tags in the sequence ●*abbacbccbbab*..., the first step of which is shown in Figure 3.5, which has a lookahead of four. Between each two symbols in the observed sequence, the algorithm calculates the probability of there being a transition within the hidden state (the right branch from each node), and the probability of there being a transition to the other hidden state (the left branch from each node).



(a)

$k=-1$	
•	1
a	1
b	1
c	1
○	1

$k=0$	
•	2
a	3
b	3
c	0
○	1

$k=+1$	•	*a*	*b*	*c*	○
•	0	0	1	0	0
a	1	0	2	0	0
b	0	2	0	0	1
c	0	0	0	0	0
○	0	1	0	0	0

(b)

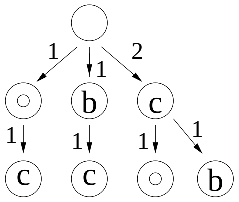
$k=-1$	
•	$\frac{1}{5}$
a	$\frac{1}{5}$
b	$\frac{1}{5}$
c	$\frac{1}{5}$
○	$\frac{1}{5}$

$k=0$	
•	$\frac{2}{10}$
a	$\frac{3}{10}$
b	$\frac{3}{10}$
c	$\frac{1}{10}$
○	$\frac{1}{10}$

$k=+1$	•	*a*	*b*	*c*	○
•	$\frac{2}{14}$	$\frac{3}{14}$	$\frac{1}{2}$	$\frac{1}{14}$	$\frac{1}{14}$
a	$\frac{1}{4}$	$\frac{3}{20}$	$\frac{1}{2}$	$\frac{1}{20}$	$\frac{1}{20}$
b	$\frac{2}{12}$	$\frac{1}{2}$	$\frac{2}{24}$	$\frac{1}{24}$	$\frac{1}{4}$
c	$\frac{2}{10}$	$\frac{3}{10}$	$\frac{3}{10}$	$\frac{1}{10}$	$\frac{1}{10}$
○	$\frac{2}{14}$	$\frac{1}{2}$	$\frac{3}{14}$	$\frac{1}{14}$	$\frac{1}{14}$

(c)

Figure 3.3: The • model built from •*aba*○*cbc*○*bab*•.



(a)

$k=-1$	
•	1
a	1
b	1
c	1
○	1

$k=0$	
•	0
a	0
b	1
c	2
○	1

$k=+1$	•	*a*	*b*	*c*	○
•	0	0	0	0	0
a	0	0	0	0	0
b	0	0	0	1	0
c	0	0	1	0	1
○	0	0	0	1	0

(b)

$k=-1$	
•	$\frac{1}{5}$
a	$\frac{1}{5}$
b	$\frac{1}{5}$
c	$\frac{1}{5}$
○	$\frac{1}{5}$

$k=0$	
•	$\frac{1}{10}$
a	$\frac{1}{10}$
b	$\frac{1}{5}$
c	$\frac{2}{5}$
○	$\frac{1}{5}$

$k=+1$	•	*a*	*b*	*c*	○
•	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{5}$	$\frac{2}{5}$	$\frac{1}{5}$
a	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{5}$	$\frac{2}{5}$	$\frac{1}{5}$
b	$\frac{1}{12}$	$\frac{1}{12}$	$\frac{2}{12}$	$\frac{1}{2}$	$\frac{2}{12}$
c	$\frac{1}{18}$	$\frac{1}{18}$	$\frac{1}{3}$	$\frac{2}{9}$	$\frac{1}{3}$
○	$\frac{1}{12}$	$\frac{1}{12}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$

(c)

Figure 3.4: The ○ model built from •*aba*○*cbc*○*bab*•.

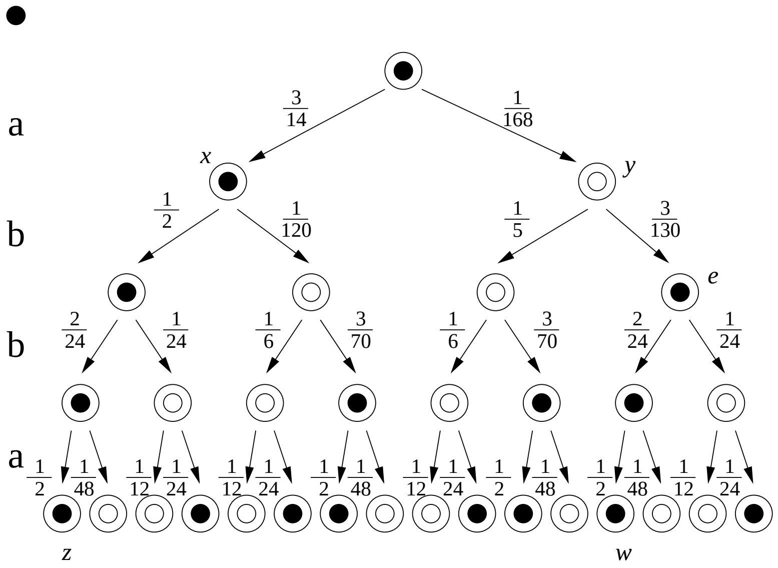


Figure 3.5: The expansion step in a Viterbi search of •*abbacbccbbab*. . . .

The probability for the left branch can be taken from the right hand tables in Figure 3.4(c) (for states in the ○ model) or Figure 3.3(c) (for states in the • model). The probability for a right branch is the product of two probabilities, that of the transition from one model into the other and of seeing the observed character.

Following the expansion step shown in Figure 3.5 is a pruning step. Either node *x* or node *y* must be pruned from the search tree, taking all descendants with it. Since node *z* is the leaf with the highest probability and a descendant of *x* rather than *y*, *y* must be pruned. Nodes *w* and *e* are discussed in Section 4.3.2.

Figure 3.6 shows the tree after pruning. Node *x* in Figure 3.5 has become x_{-1} and there are a new *x* and a new *y* based on the location of *z*, the lowest entropy leaf. Figure 3.7 shows the situation two steps later. For the first time the algorithm is about to prune the *x* branch rather than the *y* branch, and insert a ○ tag.

Viterbi search says that even for this demonstration example a lookahead of four is insufficient to guarantee an optimal tagging: the lookahead must be one more than the sum of the order of the model (1) and the longest tag length (3). Real examples typically have significantly longer tag lengths (see the samples in Appendix A) and

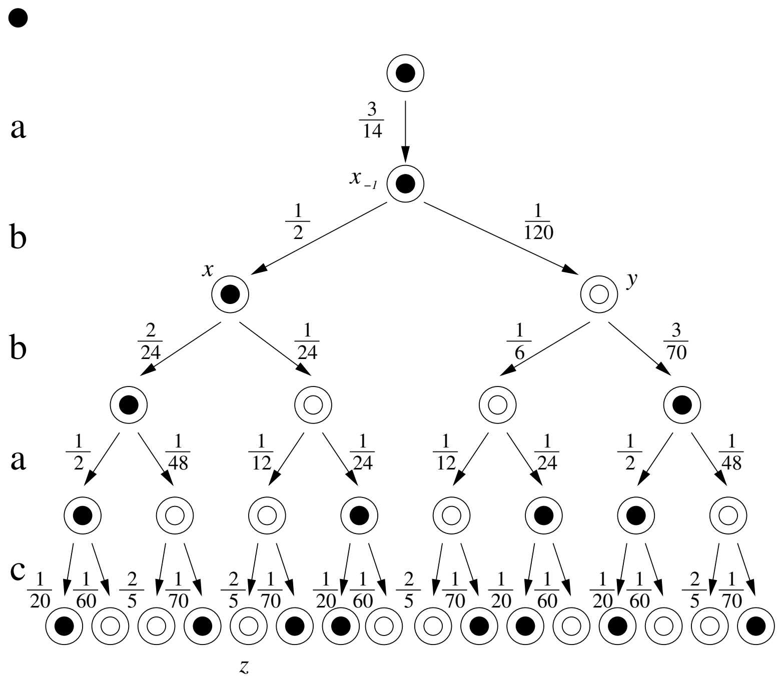


Figure 3.6: The next expansion step in a Viterbi search of •*abbacbccbbab*. . . .

often higher-order models, but for clarity a short lookahead has been used in this example.

3.5 Granularity of Models

Many published reports of text mining, information retrieval and other information systems model text as words [61]. This *a priori* assumption of segmentation into words leads to two separate problems:

1. In many contexts it is not clear what is and is not a word. In English two areas of ambiguity are contractions and abbreviations (for example ‘i.e.’ and ‘can’t’) and sometimes joined words (for example ‘real-time’ which is used variously as ‘realtime,’ ‘real time’ and ‘real-time’).

2. Words seen during testing (or practical application) that are not seen during training raise the ‘unknown-word problem’ [144]. This problem is a variant

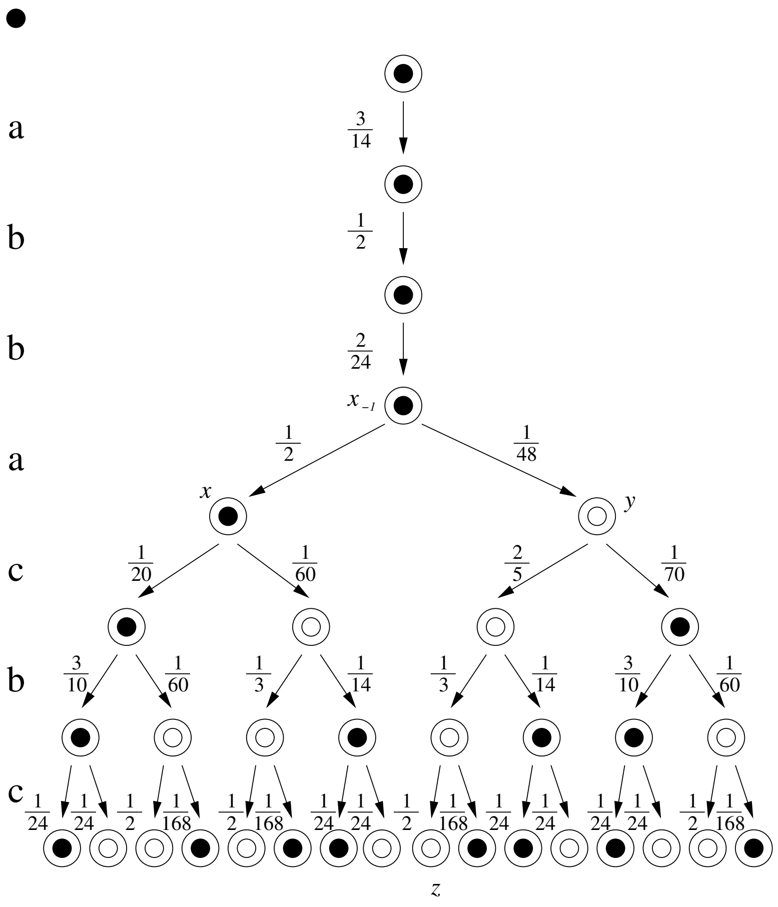


Figure 3.7: The fourth expansion step in a Viterbi search of •*abbacbccbbab*. . . .

of the zero-frequency problem (see Section 3.3). In many system-evaluation contexts, the problem is solved by leaking information from the testing set to the training set in the form of a ‘Perfect Lexicon’ containing every word in the system [17]. In production systems this approach is not possible because, unless a constraint is placed on the system vocabulary (so-called ‘controlled vocabularies’ [84, 105]), an unbounded number of words may be seen over the life of the system.

Approaches to solving the unknown-word problem include merging all unseen words into a single class and treating all unknown words the same, which works surprisingly well for news articles in which most unknown words are proper nouns, and escaping back to a character-level model, requiring two models, one at the word level and one at the character level.

An alternative to this is modelling text as a sequence of characters [133]. At first glance neither of the problems discussed above affects character-based models, but similar problems arise at a different level of granularity.

1. Unicode allows combining character sequences—characters built from a base character and combining characters, which add elements to it (i.e. accents or enclosing circles). All characters in most living natural languages (including English, Maori and Mandarin) are representable without combining characters, but should a system see them in input, handling them is problematic.

2. Though the Unicode character set is bounded, it is sufficiently large (many tens of thousands of characters) that if characters are hyper-geometrically distributed (as can be expected in natural languages [99, 169]), only rarely will a system see an instance of every character. Unicode is also expanding, with more characters being added; in theory a production system could see characters which were undefined when the system was built.

These character-level problems appear to be of a similar nature, if not a similar frequency, to the word-level problems. This suggests that the transition from word to character level has not actually solved the word level problems but rather transformed them to a lower level.

3.6 Searching in Models

Once built, the models can be used to find the most likely sequence of hidden states for a sequence of observed states. This is done using a search tree, in which each node is labelled with a state in the model. Each node is also labelled with the sum of all probabilities on the path between it and the root of the search tree. Entropy is inversely related to likelihood [126], and the most likely sequence corresponds to the leaf node with the lowest entropy.

```
$oldLeaves \leftarrow$ root;
while moreInputSymbols do
    $newLeaves \leftarrow \emptyset$;
    for leaf $\in$ oldLeaves do
        $newleaves \leftarrow$ ExpandLeaf(leaf) ;
        CalculateEntropy(newleaves);
    end
    $oldLeaves \leftarrow newLeaves$ ;
end
$result \leftarrow$ SelectLowestEntropyLeaf(oldLeaves);
```

Algorithm 1: The complete search algorithm.

An exhaustive, or complete, search for the most likely sequence involves a search space as deep as the sequence is long. This algorithm is shown in Algorithm 1. The function $ExpandLeaf$ takes a single leaf node in the search tree, examines the state in the model with which it is labelled and adds a new leaf to the search tree for each out-going transition from the state in the model. The function $CalculateEntropy$ calculates entropy of the each of these new leaves.

For many interesting sequences this search space is computationally infeasible, but the ‘Viterbi search’ [140] algorithm provides a computationally feasible searches in situations when only local information matters. The Viterbi proof [140] guarantees that Viterbi search will find the most likely sequence, provided the model determines the entropy for a node based on bounded local knowledge, rather than on global knowledge required by the exhaustive search. Fortunately Markov models, even high order Markov models, meet this criterion [90]. The length of the sequence that must be modelled for this local knowledge is called the ‘lookahead’.

```
$oldLeaves \leftarrow$ root;
**while** *moreInputSymbols* **do**
    $newLeaves \leftarrow \emptyset$;
    **for** *leaf* $\in$ *oldLeaves* **do**
        $leaves \leftarrow$ *ExpandLeaf(leaf)* ;
        *CalculateEntropy(leaves)*;
        *AddLeavesToSet(newLeaves,leaves)*;
    **end**
    $bestLeaf \leftarrow$ *SelectLowestEntropyLeaf(oldLeaves)*;
    $oldLeaves \leftarrow$ *PruneBranch(bestLeaf, newLeaves)*;
**end**
$result \leftarrow$ *SelectLowestEntropyLeaf(oldLeaves)*;
```

Algorithm 2: The Viterbi search algorithm.

Viterbi is a beam search, as shown in Algorithm 2. This is expressed as a search tree which is built independently of the Markov model in use, but with pointers in every node to a state in the model. The operation $PruneBranch$ takes a $bestLeaf$ from a selection of $newLeaves$, traces parents of $bestLeaf$ up until it finds a node which is the parent of every leaf in $newLeaves$ and prunes all daughters from that node except the one which leads to $bestLeaf$.

There is an alternative representation, that of a search lattice, in which nodes from the search tree are not pruned but ‘merged’ with other nodes with identical state in the underlying models. Merged nodes have the lowest entropy of any of the nodes from which they were merged, this representing the minimum entropy path through the search tree (now a search lattice) to the node. The search lattice is either unified with the Markov model or has a similar structure. This representation is widely used in signal processing and reflects common low-level and hardware implementations in that field [63].

The stack algorithm, a variant of Viterbi search, uses a sorted list rather than an explicit search tree. The list is sorted by the entropy of the node and initially populated with the first symbol. The lowest entropy node is removed from the list and its children calculated and added to the list. The search ends when a leaf node is found.

The Fano algorithm, related to the stack algorithm, does not use a stack but moves incrementally though the search tree guided by entropy-based thresholds, revisiting many nodes, but using only tightly-bounded memory, thus making it suitable for implementation in hardware. The creeper algorithm is a hybrid of the stack and Fano algorithms, using complex tables. All three of these algorithms are described in detail in Johannesson and Zigangirov, Chapter 6 [63].

Viterbi search implemented as a lattice or tree, the stack algorithm, the Fano algorithm, and the creeper algorithm all represent different trade-offs between time and space, and between simple and complex algorithms. The search-tree representation is traditional in computer science, because it allows a more direct comparison with other forms of searching; it is used in this thesis for a more natural representation of the pruning explored in Section 4.3.

```
$oldLeaves \leftarrow$ root;
while moreInputSymbols do
    $newLeaves \leftarrow \emptyset$;
    for leaf $\in$ oldLeaves do
        $leaves \leftarrow$ ExpandLeaf(leaf) ;
        CalculateEntropy(leaves);
        AddLeavesToSet(newLeaves,leaves);
    end
    $oldLeaves \leftarrow$ SelectNLowestEntropyLeaves(newLeaves,N);
end
$result \leftarrow$ SelectLowestEntropyLeaf(oldLeaves);
```

Algorithm 3: The Teahan search algorithm.

Algorithm 3 shows the Teahan search, a ‘Viterbi-inspired’ [136] algorithm which has been found effective [133]. Rather than search a fixed distance ahead into the search space on each increment, it only expands the N lowest entropy nodes at each level in the tree (line 4).

The Teahan search algorithm is a heuristic: it is not guaranteed to find the lowest entropy tagging. The Viterbi proof cannot usefully be applied to Teahan search. This is because the only point at which Teahan search is guaranteed to search the local search space at every step in the search is when N is the number of leaves in the exhaustive search. At this point the Teahan search and exhaustive search become identical.

For many interesting problems, limited amounts of data with correlated hidden and observable sequences are available for training, but data with only observable sequences abound. An algorithm to utilise these un-correlated observable sequences was developed by Baum and is known as the Baum–Welch algorithm [10, 11, 118]. This (Algorithm 4) is similar to Viterbi search with the addition of a step (line 5) that updates the model after the most likely branch has been found [118, 90]. The *UpdateModel* function updates the hidden Markov model to include seeing *bestLeaf*.

```
  $oldLeaves \leftarrow$ root;
  while moreInputSymbols do
      $newLeaves \leftarrow \emptyset$;
      for leaf $\in$ oldLeaves do
          $leaves \leftarrow$ ExpandLeaf(leaf) ;
          CalculateEntropy(leaves);
          AddLeavesToSet(newLeaves,leaves);
      end
      $bestLeaf \leftarrow$ SelectLowestEntropyLeaf(oldLeaves);
5     UpdateModel(bestLeaf);
      $oldLeaves \leftarrow$ PruneBranch(bestLeaf, newLeaves);
  end
  $result \leftarrow$ SelectLowestEntropyLeaf(oldLeaves);
```

Algorithm 4: The Baum–Welch algorithm.

The Baum–Welch algorithm is a specialisation of Expectation Maximisation (EM) which is widely used in machine learning [149] and statistics [60]. McLachlan and Krishnan [98] describe EM and the relationship between it and the Baum–Welch algorithm in detail and [18] discusses this relationship mathematically.

The Baum–Welch algorithm is the primary training mechanism for several information-extraction systems, for learning either the transition probabilities [82, 125, 17] or the model structure [125], or both. In this thesis, the Baum–Welch algorithm is used only for learning the transition probabilities, the Markov model structure is imposed by the PPM algorithm and the hidden Markov model structure reflects the schema of the documents seen during training.

This thesis uses a variant of the Baum–Welch algorithm, in which an entire document, or group of documents, has tags inserted which are then used to update the model, rather than to perform tag insertion and model re-estimation in such a closely-linked manner. This approach precludes the possibility of intra-document learning (lowering the entropy of a sequence of symbols in a tag because they have already been seen) but allows the efficient use of non-adaptive models, and avoids the cost of ‘unlearning’ during searching. The effect of this is likely to be most significant for long, single-subject, documents which contain frequent occurrences of proper nouns and other features which are rare within, or absent from, the training corpus. Proper and rare nouns are typically introduced in stylised forms [160] which can then used to update the model for their less stylised subsequent use. Without the ability to update the model, subsequent uses of the features are likely to be ambiguous.

Much research on the Baum–Welch algorithm is performed in the context of voice recognition [11, 118], where is it used at the phone level for adapting a model to an individual’s accent. In voice recognition, the observable sequence is a discretised representation of a continuous signal. The symbols in the discretised representation can be ordered, for example, it is possible to say that 50 dB $<$ 51 dB $<<$

1000 dB. Much of this research cannot be applied to text because the observable character set in text modelling (characters) has no useful implicit ordering.

The Baum–Welch algorithm is normally used during training. However, if the sequence being modelled is changing slowly over time, or if there is insufficient training data to characterise the sequence sufficiently it can be used during testing. Unfortunately, if a feature is mis-modelled when first seen, the reinforcement of the Baum–Welch algorithm makes it much more likely that it will be mis-modelled when seen subsequently, even in contexts which could have been clear if seen by a model without re-estimation.

3.7 XML and Unicode

This section examines some issues with Unicode and XML and their impact on the corpora and algorithms used in this thesis. These issues include the assumptions Unicode makes about text, the semantics of nested XML tags, and the order of XML attributes. These issues are important because they underpin much later work in this thesis.

XML is a standard [25] for encoding data and has emerged as the leading standard for encoding textual documents for archiving, academic study, interchange and corpus building. XML uses Unicode [138] by default, allowing a large number of languages and writing systems to be represented. Unicode makes various assumptions which make it significantly easier to reason about text, including:

- That characters are unique entities from a finite set.
- That each character falls into exactly one character class.
- That the character class of each character is known.

These assumptions do not hold universally, not even for all documents held in modern information systems. Handwritten texts or texts printed prior to the standardisation of book printing are particularly problematic because their digitisation commonly involves more semantic interpretation than the digitisation of later printed works with known conventions. The Early English Books Online project,[1] is an example of a real-world undertaking impacted by these issues. Unicode character classes are discussed in Section 4.3.3.

[1] http://www.lib.umich.edu/tcp/eebo

Chapter 4

The System

This chapter introduces the bulk of the new content in the thesis, starting with a new taxonomy for metadata markup problems. The architecture of the implementation is introduced, followed by a number of optimisations and heuristics implemented within it. The search space of these optimisations and heuristics for various metadata markup problems is then examined together with the impact of metadata problems on assessing experimental correctness.

4.1 Metadata

This thesis introduces a new taxonomy for fine granularities of metadata problems: in segmentation metadata, classification metadata, and entity metadata. The remainder of this section describes the taxa.

Metadata comprises encoded tags, in ranges of adjacent characters which share some property, and externalised as XML [25]. XML is a widely-used metadata format [156, 123, 147].

4.1.1 Segmentation

Segmentation problems involve finding the internal boundaries within text. The boundaries can be linguistic (e.g. in word or sentence boundaries), semantic (e.g. between topics) or both (e.g. between index or bibliography entries). Finding word boundaries in Chinese, Japanese or Thai text and finding suitable places to seg-

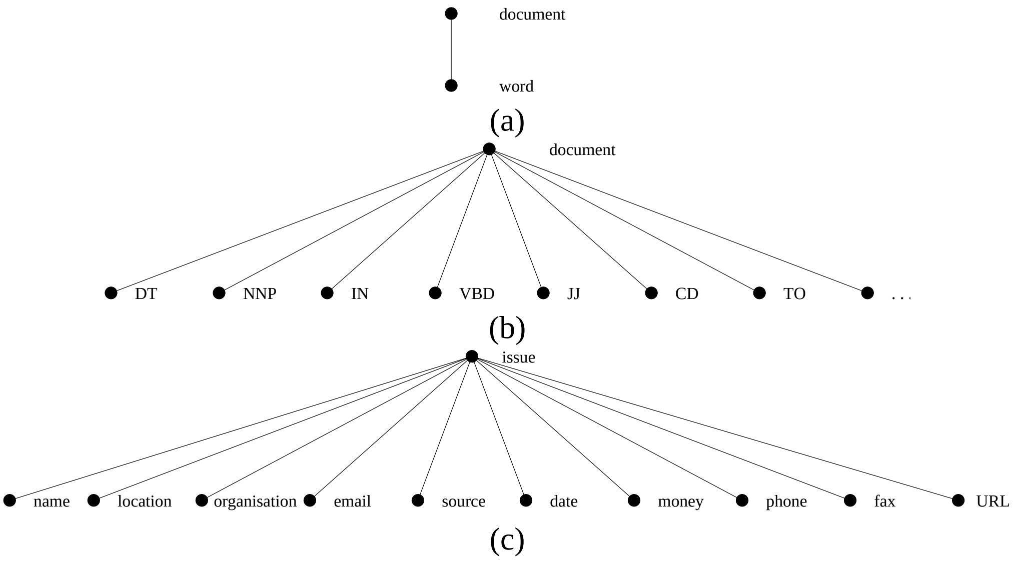


Figure 4.1: Schema structures for segmentation and classification problems. (a) The Chinese text segmentation problem. (b) The part of speech tagging classification problem. (c) The Computists' Communique classification problem. Details of these problems and corpora in which they are studied are given in Chapter 5.

ment English, German and French words for line-end hyphenation [77] and all well-known examples of segmentation problems.

As encoded in this thesis, all segmentation information is destroyed by tag merging. If adjacent tags are merged, all segmentation information is lost because information lies solely in where the tags start and end, rather than in which type of tag a piece of text falls.

Figure 4.1(a) is the schema for the Chinese text-segmentation problem. It has a single root-node and a single type of child-tag below it. There is an instance of the child-tag around each word. The schema for every segmentation problem has this shape, with a single type of child tag and all characters within instances of that tag type.

Various approaches have been used to segment text. Many early systems used simple lookup tables [157], which work surprisingly well on most text, except novel characters not seen in training. Most text segmentation systems use n-gram models or equivalent Markov models [137, 50, 117].

Recent segmentation research directions include conditional random fields [115], and using integrating segmentation with functionality such as part-of-speech tagging [58] and proper noun extraction [168]. Combining segmentation with higher-level processing allows leveraging segmentation to help solve other natural-language processing problems and the results of the higher-level processing to fine-tune the text segmentation.

4.1.2 Classification

Classification problems involve classifying textual elements (typically words or characters) into one of several classes. Many classification problems are referred to as tagging in the information extraction and document understanding communities, but this name has been avoided, because all of the problems discussed here involve inserting tags—literally 'tagging'. The term classification is used in machine learning to refer to problems which involve placing an instance into one of a set of classes, and it is used here in the same manner.

Classification metadata is immune to tag merging. If two adjacent tags of the same class are merged, no knowledge is lost, because the extracted information lies solely in which type of tag text falls. Similarly if a tag is split in two, no information is lost, provided the two new tags cover the same characters as the previous single tag.

Figures 4.1(b) and (c) show the schema structures of classification problems. The schemas have a single root node (representing the document), and each of the classes has a node directly connected to this root node. The schema for every classification problem has this shape, with a number of types of child tags and all characters within instances of these child tag types.

Much early work on classification problems was performed on part-of-speech taggers, drawing on traditional debates on the role of grammar in language. Several early systems were grounded in distinct schools of linguistic theory, but performed relatively poorly. Later approaches have used more generic statistical modelling techniques to better success.

The Brill tagger [28, 29] first trains a rule-based tagger and then learns transformation rules based on the errors of the rule-based tagger. The transformation rules allow for super-adjacency and higher-level reasoning, neither available to conventional Markov models. Super-adjacency, looking not at immediately adjacent words but at those several words away, allows wildcard-like effects. Applying rules is fast, so the whole system runs quickly, and it is widely used and well respected.

The MUC problems can be considered classification problems, but the focus is on information extraction: the inferred information is not embedded in the document text, but either included in the document header or completely separated from the document. Many problems contain higher-order reasoning outside the scope of text augmentation considered in this thesis. For example, the title *President* and the name *Bill Clinton* can be inferred to refer to the same individual combined as *President Bill Clinton*. Classification can identify title and name, both together and separately, but not perform the higher-order reasoning to link the instances or to present the separate components combined into a single sequence.

4.1.3 Entity Extraction

A superset of segmentation and classification, entity extraction, finds bounded sections of text that belong to a particular class. If adjacent tags are merged, some information may be lost, since information lies both in which symbols are in which class of tag and in where the individual tags start and finish.

Because entities have both a range and a depth, it is possible for entities to be nested, introducing extra complexity. Nesting of a tag within another of the same type is a technique used relatively widely in grammar-based linguistics. It is not inherently more complex than nesting a tag within a different type of tag.[1]

[1] However, the current work does not handle such cases gracefully, as explained in section 7.4

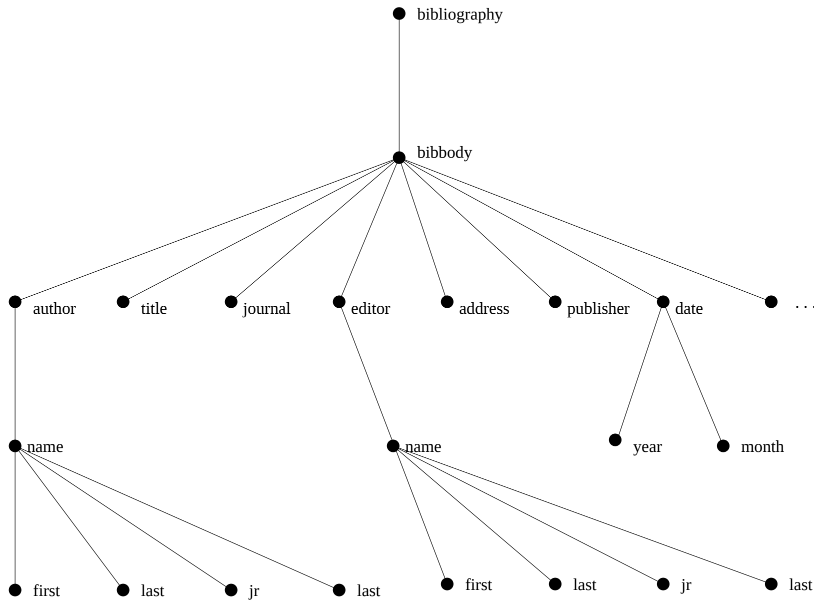


Figure 4.2: Schema structure for the bibliography entity extraction problem. Details of this problem and corpus in which it is studied are given in Chapter 5.

Figure 4.2 shows the schema structure for the bibliography corpus, an example of entity extraction in which the entities such as author names, article, titles and conference names are marked up. The schema for entity extraction problems allows arbitrary nesting of tags.

Bray [26] showed that, on a small sample, hierarchical tagging of personal names into first and last parts hindered the overall identification of names, but the hierarchical tagging of email addresses into username and host parts aided the identification of email addresses. The failure of hierarchical tagging of names in this case appears to be at least in part caused by the small number of names used. Wen [144] used eight tags from an early version of the bibliography corpus (see Section 5.2) and achieved an F-measure of 76%.

4.1.4 Limitations and Constraints

Text augmentation is not a universal method of inferring metadata. There is a range of text-augmentation problems that fall outside this taxonomy, including those with overlapping structures, those with attributes that are continuous numeric values, and those with escapes to the XML Schema ANY tag. The taxonomy is unsuitable for the coarser-grained metadata, such as document level or collection level information.

There are certain constraints derived from the XML tagging used (see Section 2.5):

1. Half the tags are opening tags $t_{tagname}$ and half are closing tags $t_{/tagname}$.
2. Only the most recently opened unclosed tag may be closed next.
3. Each opening tag must be separated from the corresponding closing tag by at least one data point from the underlying sequence.
4. No two tags of the same type are opened between any two characters.
5. Tags do not have attributes.

Constraints 1 and 2 are a restatement of the well-balancedness constraint of XML. Constraint 3 is not present in XML, but is present in the current representation to rule out the proliferation of arbitrary numbers of empty tags.

Constraint 4 is also not present in XML but is introduced here in order to make the sets of tags enumerable, both a consequence of implementation choices and a prerequisite for calculating the size of search spaces. The lack of attributes has been discussed in Section 2.5.2.

4.2 Architecture

The implementation used in this thesis is called ‘Colloquial Entropy Markup’ or CEM. CEM is built in pure Java [52], no platform-dependent library bring used. All input and output of data is performed using the Apache / Xerces implementation of the standard Java XML Document Object Model (DOM) [154]. In this thesis a

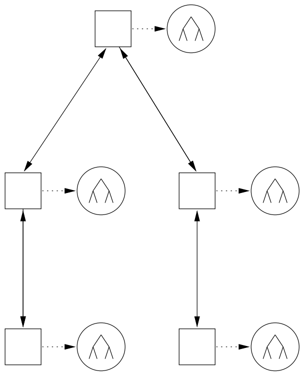


Figure 4.3: The structure of a CEM model, hidden states (square boxes) with associated PPM models (circles).

deliberately standards-based approach was taken largely in response to difficulties encountered Teahan's [133] implementation.

CEM uses Unicode throughout and recursive modelling of tags, the latter enabling it to tackle the more challenging entity-extraction tasks, as well as those of segmentation and classification. There are two main internal data-structures, the model and the search tree. DOM is not used in the internal data-structures, because when the software was first designed, the DOM was immature and it was not clear that it would prove as stable and effective as it has done.

4.2.1 The Model

The structure of the hidden Markov models implemented in CEM is shown in Figure 4.3. Each of the circles is a PPM model in the form of a suffix tree, as shown in Figure 3.2. Each of the squares is a hidden state in the hidden Markov model; the associated PPM model is the Markov model for that hidden state.

The presence of two characters without a tag between them is represented as a transition between two states within the same PPM model. The presence of two characters with one or more tags between them is represented as a series of one or more transitions between states in different PPM models (or between states in the same PPM model in the case of closing tags immediately followed by opening tags). Closing tags indicate transitions up, towards the root of the hidden Markov model and opening tags indicate transitions down, towards the leaves of the model. XML well-formedness is enforced by starting in the root of the hidden model at the start of the sequence and by forcing a return to the root by inserting close tags at the end of the sequence.

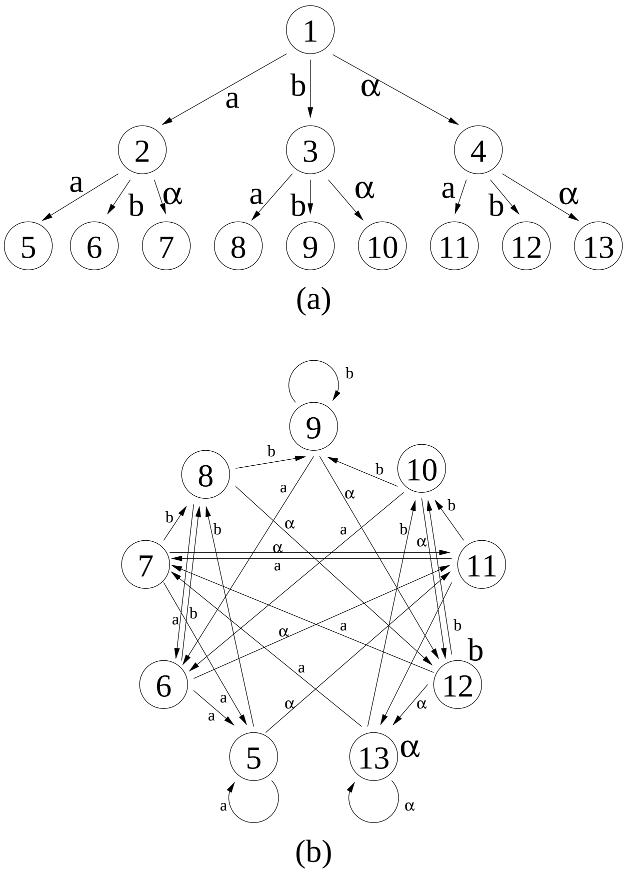
Figure 4.4 shows the relationship between the suffix tree representation of Markov models used in CEM and a more traditional representation. Nodes are numbered for identification. The implementation uses only the suffix tree during training and testing, although it can output low-order Markov models for manual validation. Figure 4.4(b) is directly convertible to a tabular format.

Each state is adjacent to an end state, because each state has an α transition from it. When building PPM models, α is treated as just another letter in the alphabet: α represents one third of the alphabet in Figure 4.4(b). Having multiple start and end states is unusual for a Markov model used in an HMM, but is natural and efficient to implement when suffix trees are used, because the suffixes can have the extra character added for hidden state transition prepended (α in this case), and be carried from one hidden state to the next.

The CEM model is implemented as shown in Figure 4.4(a): a simple tree, with each node labelled with a character and a number. The tree representation allows branches to be expanded as and when they are first seen during training, saving memory on unseen branches.

Transition probabilities are computed dynamically from counts, using escape methods, in the manner of adaptive text compressors [146]. Counts rather than probabilities are stored, so the escape method can be changed after training. This feature is desirable during experimentation, but unlikely to be important in production environments.

CEM models are serialisable: they can be streamed to a file using standard Java serialisation and later streamed back into memory intact. Models are streamed



(a)

(b)

Figure 4.4: The structure of a PPM model, (a) as a suffix tree, in which leaf nodes (5–13) are reached by navigating from the top of the tree each time an entropy is calculated, using the suffix of recently seen characters, and (b) as a finite state machine using traditional Markov model notation, in which a pointer to a node is used for state rather than a suffix and the next node is found by traversing the transition labelled with the current character.

through a gzip [88] stream reducing their size by approximately 90%, primarily because Java serialisation focuses on issues such as portability and flexibility rather than output size. No experiment was undertaken relating the size of training texts to the size of streamed or in-memory models. Streaming models to and from disk allows the reuse of models across testing sessions.

4.2.2 Differences between CEM and other systems

There are two key architectural differences between CEM Markov models and comparable systems: the handling of context between models and the symmetric, recursive structure of the hidden states. This section examines these differences in more detail.

Systems such as HTK and SMI have Markov models with a single start state, so that no matter how much context is taken into account within the models, each transition between hidden states results in a complete loss of context. HTK partly overcomes this by having a large number of hidden states in a complex structure. When moving between hidden states, CEM prepends a single character to the context for each transition (and thus each tag that is opened or closed). This is seen, for example, in the α symbol in Figure 4.4. For tagging problems with many fine-grained, deeply-nested tags this can represent a considerable loss of context, but for lightly-tagged text with a PPM model of non-trivial order the loss is less significant.

This retention of context allows for the efficient modelling of the situation in which tags are marked by a distinctive characters. For example, consider the fragments:

. . . <x> [a] </x> b

and

. . . [<x> a </x>] b

When CEM calculates the entropy for *b* with an order 3 model, in each case it has a full context to use for the calculation, and avoids the need to escape to a lower-order model. This is not true for most other Markov model implementations.

CEM hidden models have a symmetric, recursive structure, reflecting the well-formedness requirement of the XML from which it is automatically generated. This differs from the flat (non recursive) model of SMI and generic finite-state machine model of HTK and other voice-recognition systems. The flat model is sufficient for segmentation and classification problems, but not for entity extraction problems. The added complexity of a generic finite-state machine model is used in voice recognition to represent models of sentence-level structure, based on separate analysis and testing. While there are certainly areas of text augmentation which might benefit from such generic models, it is hard to imagine how they would be readily incorporated into CEM's low human-input approach.

4.2.3 The Search Tree

The search tree is the second of the two main data structures in CEM. Each node in the search tree is labelled with:

- the current character from the input stream;
- any XML tags inserted immediately before the current character;
- the current states in the hidden Markov and PPM models; and
- the cumulative entropy of traversing from the root of the search tree to this node.

There are two types of search tree implemented in CEM: Teahan search (see Algorithm 3 on page 46) and maximum lookahead search. When the maximum lookahead is used with a sufficiently long lookahead, it is a true Viterbi search. Except where explicitly stated, the maximum lookahead search (see Algorithm 2 on page 45) is used.

4.2.4 Full Exclusion

The PPM escape methods, as implemented in this thesis, differ from the standard escape methods because they do not use full exclusion. That is, when an order n model is escaped from back to an $n-1$ model, the $n-1$ model is not modified by removing characters which appear in the order n model. Removal of these characters from the $n-1$ model is safe because they have already been considered in the n model. This variant has been dubbed PPM-SY after the initials of the author, to differentiate it from other forms of PPM.

The effect of not using full exclusion is to modify slightly the action of the escape methods used. As noted on page 32, there is no *a priori* reason either to think that one escape method should model a sequence better than another, or when using PPM for text augmentation to suggest that PPMD should give better results than PPM-SY.

When using PPM to drive an arithmetic encoder, using PPM-SY would squander a small amount of probability whenever a model is escaped from, resulting in a longer coded text, and would thus be undesirable. In text-augmentation applications, the absolute entropy values are not important, only the relative values: the coded text is never used or produced so the length is irrelevant.

The choice not to use full exclusion was made for reasons of efficiency: performing set operations on large character sets in the inner loop of a computation is understandably expensive. It is expected that the cost of full exclusion will be substantially higher for larger character sets than for small ones. A version of PPM with full exclusion is tested in Section 6.1.

The implementation of full exclusion calculates the exclusion dynamically as it occurs. An alternative implementation was considered in which exclusions were calculated the first time they were used, and then cached for reuse thereafter. This would have consumed considerable extra memory, particularly for the large character-set segmentation corpus (see Section 5.3), for which the size of the model was an issue.

4.3 Optimisations and Heuristics

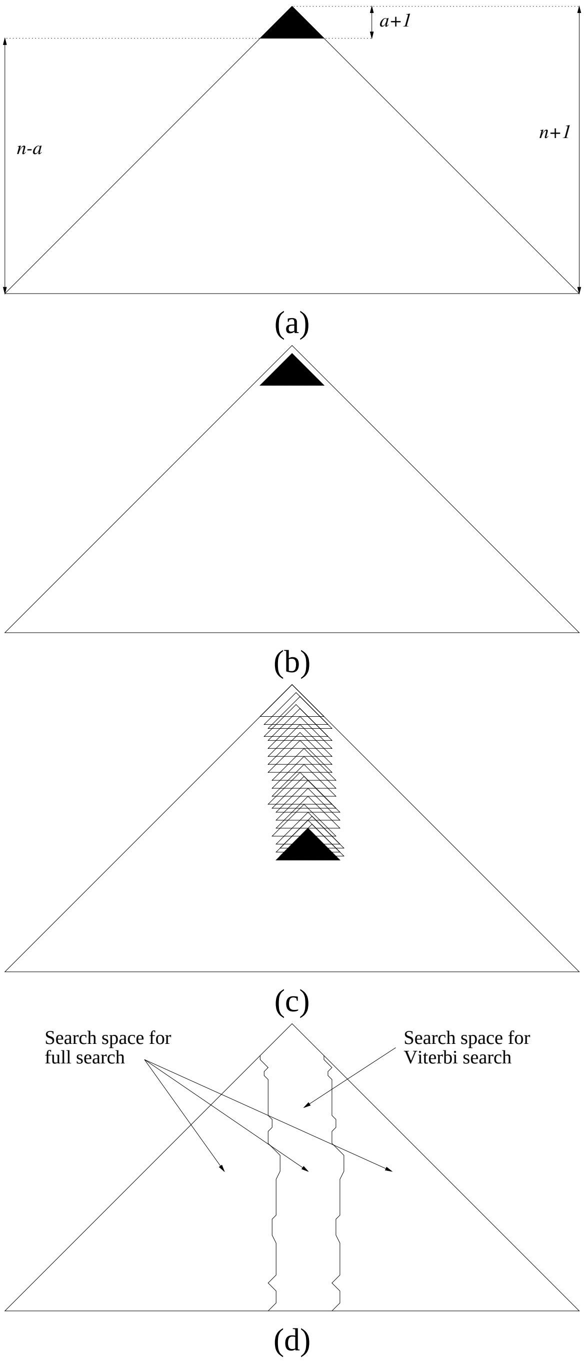
The pruning of search trees using optimisations and heuristics to enable them to be searched as efficiently as possible has a long history in computer science [71]. This section applies this tradition to the search space of text augmentation. Optimisations are techniques that improve the efficiency of problem solving without altering correctness. Heuristics are techniques that improve the efficiency of problem solving but may potentially reduce correctness. This section looks first at techniques and then at how some of them affect the search spaces in three different classes of text augmentation.

4.3.1 Viterbi Optimisation

Viterbi search [140, 141] (Algorithm 2, page 45) is an optimisation of complete search (Algorithm 1, page 44), which Viterbi proved [140] has no impact on correctness provided the lookahead a is large enough and the encoding scheme has the right properties. For text-augmentation problems 'large enough' is the maximum possible length of a tag, plus the order of the PPM model in use, plus one.

Relating search-space size to the maximum length of the tags being inserted means that some tags require smaller search spaces than others. Inserting short tags, such as personal names or parts of speech, gains more advantage from the Viterbi search than do large tags such as the <*html*> or <*body*> tags in XHTML [114] which contain an entire document.

Figure 4.5 shows an example of Viterbi search space, with each small black triangle being the search space for the current increment, page 45) and the large triangle being the full search space (respectively the for and the *while* loops in Algorithm 2, page 45). Figure 4.5(a) shows the initial search space of depth $a + 1$,



(a)

(b)

(c)

(d)

Figure 4.5: Viterbi search of a large search space.

before the first pruning of the search space, and the full search space of depth $n+1$. Figure 4.5(b) shows the second search space of depth a after the first pruning. Figure 4.5(c) shows the search half-way though, and Figure 4.5(d) shows the completed search.

aaa({)(}{)· · ·(}{)· · ·(}{)(}{)(}{x

(a)

aaaa({)(}{)(}{)(}{)(}{)}aa

(b)

○	a	{	}	(	)	○	⋆	◇
a	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{y}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$
{	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{6}{7}$	$\frac{1}{49}$
}	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$
(	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{6}{7}$
)	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$
○	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$
⋆	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{6}{7}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$
◇	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{x}{49}$	$\frac{1}{49}$	$\frac{6}{7}$	$\frac{1}{49}$	$\frac{1}{49}$	$\frac{1}{49}$

(c)

⋆	a	{	}	(	)	○	⋆	◇
a	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	0	$\frac{1}{6}$	0
{	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	0	$\frac{1}{6}$	0
}	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	0	$\frac{1}{6}$	0
(	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	0	$\frac{1}{6}$	0
)	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	0	$\frac{1}{6}$	0
○	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	0	$\frac{1}{6}$	0
⋆	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	0	$\frac{1}{6}$	0
◇	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	0	$\frac{1}{6}$	0

(d)

◇	a	{	}	(	)	○	⋆	◇
a	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	0	0	$\frac{1}{6}$
{	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	0	0	$\frac{1}{6}$
}	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	0	0	$\frac{1}{6}$
(	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	0	0	$\frac{1}{6}$
)	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	0	0	$\frac{1}{6}$
○	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	0	0	$\frac{1}{6}$
⋆	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	0	0	$\frac{1}{6}$
◇	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{6}$	0	0	$\frac{1}{6}$

(e)

Figure 4.6: A set of models and sequences for which the Viterbi assumption does not hold and Viterbi search fails. (a) a class of difficult sequences (b) a single sequence (c) top-level Markov model (d) model for ⋆ (e) model for ◇. x=1 and y=1.

As the following contrived example illustrates, it is not obvious that the Markov assumption, and with it Viterbi proof, in any form holds for natural language text. Figure 4.6 shows a Markov model with three hidden states and an alphabet of eight symbols. Hidden model ⋆ models the contents of matched { } braces (d). Hidden model ◇ models the contents of matched *( )* parentheses (d). The columns of zeros in the ⋆ and ◇ models indicate that no direct transitions between them are possible, and that transitions must be via the top-level Markov model for the ○ hidden state.

Figure 4.6(a) shows a class of sequences which is problematic with respect to this model: parentheses and brackets used in ways that do not match. Furthermore, a repeating chain of parentheses and brackets can extend the ambiguity indefinitely until some other symbols, such as an *x* are seen. Figure 4.6(b) shows a string for which Viterbi search will yield two equally likely hidden sequences. The model may be changed to prefer one over the other by changing x in $\circ$ (and adjusting the other probabilities so that the sum is 1). However, such a solution still requires that the search sees the end of the chain before pruning the search tree at the start of the chain.

Fortunately such situations are rare, none of the datasets presented in this thesis appears to contain such sequences, and none has been reported in the literature. Experience [136, 144, 145, 135, 26, 163] has shown that in practice Viterbi search does work on natural language text.

Figure 4.6 shows a situation in which Teahan Search (Algorithm 3 on page 46) performs admirably. Teahan Search expands a fixed number of nodes at each level in the search tree so it is capable of exploring equal entropy branches of the search tree to an arbitrary depth, providing at each level one node from each branch is expanded. However, if a branch has higher entropy (for example, y in Figure 4.6(c) is raised), then it will probably get pruned, even if the lowest global entropy lies down that branch of the search tree.

4.3.2 Best First Optimisation

The best first optimisation is based on the observation that once a candidate augmentation has been found and the entropy calculated, all nodes within the search space with higher entropy can be pruned immediately. If a likely candidate augmentation can be found computationally cheaply, and the probability distribution function is steep (i.e. the model has high discrimination), the search space can be reduced considerably. In Figure 3.5, e has an log probability of $\frac{3}{168\times130} = \frac{3}{21840}$, and node z has an log probability of $\frac{6}{14\times2\times24\times2} = \frac{6}{1344} = \frac{1}{224}$: neither w nor any other child of e can have a lower log probability (and thus entropy) than that of node z, so node e need not be expanded.

The savings made from best first are difficult to calculate, because they depend on the probability distribution function for each state in the model and the exact sequence of symbols seen. In general, however, the savings are larger for probability density functions that are highly discriminative. Discrimination generally increases as models are better trained.

The CEM implementation finds a best first candidate by calculating the entropy of the left most leaf (the only leaf reachable without inserting any tags). This is the computationally cheapest leaf to find and in many situations it is a low-entropy leaf, if not the lowest. Hardware and Field Programmable Gate Array (FPGA) implementations of Viterbi search may avoid the need for the best first optimisation by performing this part of the algorithm in parallel [140, 141, 121]. Such treatment is not possible with text augmentation because of the significantly larger lookahead.

4.3.3 Automatic Tokenisation Heuristic

The automatic tokenisation heuristic is based on the observation that in many problems there are classes of characters between which no tag ever occurs. For example, in the Computists' and bibliography corpora, no tag ever occurs between a pair of lower-case letters or between a pair of whitespace characters. If no tag is ever seen in a situation during training, and a sufficient amount of training data has been seen, it is reasonable not to consider inserting tags in such positions during testing. This assumption may prove false, which is why automatic tokenisation is a heuristic not an optimisation.

The saving in search space depends on the structure of the text. However, if text were uniform words of four letters starting with a capital letter and separated from the next by a space $(\ldots\textvisiblespace\mathit{Abcd}\textvisiblespace\mathit{Efgh}\textvisiblespace\ldots)$ and automatic tokenisation meant the search did not have to consider inserting tags between pairs of lower-case letters, two of every five nodes in the search space would not need to be expanded. This $\frac{2}{5}$ approximation is assumed throughout this chapter.

Some types of contraction and abbreviation have a direct impact on automatic tokenisation. For example, the string *John Anthony Smith* may have the same search space as *J. A. Smith*, even though they differ markedly in length.

The CEM implementation keeps an occurrence table of possible pairs of Unicode character classes [138], and counts how many tags are seen between each pair. During augmentation, each node in the search tree is checked to see whether more than a threshold number of tags has been seen between the current pair of character classes, before considering whether to expand the search tree. Common threshold values include -1, 0, 1 and the default 5.

Unicode characters are divided into a set of 28 classes. The most common classes seen in the corpora used in this thesis are lowercase letter, uppercase letter, other letter (common in the segmentation corpus), space separator, line separator, decimal digit number, and various classes of punctuation. The classes are particularly convenient in Java, which uses Unicode throughout [52]. The ANSI C [59] functions *isspace(), isupper(), isdigit()*, etc. have a long history in parsing applications [2] and would almost certainly have performed well in this role for the English language corpora. There are been proposals [5] for much more sophisticated character-level metadata systems in Unicode, but these are not considered here.

One Unicode character class, the private use class, is reserved for ‘use by software developers and end users who need a special set of characters for their applications. [These characters] are reserved for private use and do not have defined, interpretable semantics except by private agreement’ [138]. CEM uses these to represent tags in character-level models, assigning a character to each tag to enable it to be modelled as just another character within the PPM models: the α in Figure 4.4 and the $\diamond$, $\star$ and $\circ$ in Figure 4.6. These characters are used by CEM only internally, and always mapped to or from full XML representations of the tags when externalised.

4.3.4 Alphabet Reduction

Alphabet reduction is a heuristic based on the same character classes as automatic tokenisation. In the bibliography corpus, repeating patterns of punctuation and capitalisation involving names in bibliographies were noticed. Names, which are commonly unique strings, remain a problem for the PPM model which sees limited context.

Alphabet reduction merges a class of characters into a single character in the model. For example, merging all upper case letters to *A* and all lower case letters to *a* means that *John A. Smith and*, *Jill K. Jones and* and *Yong X. Xiong and* all merge to *Aaaa A. Aaaaa aaa*. Throwing away this information homogenises these names. Considerably less memory and training data are needed to produce high-order models because alphabet reduction reduces the size of the alphabet so drastically. Empirically, alphabet reduction has raised the maximum order of the model to between 15 and 25. The performance of alphabet reduction in practice is examined in detail in Section 6.4.3.

This method is related to methods used elsewhere for finding acronyms [32, 160] using capitalisation patterns for generating candidate acronyms, which are then winnowed using other techniques. The benefits of alphabet reduction are hard to model, as they depend on the gains from modelling at a higher order compared with the loss of information about each character.

4.3.5 Maximum Lookahead Heuristic

The lookahead a required by the Viterbi proof is not always needed in practice, and previous work [133] suggests that the results of tag insertion commonly converge at lookaheads much lower than a. The maximum lookahead heuristic is to select a lower lookahead that represents a trade-off between correctness and efficiency. The lower lookahead is denoted a'. If a' is too low, the lowest entropy tagging may not be found; this may be detectable during evaluation (see Section 2.3.4). If a' is too high, the search space is unnecessarily large.

The CEM implementation collects statistics on the maximum size of every tag, but leaves the selection of a lower lookahead to the user. The performance of maximum lookahead in practice is examined in detail in Section 6.4.4. Various methods for limiting the depth of Viterbi search are discussed in [118].

4.3.6 TagC Heuristic

As presented so far, CEM considers every possible combination of tags whenever it considers inserting any tags. In real documents, however, only limited ranges of permutations of tags are found. The TagC heuristic involves tracking during training the set of all tag permutations seen. For example, the training text *<entry> <author><forenames> Donald E.</forenames> <surname>Knuth. </surname></author>*. . . would add { (*<entry> <author> <forenames>*), (*</forenames><surname>*) and (*</surname> </author>*)} to the set of permutations. When tags are inserted, only the permutations seen in training are considered for insertion (plus closing tags at the end of the file to guarantee that all tags are closed).

The TagC heuristic has no effect on segmentation problems (since there are only two states) and only limited effect on classification, because only one tag can be closed and one opened, limiting the number of permutations. The significantly more complex schemas involved in entity extraction (see Figure 4.2) give considerable scope for savings to be made. The savings will be greater for complex schemas when a relatively small set of permutations is seen during training. The performance of the TagC heuristic is discussed in Section 6.4.5.

4.3.7 State Tying

State tying is a widely-used heuristic in speech recognition [60], which appears not to have been used before in text modelling. The insight on which state tying is

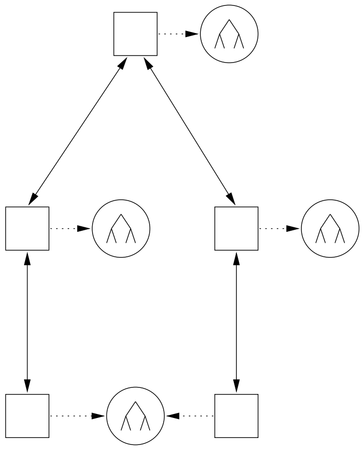


Figure 4.7: The structure of a hidden Markov model, with state tying. The squares are hidden states, linked by the solid arrows of the model structure and by dotted arrows to their associated models.

built is that some states in a large model are similar not by chance but because they model similar concepts. Thus in a speech-recognition system, the models for the second half of the words ‘hair’ and ‘pair’ are similar (or at least they are for certain dialects) even though the words themselves are different and they may represent different parts of speech. State tying uses a single underlying Markov model to model several hidden states. The hidden states are not merged—at a higher level the model tracks the difference between them—but they share a PPM model and should require significantly less training data. Figure 4.7 shows the hidden Markov model shown in Figure 4.3 with two leaf states tied.

The key benefit of state tying is the ability to share training data between relatively common and relatively rare tags so as to achieve better performance from the same amount of training data. State tying only works on entity extraction problems, because it requires at least two levels below the document root to tie together. Tying two states in a classification problem would leave two indistinguishable states. In a segmentation problem there is one (non-root) state, which cannot be tied to itself.

By default CEM performs state tying on all states with the same tag *name*. The effect of *not* tying the *name* tag is examined in Section 6.4.6.

4.4 Search Space

As discussed in Section 2.4, the efficiency of abstract computer operations is expressed by complexity, using the $\mathcal{O}(x)$. In the case of the tag insertion methodology presented here, the parameters are the numbers of tags (t), the lookahead (a) and the size of the input is the length (n) of the text. This complexity is a reflection of tagging action, rather than the complexity of the underlying intellectual or syntactic complexity [16].

If u is a constant and x and y are unbounded positive variables, $\mathcal{O}(u) \ll \mathcal{O}(x) \ll \mathcal{O}(x^u) \ll \mathcal{O}(u^y) \ll \mathcal{O}(x^y)$. Algorithms with $\mathcal{O}(u^y)$ or greater are referred to as intractable and run in non-polynomial time on conventional computer equipment.

A line of investigation in the MUC conferences (see Section 2.2.4) was measuring the inherent complexity in the web of atoms in the named entity tasks [7]. This approach relied on a uniform model of textual atoms extracted into a relational database and a network of inferred relations between them, not readily adaptable to the approach under consideration in this thesis. It was discovered was that tasks considered in MUC-5, MUC-6 and MUC-7 had surprisingly similar complexity, suggesting that the underlying complexity of textual understanding tasks may not be as great as that of the solutions presented here. This approach is not applicable to the present work because no web of atoms or equivalent structure is constructed by systems such as CEM.

This thesis examines only the efficiency of text augmentation by tag insertion, rather than the building of models which is a prerequisite to this activity. There is other work in the area of efficiently building models [97, 133], but it is outside the scope of this thesis. CEM builds the suffix tree with a hash table from the standard Java libraries. The hash key is the character leading to the node stored in the hash value. Character counts are stored in the child node. Character counts are stored as Java longs and never rescaled (none of the corpora dealt with in this thesis are sufficiently large to overflow a long).

This analysis of search space is dependent on the constraints introduced in Section 4.1.4. Removing Constraint 3 would add an infinite number of empty tags into the search space, and removing Constraint 4 would add an infinite number of non-empty tags. Therefore analysis includes recursive tags, but only when there is at least one character between each two open tags of each type.

If a document contains a single character, it could potentially have tags inserted either before or after that character. By Constraint 3, which forbids empty tags, any tags inserted into such a document must open before the character and close after it. By constraint 4, each tag can only open once. If the document is being marked up using a set of t tags, then $0, 1, 2, 3, \ldots$ or t tags could occur before the character, with the tags chosen being a permutation of the t tags. Thus, the number of combinations of tags that might be inserted prior to the first character is:

$$\sum_{i=0}^{t} {}_tP_i \qquad = \qquad \sum_{i=0}^{t} \frac{t!}{(t-i)!}$$

Constraint 3, which prevents the opening of tags that would be empty, and Constraint 2 which requires that all open tags must be closed, means the only tags following the final character in any document are close tags matching those tags remaining unclosed. Thus the number of taggings of the entire document is the same as the combinations of tags that might be inserted prior to the first character.

If a document with the single character ‘a’ is tagged with the two tags, ‘<x>’ and ‘<y>’, then there are $\sum_{i=0}^{2} {}_2P_i = 1 + 2 + 2 = 5$ possible taggings.

In a document of two characters, the same tags might be inserted prior to the first character as in the case of a one-character document. More tags may occur between the first and second characters: tags may be closed as well as opened. The maximum number of tags that may be opened is directly related to the number of tags previously opened:

$$\sum_{j=0}^{i}(\sum_{k=0}^{t} {}_tP_k)$$

where i is the number of tags opened before the first character.

As before, the tags following the last character can only be the closing tags of already open tags. This gives the total number of taggings for a two character document as:

$$\sum_{i=0}^{t}\left({}_tP_i \times \sum_{j=0}^{i}\sum_{k=0}^{t} {}_tP_k\right) = \sum_{i=0}^{t}((i+1) \times {}_tP_i) \times \sum_{k=0}^{t} {}_tP_k$$

Thus, if a document with the two characters ‘ab’ is tagged with the two tags the ‘<x>’ and ‘<y>’, then there are $1 \times 1 \times 5 + 2 \times 2 \times 5 + 2 \times 3 \times 5 = 55$ possible taggings. The formula on the right can be considerably simplified, but the $\sum_{j_x=0}^{t\ldots}\sum_{k_x=0}^{t} {}_tP_{k_x}$ factor can be factored out.

The number of taggings for a three-character document follows from this:

$$\sum_{i=0}^{t}\left({}_tP_i \times \sum_{j_1=0}^{i}\sum_{k_1=0}^{t}\left({}_tP_{k_1} \times \sum_{j_2=0}^{i-j_1+k_1}\sum_{k_2=0}^{t} {}_tP_{k_2}\right)\right)$$

$$= \sum_{i=0}^{t} {}_tP_i \times \sum_{j_1=0}^{i}\sum_{k_1=0}^{t}(k_1 - j_1 + i + 1) \times \sum_{k_2=0}^{t} {}_tP_{k_2}$$

and each additional character in the document adds a $\sum_{j_x=0}^{t\ldots}\sum_{k_x=0}^{t} {}_tP_{k_x}$ term to the number of taggings, which is $\mathcal{O}(t^2t!) = \mathcal{O}(t!) = \mathcal{O}(t^t)$.

Classification is significantly simpler, because each character can be put into only one of t classes, giving ${}_tP_1$ or t options, which is $\mathcal{O}(t)$. Segmentation is even simpler: either a tag is inserted or no tag is inserted, a binary decision, giving $\mathcal{O}(c)$ where c is a constant.

Table 4.1 gives the number of nodes in search spaces, first for inserting tags between two characters in a document and then for inserting tags into an entire document for each variant.

4.4.1 The Semantics of Nested Tags

Permutation is a significant contributor to the search space, particularly when t is large. If the semantics of nested tags (see Section 2.5.1) were changed so that opening tags occurring between two adjacent characters are semantically equivalent, independent of order (i.e. widely expected HTML / XHTML semantics), this would change the permutation to a combination, substantially reducing the search space for entity extraction. Changing the semantics of nested tags also drastically reduces the maximum number of Markov models which would be needed in the case where tags are not used consistently, increasing the usefulness of state tying (see Section 4.3.7).

Segmentation and classification do not involve nested tags, so their semantics are irrelevant.

4.5 Teahan Search

Not all of the optimisations and heuristics described above can be applied to the Teahan search algorithm. In particular, those that relate to pruning the depth of the search space (the Viterbi and best-first optimisations, and the maximum lookahead heuristic) cannot be used because the Teahan search does not consider depth of search. Automatic tokenisation, which applies to the nodes at which the search tree can branch, can be used with Teahan search, as can the TagC heuristic, which relates to the width of the branching.

Algorithm	Segmentation	Classification	Entity Extraction
per Character	$\mathcal{O}(c)$	$\mathcal{O}(t)$	$\mathcal{O}(t^t)$
Complete	$\mathcal{O}(c^n)$	$\mathcal{O}(t^n)$	$\mathcal{O}(t^{tn})$
Viterbi	$\mathcal{O}(c^a)$	$\mathcal{O}(t^a)$	$\mathcal{O}(t^{ta})$
Maximum Lookahead	$\mathcal{O}(c^{a'})$	$\mathcal{O}(t^{a'})$	$\mathcal{O}(t^{ta'})$

Table 4.1: Search space size. t is the number of tags, t is the document length, a is the lookahead for Viterbi search, a' is the lookahead for maximum lookahead search and c is a constant.

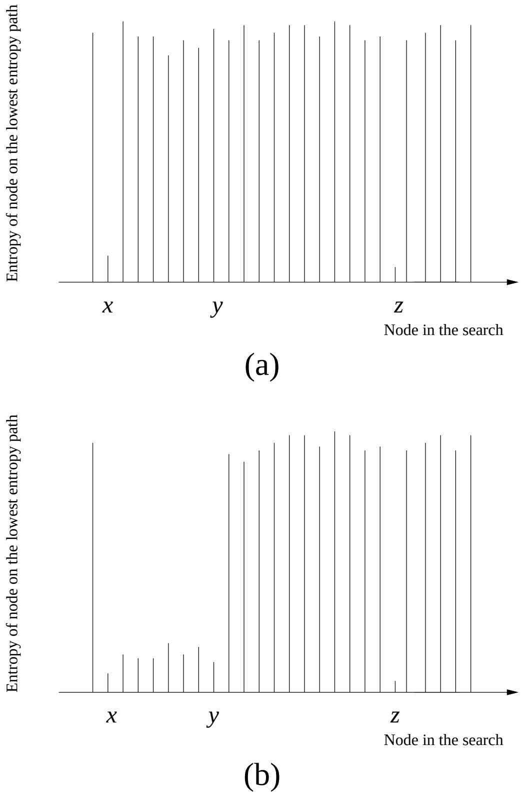


Figure 4.8: Scenarios in which Teahan search and Viterbi search can be expected to perform differently, (a) Teahan search performs well and (b) Viterbi search performs well.

Both Teahan search and Viterbi search with maximum lookahead are heuristics and it makes sense to ask which can be expected to perform better, or *might* perform better, than the other. There is no *a priori* reason to believe that one will perform better in the general case, but in specific cases they perform differently. Viterbi search can be expected to perform well in situations in which there is a great deal of ambiguity (a small entropy difference between a large number of nodes at the same level) in the search tree, because it focuses on searching the current, immediate context. Teahan search will perform better when the search contains long sequences of low ambiguity interspersed with short sequences of high ambiguity because, by counting only the leaves, it is able to look effectively past the long sequences of low ambiguity.

Figure 4.8 shows two scenarios which illustrate such situations. It shows the entropy implications of inserting a single tag at various points in a sequence. In Figure 4.8(a) all the points are high-entropy, except *x* and *z* which are low entropy. Viterbi search with maximum lookahead is only capable of determining whether *x* or *z* is the better place to insert the tag if the difference between them is a' or less. Teahan search is capable of making the differentiation no matter what the separation, provided there are no (or relatively few) other low entropy branching options between *x* and *z*. Figure 4.8(b) still has *x* and *z* but also has a range of relatively low-entropy branching options between *x* and *y*. In such a situation Teahan search is likely to prune prematurely at *x*, whereas Viterbi search with maximum lookahead is guaranteed to find the best option within the a' maximum lookahead.

4.6 Evaluation

This section examines how the measures of correctness first introduced in Section 2.3 can be used in conjunction with the metadata taxonomy introduced in Section 4.1. For each of the measures, each of the three taxa is examined. A new correctness measure, *type confusion matrices*, is introduced.

4.6.1 Recall and Precision

Recall, precision, and their combination in the F-measure, are the primary means of evaluating correctness in information-retrieval systems, but the definition of what constitutes a document varies for each type of text-augmentation problem.

Segmentation

For segmentation problems the evaluation question is ‘Does a segment end between one symbol and the next and was that segment end found?’ Recall and precision are good measures for evaluating segmentation problems because both operate on *to␣be␣or␣not␣to␣be*

(a)

<to>to</to>␣<be>be</be>␣<cc>or</cc>␣<xnot>not</xnot>
␣<to>to</to>␣<be>be</be>

(b)

Figure 4.9: A short quote from Hamlet. (a) without and (b) with part of speech tags.

a binary distinction. Recall and precision are the standard methodology for measuring correctness in the fields of Chinese text segmentation [137, 145, 12, 50] and Japanese text segmentation [3], both widely-studied segmentation problems.

Classification

For classification problems, the evaluation question is ‘Is the class predicted for symbol n correct?’, where symbols are the characters, words, sentences or documents being placed into classes. Recall and precision are standard methodology for measuring correctness in the fields of part-of-speech tagging [28, 76, 94] and genre classification [66], which are probably the most widely-studied textual classification problems.

Figure 4.9(a) shows a short quote from Hamlet and Figure 4.9(b) the same quote marked up using the tags of the Lancaster Oslo/Bergen part-of-speech corpus [64]. Teahan’s work (from which this example is taken) [133] is a word-based approach and uses word-based evaluation mechanisms: there are 6 words in the sample and they are all correctly tagged, giving 6 true-positives. Character-based approaches see only characters not words: there are 18 characters, including 5 spaces, all correctly tagged, giving 18 true-positives. Evaluation of the output from a character-based system using a word-based evaluation might be considered. However, this works for mistakes such as misclassification of an entire word, but fails when only part of a word or a non-word character is misclassified. There are similar problems in evaluating Optical Character Recognition (OCR) at a word level when word boundaries can be incorrectly identified [73].

The core problem is that character-based approaches are more expressive and can be wrong in ways that cannot be represented in conventional word-based approaches. The reverse is not the case, however, and the output of a word-based system can be compared to that of a character-based system at the character level.

The expressiveness of character-based approaches definitely has advantages in some corpora. For example, dates in the Computists' corpus (Section 5.1) are expressed as a single word in the form *19Jan98* which word-based approaches see as a single word (unless they have customised word boundaries heuristics) and are unable to do better that identifying it as a date (<*date*>*19Jan98*</*date*>). Character-based approaches are capable of breaking the date into component parts (<*date*><*day*>*19*</*day*><*month*>*Jan*</*month*><*year*>*98*</*year*></*date*>).

The difference in expressiveness applies to all three types of text augmentation problem if the standard measurement technique is word-based, but is most obvious in classification problems such as part of speech tagging.

Entity Extraction

Measuring entity extraction as an information retrieval problem is challenging. The four basic classes (true positives, false positives, false negatives and true negatives) are accumulated over successive independent trials, but the XML well-balancedness constraint (see page 56) introduces inter-dependencies between trials.

Figure 4.10 shows inter-dependencies in a small entity extraction problem. The untagged input text is shown in Figure 4.10(a). The task is to insert <*name*> and <*title*> tags into the text, as shown in Figure 4.10(b). Figure 4.10(c) shows an error: the boundary between the first two names has been inserted in an incorrect place: the tag <*name*>*Smolensky, P., Fox,* </*name*> is a false positive. The independence criterion is broken because seeing this false positive does not just preclude the possibility of seeing the tag <*name*>*Smolensky, P.,* </*name*>. It also precludes the

Smolensky, P., Fox, B., King, R., and Lewis, C. Computer-aided reasoned discourse...

(a)

<name>Smolensky, P., </name><name>Fox, B., </name><name>King, R. </name>, and <name>Lewis, C. </name><title>Computer-aided reasoned discourse... </title>

(b)

<name> Smolensky, P., Fox, </name> B., <name> King, R. </name> , and <name> Lewis, C. </name> <title> Computer-aided reasoned discourse... </title>

(c)

Figure 4.10: Inter-dependencies in a small entity extraction problem.

possibility of seeing the tag *<name>Fox, B., </name>*.

The possibility of *<name>Smolensky, P., Fox, </name>*, *<name>Smolensky, P., </name>* and *<name>Fox, B., </name>* as names is not precluded if the data is segmented into a relation before processing. However, such segmented results could not be merged back into XML using tags such as we are using if these three names are included.

It is unclear whether breaking of the independence criterion matters. Certainly it means that recall and precision results from entity-extraction problems are in some way different from segmentation and classification results, and not directly comparable. Recall and precision are the primary means of comparison in the TREC, MUC and DUC conferences (see Section 2.2.4).

4.6.2 Edit Distance

The correctness of all kinds of metadata used in text augmentation can be measured using edit distance.

4.6.3 Confusion Matrices

As with recall and precision, the effectiveness of confusion matrices on different kinds of text augmentation problems varies.

Segmentation

Confusion matrices of segmentation problems represent a degenerate case in which there are only two classes. The matrix contains the four basic measures from the information retrieval paradigm and is a contingency table:

$$\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} = \begin{bmatrix} true\,positives & false\,positives \\ false\,negatives & true\,negatives \end{bmatrix}$$

For this reason evaluating segmentation using a confusion matrix or the information retrieval metrics produce the same results, but the information retrieval metrics have higher level metrics (recall and precision) built upon them.

Classification

Confusion matrices are the standard method of evaluating classification tasks [149]. Their only disadvantage is that they are somewhat verbose, especially for problems (such as part-of-speech tagging) which have a large number of classes.

Entity Extraction

Confusion matrices have identical independence problems to recall and precision when used to evaluation entity extraction from text. Confusion matrices assume an underlying many-class classification task, but entity extraction in the most general form is more general than this; it is a *hierarchical* many-class classification task. If the hierarchy depth is bounded in some way, it is possible to re-define the problem such that every possible state in the hierarchy is a new class. This approach suffers from problems of combinatoric explosion, leading to large, sparse, matrices which cannot be normalised, since this leads to division by zero.

4.6.4 Type Confusion Matrices

Type confusion matrices are a new extension of confusion matrices suitable for application to hierarchical many-class classification tasks. Every node in the hierarchy is assigned a type, which is the most recently opened tag. The type confusion matrix for a hierarchical classification problem with i classes is:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,i} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,i} \\ \vdots & \vdots & \ddots & \vdots \\ a_{i,1} & a_{i,2} & \cdots & a_{i,i} \end{bmatrix}$$

$a_{m,n}$ in column n and row m is the number of symbols that should have been classified in a node of class n that were actually classified in a node of class m.

Type confusion matrices can be used similarly to confusion matrices, but it should be noted that information has been thrown away. For example, if the sequence *. . . S. Kraus, and V. Subrahmanian. . .* is marked up as:

. . . <editor><name><first>S.</first><last>Kraus,</last></name>and
<name><first>V.</first><last>Subrahmanian</last></name></editor>. . .

rather than as:

. . . <author><name><first>S.</first><last>Kraus,</last></name>and -
<name><first>V.</first><last>Subrahmanian</last></name></author>. . .

the *author* / *editor* confusion would only be apparent in the *and* sub-sequence. Other sub-sequences such as *Kraus,* do not have the erroneous tag as an immediately enclosing tag. This situation is much worse when dealing with classes whose only content is other classes such as the *bibbody* tag which always contains a single other tag.

Type confusion matrices are applicable to any tag insertion problems. However, when applied to a classification problem, they degenerate to a confusion matrix because the immediately enclosing tag is the only tag. When applied to segmentation problems, type confusion matrices degenerate to a contingency table (see page 81).

4.6.5 Entropy

All types of text augmentation can be evaluated using entropy. Care does need to be taken to avoid using the same model or a model built from the same data for both augmentation and evaluation. If entropy is being used for evaluation, it is normal to either use an empty adaptive model or a model built from data which is distinct from the training, re-estimation or testing data.

When a tag insertion using a Viterbi algorithm, produces an incorrect result, entropy measurements can be used to determine whether the fault lies with the model or the searching algorithm. If the result produced by tag insertion has lower entropy than the baseline (or ground truth) text, the model is flawed (i.e. has not seen enough training data, is not of sufficient order, or is attempting to linguistically model non-linguistic features). If the experimental result has higher entropy than the baseline (or ground truth), the searching algorithm is flawed (i.e. one of the heuristics is making an assumption that does not hold for this text). This technique is used in Section 6.4.3 to examine the effectiveness of the alphabet-reduction heuristic.

Chapter 5

The Text

In this chapter the four corpora used in this thesis are introduced, the problems posed by the corpora are described and previous work solving these, or similar, problems is discussed.

In the information-retrieval paradigm, a collection of documents is called a 'corpus' and is assumed to have some commonality: the documents are either from the same source, cover the same topic, or are a representative sample of a larger population of documents. Building corpora, especially those with rich metadata about and within the documents, can be expensive and time-consuming.

In the research community, corpora serve as pools of data for exploratory research [91, 92] and as benchmarks for comparative research [65, 64]. This thesis uses them for both these purposes. The corpora used here are referred to as: the Computists' corpus, the bibliography corpus, the Chinese text-segmentation corpus and the Reuters' corpus. Each of these is discussed in the following sections. Short samples of each can be found in Appendix 1.

5.1 Computists' Corpus

The Computists' corpus [136, 135, 148, 26, 144] is composed of issues of a magazine called 'The Computists' Communique' converted from ASCII text to XML. Each of the 38 issues is approximately 1200 words in length and consists of a number of short articles usually followed by a list of job openings. Previous workers marked up ten features (*name*, *location*, *organisation*, *email*, *source*, *date*, *money*, *phone*, *fax* and *url*) by hand, and then made corrections based on the results of the Teahan's TMT [135].

```
(937) 255-2902. <http://web.fie.com/htdoc/fed/afr/wri/any
/proc/any/07209802.htm>. [CBD, 20Jul98.]
```

(a)

```
<p>(937) 255-2902</p>. <<<u> http://web.fie.com/htdoc/fed/afr/wri/any</u>
/proc/any/07209802.htm>. [<s> CBD</s>, <d> 20Jul98</d>.]
```

(b)

```
<p>(937) 255-2902</p>. &lt;<u> http://web.fie.com/htdoc/fed/afr/wri/any
/proc/any/07209802.htm</u>&gt;. [<s> CBD</s>, <d>20Jul98</d>.]
```

(c)

Figure 5.1: Corrections in the Computists' Communique. (a) the original text (b) the text as received (c) the text used in this thesis.

For this thesis the data was converted from the XML-like format used by TMT into well-formed XML and a number of systemic errors corrected. Figure 5.1(a) shows two lines from corpus as it appears in the original text, notice that a URL has been broken across a line break. Figure 5.1(b) shows the text as used by Teahan, Bray and Wen [135, 26, 144]. Four tags have been added: phone number, URL, source and date. Only the first part of the URL has been marked-up as a URL. The insertion of the URL and email address (not shown) tags was done automatically, inserted extra '<' and the URL detection failed when the URL had been line-wrapped. The text also has un-escaped '<', '>' and '&' (not shown) characters, which are non-well-formed XML. Figure 5.1(c) shows the same text with these deficiencies corrected. This is the version used in this thesis.

The corpus has a number of endemic ambiguity issues: (a) mailing-list names are listed as sources when derived from the mailing list but not when creation of the mailing list is announced; (b) many of the organisation names (particularly *Apple*) were marked up intermittently and (c) many words are marked up coincidentally.

For example, in a discussion about computers from IBM and Sun Microsystems, *Sun* is marked as an organisation even when used as a class of computers. *PC* is never marked as an organisation. These issues, and the fact that organisations and sources are named after places and people, and that place names are often coined from personal names, account for the many of errors previously reported [26, 144].

Several corrections to the corpus are made in this thesis to attempt to resolve (b) and (c). Two passes were made over the corpus, marking-up organisations (and to a lesser extent sources) which had not been marked-up in previous work. This revised corpus is used everywhere in this thesis except Section 6.2.2, where results are compared with previous work and therefore the uncorrected data must be used. To the author's knowledge the corpus is in the public domain. Copies are available from the author.

Inserting the ten features into the Computists' corpus is a classification problem. Figure 4.1(c) shows the schema structure for the problem. The MUC named entity problems from the MUC conferences have strong correspondences to the *name*, *location*, *organisation*, *source*, *date* and *money* tags.

5.2 Bibliography Corpus

The bibliography corpus was created specifically for this thesis from bibliography records. It was designed to resemble the bibliographies found in the computer science technical report collection at the New Zealand Digital Library [153, 109]. The corpus consists of a large number of bibliographies generated by the LATEX / BIBTEX tool-chain which is widely used throughout technically-oriented scientific disciplines. It is anticipated that a model trained on the bibliography corpus may be adaptable for academic fields which use humanities citation conventions by using the Baum–Welch algorithm (see Section 3.6), but this is not explored in this thesis. Marking up bibliographies is a first step for several activities, including document linking, bibliometrics [111] and a range of possible integrated reading list, bibliography and citation systems, making it a desirable feature for a digital library.

A collection of publicly available bibliographic databases[1] has been maintained and expanded by other workers for a number of years. Samples of bibliographic entries were taken from the same sources as this collection, split into 14682 bibliographies with up to 25 entries and formatted using the BibTeX and LaTeX [77] text-formatting systems. Seven of the standard bibliography styles (*abbrv*, *alpha*, *apalike*, *ieeetr*, *plain*, *siam* and *unsrt*) and several different page layout techniques (*article*, *book* and *report*) were used to mitigate secondary effects due to line, column and page wrapping.

Addition of metadata tags into the bibliographies changed the layout of entries; line breaks and hyphenation, in particular, were radically changed. To avoid this, each bibliography was processed twice, once using the standard style file and once using a modified style file which inserted metadata tags around parts of the entries. This process is shown in Figure 5.2. The upper half of the figure shows the processing of the bibliography (.bib) using the unmodified style file (.sty) to produce the laid-out bibliography (.bbl) using BibTeX. This laid-out bibliography was then processed to a PostScript (.ps) document using LaTeX and dvips, and then the PostScript document processed to a text file (.txt) using ps2txt. The lower half of the figure shows the processing of the bibliography using the modified style file to insert escaped XML tags. The resulting two text files were then merged into a single XML document, taking the layout, whitespace and punctuation from the text derived from the unmodified style file and un-escaping the escaped XML tags from the text derived from the modified style file. The resulting bibliographies were processed using the XML ‘preserve-space’ style to preserve whitespace.

There are several peculiarities in the corpus, largely because of how it was constructed.

[1] http://liinwww.ira.uka.de/bibliography/index.html

1. All first names are marked-up in a single tag rather than each first name in a separate tag. The BST language[2] in which the style files are written has primitives for laying out names. Marking-up individual first names separately would have required a modified BST interpreter rather than modified BST programs.

2. There are inconsistencies in the relative location of punctuation and close tags at the end of words. The period following an initial is an indication of contraction, semantically part of the initial, whereas the period at the end of a sentence is semantically separated from the word it follows. The tagging attempts to reflect this, but there are some deeply ambiguous cases, particularly where an initial falls at the end of a sentence and the period fills both roles. In such cases the punctuation has been included within the tag.

3. Splitting a large bibliography into many smaller ones breaks cross-references between entries unless both referrer and referent happen to appear in the same smaller bibliography. Broken cross references appear as '*[?]*'.

LaTeX commands to generate non-ASCII characters in the text are escaped to Unicode characters. The conversion is based upon the commands observed in the corpus rather than a comprehensive list of commands, but includes many common mathematical symbols and letters from a wide variety of Western European languages (Portuguese, Spanish, German, Polish, Swedish, etc.). Most of the letters appear in names, either in the name field or as references to people in titles. A few of the entries were entirely in French. Many bibliography entries with non-ASCII characters also occur in a Romanised form, with the non-ASCII characters converted to ASCII characters by bibliography creators.

[2] The author knows of no comprehensive description of the BST language; the implementation is part of BIBTEX. It is a stack-based language in which sets of non-recursive macros (called 'style files') are used to format convert entries in a standard format (for which again, a canonical description appears to be lacking) into bibliography entries conforming to the stylistic conventions of a particular publication.

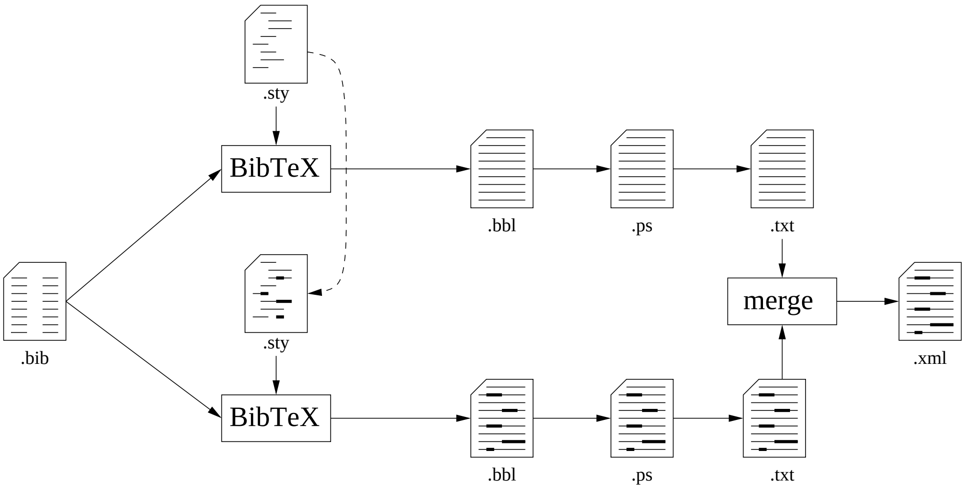


Figure 5.2: Data-flow diagram for creating the bibliography collection.

Escaping non-ASCII characters rather then dropping them out of the corpus made the corpus significantly less close to the computer science technical report collection, but significantly closer to bibliographies as they appear in the majority of electronic documents, and closer to how they were intended to appear. Other researchers have discarded such bibliographies, at the rate of 6.5% [125].

Many of the discarded bibliographies contain LaTeX macros which could never be processed by standard LaTeXṠome appear to be mis-typed macros, but there is no way to distinguish these from macros which individual researchers have defined locally. There are also many sets of macros circulating in subject- and language-specific communities to represent features of interest within those communities. The lack of namespaces in LaTeX means that there is no easy way to differentiate these, and because macro files are imported into the document rather than the bibliography, isolated bibliographies contain no reference to the file name which defines (or redefines) macros.

The structure of the schema is shown in Figure 5.3. The tags at levels B and C indicate bibliographies marked-up according to certain bibliography and document styles respectively. All combinations of these were used when creating the corpus. Tags at level E correspond to tags of different types of documents being referenced.

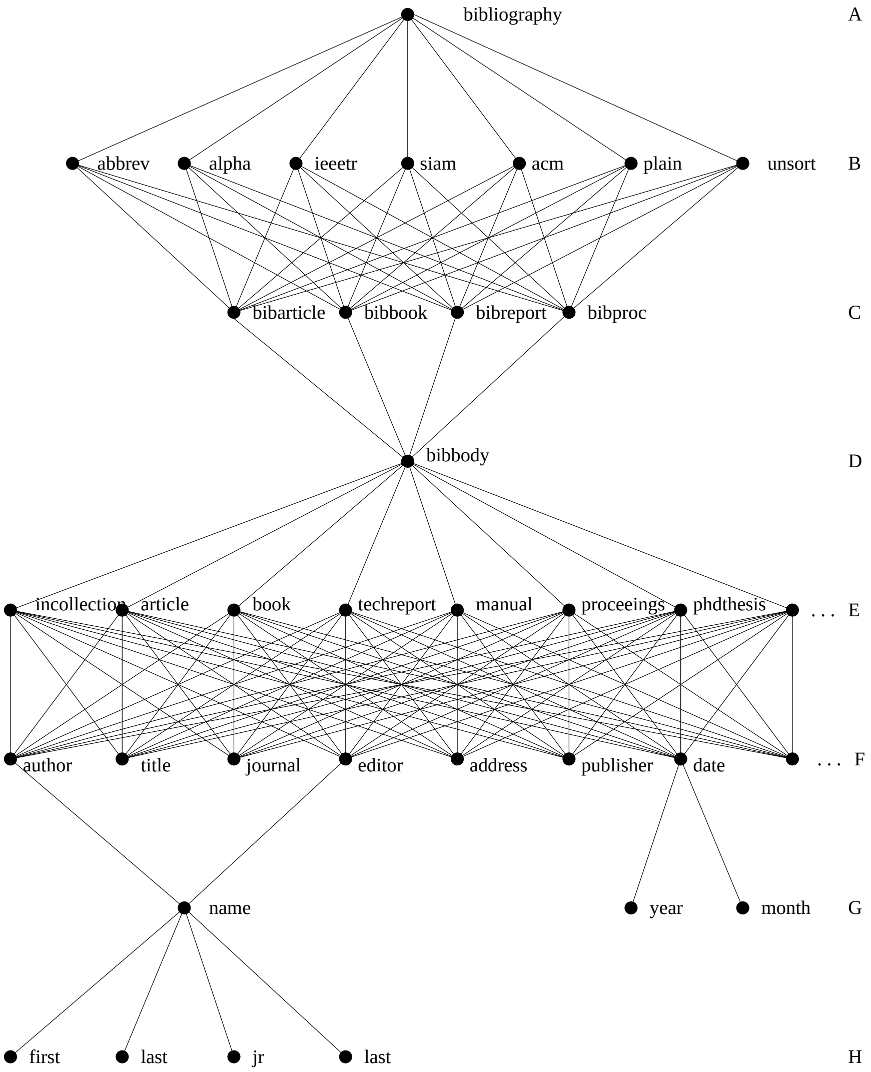


Figure 5.3: Schema for the bibliography corpus with all tags.

Tags at level F correspond to the fields in bibliographic records.

The structure of names in the BIBTEX format is somewhat unusual. With four parts (first, last, van and jr), the structure reflects American English names as conceptualised in the 1980s, but handles rather poorly a number of features of names as used internationally, particularly double-barrelled surnames, von parts[3] starting with a capital and names in which the given name follows the surname. One of the causes is systematic confusion between the portion of the name which is written first and the given (as opposed to inherited, parental) portion. These issues are compounded by the difficulties representing non-ASCII characters in LATEX, for example the need to encode '*Céline*' as '`C{\'{e}}line`,' and the use of a simplistic sorting algorithm for ordering the entries.

A number of different workarounds have been developed to force BIBTEX and LATEX to 'do the right thing' in sorting, formatting and hyphenating particular names. A collection of these can be found in the archives of the `comp.lang.tex` newsgroup. Other name formats, such as the Library of Congress authority lists [112] used in the MARC [108, 48] format are actively curated, enabling such issues to be handled systematically, if not optimally. In this thesis, the original BIBTEX terminology is used because it is precise and clear to workers and tool builders in the field [77, 101].

Not all the tags shown in Figure 5.3 are used in this thesis. Figure 5.4 shows only those tags in the corpus which are used in experiments in this thesis. Note, in particular, that the tags at levels B, C and E in Figure 5.3 are missing in Figure 5.4. The variant schema structure shown in Figure 5.5, and explained in Section 4.3.7, is used in experiments with state tying.

Freitag and McCallum [46, 96] report work on a similar, although non-hierarchical, corpus initially hand-crafted, then incrementally improved using Markov models. Citeseer [80] (see Section 2.2.4) also involves bibliographic data,

[3] In the BIBTEX model of names, fragments such as '*von*' and '*van der*' are referred to as the 'von part'.

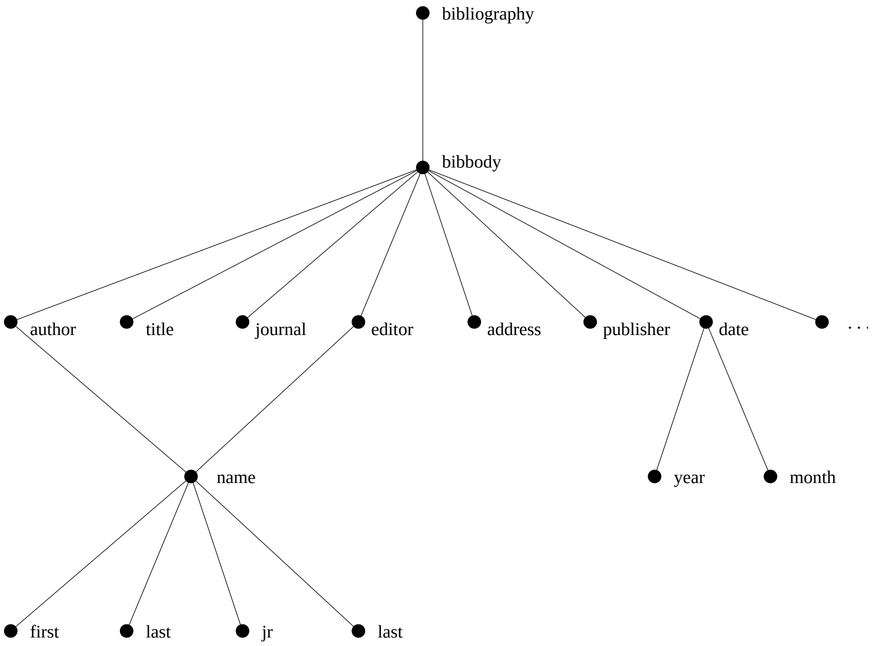


Figure 5.4: Schema for bibliography corpus with tags used in this thesis (with state tying).

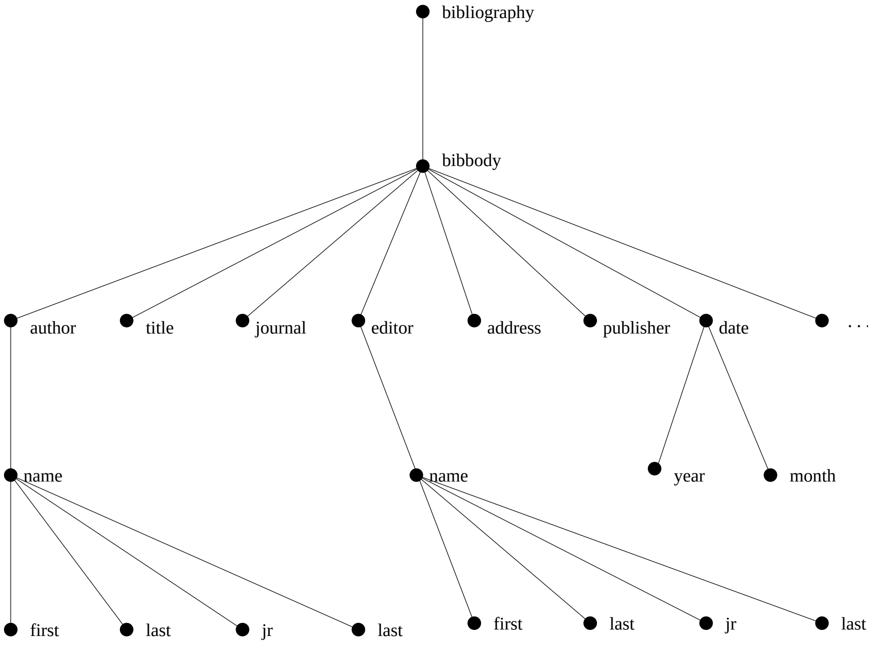


Figure 5.5: Schema for the bibliography corpus without state tying.

using a handcrafted multi-step algorithm.

5.3 Segmentation Corpus

The segmentation corpus was derived from the ROCLING segmentation corpus. which contains about two million pre-segmented words, represented in the Big5 coding scheme. The corpus was converted from Big5 encoding to GB (Guojia Biaozhun) by Wen [137].

The corpus was further converted from GB encoding to Unicode. After inserting *word* tags, whitespace (but not punctuation) was removed and the text split on sentence boundaries into 1000 documents of approximately the same size. The XML was output as ASCII to force all non-8-bit clean characters to be converted into Unicode escapes to reduce the chance of handling errors.

In the resulting corpus, a two character word looks like: *<word>&#x065f6;-&#x05019;</word>*. The corpus also includes western terms (for example, proper nouns and currency symbols). A thorough review of Chinese text segmentation is given in Teahan and Wen [137]. As the author neither reads nor speak Chinese, he is unable to give a detailed analysis. The results of previous workers are shown in Table 6.7.

The segmentation corpus appears to suffer from the overly 'optimistic segmentation' described by Wu and Fung [157]. This phenomenon is caused by the tendency for many segmentation algorithms to be biased towards smaller segments when faced with even genuine ambiguity.

Inserting *word* tags into the segmentation corpus is a segmentation problem. Figure 4.1(a) shows the schema structure for the problem.

5.4 Reuters' Corpus

The Reuters' corpus is a collection of news articles taken from the Reuters' news wire and referred to by Reuters as 'Reuters Corpus, Volume 1, English language, 1996-08-20 to 1997-08-19'. The articles range from two-paragraph summaries of financial information to in-depth articles on political or literary topics. The corpus has been widely studied for a number of purposes, including text categorisation and clustering [62, 55], information extraction [45, 46, 119], authorship [68], and part of speech tagging [46].

This is the sort of news discussed on page 1: automatically inserting tags, either as a first step in a more sophisticated information-extraction process, or simply to tag articles as being connected to the organisations and locations. This process, or one similar to it, is performed ubiquitously in the field of news aggregation.

The corpus was prepared for this thesis by taking the first 7471 articles from the full Reuters' corpus, removing the document level metadata (title, author, topic and copyright information) and passing it through the Brill tagger [28], a widely used part-of-speech tagger that tags every word with a label that indicates the role it plays in speech. The tagger's notion of what constitutes a word is sometimes unusual—*Don't* is regarded as two words and *dollar/yen* as one word—but the tagger was used 'out of the box' according to accepted practice [46, 119]. 11 documents containing URLs, which confused the tagger's parser, were removed. The full Reuters' corpus contains many duplicates [69], but as with other corpora and information systems, the presence or absence of duplicates is not as important as whether the corpus is a representative sample of the larger population of documents. Given that identical or similar news articles commonly appear in a number of publication outlets, having duplicates and near-duplicates in the Reuters' corpus is a sign of correlation with 'real-world' news sources, rather than a sign of a flaw.

The full Reuters' corpus is large (over 800,000 articles), but only the first block of articles is used here, since the behaviour of text augmentation on large bodies of text is not the primary interest of this thesis and has been studied elsewhere [133]. A complete explanation of the meanings of each of the 38 tags is contained in [94]. The text of the Reuters' corpus is copyright Reuters and not for redistribution. Copies of the corpus are, however, available from Reuters.

Inserting part of speech tags into the Reuters' corpus is a classification problem. Figure 4.1(b) on page 52 shows the schema structure for the problem.

Chapter 6

Results

In this chapter the effects of applying the earlier discussed optimisations and heuristics to the four corpora discussed in the previous chapter are examined. The correctness results are then given and, where possible, compared against experimental results given in the literature. The effects of Baum–Welch re-estimation are examined and, finally, the effectiveness of individual optimisations and heuristics are examined.

6.1 PPM-SY versus PPMD

CEM normally uses PPM-SY, and in this section it is compared with PPMD . Figure 6.1 shows the search time per node of the search in the Computists' corpus, for a range of orders of model and a lookahead of six. The search time increases less than linearly for PPM-SY and more than linearly for PPMD.

Despite the use of leave-one-out cross-validation, the correctness of PPM-SY and PPMD was identical in all cases except for the case of the location *Capitol Hill*, which was correctly identified as a location by PPMD using models of order three and four when PPM-SY incorrectly identified it as an organisation. Using an order-five model correctly identified it as a location.

Figure 6.2 shows the search time per node of the search in the Chinese segmentation corpus, for a range of orders of model and a lookahead of four. The time increases less than linearly for PPM-SY and more than linearly for PPMD. This increase in the cost is substantially larger than in the Computists' corpus, probably

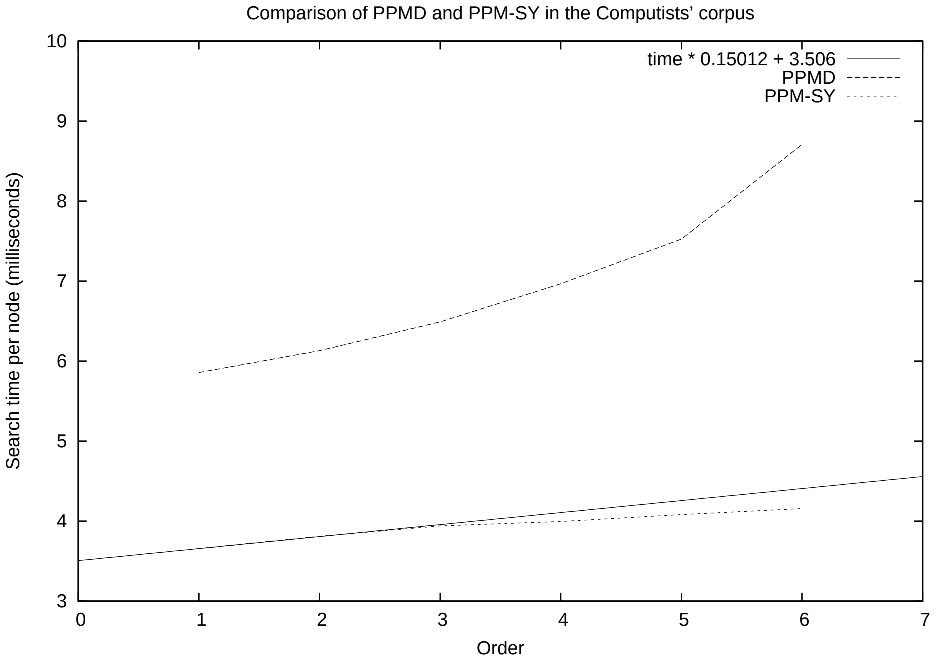


Figure 6.1: Graph showing the speed of searching in the Computists' corpus for PPMD and PPM-SY. A reference line is included to show that the speed for PPM-SY is growing less than linearly with respect to model order. Timings are averaged over leave-one-out cross-validation.

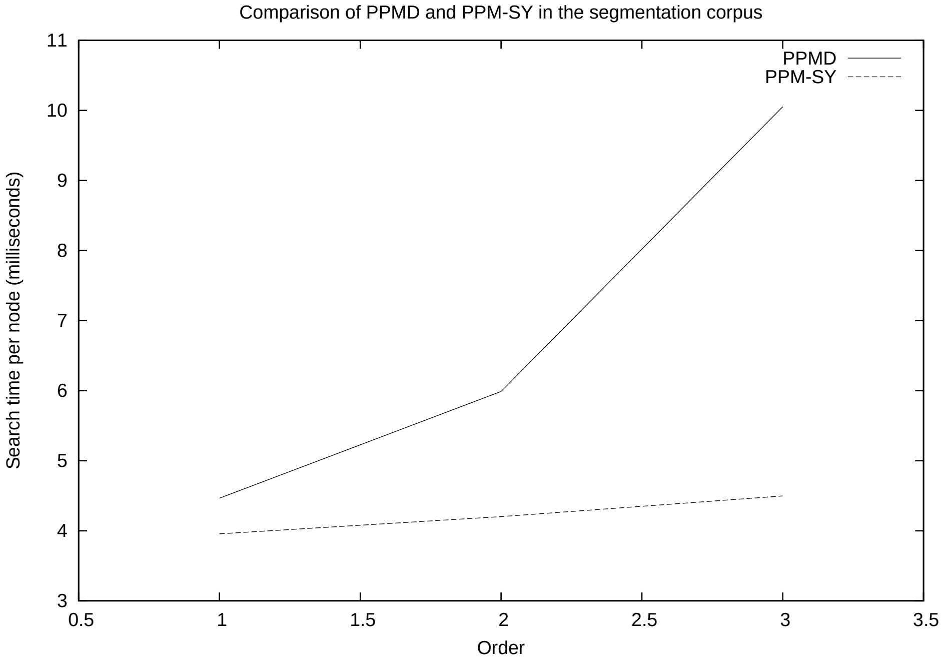


Figure 6.2: Graph showing the speed of searching in the segmentation corpus for PPMD and PPM-SY. All runs use 900 training documents and a single testing document. Results shown are averages over 100 runs.

because of the significantly larger character set involved. PPMD gave better results, on average, than PPM-SY, with a difference in F-measure of $+0.03$%, $+0.02$% and $+0.04$% for orders one, two and three respectively.

6.2 Correctness

Correctness (see Section 2.3) is studied on a corpus-by-corpus basis. Leave-one-out cross-validation is used only for the Computists' corpus, because that corpus is so small. In all other experiments, no cross validation is used except where specifically stated.

6.2.1 Granularity and Heterogeneity

Unfortunately text-mining systems of the type being examined in this thesis make the assumption that text seen during training is the same as the text seen during testing. In this sense they are not general-purpose systems in the way that PPM [133], bzip [88] or gzip [124] are. How well this assumption hold varies from corpus to corpus depending in the internal granularity and heterogeneity. For the corpora described in Chapter 5:

- The Chinese text-segmentation corpus was built from pre-homogenised data, no variation among the 1000 documents is apparent to the author.
- The Computists' corpus contains documents which all have the same structure, but with considerable variation on subject matter.
- The Bibliography corpus contains relatively homogeneous documents with two exceptions: (a) those documents generated from personal bibliographies containing all publications by an individual, and (b) those documents generated from forum bibliographies containing all publications appearing in a journal, conference or book series. These documents are entirely an artifact of the way the data was prepared—an insignificant number of peer reviewed articles are published in computer science which contain references to only a single author or source.
- The Reuters' corpus, by contrast, contains genuinely heterogeneous articles, ranging from short market-report articles, with columns of numeric figures, to long in-depth articles of political commentary.

Only the Reuters' corpus is evaluated both at a corpus level and at a document level (see Section 6.2.5). The other corpora are evaluated at the corpus level.

6.2.2 Computists' Corpus

The Computists' corpus has been previously studied by Bray [26], using TMT, and Wen [144]. Bray evaluated extraction based upon a confusion matrix (see Section 2.3.3) and this is reproduced in Table 6.1(a). Tables 6.1(b) and (c) show the confusion matrices for CEM on the corrected data using maximum lookahead search and Teahan search respectively. The values in (a) are measured in words, the values in (b), (c) and (d) are in characters. The *issue* tag is the background: both TMT and CEM build Markov models for the *issue* tag but Bray does not report the full results for this, so the CEM results in (b), (c) and (d) have an extra row.

For most of the tags the CEM results were comparable to, but slightly worse than the results given in Bray. Because the Bray results are percentages of words correctly classified and the CEM results are percentages of characters correctly classified, direct comparison between these results is difficult. Many of the mistakes shown in Table 6.1 for both systems appear be connected to inconsistencies, as described in Section 5.1.

Three of the tags with the best performance (*url*, *email* and *money*), deserve close attention. The first two can be described using a regular expression and the last is uniquely and exclusively identified by a single character (*$*). These properties make tag insertion much more consistent; they also make modelling such tags easier for certain kinds of models. Unfortunately it also makes marking-up using Markov models pointless: except in extreme cases marking up by regular expression is always more efficient than marking-up using Markov models and searching.

The systemic confusion between *name*, *source*, *location* and *organisation*, as discussed in Chapter 5, is clear in all three confusion tables, with greater confusion for CEM than for TMT.

Another situation in which CEM performs much worse than the Bray analysis is the *fax* tag. The most common type of error with *fax* and *phone* tags in both systems is where the fax numbers are mistaken for phone numbers: *<p>617-373-*

	d	n	s	l	o	u	e	p	f	m	i
[d]ate	**93.46**		+								6.40
[n]ame		**89.35**	+	1.31	1.50						7.48
[s]ource		+	**60.09**		2.85						36.62
[l]ocaton		+		**81.64**	4.69						12.89
[o]rg		2.56	2.56	1.63	**69.23**						24.01
[u]rl						**100.00**					
[e]mail							**97.34**				2.66
[p]hone								**82.29**	10.71		
[f]ax									**100.00**		
[m]oney										**100.00**	

(a)

	d	n	s	l	o	u	e	p	f	m	i	#
[d]ate	**91.18**	+	+	+	+			+	+		8.12	10070
[n]ame	+	**85.49**	2.75	1.78	2.02		+		+		7.27	10494
[s]ource		1.13	**51.97**	+	3.02		+		+		41.71	9983
[l]ocation	+	2.66	1.96	**72,38**	4.79		+		+		17.65	5155
[o]rg	+	3.48	2.99	3.66	**27.50**	+	+		+		60.99	5688
[u]rl	+	+	+	+	+	**95.23**	+	+			3.11	20023
[e]mail	+	+	1.14	+	+	+	**93.60**	+	+		3.30	12164
[p]hone								**88.69**	9.95		1.36	955
[f]ax								27.86	**69.14**		3.01	499
[m]oney									+	**99.47**	+	1133
[i]ssue	+	+	1.14	+	1.47	+	+	+	+		**95.92**	317169

(b)

	d	n	s	l	o	u	e	p	f	m	i	#
[d]ate	**91.04**		+	+	+		+				8.22	10098
[n]ame		**87.92**	2.19	1.26	3.92		+				4.64	11167
[s]source	+	1.27	**64.02**	+	5.99		+				27.81	14229
[l]ocation	+	2.46	1.26	**75.82**	11.38	+					8.75	5534
[o]rg		2.57	2.13	4.27	**58.48**		+				32.40	12212
[u]rl		+	+		+	**95.90**	+		+		2.88	20089
[e]mail	+	+	+	+	+	1.08	**94.70**				3.38	12186
[p]hone								**75.03**	8.88		16.10	969
[f]ax								16.43	**57.11**		26.45	499
[m]oney			+					+	+	**90.35**	7.98	1140
[i]ssue	+	+	+	+	1.28	+	+	+	+	+	**96.94**	303100

(c)

	d	n	s	l	o	u	e	p	f	m	i	#
[d]ate	**92.23**	+	+	+			+				6.99	10075
[n]ame		**92.46**	+	+	2.65						3.49	11135
[s]source	+	+	**68.11**	+	5.12		+				25.57	13881
[l]ocation		1.62	+	**84.53**	7.87						5.86	5619
[o]rg	+	2.03	2.47	2.39	**66.47**		+				26.53	12169
[u]rl		+	+		+	**96.48**	1.00				2.25	19668
[e]mail	+	+	+	+		+	**96.55**				2.23	12436
[p]hone								**72.55**	6.60		20.85	969
[f]ax		1.20						4.21	**70.14**		24.45	499
[m]oney			+					+	+	**88.80**	9.58	1107
[i]ssue	+	+	+	+	1.09	+	+	+	+	+	**97.48**	301326

(d)

Table 6.1: Confusion matrices for the Computists' corpus (a) from Bray using TMT [26] page 70, (b) from CEM/maximum lookahead using the same data as Bray, (c) from CEM/maximum lookahead using corrected data, (d) from CEM/Teahan search using corrected data. Character counts (#) are in characters, all other values are in percent, '+' indicates a figure lower than 0.99%. A lookahead of 6 was used.

Author	Recall	Precision	F-measure
Wen	65.29	73.35	69.09
CEM/maximum lookahead (Wen's data)	49.17	63.38	55.38
CEM/maximum lookahead (corrected)	71.06	61.21	66.13
CEM/Teahan (corrected)	74.65	67.71	71.18

Table 6.2: Accuracy for the Computists' corpus, from Wen [144] page 75 and from the current work. A lookahead of 6 was used.

5358</p>, <p>617-373-5121</p><f>Fax</f>. In CEM, because of the small number of *fax* tags seen (28 at most), the model for the *fax* tag is the closest to an untrained model: it is the least biased against apparently random sequences. The range of characters seen in the *fax* tag is narrow, but not significantly narrower than *phone* tag. This results in errors such as: *<f>REAL</f>basic*, *<f>pp. 43-45</f>*, and *Unix <f>ht://Di</f>g search*.

As predicted in Section 4.5, CEM with Viterbi search performed differently from CEM with Teahan search. With the ability of Teahan search to 'see' long distances it might have been expected to correctly classify phone and fax numbers, which commonly have the differentiator at the end. Unfortunately the numeric content of these tags, being effectively random digits, has high entropy which limited the gains made here. The clearest improvements were situations such as *(703) 306-0599 Fax* which maximum lookahead search broke in two as: *<p>(703) 306-0599</p><f>Fax</f>*, whereas Teahan search correctly marked-up as *<f>(703) 306-0599 Fax</f>*.

Wen [144] expresses accuracy in terms of recall, precision and error rates for each type of tag, as shown in Table 6.2. The Wen model is trained on 25 documents, whereas this thesis uses leave-one-out cross-validation for the Computists' corpus. The apparent reason for the better performance of Teahan search in this case is that many of the ambiguities are of type (a) rather than type (b), as shown in Figure 4.8. The values in Table 6.1 bear no direct relationship with those in Table 6.2 because the former are at the word (or character) level, whereas the latter is the recall and precision of whole tags (excluding the issue tag).

name	pages	date	volume	number	title	journal	booktitle	publisher	address	bibliography	count character
97.80					1.46	+	+	+	+	+	626,917
	98.79		+		+		+			+	140,260
+	+	**97.32**	+		1.07	+	+	+	+	+	161,128
+	+	+	**95.32**	+	1.05	+	+	+	+	2.99	31,182
	+	+	1.32	**96.54**	+		+		+	1.03	11,321
1.47		+			**92.58**	+	3.84	+	+	1.53	1,510,757
+		+	+		1.31	**94.55**	+	+	+	3.17	276,285
+		+	+		7.54	+	**88.61**	+	+	2.83	272,458
+	+	+	+		3.45	1.72	+	**82.71**	3.82	6.93	98,670
+	+	+	+		2.32	+	1.62	3.92	**86.77**	3.99	101,187
0.61	+	+	+	+	2.31	6.55	2.56	1.37	1.01	**84.68**	937,928

Table 6.3: Confusion matrix for the bibliography corpus without *note*. Counts are in characters, all other values are in percent, a ‘+’ indicates a figure lower than 0.99%. Order 6 models trained on 6000 documents and tested on 1000 documents with a lookahead of 5.

6.2.3 Bibliography Corpus

Because the bibliography corpus was developed in the present study, there is not a wide range of results from other systems to compare the results from CEM against. Wen [144] gives some results on three tags (*publisher, date* and *pages*) from an early version of the corpus, but these results are not sufficiently detailed for comparison.

Table 6.3 shows the confusion matrix for a large number of tags in the bibliography corpus. A significant number of the errors were caused by use of the note field in BIBTEX. This field allows arbitrary text to be inserted at the end of an entry. Often this extra text is an abbreviated reference (for example: *(Published version of*

UWCS Tech. Report No. 226., 1974)), information which should ideally be in other fields of the reference (such as *Lecture Notes in Computer Science 866* should be in the series and number fields) or a citation (such as *Erratum in it JPL 25:5, 2000, pp. 541–542.*). In Table 6.3 the note tags were stripped prior to tagging, the text previously included in them appeared at the document level, polluting the trained model by adding noise.

The root of these errors is that the generation of the corpus (and all BIBTEX processing) assumes that the BIBTEX file format is prescriptive, when in fact it is descriptive: users will put whatever they need to into a BIBTEX file to get the entry to look 'right' in the style they are using. This leads to a situation in which the meaning of bibliographic entries (when formatted for publication) is clear to researchers and librarians passingly familiar with the field, but the content of the BIBTEX fields does not correspond to field definitions. No increase in lookahead, training data or model order can remedy such a problem.

A different kind of error is seen at the boundary between the author list and the document title because of the wide variation in layout of the author list and the tendency of titles to start with lengthy proper nouns which are easily mistaken for author names. The first word or two of the title are sometimes tagged as author names, either as part of the last genuine author name or as a separate name. This kind of error is strongly linked to the lookahead (see section 6.4.4): as more context is taken into account these errors diminish.

Table 6.4 shows a confusion matrix with the *note* tag added. The overall performance is not substantially different, but that for the *number* tag drops considerably. This appears to be because many of the *note* tags contained numeric sequences (see examples above) and separating *note* tags out from the background model enables it to effectively model numbers.

Table 6.5 shows the type confusion matrix for the bibliography corpus. The *bibliography* tag is still the document tag, but almost all the content is now with

name	pages	date	volume	number	title	journal	booktitle	publisher	address	note	bibliography	count character
97.75			+		1.14	+	+	+		+	+	67807
	98.32				+					+	+	13245
+		**96.40**		+	1.41	0.05	1.16	+		+	+	15275
		+	**96.09**	+	1.17	0.09		+		+	1.63	3426
			1.22	**87.13**					1.82	2.84	6.99	987
2.20		+	+	+	**91.89**	+	3.84	+	+	+	+	150900
			+		1.08	**94.30**	+	1.18		1.22	2.05	29738
+	+	+			5.42	+	**90.90**	+	+	+	1.49	27407
1.14					4.96	1.89	1.12	**83.20**	2.11	1.90	3.69	11178
+		+	+		1.78	+	1.39	6.82	**85.14**	1.38	1.95	12715
+	+	+			2.43	+		1.55	+	**93.40**	1.17	16476
+	+	+	+		1.57	3.12	3.54	1.49	+	+	**87.49**	82518

Table 6.4: Confusion matrix for the bibliography corpus with *note*. Counts are in characters, all other values are in percentage, a '+' indicates a figure lower than 0.99%. Order 6 models trained on 6000 documents and tested on 100 documents with a lookahead of 5.

bibbody tags which contain the bodies of the references (but not the leading reference key in bibliography styles which use one).

Many of the characters mistakenly marked-up as *bibbody* are punctuation (and the *note* tag as explained above), whereas the errors in the *title* column mainly represent the first few words of the *title* confused with the end of the preceding *author* tag. As in Tables 6.3 and 6.4, there is confusion between *title* and *booktitle* because *booktitle* is used in the place of *title* when there are two titles to a document (i.e. a chapter title and a book title, or an article title and a collection title).

There is confusion between the *publisher* and *address* tags because many *publisher* tags have the address of the publisher included within them, especially in entries for *proceedings* and *inproceedings* in which the *address* tag is reserved for the address of the conference rather than the publisher.

In Table 6.4, the *name* from Table 6.3 has been split into five separate tags: *editor*, *author*, *name*, *first* and *last*. There is considerable confusion among the various tags, but surprisingly little difference between the *editor* and name *tags*, because the *name* is almost always immediately following a *bibbody* start tag while an *editor* tag is in the middle of the *bibbody* tag.

Table 6.6 shows the effect of increasing model order—as the model order increases, the experimental result converges with the expected results, the number of defects falling. Placing name tags is particularly challenging because of the diversity in the way names are laid out in the training text.

The results given here appear much better than the figures given for other systems, such as [46]. However, such a direct comparison is at best an approximation because of the different granularity at which the results are measured and the different number of tags. Informal comparison of these results to uncorrected results[1] listed on the Citeseer website[2] suggest that a significantly better determination of

[1] The Citeseer system allows for users to correct or complete bibliographic information. These corrected entries are not considered here.

[2] `http://citeseer.nj.nec.com/cs`

bibbody	editor	author	name	first	last	pages	date	volume	number	title	journal	booktitle	publisher	address	bibliography	count character
79.51	+	+	+	+	+	+	+	+	+	+	7.47	5.17	1.78	1.55		255,064
3.15	**85.11**	2.98	1.94	+	3.84					2.57		+	+	+		12,432
+	+	**92.58**	2.47	+	+					3.86	+		+			26,446
1.46	+	4.17	**89.51**	+	1.82					2.11	+	+	+			20,261
+		+	+	**94.68**	2.87					1.90	+	+	+	+		79,359
+		+	+	1.33	**94.61**	+				3.32	+	+	+	+		149,929
+	+			+	+	**99.09**	+	+		+			+	+		56,241
+				+		+	**97.17**	+	+	1.01	+	1.07	+	+		64,928
2.56						+	+	**95.31**	+	+	+	+	+	+		12,805
+						+	+	+	**97.73**	+		+				4,321
1.25		+	+	+	1.25		+	+		**93.73**	+	2.61	+	0.16		602,944
3.19				+	+	+		+		1.01	**95.00**	+	+	+		118,350
2.99			+	+	+		+			10.17	+	**85.37**	+	+		114,846
9.29	+		+	+	+	+	+	+		4.65	1.53	+	**80.04**	3.20		40,286
3.87	+		+	+	+	+	+	+	+	2.41	+	2.16	2.33	**87.94**		48,225
+				+						+		+			**99.84**	77,247

Table 6.5: Type confusion matrix for the bibliography corpus for many tags. Character counts (#) are in characters, all other values are in percent, a '+' indicates a figure lower than 0.99%. Order 6 models trained on 6000 documents and tested on 300 documents with a lookahead of 5.

Order	Text
0	[5] <name><first>T. </first><last>Matsui,</last> ◇ ◇ <first>T. </first><last>Matsuoka,</last> </name>and <name><first>S.</first> <last>Furui,</last></name> <title>/Smoothed N-best-based speaker adaptation for speech recognition,"◇ in ◇Proc. ICASSP◇*</title>* ’*<pages>*97,*</pages>* (*<journal>*Munich, Germany*</journal>*), pp. <pages>1015–1018,</pages> Apr. *<date>*1997</date>.
1	[5] <name> <first> T. </first> <last> Matsui,</last> <first> T. </first> <last> Matsuoka,</last> </name> and <name> <first> S.</first> <last> Furui,</last> </name> <title> /Smoothed N-best-based speaker adaptation for speech recognition,"◇ in ◇Proc. ICASSP*</title>* ’*<pages>* 97,*</pages>* (*<journal>* Munich, Germany*</journal>*), pp. <pages> 1015–1018,</pages> Apr. <date>1997</date>.
2	[5] <name> <first> T.</first> <last> Matsui,</last> </name> <name> <first> T.</first> <last> Matsuoka,</last> </name> and <name> <first> S.</first><last> Furui,</last> </name> <title> /Smoothed N-best-based speaker adaptation for speech recognition,"</title> in <booktitle> Proc. ICASSP ’97◇, (◇Munich, Germany◇),*</booktitle>* pp. <pages> 1015–1018,</pages> <date>Apr. 1997</date>.
3	[5] <name> <first> T.</first>◇ ◇ <last> Matsui,</last> <first> T.</first><last> Matsuoka,</last> </name> and <name> <first> S.</first> <last>Furui,</last> </name> <title> /Smoothed N-best-based speaker adaptation for speech recognition,"</title> in <booktitle> Proc. ICASSP ’97,</booktitle>(<address> Munich, Germany</address>), pp. <pages> 1015–1018,</pages><date> Apr. 1997</date>.
4	[5] <name> <first> T.</first> ◇ ◇<last> Matsui,</last> <first> T.</first><last> Matsuoka,</last> </name> and <name> <first> S.</first> <last>Furui,</last> </name> <title> /Smoothed N-best-based speaker adaptation for speech recognition,"</title> in <booktitle> Proc. ICASSP ’97,</booktitle>(<address> Munich, Germany</address>), pp. <pages> 1015–1018,</pages><date> Apr. 1997</date>.
5	[5] <name> <first> T.</first> <last> Matsui,</last> </name> <name> <first> T.</first> <last> Matsuoka,</last> </name> and <name> <first> S.</first><last> Furui,</last> </name> <title> /Smoothed N-best-based speaker adaptation for speech recognition,"</title> in <booktitle> Proc. ICASSP ’97,</booktitle> (<address> Munich, Germany</address>), pp. <pages> 1015–1018,</pages><date> Apr. 1997</date>.
Expected	[5] <name> <first> T.</first> <last> Matsui,</last> </name> <name> <first> T.</first> <last> Matsuoka,</last> </name> and <name> <first> S.</first><last> Furui,</last> </name> <title> /Smoothed N-best-based speaker adaptation for speech recognition,"</title> in <booktitle> Proc. ICASSP ’97,</booktitle>(<address> Munich, Germany</address>), pp. <pages> 1015–1018</pages> , <date> Apr. 1997</date>.

Table 6.6: Example of effect of model size on defects, using models trained on 4000 documents and a lookahead of 5. Tags in *italics* are incorrectly placed. ◇ indicates a missing tag.

Author	Corpus	Recall	Precision	F-measure	Perfect	Ref.
Peng	People’s Daily & Treebank	74.0	75.1	74.2	Yes	[116]
Ponte & Croft	People’s Daily & Xinhua	93.6	96.0	94.8	Yes	[117]
Ponte & Croft	People’s Daily & Xinhua	89.8	84.4	87.0	No	[117]
Palmer	TREC-5	—	—	82.7	Yes	[113]
Teahan	Xinhua	93.4	89.6	91.5	No	[137]
CEM/Teahan	ROCLING	97.8	98.1	97.9	No	
CEM/Viterbi	ROCLING	98.2	98.0	98.1	No	

Table 6.7: Performance of Chinese text segmentors. Perfect indicates that the system uses a perfect lexicon.

non-name structures by CEM and similar determination of names by CEM and McCallum’s system described in [47, 75].

6.2.4 Segmentation Corpus

Segmentation of Chinese text is an archetypical segmentation task and there are many published recall and precision figures for this task. Table 6.7 shows a selection of these, together with the best-case results obtained in the present study for CEM on the segmentation corpus described in Section 5.3. Many systems use a perfect lexicon: a list of all words which may be seen during testing and effectively solves the zero frequency problem [146] but prevents the results from being transferred to many real-world problems. The difference between the two Ponte and Croft results[117] in Table 6.7 shows the drop in performance of a system used with and without a perfect lexicon. Production systems typically cannot assume access to a perfect lexicon. There is a relationship between the perfect lexicon and the order -1 (or 0-gram) model in PPM, which includes all characters representable in the character set,

The results from CEM using maximum lookahead search and CEM using Teahan search are similar, with the maximum lookahead search performing marginally better. The Teahan search used 2000 leaves and averaged 5983 nodes per character. The maximum lookahead search used a lookahead of 6 and averaged 4081 nodes per character. Both used an order 3 model trained on 900 documents and 10 testing documents.

Taken at face value, the results for CEM are clearly better than those for the other segmentation systems. However, most of the other systems appear to be assessing recall and precision on the number of whole words rather than on word boundaries, which can double the perceived number of false positives and false negatives for isolated errors. This is because a single segmentation error can cause the words on either side of a boundary both to become false negatives. Another issue is that the data used in the present work was sorted at the sentence level, and it is not clear that this was the case for the other reported results. Data was used in the form it was obtained in, and with no notes on the sorting or otherwise in the literature, no extra processing was performed.

CEM differs from Teahan's TMT system in internal character handling. TMT uses ASCII internally, breaking Unicode characters into multiple characters. Because of the way in which Unicode characters are laid out in the available 32 bits (in 'code pages') there are a number of artifacts, the primary one being that novel Unicode characters are always mapped to novel characters within CEM, escaping back to the order -1 model, but within TMT they may not escape back only as far as the code page. As noted earlier, there is no *a priori* reason for preferring one escape method over another (see Section 3.4) and these results are unlikely to be generalisable beyond Chinese text segmentation.

Because of the large alphabet used in Chinese, the models for even modest orders are large, making the problem significantly more difficult than it would be in a smaller alphabet language such as English. No attempt has been made to optimise the memory usage by CEM models, meaning that it cannot be used to build such large models as Teahan's TMT.

6.2.5 Reuters' Corpus

Figures 6.3 and 6.4 show both recall and precision curves for the entity extraction task in the Reuters' corpus, with training on 7100 documents and testing on 100

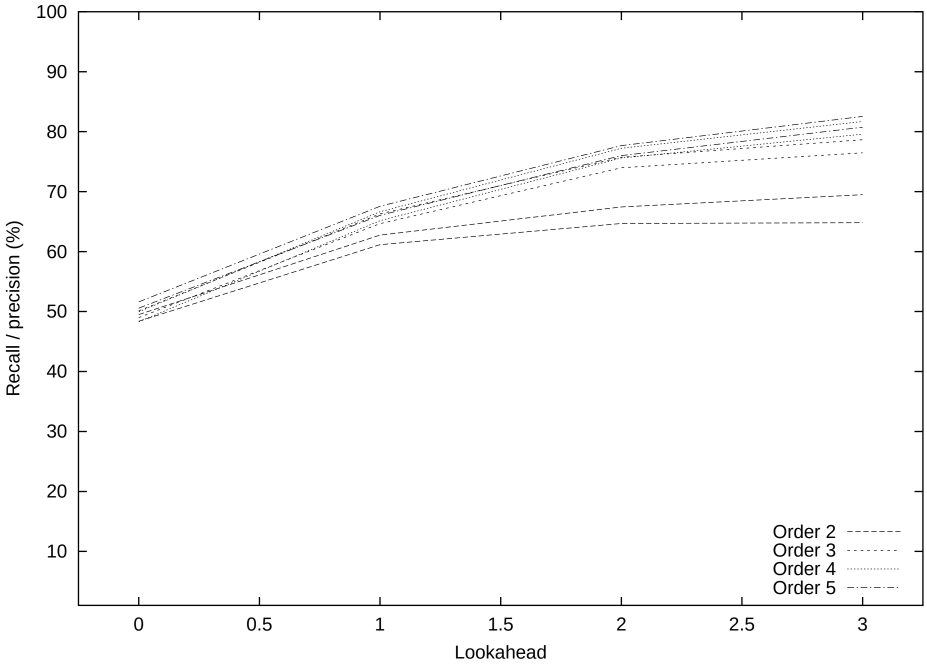


Figure 6.3: Graph of recall and precision against lookahead for various orders of models for documents in the Reuters' corpus.

documents. The difference between Figures 6.3 and 6.4 is granularity, as explained in Section 6.2.1. Figure 6.3 shows recall and precision calculated for each document and then averaged over the testing set. Figure 6.4 shows the recall and precision calculated over the entire testing set. In every case shown, recall and precision are highly correlated and similar.

The difference between Figures 6.3 and 6.4, up to six percent and greatest at low lookaheads, is caused by a number of shorter market-report articles with columns of figures which are easier to tag than are longer articles of a more literary nature. Fortunately, while the results are different, the trends are still clearly the same: incremental gains as the lookahead is increased. Unfortunately the prohibitive size of large models prevented the creation of higher order models.

Overall, the performance of CEM was poor, as state-of-the-art taggers routinely have recall and precision measures in the 90% range [28]. The results are particularly disappointing since the baseline data was generated using a finite-state based

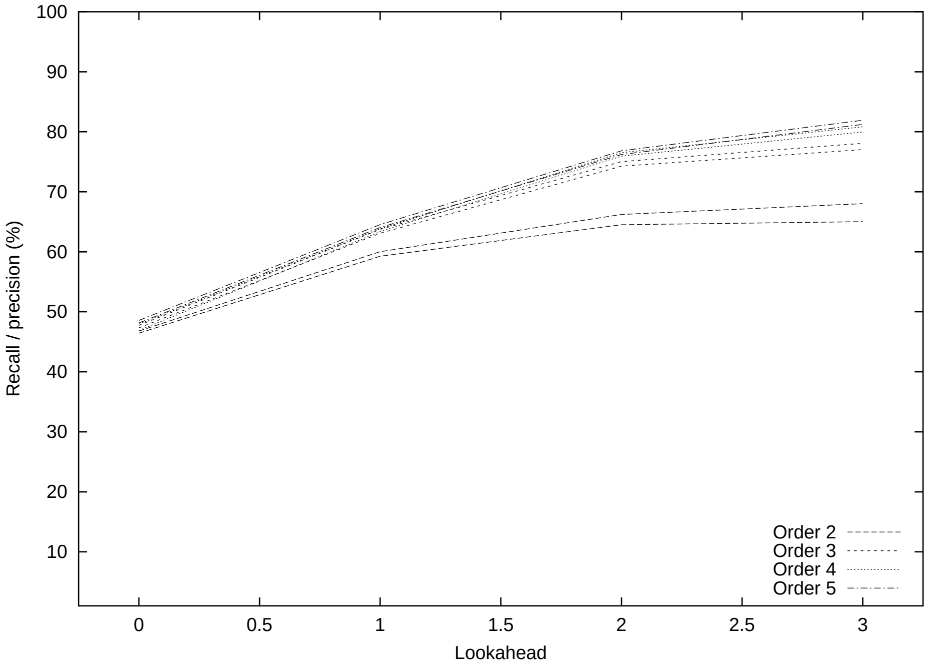


Figure 6.4: Graph of recall and precision against lookahead for various orders of models for the Reuters' corpus taken as a whole.

system (the Brill tagger) which word-level taggers have been able to emulate relatively easily. There are two possible causes. Firstly, whereas the Brill tagger uses a model and search context of a handful of words, CEM uses a model and search context of a handful of characters. Secondly, CEM's linear context and lack of super-adjacency handicapped it against the Brill tagger which uses rule-based post-processing which can examine not just immediate words, but more remote words. Small-scale investigations suggested that increasing model order and lookahead had little effect.

6.3 Baum–Welch Re-estimation

The Baum–Welch algorithm (see Section 3.6) allows untagged data to be used to boost models' performance. This section looks at the application of Baum–Welch re-estimation in the bibliography corpus. This is pertinent, because, as has been

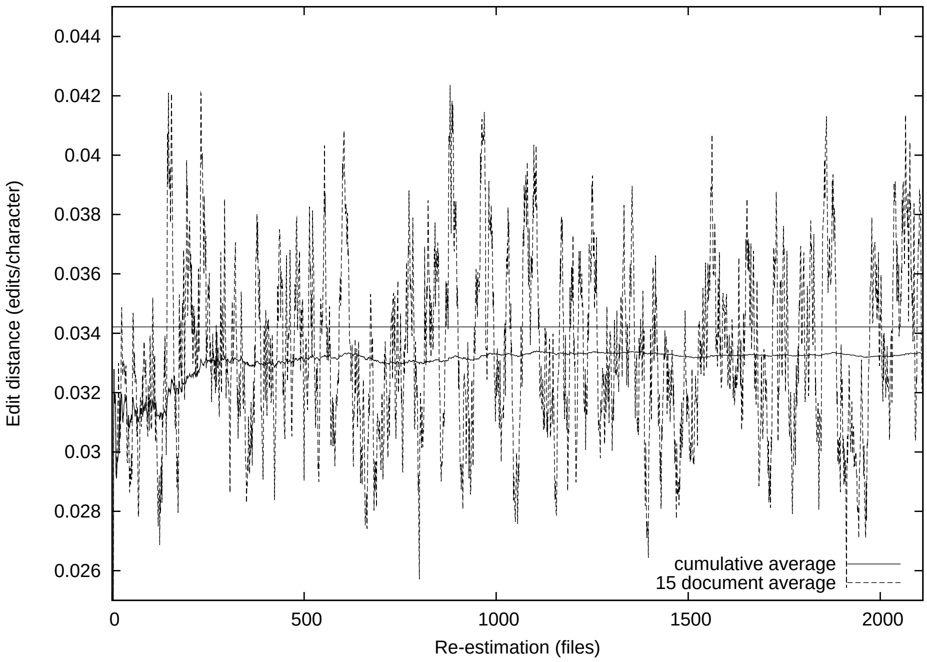


Figure 6.5: Graph of edit distance with increasing re-estimation. Trained with 2110 *abbrv* documents, re-estimated with up to 2111 *acm* documents, using the *first* and *last* tags only, order 4 and lookahead of 3.

pointed out in Sections 5.2 and 6.2.3, the bibliography corpus is significantly less diverse than an uncurated bibliography collection in a digital library and it would be beneficial to be able to generalise the models built on the bibliography corpus to these more diverse collections.

Figure 6.5 shows an attempt to generalise from the *abbrv* bibliography format to the *acm* bibliography format. The *abbrv* format is an abbreviated form with author forenames initialised, while the *acm* format is more standard style which includes the full author forenames, if known. Only the *first* and *last* tags are considered.

As might be expected, a model built on the *abbrv* format and tested on the *acm* format makes many errors. The line across Figure 6.5 at 0.0342 edits per character is the average number of edits over the entire 2111 *acm* documents without any re-estimation. The most common error is the tagging of a *first* tag as a *last* tag, which is seen by the edit distance metric as four separate errors: removing one opening and one closing tag, and adding one opening and one closing tag. A novel error is the misidentification of *eds* (the token indicating the start of an editor list in the *acm* format) as last name.

The 15-document average is a running average of the previous 15 points. It shows a great deal of noise and no obvious pattern of increase or decrease. The cumulative average reaches 0.0323 edits per character after all 1269 documents, a significant drop from the 0.0342 edits per character without re-estimation. Re-estimation clearly reduces the edit distance in this case, lowering the average edit distance for the *acm* documents. EM theory [60] predicts this is not a true convergence (as an increasing proportion of the data is estimated rather than true data, the fidelity of the model slowly falls) but there is insufficient re-estimation data in this example for this to become apparent.

The documents are processed here in random order, but these figures are particularly sensitive to the order in which the documents are processed. The first handful of documents used in the re-estimation appear to be important. It may be worth exploring whether documents should be used ordered in some manner, perhaps those with the lowest mutual-entropy first.

6.4 Effectiveness of Optimisations and Heuristics

The bibliography corpus is a useful dataset for evaluating the effectiveness of optimisations and heuristics because the wide variety of tags in the corpus allows a selection of tags to be examined. The segmentation corpus is also used because it represents a widely-studied problem and a sharp contrast to the bibliography corpus.

6.4.1 Best First

Best first (Section 4.3.2) is an optimisation that exploits the nature of the maximum lookahead search, linking the discrimination of the models to the search space

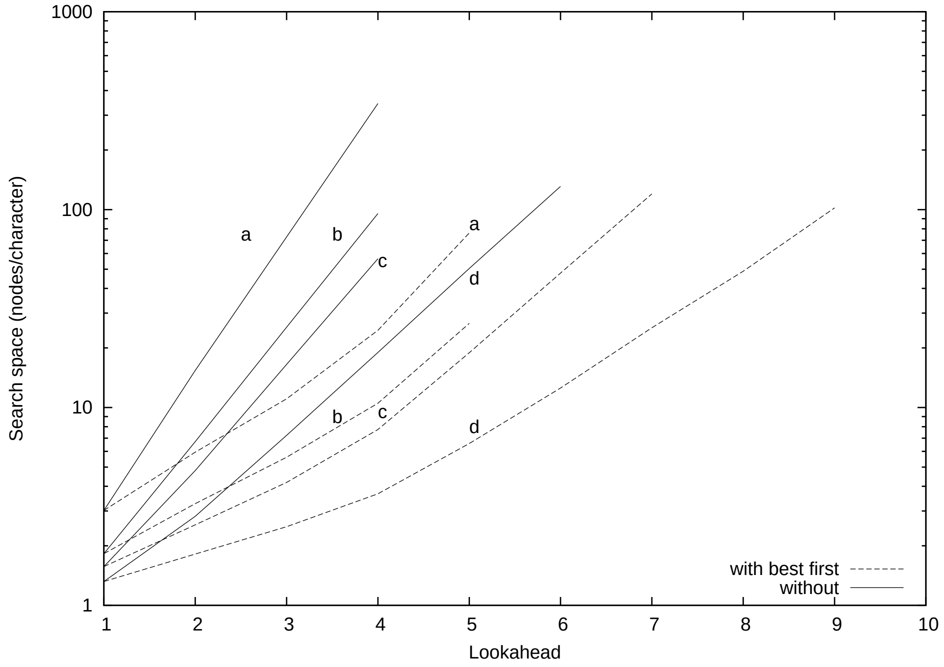


Figure 6.6: Best first optimisation in hierarchical tag insertion. The lines are: a *author*, *editor*, *name*, *first* and *last*; b *name*, *first* and *last*; c *name* and *last*; d *name*. All runs used an order 3 model with 200 training documents and a single testing document.

required to find the lowest entropy tagging of a sequence with respect to that model.

Figure 6.6 shows the effect of the best first optimisation on the hierarchical (nested) tags *author*, *editor*, *name*, *first* and *last* in the bibliography corpus. In all cases where the lookahead is > 1, the search space was significantly reduced. The effect was greatest with the largest number of tags, because as the number of tags increases, the chance that an observed sequence will have low entropy relative to a particular model increases.

Figure 6.7 shows the effect of the best first optimisation on the non-hierarchical tags *name*, *pages*, *date*, *volume* and *number* in the bibliography corpus.

Figure 6.8 shows the effect of the best first optimisation on the *word* tag in the segmentation corpus. Without best first, the order of the model has no impact on the search space. Best first reduces the search space (a versus b), with the effect increasing as the order increases the discrimination of the model (b, c, d, and e).

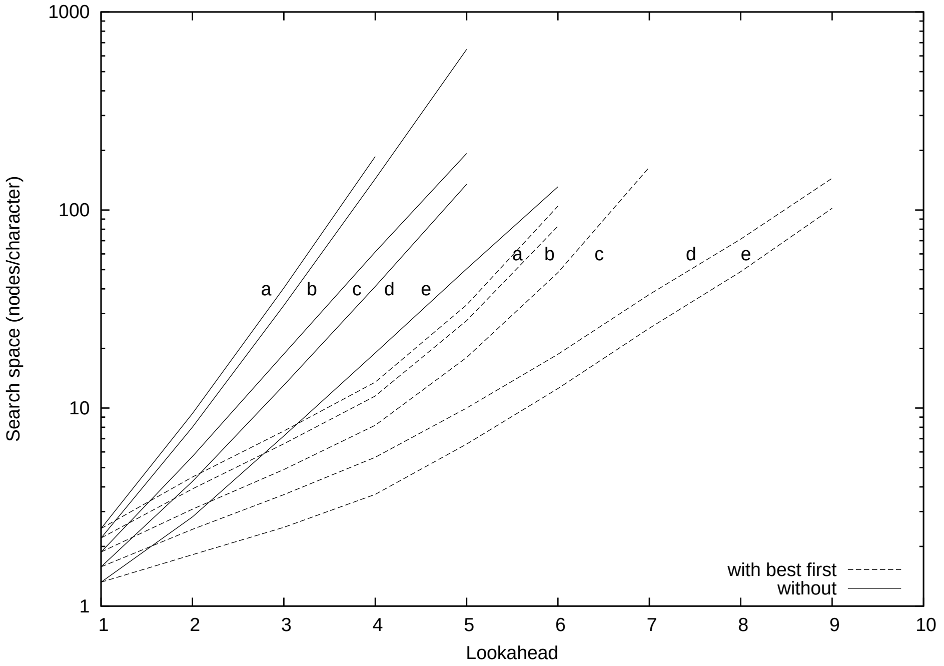


Figure 6.7: Best first optimisation in non-hierarchical tag insertion. The lines are: a *name*, *pages*, *date*, *volume* and *number*; b *name*, *pages*, *date* and *volume*; c *name*, *pages* and *date*; d *name* and *pages*; e *name*. All runs used an order 3 model with 200 training documents and a single testing document.

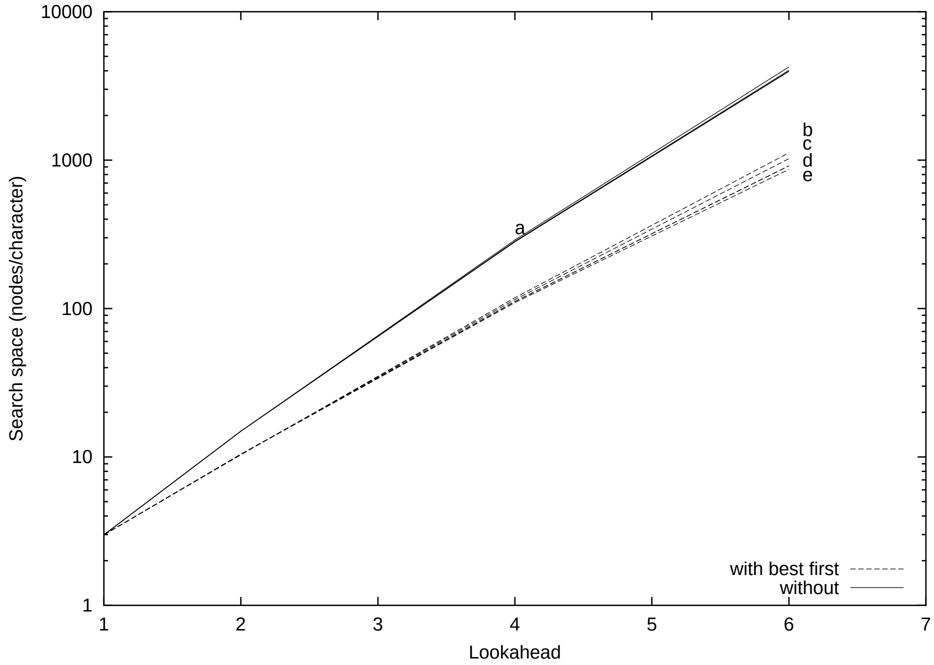


Figure 6.8: The effect of best first on *word* for varying model orders. a labels nearly co-incident quadruple lines representing the search spaces for orders 1, 2, 3 and 4 without best first; b is order 1 with best first; c is order 2 with best first; d is order 3 with best first; e is order 4 with best first. All runs used 900 training documents and a single testing document.

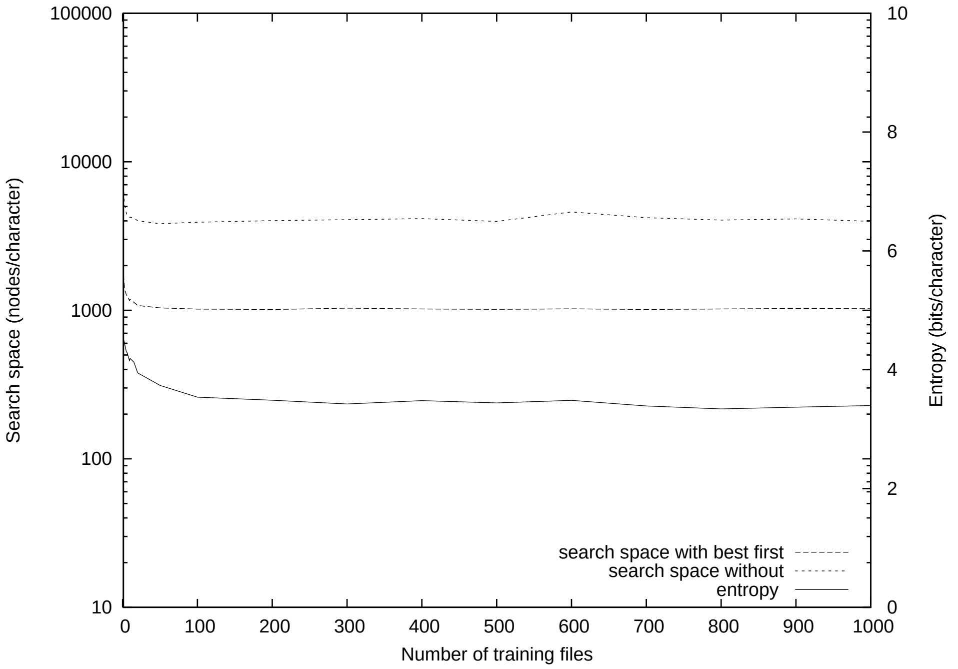


Figure 6.9: Effect of best first when the number of training documents is varied. All runs use order 3 models with a lookahead of 6 and a single testing document from the segmentation corpus. Entropy is the entropy of the entire entire document with respect to the model using for text augmentation, normalised for document length.

The documents in the segmentation corpus are significantly more homogeneous than those in the bibliography corpus, resulting in less noise in their respective graphs.

Figure 6.9 shows how little the effectiveness of the best first increases with the amount of training in the segmentation corpus. Without best first, the search space is independent of the number of documents trained on, but with best first the search space drops. Most of the drop occurred over the first 200 training documents, with relatively little drop over the remaining 799 documents (one document was always withheld for testing).

Figures 6.7, 6.8 and 6.9 each show the results for a single document. This is because while the trends are the same (in all cases best first improves performance and that improvement increases with model order) the size of the improvement varies considerably depending on the problem, and indeed the document, being tackled. In all cases the results are representative of larger-scale experimentation, but averaged results are naturally smoother.

These findings are consistent with the expectations from Section 4.3.2. Well-trained, high-order models allow the probability distribution function to distinguish accurately between likely and unlikely branches, and models with many tags have many more unlikely branches to prune. Given the good performance, the relatively simple implementation and fact that no extra state is required in the model, the best first optimisation is valuable in these tag insertion problems.

6.4.2 Automatic Tokenisation

Automatic tokenisation (see Section 4.3.3) is explored using occurrence tables for illustrative purposes. Table 6.8 shows an occurrence table for the Reuters' corpus after the start and end tags have been converted to special-use characters. In Table 6.8(a) each row contains counts of characters appearing in the corpus belonging to each Unicode character class. Each column contains counts of the character class of the characters immediately following them. In Table 6.8(b) each row contains counts of characters in a Unicode character class that occur immediately prior to a tag (either a start tag or an end tag). Each column contains counts of the class of the character immediately following a tag. An empty cell in Table 6.8(b) indicates that a pair of classes between which a tag has not been seen and which it is reasonable to assume need not be considered for inserting tags. Cells that are empty in Table 6.8(b) but occupied in Table 6.8(a) represent a genuine saving, particularly if the number in the cell in Table 6.8(a) is high, as these are pairs of characters between which the search is not considered inserting tags.

The distinctive cross-shape in Table 6.8(b) is due to the fact that opening tags usually follow a space character and are followed by almost anything, while closing tags can be preceded by almost anything but are followed by a space or '\n' character. This effect is reinforced by the uniform formatting of the corpus. The

First Character		Second Character															Sum
Symbol	#	0	1	2	9	12	15	18	20	21	22	23	24	25	26	27	
UNASSIGNED	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
UPPERCASE LETTER	1	-	85k	223k	1k	-	-	30k	1k	57	2k	2	13k	73	1k	1	360k
LOWERCASE LETTER	2	-	2k	5m	254	-	-	1m	13k	35	4k	-	134k	24	6	-	6m
DECIMAL DIGIT NUMBER	9	-	1k	1k	145k	-	-	66k	5k	36	2k	-	53k	267	52	-	275k
SPACE SEPARATOR	12	-	-	-	-	-	62k	1m	-	-	-	-	-	-	-	-	1m
CONTROL	15	-	-	-	-	-	54	69k	-	-	-	-	-	-	-	-	69k
PRIVATE USE	18	7k	248k	1m	65k	1m	7k	7k	8k	8k	17	-	14k	2k	6k	4	2m
DASH PUNCTUATION	20	-	3k	13k	5k	-	-	6k	4k	5	9	-	27	-	322	-	33k
START PUNCTUATION	21	-	4k	1k	1k	-	-	206	131	-	-	-	92	99	713	-	8k
END PUNCTUATION	22	-	27	6	20	-	-	7k	8	4	1	-	1k	-	2	-	8k
CONNECTOR PUNCTUATION	23	-	-	-	-	-	-	3	-	-	-	119	-	-	-	-	122
OTHER PUNCTUATION	24	-	15k	18k	44k	-	-	139k	141	78	303	1	20k	8	18	-	238k
MATH SYMBOL	25	-	50	19	2k	-	-	475	8	4	1	-	39	16	44	-	2k
CURRENCY SYMBOL	26	-	13	36	9k	-	-	344	-	10	6	-	32	18	-	-	9k
MODIFIER SYMBOL	27	-	-	5	-	-	-	-	-	-	-	-	-	-	-	-	5
Sum		7k	360k	6m	275k	1m	69k	2m	33k	8k	8k	122	238k	2k	9k	5	11m

(a)

First Character		Second Character															Sum
Symbol	#	0	1	2	9	12	15	18	20	21	22	23	24	25	26	27	
UNASSIGNED	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
UPPERCASE LETTER	1	-	-	-	-	30k	-	-	-	-	-	-	-	-	-	-	30k
LOWERCASE LETTER	2	-	-	-	-	1m	-	-	-	-	-	-	-	-	-	-	1m
DECIMAL DIGIT NUMBER	9	-	-	-	-	66k	-	-	-	-	-	-	-	-	-	-	66k
SPACE SEPARATOR	12	-	201k	1m	62k	-	-	-	5k	8k	17	-	6k	2k	6k	4	1m
CONTROL	15	7k	46k	302	3k	3	-	-	3k	406	-	-	7k	89	63	-	69k
PRIVATE USE	18	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DASH PUNCTUATION	20	-	-	-	-	6k	-	-	-	-	-	-	-	-	-	-	6k
START PUNCTUATION	21	-	-	-	-	206	-	-	-	-	-	-	-	-	-	-	206
END PUNCTUATION	22	-	-	-	-	7k	-	-	-	-	-	-	-	-	-	-	7k
CONNECTOR PUNCTUATION	23	-	-	-	-	3	-	-	-	-	-	-	-	-	-	-	3
OTHER PUNCTUATION	24	-	-	-	-	139k	-	-	-	-	-	-	-	-	-	-	139k
MATH SYMBOL	25	-	-	-	-	475	-	-	-	-	-	-	-	-	-	-	475
CURRENCY SYMBOL	26	-	-	-	-	344	-	-	-	-	-	-	-	-	-	-	344
MODIFIER SYMBOL	27	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Sum		7k	248k	1m	65k	1m	-	-	8k	8k	17	-	14k	2k	6k	4	2m

(b)

Table 6.8: Occurrence tables for the Reuters' corpus. (a) Table of all pairs of characters. (b) Table of pairs of characters either side of a tag. 'k' and 'm' indicate units of a thousand and a million respectively.

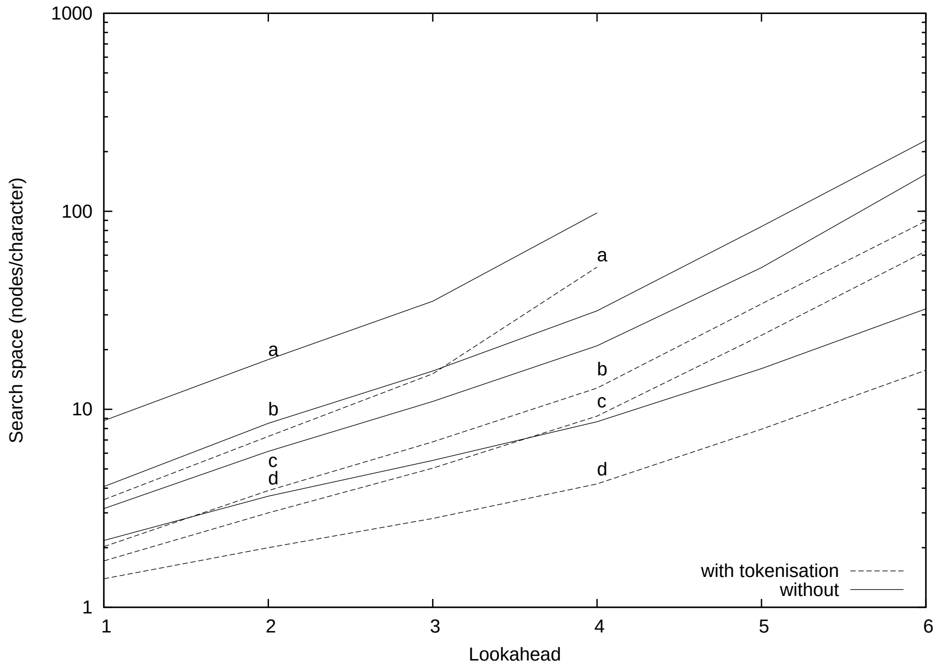


Figure 6.10: Effect of tokenisation on a group of hierarchical tags. The lines are: a *name*, *last*, *first*, *editor* and *author*; b *name*, *last* and *first*; c *name* and *last*; d *name*. Each run was performed with 2000 training documents, one testing document and order 3 models.

CONTROL[3] character class includes '\n', '\r' and EOF.

Figures 6.10 and 6.11 show the effect of tokenisation of hierarchical and non-hierarchical tags in the bibliography corpus. The reason for the differences between hierarchical and non-hierarchical tags is shown in Table 6.9. Table 6.9(a) shows all pairs of characters; Table 6.9(b) shows those either side of the *name* tag, the sparseness of the latter indicating that a procedure such as tokenisation has the potential to make an improvement. The hierarchical tags shown in Table 6.9(c) are similar to the non-hierarchical tags shown in Table 6.9(b), not because they are hierarchical but because they are sequences of case-sensitive characters delimited with spaces, commas and full-stops. The non-hierarchical tags shown in Table 6.9(d) by comparison have a significantly more diverse context. The *date* tag is a sequence of digits and case-sensitive characters and *volume* and *number* tags are strings of dig-

[3] The standard method of writing the names of Unicode characters and character classes is in capitals.

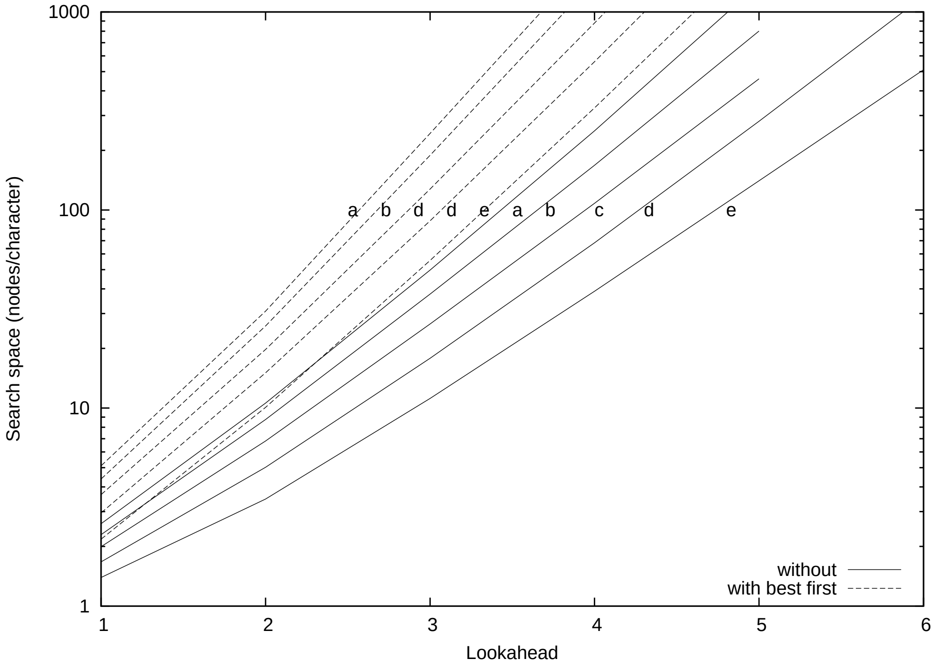


Figure 6.11: Effect of tokenisation on a group of non-hierarchical tags. The lines are: a *name*, *pages*, *date*, *volume* and *number*; b *name*, *pages*, *date* and *volume*; c *name*, *pages* and *date*; d *name* and *pages*; e *name*. Each run was performed with 2000 training documents, one testing document and order 3 models.

its commonly delimited by brackets and semicolons. The resultant occurrence table is much less sparse than the previous table.

Tokenisation potentially interacts with other errors. For example, in Table 6.10 some errors on the bibliography corpus result from problems finding the boundary between the author list and the title tag. In this example, *Athena*, the first word of the article title, has been split in two. The string *Athen* has a slightly lower entropy in the last tag than in the title tag, but *a:* has never been seen in the last tag. The *a:* has not been seen when the decision is taken whether or not to start the tag name tag, so the word is split in two.

Whether the first or the second error is preferable will probably depend on the application. As lookahead gets longer, such errors are greatly reduced, but the proper nouns commonly found at the start of titles are often long words (particularly corporate, place and personal names transliterated into English) and remain problematic even at long lookaheads.

Of 100 differences in correctness examined in the bibliography corpus, using the experimental scenario from Figure 6.10 but using 500 testing documents, 98 were errors of the type shown by Table 6.10. Both the tokenisation and non-tokenisation results were incorrect but the non-tokenisation results recovered more quickly. The remaining were situations in which every tag occurred between rare pairs of character classes.

The appearance of tags between novel or rare pairs of character classes could be guarded against by also inserting tags between character classes seen fewer times than a separate threshold (of the order of 25). In all cases examined this would have solved the problem. If the training corpora is representative, this should have little effect on the search space.

Table 6.11(a) and (b) show the occurrence tables for the Computists' corpus and all the tags within it. Table 6.11(b) is significantly less sparse than Table 6.8(a). However, the frequently-occurring alpha-numeric pairs in the upper left corner are

First Character		Second Character															Sum
Symbol	#	0	1	2	9	12	15	18	20	21	22	23	24	25	26	27	
UNASSIGNED	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
UPPERCASE LETTER	1	-	100k	375k	5k	26k	2k	127	2k	763	1k	-	132k	362	2	19	649k
LOWERCASE LETTER	2	-	2k	3m	4k	479k	60k	10k	16k	1k	5k	1	233k	1k	-	57	4m
DECIMAL DIGIT NUMBER	9	243	288	3k	384k	13k	4k	2	1k	56k	81k	-	86k	88	-	1	632k
SPACE SEPARATOR	12	-	382k	367k	98k	37k	-	84k	225	20k	756	9	6k	825	1	103	1m
CONTROL	15	1k	34k	43k	17k	2	13k	1k	9	50k	3	-	359	47	1	14	162k
PRIVATE USE	18	-	87k	469	6	83k	4k	3	-	1	-	1	70	17	-	1	176k
DASH PUNCTUATION	20	-	7k	11k	2k	277	556	-	-	2	14	-	49	-	1	-	22k
START PUNCTUATION	21	-	26k	2k	96k	128	2k	62	5	9	4	-	1k	15	-	1	130k
END PUNCTUATION	22	-	9	75	17	49k	143	3	112	26	142	-	40k	24	1	-	90k
CONNECTOR PUNCTUATION	23	-	6	5	-	-	-	-	-	-	-	-	-	-	-	-	11
OTHER PUNCTUATION	24	-	6k	8k	23k	309k	76k	75k	1k	202	1k	-	27k	32	-	1	529k
MATH SYMBOL	25	-	622	589	282	747	152	-	36	6	24	-	81	579	-	2	3k
CURRENCY SYMBOL	26	-	-	1	4	-	-	-	-	-	-	-	1	-	-	-	6
MODIFIER SYMBOL	27	-	66	104	13	5	-	-	2	-	1	-	8	-	-	-	199
Sum		2k	649k	4m	632k	1m	164k	172k	22k	130k	90k	11	529k	3k	6	199	7m

(a)

First Character		Second Character															Sum
Symbol	#	0	1	2	9	12	15	18	20	21	22	23	24	25	26	27	
UNASSIGNED	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
UPPERCASE LETTER	1	-	-	-	-	124	-	-	-	-	-	-	3	-	-	-	127
LOWERCASE LETTER	2	-	-	2	-	10k	44	-	-	-	-	-	5	-	-	-	10k
DECIMAL DIGIT NUMBER	9	-	-	-	-	2	-	-	-	-	-	-	-	-	-	-	2
SPACE SEPARATOR	12	-	84k	454	2	-	-	-	-	1	-	1	59	17	-	1	84k
CONTROL	15	-	1k	7	2	-	-	-	-	-	-	-	2	-	-	-	1k
PRIVATE USE	18	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DASH PUNCTUATION	20	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
START PUNCTUATION	21	-	62	-	-	-	-	-	-	-	-	-	-	-	-	-	62
END PUNCTUATION	22	-	-	-	-	3	-	-	-	-	-	-	-	-	-	-	3
CONNECTOR PUNCTUATION	23	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
OTHER PUNCTUATION	24	-	1	-	-	73k	2k	-	-	-	-	-	-	-	-	-	75k
MATH SYMBOL	25	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
CURRENCY SYMBOL	26	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
MODIFIER SYMBOL	27	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Sum		-	85k	463	4	83k	2k	-	-	1	-	1	69	17	-	1	172k

(b)

First Character		Second Character															Sum
Symbol	#	0	1	2	9	12	15	18	20	21	22	23	24	25	26	27	
UNASSIGNED	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
UPPERCASE_LETTER	1	-	-	-	-	192	2	-	-	-	-	-	4	-	-	-	198
LOWERCASE_LETTER	2	-	2	2	-	16k	208	-	-	-	-	-	168	-	-	-	17k
DECIMAL_DIGIT_NUMBER	9	-	-	-	-	3	-	-	-	-	-	-	-	-	-	-	3
SPACE_SEPARATOR	12	-	168k	659	2	-	-	-	-	5	-	4	93	37	-	-	169k
CONTROL	15	-	2k	6	2	-	-	-	-	1	-	-	1	-	-	-	2k
PRIVATE_USE	18	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DASH_PUNCTUATION	20	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
START_PUNCTUATION	21	-	56	-	-	-	-	-	-	-	-	-	-	-	-	-	56
END_PUNCTUATION	22	-	-	-	-	32	1	-	-	-	-	-	-	-	-	-	33
CONNECTOR_PUNCTUATION	23	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
OTHER_PUNCTUATION	24	-	1	-	-	152k	3k	-	-	-	56	-	-	-	-	-	156k
MATH_SYMBOL	25	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	1
CURRENCY_SYMBOL	26	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
MODIFIER_SYMBOL	27	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Sum		-	171k	667	4	169k	3k	-	-	6	56	4	266	37	-	-	345k

(c)

First Character		Second Character															Sum
Symbol	#	0	1	2	9	12	15	18	20	21	22	23	24	25	26	27	
UNASSIGNED	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
UPPERCASE_LETTER	1	-	-	-	-	143	1	-	-	4	5	-	35	-	-	-	188
LOWERCASE_LETTER	2	-	2	3	-	10k	55	-	-	-	619	-	290	-	-	-	11k
DECIMAL_DIGIT_NUMBER	9	-	-	-	-	8k	784	-	-	11k	33k	-	44k	-	-	-	99k
SPACE_SEPARATOR	12	-	97k	518	57k	-	-	-	-	3	-	-	708	20	-	2	156k
CONTROL	15	-	2k	23	6k	-	-	-	-	-	-	-	55	-	-	-	9k
PRIVATE_USE	18	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DASH_PUNCTUATION	20	-	-	-	-	-	-	-	-	-	1	-	2	-	-	-	3
START_PUNCTUATION	21	-	4k	6	29k	-	-	-	1	1	-	-	247	-	-	-	33k
END_PUNCTUATION	22	-	-	-	-	42	5	-	-	-	72	-	37	-	-	-	156
CONNECTOR_PUNCTUATION	23	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
OTHER_PUNCTUATION	24	-	58	11	20k	101k	5k	-	-	8	31	-	38	-	-	-	127k
MATH_SYMBOL	25	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
CURRENCY_SYMBOL	26	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
MODIFIER_SYMBOL	27	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

(d)

Table 6.9: Occurrence tables for the bibliography corpus. (a) Table of all pairs of characters. (b) Table of pairs of characters either side of a *name* tag. (c) Table of pairs of characters either side of *name*, *last*, *first*, *editor* and *author* tags. (d) Table of pairs of characters either side of *name*, *pages*, *date*, *volume* and *number* tags.

Case	Text
0	[Son] D. Song. Athena: A new efficient automatic checker for security protocol analysis.
1	[Son] <name> <first> D.</first> <last> Song.</last> </name>-<name> <last> Athena:</last> </name> <title> A new efficient automatic checker for security protocol analysis.</title>
2	[Son] <name> <first> D.</first> <last> Song.</last> </name>-<name> <last> Athen</last> </name> <title> a: A new efficient automatic checker for security protocol analysis.</title>

Table 6.10: Interaction between errors. The unmarked-up text (0), the text with a markup error (1) and with the first error confounded by a second error which splits a word in two (2).

First Character		Second Character														Sum
Symbol	#	0	1	2	9	12	15	18	20	21	22	23	24	25	26	
UNASSIGNED	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
UPPERCASE LETTER	1	-	5k	10k	55	1k	39	1k	174	-	110	4	971	27	-	20k
LOWERCASE LETTER	2	-	451	198k	974	33k	1k	2k	896	-	116	40	10k	9	-	249k
DECIMAL DIGIT NUMBER	9	-	1k	52	4k	339	50	1k	234	-	88	1	746	359	-	8k
SPACE SEPARATOR	12	-	8k	29k	1k	8k	2k	2k	163	1k	-	1	418	609	-	54k
CONTROL	15	-	897	2k	42	1k	2k	679	73	441	-	71	178	317	-	8k
PRIVATE USE	18	36	2k	1k	904	1k	179	36	31	51	356	-	2k	1k	275	10k
DASH PUNCTUATION	20	-	215	864	201	225	23	44	233	-	5	-	3	-	-	1k
START PUNCTUATION	21	-	616	171	67	1	-	611	-	-	-	-	31	129	-	1k
END PUNCTUATION	22	-	-	-	-	352	828	14	-	-	10	-	435	-	-	1k
CONNECTOR PUNCTUATION	23	-	10	32	2	-	72	-	-	-	-	4k	1	-	-	4k
OTHER PUNCTUATION	24	-	607	4k	350	7k	1k	303	8	-	948	-	1k	35	-	18k
MATH SYMBOL	25	-	3	41	18	427	30	1k	1	-	6	-	966	190	-	2k
CURRENCY SYMBOL	26	-	-	-	275	-	-	-	-	-	-	-	-	-	-	275
Sum		36	20k	249k	8k	54k	8k	10k	1k	1k	1k	4k	18k	2k	275	383k

(a)

First Character		Second Character														Sum
Symbol	#	0	1	2	9	12	15	18	20	21	22	23	24	25	26	
UNASSIGNED	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
UPPERCASE LETTER	1	-	-	-	-	308	48	-	16	-	179	-	603	6	-	1k
LOWERCASE LETTER	2	-	-	35	-	750	48	-	7	-	164	-	880	908	-	2k
DECIMAL DIGIT NUMBER	9	-	-	-	-	135	42	-	8	-	13	-	924	18	-	1k
SPACE SEPARATOR	12	-	1k	397	804	-	-	-	-	38	-	-	4	8	230	2k
CONTROL	15	36	478	61	62	-	-	-	-	13	-	-	5	3	21	679
PRIVATE USE	18	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DASH PUNCTUATION	20	-	3	4	19	-	-	-	-	-	-	-	-	-	18	44
START PUNCTUATION	21	-	543	45	17	-	-	-	-	-	-	-	-	-	6	611
END PUNCTUATION	22	-	-	-	-	9	-	-	-	-	-	-	5	-	-	14
CONNECTOR PUNCTUATION	23	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
OTHER PUNCTUATION	24	-	57	26	-	75	5	-	-	-	-	-	21	119	-	303
MATH SYMBOL	25	-	24	1k	2	-	-	-	-	-	-	-	2	-	-	1k
CURRENCY SYMBOL	26	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
Sum		36	2k	1k	904	1k	143	-	31	51	356	-	2k	1k	275	10k

(b)

Table 6.11: Occurrence tables for the Computists' corpus. (a) Table of all pairs of characters. (b) Table of pairs of characters either side of a tag.

First Character		Second Character														Sum
Symbol	#	0	1	2	5	9	11	18	20	21	22	23	24	25	26	
UNASSIGNED	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
UPPERCASE LETTER	1	-	3k	2k	470	61	-	1k	1	1	-	18	1	-	1	8k
LOWERCASE LETTER	2	-	146	17k	696	4	-	4k	1	1	1	-	-	-	7	22k
OTHER LETTER	5	-	45	259	1m	524	-	1m	-	56	389	225	-	-	572	3m
DECIMAL DIGIT NUMBER	9	-	18	346	1k	4k	-	1k	2	-	383	262	-	-	62	7k
OTHER NUMBER	11	-	-	-	-	-	-	3	-	-	-	-	-	-	-	3
PRIVATE USE	18	999	3k	1k	1m	2k	1	2m	-	18k	20k	314k	28	9	1k	4m
DASH PUNCTUATION	20	-	-	2	-	2	-	-	-	-	-	-	-	-	-	4
START PUNCTUATION	21	-	-	-	393	383	-	26k	-	1	23	20	-	-	-	27k
END PUNCTUATION	22	-	2	-	49	1	-	21k	-	3	-	4	-	-	1	21k
OTHER PUNCTUATION	24	-	601	487	276k	54	2	28k	-	8k	19	1	5	-	348	315k
MATH SYMBOL	25	-	8	13	-	-	-	13	-	-	-	-	-	-	-	34
CURRENCY SYMBOL	26	-	-	-	-	-	-	9	-	-	-	-	-	-	-	9
OTHER SYMBOL	28	-	2	72	604	107	-	1k	-	13	-	-	-	-	9	2k
Sum		999	8k	22k	3m	7k	3	4m	4	27k	21k	315k	34	9	2k	7m

(a)

First Character		Second Character														Sum
Symbol	#	0	1	2	5	9	11	18	20	21	22	23	24	25	26	
UNASSIGNED	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
UPPERCASE LETTER	1	-	126	15	875	34	-	-	-	46	259	482	4	-	45	1k
LOWERCASE LETTER	2	-	721	616	1k	30	-	-	-	33	1k	509	-	-	98	4k
OTHER LETTER	5	-	1k	347	1m	2k	1	-	-	17k	18k	304k	-	8	1k	1m
DECIMAL DIGIT NUMBER	9	-	29	58	878	22	-	-	-	12	110	246	24	-	26	1k
OTHER NUMBER	11	-	-	-	3	-	-	-	-	-	-	-	-	-	-	3
PRIVATE USE	18	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
DASH PUNCTUATION	20	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
START PUNCTUATION	21	-	1k	343	24k	130	-	-	-	92	6	55	-	-	21	26k
END PUNCTUATION	22	-	15	2	12k	44	-	-	-	405	150	8k	-	-	22	21k
OTHER PUNCTUATION	24	998	255	13	26k	67	-	-	-	821	30	221	-	1	21	28k
MATH SYMBOL	25	-	-	-	-	-	-	-	-	-	5	8	-	-	-	13
CURRENCY SYMBOL	26	-	-	-	-	9	-	-	-	-	-	-	-	-	-	9
OTHER SYMBOL	28	1	70	83	1k	25	-	-	-	51	16	318	-	-	87	1k
Sum		999	3k	1k	1m	2k	1	-	-	18k	20k	314k	28	9	1k	2m

(b)

Table 6.12: Occurrence tables for the segmentation corpus. (a) Table of all pairs of characters. (b) Table of pairs of characters either side of a tag.

mainly zero, so the heuristic is of some benefit.

Table 6.12 is the occurrence table for the segmentation corpus and indicates that the OTHER LETTER is by far the most common character class, which is to be expected since most Chinese characters fall into this class. The nature of the corpus means that all of the frequently-occurring pairs in Table 6.12(a) also appear in Table 6.12(b) (as non-zeros), indicating that automatic tokenisation is going to have little effect on the search space in this corpus.

Figure 6.12 shows the interaction between best first and tokenisation for the *name* tag. The addition of tokenisation to best first always reduces the search space, but the effect is most noticeable at low lookaheads when best first is less effective. This is because automatic tokenisation prunes branches of the search tree without having to expand the first node in the branch to calculate the entropy.

Consistent with the expectations from Section 4.3.3, these results show that automatic tokenisation improves performance on some datasets. However, it does not

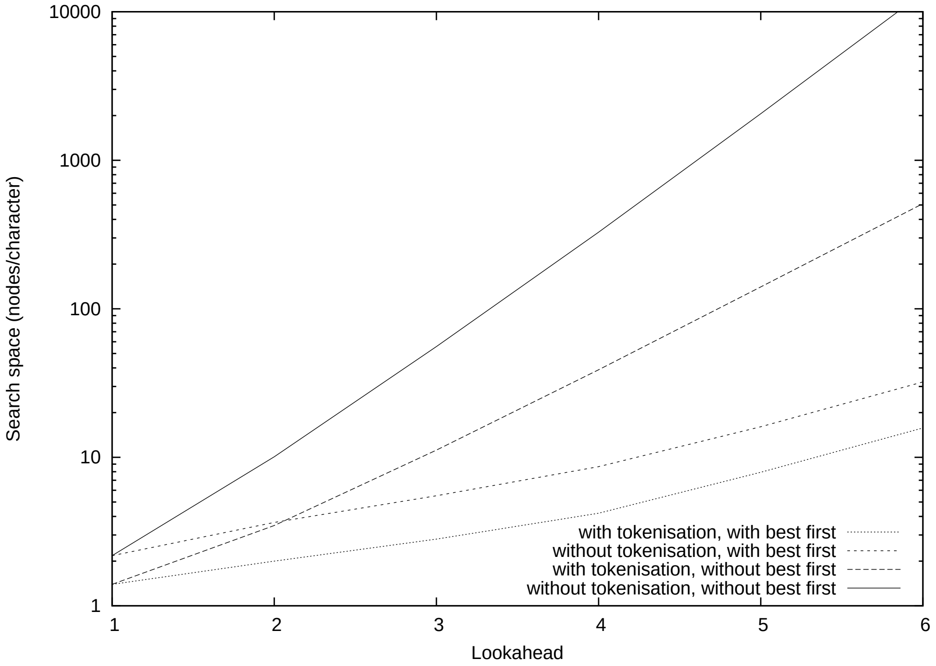


Figure 6.12: Effect of best first and automatic tokenisation on *name* tag. Each run was performed with 2000 training documents, one testing document and order 3 models.

perform consistently well across all datasets, and a number of the corpora have noise in the occurrence tables. Such noise is likely to be significantly greater in digital library collections of heterogeneous documents of diverse origin than in the curated corpora used here. Anecdotal evidence of HTML and XHTML documents from the Internet suggest that tags do occur in a significantly wider variety of places than in the corpora examined here. Automatic tokenisation requires a small and tightly-bounded amount of extra state per model in the form of an occurrence table.

Unlike best first, automatic tokenisation is not linked to the discrimination of the models. This means it can perform well even for a poorly trained model. The reason that automatic tokenisation does not perform as well as the occurrence table method is that the PPM model already discriminates between these situations and that best first ensures that the branches that get pruned by automatic tokenisation are not explored anyway.

Name	Symbol	Example
Null folder	N	*Jones,␣Jill␣K.␣and*
Capitals folder	c	*JONES,␣JILL␣K␣AND*
Case folder	C	*Aaaaa,␣Aaaa␣A.␣aaa*
Unicode folder	u	*AaaaaPSAaaaSAPSaaa*
Vowel folder	V	*nvnvn,␣nvnn␣n.␣vnn*
Vowel & case folder	VC	*Nvnvn,␣Nvnn␣N.␣vnn*

Table 6.13: Folders used in alphabet reduction.

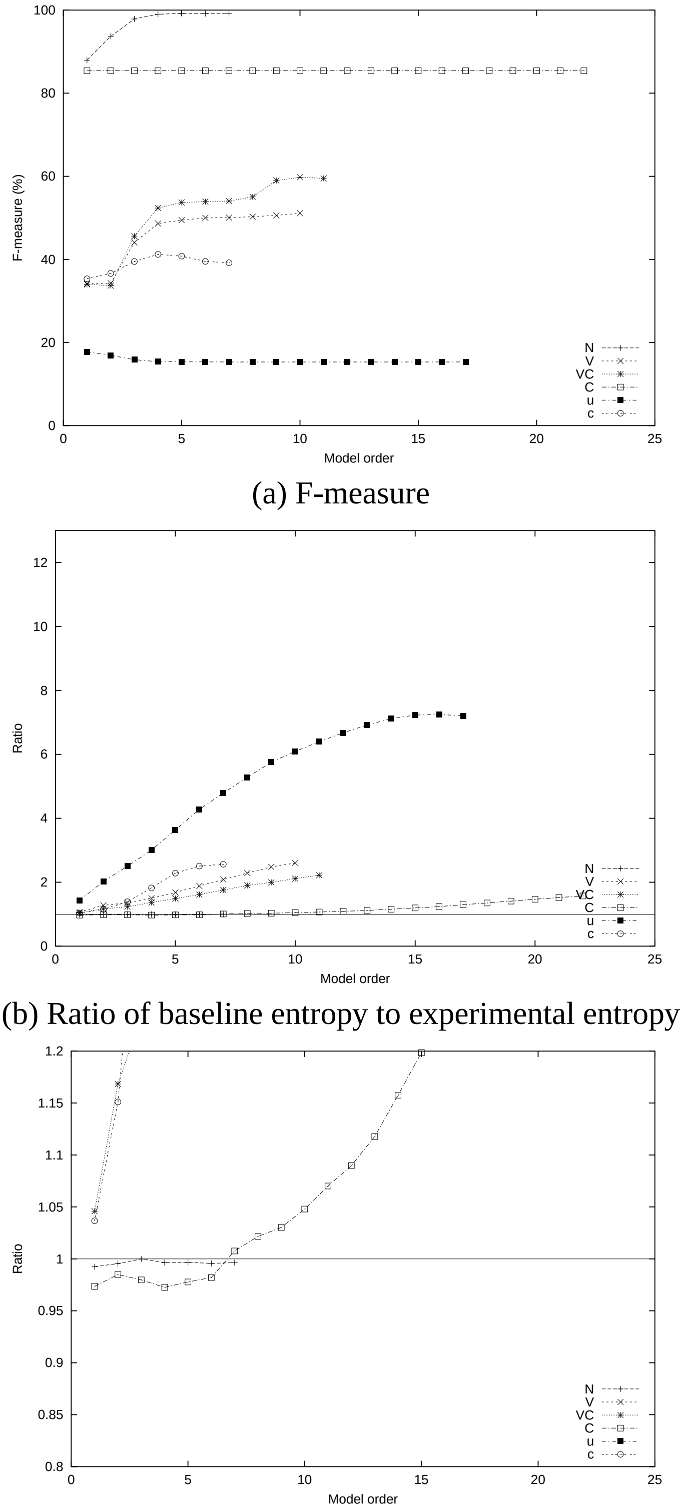
6.4.3 Alphabet Reduction

Table 6.13 shows the six ‘folders’ used in the alphabet reduction experiment. They ‘fold’ the alphabet used in the model, as their effects on a sample string show.

The Null folder does not change the alphabet at all. The Capitals folder removes the distinction between upper and lower case. The Case folder folds all uppercase letters to a single letter and all lowercase letters to a single letter. The Unicode folder folds each of the Unicode character classes (see Section 4.3.3) to a single character per class. The Vowel folder folds all vowels to a single letter and all non-vowels to a single letter. The Vowel and Case folder folds uppercase vowels to a single letter, lowercase vowels to a single letter, uppercase non-vowels to a single letter and lowercase non-vowels to a single letter.

Figure 6.13 shows the results of these six folders on *name* in the bibliography corpus. Figure 6.13(a) shows the F-measure against the order of the model for each of the folders. The experiment was performed in 750 megabytes of heap memory, and the data is shown only for those models and lookaheads which could be built and used in that memory.

The N folder performed best, but N models could only be built to order seven, because of the large alphabet. The C models also performed well and could be built to order 23. However, increasing order did not increase the performance because useful information was thrown away by the folder. The c, V and VC models all performed similarly poorly and could be built to orders between seven and ten. The



(a) F-measure

(b) Ratio of baseline entropy to experimental entropy

(c) Ratio of baseline entropy to experimental entropy (detail)

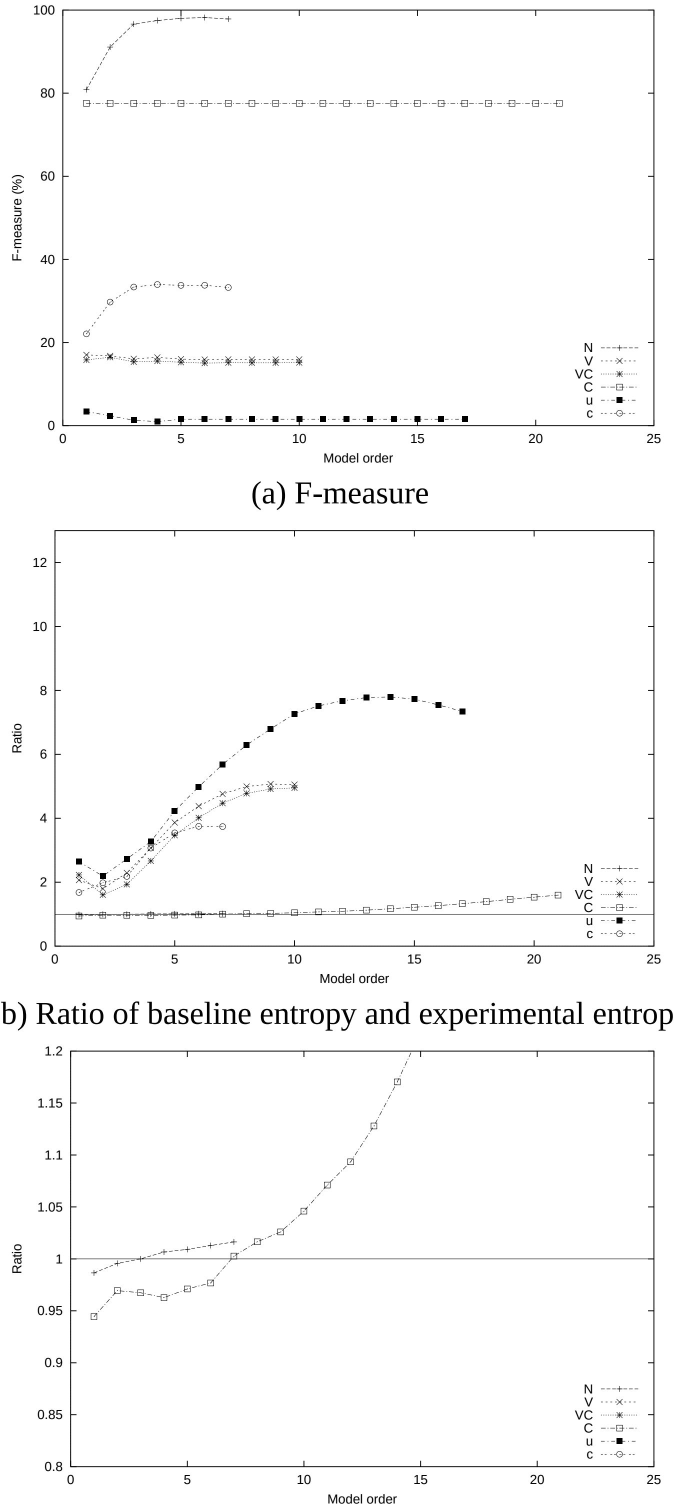
Figure 6.13: The effects of alphabet reduction on finding the *name* tag of bibliography corpus. Lookahead of 8, trained on 2000 documents and tested on 20 documents.

u models performed badly with an F-measure less than twenty despite being able to be built to order 18. This is particularly surprising given that the u folder has a close relationship with the C folder, which performed well. The reason for this difference appears to be that '.' and ',' are important in delimiting names and other features in bibliographies and the u models were unable to distinguish between these characters.

Figure 6.13(b) shows the ratio of baseline to experimental entropy for the same experiments while Figure 6.13(c) shows detail of the same relationship where the ratio approaches one. As discussed previously (see page 83), the entropy can be used to determine whether the model or the search is responsible for a mis-tagging. All data points with a ratio less than one indicate that the search was deficient (i.e. the lookahead could be increased for greater correctness). All data points where the ratio is greater than one indicate that the model is deficient in some regard; in the ideal situation the ratio is 1:1. There are three likely ways in which the model can be deficient: it may have seen insufficient training data, it may be of insufficient order, or it may be failing to capture important features of the data. 2000 training bibliographies (approximately 45,000 bibliographic entries) would appear to be sufficient training data: models with smaller alphabets generally require less training data. Increasing the order of the u, V, c and VC models clearly moves the ratio further from 1:1. Thus the problem is likely to be that these models are not capturing important features of the data.

The upward trend in the entropy ratio for the C models of order higher than 6 (Figure 6.13(c)) is consistent with the behaviour of PPM models when the order is increased beyond optimal. This species of over-fitting is caused by the building of a higher order model than there is training data available to train effectively, leading to many common states having their probabilities generated via the escape method.

The increase in noise for the ratio of entropies (particularly for the u model) as order increases is due to sampling effects.



(a) F-measure

(b) Ratio of baseline entropy and experimental entropy

(c) Ratio of baseline entropy and experimental entropy (detail)

Figure 6.14: The effects of alphabet reduction on finding multiple tags in the bibliography corpus. Lookahead of 4, trained on 2000 documents and tested on 20 documents.

Figure 6.14 shows the same details as Figure 6.13 for tags *name*, *pages*, *date*, *volume* and *number* at a lower lookahead (necessary because of the greatly increased search space caused by the additional tags). Performance in Figure 6.14 was consistently poorer than that in Figure 6.13, but the relative performance of the folders was similar. The one deviation from this is the c folder, whose F-measure is similar to the VC and V folders in Figure 6.13, but clearly superior in Figure 6.14. This is because the *pages*, *date*, *volume* and *number* tags in Figure 6.14 are number-centric rather than text-centric, so the loss of capitalisation does not effect them as badly.

The large reductions in correctness shown in Figures 6.13 and 6.14 strongly suggest that, with the possible exception of C, alphabet reduction is unlikely to be useful in production systems for such corpora.

6.4.4 Maximum Lookahead Heuristic

For the majority of tag-insertion problems, maximum lookahead is problematic because the lookahead at which the accuracy becomes asymptotic is computationally infeasible. For problems with a small number of tags, maximum lookahead is obtainable. Table 6.14 shows the effect of various lookahead values on a single bibliographic entry. The result converges on the expected text within a lookahead of 5, much shorter than the maximum tag length of ~ 60 which Viterbi search suggests would be required.

The defects displayed in Table 6.14 are mainly of types already discussed in Section 6.2.3: confusion caused by the wide variety of name formats and confusion between article titles and book titles. Similar defects were also seen in Table 6.6, in which the same reference was used to examine the performance with varying model orders. However, as shown in Figure 6.15, there is often a great deal of noise, and it may not be clear whether the asymptote has been reached or whether the lookahead must be increased.

The primary sources of errors when inserting the *pages* tag were four-digit page

lookahead	text
1	[5] <name> <first> T.</first> <last> Matsui,</last> </name> ◇ ◇*<title>* T.◇ ◇ *</title>* *<journal>*◇ ◇ Matsuoka,*</journal>* and <name> <first> S.</first> <last> Furui,</last> </name> <title> /N-best-based speaker adaptation for speech recognition,"</title> in <booktitle> Proc. ICASSP ’97,</booktitle> (◇*<name>* *<first>* Munich,*</first>* *</name>* *<title>* Germany ◇),*</title>* pp. <pages> 1015–1018,</pages> <date> Apr. 1997</date>.
2	[5] <name> <first> T.</first><last> Matsui,</last> ◇ ◇ <first> T.</first> <last> Matsuoka,</last> </name> and <name> <first> S.</first> <last> Furui,</last> </name> <title> /N-best-based speaker adaptation for speech recognition,"</title> in <booktitle> Proc. ICASSP ’97,</booktitle> (<address> Munich, Germany</address>), pp. <pages> 1015–1018,</pages> <date> Apr. 1997</date>.
3	[5] <name> <first> T.</first><last> Matsui,</last>◇ ◇ <first> T.</first> <last> Matsuoka,</last> </name> and <name> <first> S.</first> <last> Furui,</last> </name> <title> /N-best-based speaker adaptation for speech recognition,"</title> in <booktitle> Proc. ICASSP ’97,</booktitle> (<address> Munich, Germany</address>), pp. <pages> 1015–1018,</pages> <date> Apr. 1997</date>.
4	[5] <name> <first> T.</first> <last> Matsui,</last>◇ ◇<first> T.</first> <last> Matsuoka,</last> </name> and <name> <first> S.</first> <last> Furui,</last> </name> <title> /N-best-based speaker adaptation for speech recognition,"</title> in <booktitle> Proc. ICASSP ’97,</booktitle> (<address> Munich, Germany</address>), pp. <pages> 1015–1018,</pages> <date> Apr. 1997</date>.
5	[5] <name> <first> T.</first> <last> Matsui,</last> </name> <name> <first> T.</first> <last> Matsuoka,</last> </name> and <name> <first> S.</first> <last> Furui,</last> </name> <title> /N-best-based speaker adaptation for speech recognition,"</title> in <booktitle> Proc. ICASSP ’97,</booktitle> (<address> Munich, Germany</address>), pp. <pages> 1015–1018,</pages> <date> Apr. 1997</date>.
baseline	[5] <name> <first> T.</first> <last> Matsui,</last> </name> <name> <first> T.</first> <last> Matsuoka,</last> </name> and <name> <first> S.</first> <last> Furui,</last> </name> <title> /N-best-based speaker adaptation for speech recognition,"</title> in <booktitle> Proc. ICASSP ’97,</booktitle> (<address> Munich, Germany</address>), pp. <pages> 1015–1018,</pages> <date> Apr. 1997</date>.

Table 6.14: Example of effect of lookahead on defects, using order 4 models trained on 4000 documents. Tags in *italics* are incorrectly placed. ◇ indicates a missing tag.

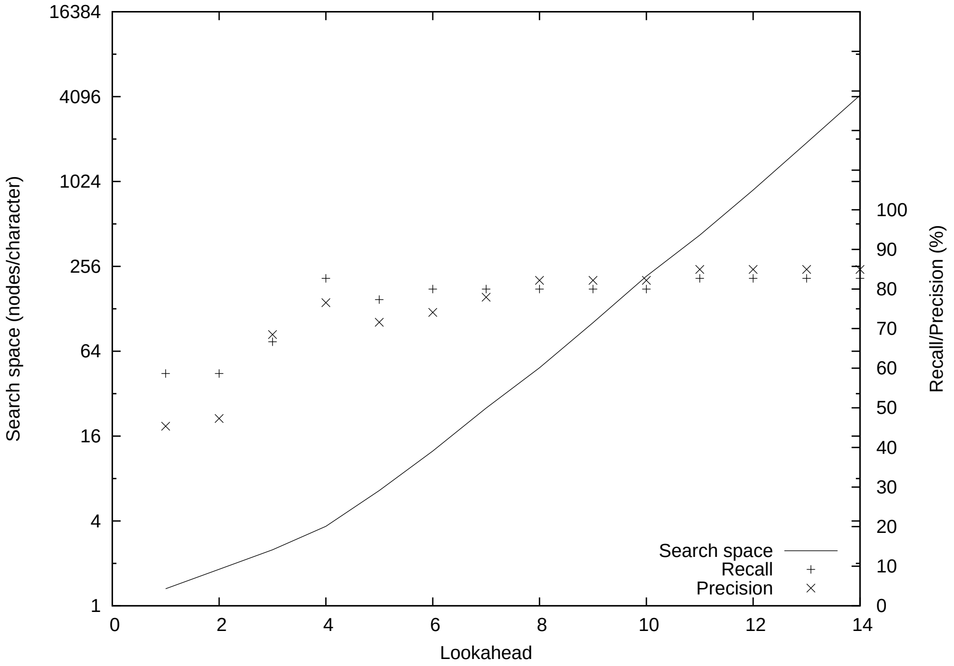


Figure 6.15: Graph of recall, precision and search space against lookahead for the single *name* tag. Models trained on 2000 documents and tested on one document.

numbers that looked like years such as *1993–2002* and features such as n–n+4, which is a common format when the citation is taken from an electronic copy and the document length is known but not the location within the larger journal or collection. These sources of noise are compounded by variability in the length of bibliographies, which may be as short as a single entry with only one *pages* tag and only one *name*. These problems are not resolved by increasing the lookahead.

Figure 6.16 shows the same analysis for the *word* tag in the segmentation corpus. The data from this graph (Table 6.15) show that while the search space increased by five orders of magnitude, the recall and precision increased by less than one percent. It is not clear why recall and precision cross-over in Figures 6.15 and 6.16 as lookahead increases, but the levelling-off of increase in recall and precision, indicative and representative of larger samples, suggests that the model does not contain all the information needed to make the underlying relevancy decisions.

These results show that the maximum lookahead heuristic can be effective. In-

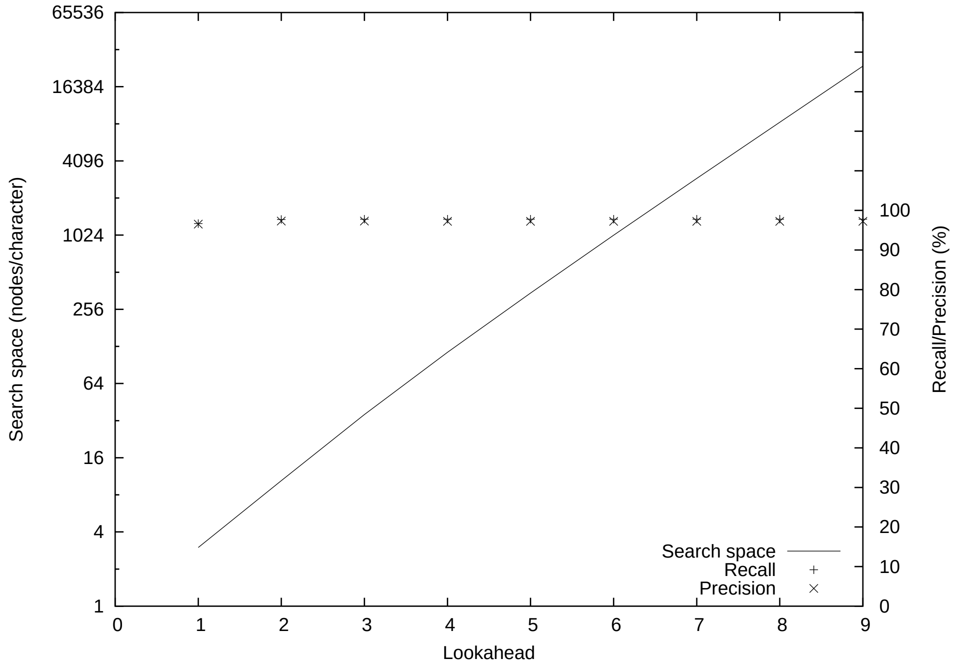


Figure 6.16: Graph of recall, precision and search space against lookahead for the *word* tag. Models trained on 2000 documents and tested on one document.

Lookahead	Search space (nodes per character)	Recall (%)	Precision (%)
1	6.00	97.10	97.37
2	27.26	97.83	97.79
3	86.22	97.82	97.53
4	241.07	97.73	98.21
5	633.54	97.74	98.21
6	1598.50	98.30	98.06
7	3976.08	97.72	97.59
8	9801.47	97.61	98.16
9	23457.08	97.77	97.87
10	58153.64	97.84	98.09
11	139079.05	97.71	98.02

Table 6.15: Table of recall, precision and search space against lookahead for the *word* tag. The data is plotted in Figure 6.16.

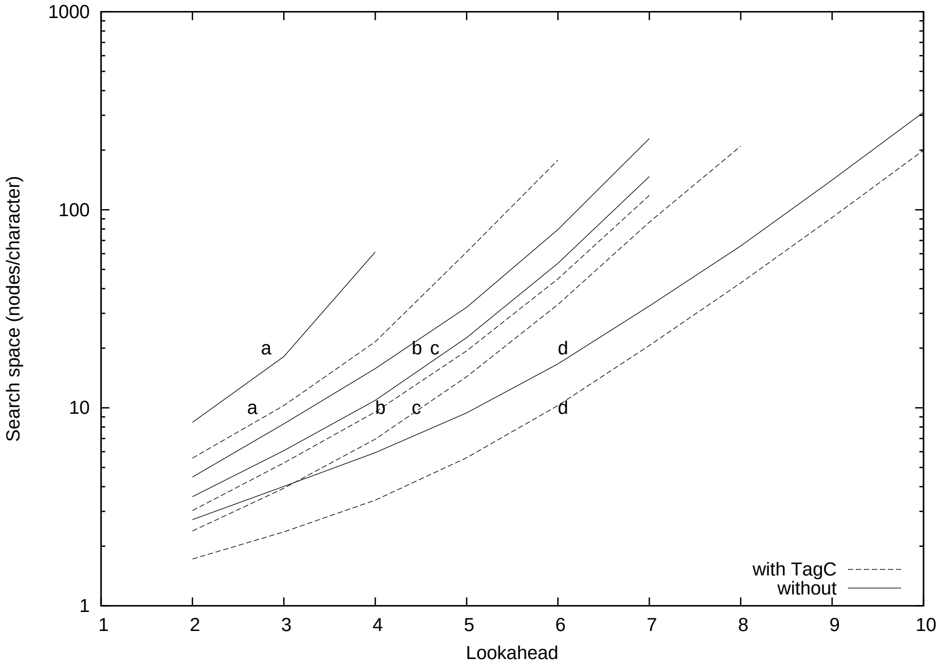


Figure 6.17: TagC heuristic in hierarchical tag insertion. From steepest to shallowest the lines are: (a) *author*, *editor*, *name*, *first* and *last*; (b) *name*, *first* and *last*; (c) *name* and *last*; (d) *name*. All runs used an order 3 model with 200 training documents and a single testing document.

creasing the lookahead beyond six has, in this case, no obvious benefit to recall and precision but is of great detriment to the search space.

6.4.5 TagC Heuristic

The TagC heuristic (Section 4.3.6) limits the number of tags to be considered for insertion between two characters in a document. Figure 6.17 shows the effect of the TagC heuristic on the hierarchical tags *author*, *editor*, *name*, *first* and *last* in the bibliography corpus. In all cases the search space was reduced. Figure 6.18 shows the effect of the TagC heuristic on the non-hierarchical tags *name*, *pages*, *date*, *volume* and *number* in the bibliography corpus.

Results show the TagC heuristic to be consistent and significant. Much of the pruning of the TagC heuristic is similar to that of the best first optimisation. A tag that is ruled out by the TagC heuristic has not been seen in this model before,

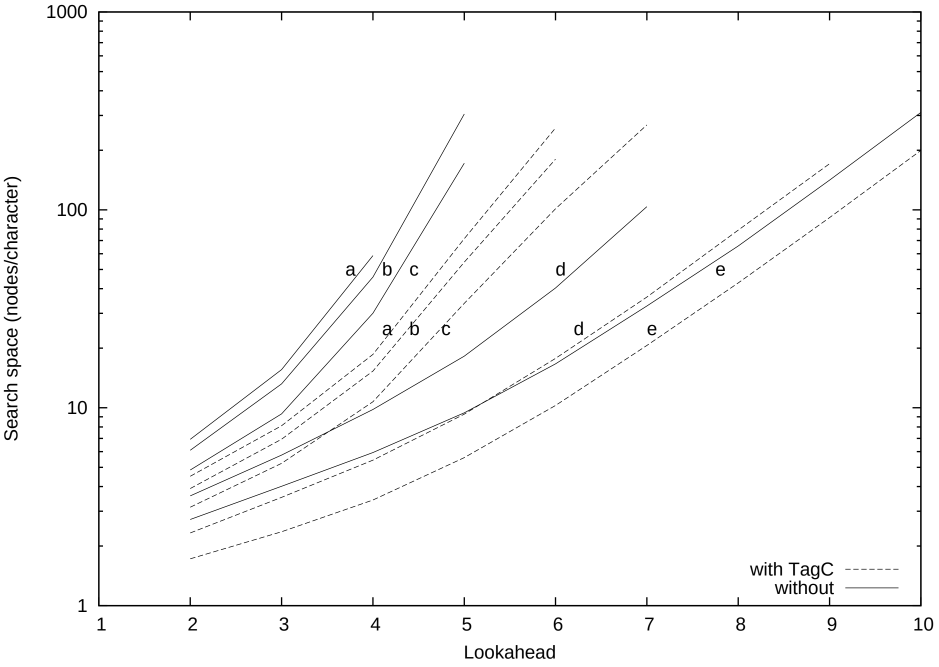


Figure 6.18: TagC optimisation in non-hierarchical tag insertion. From steepest to shallowest the lines are: (a) *name*, *pages*, *date*, *volume* and *number*; (b) *name*, *pages*, *date* and *volume*; (c) *name*, *pages* and *date*; (d) *name* and *pages*; (e) *name*. All runs used an order 3 model with 200 training documents and a single testing document.

meaning the PPM model must escape back to order -1 (see Section 3.4), and implying high entropy transitions. The structure of PPM models means an order a transition can be followed, at most, by an order $a+1$ transition (except for the start of sequence symbol), so an order -1 transition can be penalised over an order n transition for $n+1$ transitions. Many of the tags and tag sequences ruled out by the TagC heuristic would mean three or four order -1 transitions and can be rapidly pruned by the best first under normal circumstances.

The set of observed tag combinations is smaller in the bibliography corpus than it may be in real-world corpora because, when integrating the tagged and untagged bibliographies (see Figure 5.2), placement of tags with respect to inter-word whitespace was performed automatically and therefore consistently. Diverse, real-world, uncurated sources are unlikely to display this degree of consistency.

6.4.6 State Tying

The opportunity to apply the state tying heuristic (see Section 4.3.7) occurred only once in the corpora studied, on the *name* tag which may occur within the *editor* or the *author* tag in the bibliography corpus. The schema for the bibliography dataset with and without state tying are shown in Figure 5.4 and Figure 5.5 respectively. Figure 5.4 differs from Figure 5.5 in that the *name* subtree has been cloned and a copy appears for each parent. This section examines the effect this duplication has on the performance of the model.

Table 6.16 shows the type confusion matrices, with and without state tying, for the bibliography corpus. Perhaps surprisingly, the two key leaf tags *first* and *last* perform similarly in the two models. This is evidence that good models were built for these tags both with and without tying. At a slightly higher level, the tying performed noticeably better (more than 1%) at identifying *name* tags, while without tying performed noticeably better (more than 1%) at identifying *editor* tags. This later improvement appears to be because that proceedings editors often only have

99.88	+	+		+	+	+					+	+	+		+
+	**80.06**	+	+	+	+	+	+	+	+	+	3.02	7.09	4.64	1.80	1.54
	3.01	**83.93**	3.76	2.21	+	3.49					2.85		+	+	+
	+	+	**92.59**	2.43	+	+					3.91	+	+	+	+
+	1.52	+	4.27	**89.50**	+	1.87					1.94	+	+	+	+
	+	+	+	+	**94.58**	2.92					1.87	+	+	+	+
	+	+	+	+	1.28	**94.78**	+		+		3.21	+	+	+	+
	+	+		+	+	+	**99.04**	+	+		+			+	+
	+	+		+	+	+	+	**97.27**	+	+	1.01	+	+	+	+
	3.32					+	+	+	**94.79**	+	+	+	+	+	+
	1.46						+	+	+	**96.97**	+		+		
+	1.28	+	+	+	+	1.35	+	+	+	+	**93.91**	+	2.41	+	+
+	3.28	+		+	+	+	+	+	+	+	1.27	**94.52**	+	+	+
	3.13	+	+	+	+	+		+	+	+	7.94	+	**87.29**	+	+
	8.01	+		+	+	+	+	+	+	+	3.61	1.57	+	**81.70**	3.98
	4.19	+		+	+	+	+	+	+	+	2.60	+	1.85	2.68	**87.30**
bibliography	bibbody	editor	author	name	first	last	pages	date	volume	number	title	journal	booktitle	publisher	address
99.87	+	+				+	+				+		+	+	+
+	**77.14**	+	+	+	+	+	+	+	+	+	3.01	8.25	6.52	1.82	1.39
	3.31	**85.88**	3.33	1.39	+	2.82					2.62		+	+	+
	+	+	**92.78**	2.37	+	+					3.56		+	+	+
	1.59	+	5.30	**87.70**	+	2.49					1.91	+	+	+	+
	+	+	+	+	**94.35**	3.31					1.55	+	+	+	+
	+	+	+	+	1.45	**94.49**			+		3.10	+	+	+	+
	+						**98.95**	+	+	+	+			+	+
	+	+		+	+	+	+	**96.87**	+		1.06		1.18	+	+
	2.09			+	+	+	+	+	**95.99**	+	1.11	+	+	+	+
	+						+	+	+	**96.09**	1.53		+		
+	1.17	+	+	+	+	1.70	+	+	+	+	**93.08**	+	3.05	+	+
	2.63	+	+	+	+	+	+	+	+		1.11	**95.54**	+	+	+
+	3.71	+	+	+	+	+	+	+	+		10.23	+	**84.59**	+	+
	7.25	+	+	+	+	+	+	+	+		4.57	1.85	+	**81.47**	3.77
	3.54	+		+	+	1.18	+	+	+		3.02	+	2.35	2.86	**85.82**

Table 6.16: Type confusion matrices for the bibliography corpus. The matrix on the left is with state tying and the matrix on the right is without state tying. All values are in percent, a '+' indicates a figure lower than 0.99%. Order 6 models trained on 6000 documents and tested on 500 documents with a lookahead of 5.

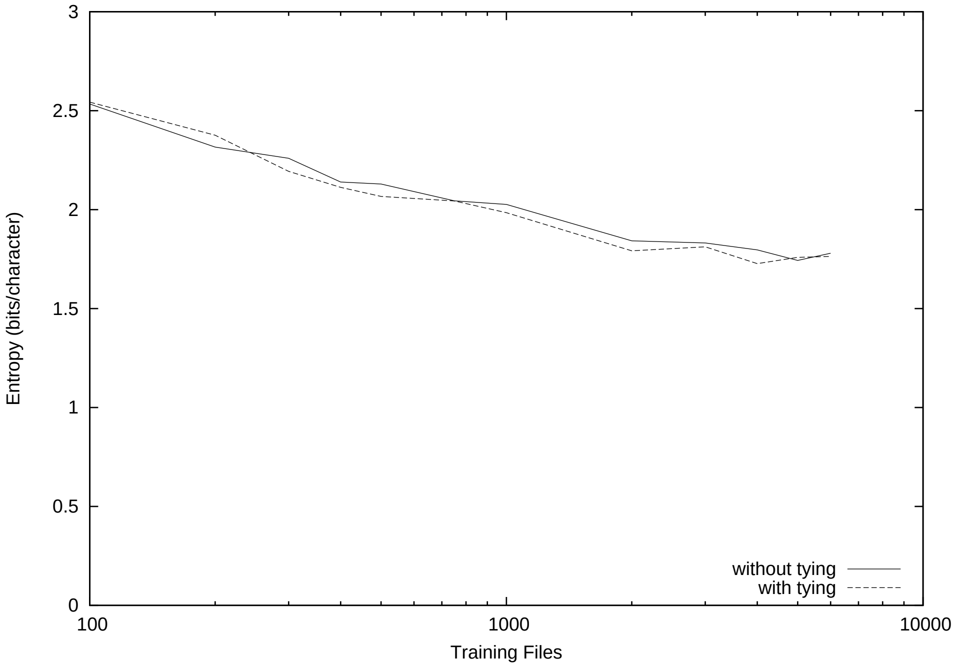


Figure 6.19: Entropy dropping with increased training data, with and without state tying. Order 6 models tested on 500 documents with a lookahead of 5.

their last name given in bibliography entries and modelling *editor* tags separately from *author* tags allowed this information to be captured.

In the tags not directly related to names, the state tying results are slightly better than the without state tying results, having a higher number of results on the leading diagonal in nine of eleven cases. This is, perhaps, because the state tying presented a more consistent model of the concepts of names to the rest of the model. Other features of type confusion matrices for the bibliography corpus are explained in Section 6.2.3.

Figure 6.19 shows how entropy drops with increased training data, with and without state tying, for the tags shown in Table 6.16. Entropy with state tying appears to be slightly less, but not consistently less, than entropy without state tying. This is somewhat surprising since the motivation for state tying was to achieve better performance from the same amount of training data (Section 4.3.7), and this appears not to be happening consistently. This is probably because the effect which

98.98	+	+	+	+	+
2.85	**88.50**	+	4.86	+	2.79
9.91	+	**81.03**	6.36	+	2.20
2.86	1.88	1.68	**93.38**	+	+
1.10	+	+	+	**94.73**	3.85
2.68	+	+	+	1.49	**95.64**
bibliography	name	editor	author	first	last
98.53	+	+	+	+	1.29
2.57	**89.04**	+	5.45	1.01	1.74
10.52	+	**81.36**	5.90	+	1.78
3.41	2.44	2.29	**91.78**	+	+
1.16	+	+	+	**95.96**	2.67
2.48	+		+	1.81	**95.56**

Table 6.17: Type confusion matrices for the bibliography corpus. The matrix on the left is with state tying and the matrix on the right is without state tying. All values are in percent, a ‘+’ indicates a figure lower than 0.99%. Order 6 models trained on 6000 documents and tested on 100 documents with a lookahead of 5.

is noticeable in Table 6.16 is too small to be detected over the sampling error.

Table 6.17 shows the type confusion matrices, with and without state tying, for the bibliography corpus for a greatly reduced set of tags compared with Table 6.16. The results do not show a clear pattern of similarity with those shown in Table 6.16 for the larger set of tags, suggesting that the results are not generally applicable.

An unanticipated benefit of state tying is that the combined models are significantly smaller than the separate models. The memory consumption of models increases linearly with extra tags but less than linearly with extra training data: if two tags are tied together to use the same PPM model, memory can be saved. The CEM implementation uses memory näively, no experimentation or tuning has been used to reduce the memory consumption.

The state tying optimisation gives at best a marginal improvement in results, but can be expected to lead to smaller models. Occam’s Razor (also called the ‘principle of parsimony’ or the ‘principle of simplicity’) asserts that a simpler or smaller model of a phenomenon is to be preferred over a more complex or larger one.

Chapter 7
Conclusions

This thesis extended text augmentation to cover entity extraction problems. It investigated three classes of text augmentation: segmentation, classification and entity extraction, and described how they are connected to data mining, text mining and related fields.

Segmentation, the computationally simplest class, involves segmenting the text. Information is encoded in where one segment ends and the next starts. Tasks such as Chinese text segmentation were evaluated using recall and precision on the segment boundaries.

Classification, which is more computationally expensive than segmentation, involves classifying textual elements into one of several classes. Information is encoded in the class an element falls into. Classification tasks, such as part of speech tagging, have close ties to machine learning, and share with it the confusion matrix evaluation method.

Entity extraction is the most computationally expensive class of text augmentation. It marks-up textual fragments with a nested hierarchy of classes and information is encoded both in where fragments start and finish and in their type. Inserting attribute-free XML into text is an entity-extraction task. Entity extraction was evaluated using type confusion matrixes and using edit distances.

7.1 Review of Aims

In Section 1.2 various aims were introduced; in this section they are examined to determine whether they have been met.

1. *Examine text-augmentation problems, in the large, to attempt to determine which are susceptible to automated text augmentation and whether some sets of problems are inherently easier than other.*

   Section 4.1 built a taxonomy of three taxa of text-augmentation problems: segmentation, classification and entity extraction. Collection and document level metadata are poorly catered for. Section 4.1.4 covers a number of forms of fine-grained metadata which does not sit within the taxonomy. Sections 4.4 and 6.4 examines the different static and dynamic performance of various searches over the different problems. Segmentation is computationally easier than classification, which is computationally easier than entity extraction. This aim has been met.

2. *Build a text-augmentation system capable of solving at least as wide a range of problems as existing low-human-input systems, with an eye to eventual inclusion as part of a digital library system.*

   Section 4.2 describes CEM, a system capable of solving a wider range of text-augmentations problems than the immediately previous systems TMT and SMI, which did not solve entity-extraction problems. CEM has low-human-input and has a number of design characteristics such as using Unicode throughout and using standard XML documents. This aim has been met.

3. *Locate and/or build corpora to test this system.*

   The four corpora used in this thesis are described in Chapter 5. The Computists' corpus was developed from an earlier corpus; the Chinese text segmentation and Reuters' corpora were existing corpora adapted for use. The bibliography corpus was built as a model entity-extraction corpus. This aim has been met.

4. Use specific heuristics and optimisations which perform well in relation to a particular set of augmentation problems.

   The best first optimisation and automatic tokenisation, alphabet reduction, maximum lookahead, TagC and state tying heuristics are described in Chapter 5 and used with particular types of augmentation problems. State tying is effective only on entity extraction problems (Section 4.3.7) and TagC only works on entity extraction and classification problems (Section 4.3.6). This aim has been met.

5. *Evaluate both the text-augmentation system and the heuristics and optimisations in the system.*

   Chapter 6 contains a systematic evaluation of both the system as a whole and individual heuristics and optimisations. This aim his been met.

7.2 Performance of CEM and the New Techniques

The implementation, CEM, created for this thesis uses a substantially different form of model from that used by previous workers. The model not only allows fully recursive modelling to deeply tagged XML, it also carries context between hidden states, which avoids prejudicing entry to these states by avoiding escaping back to low-order models. CEM also uses a significantly more efficient variation on the PPMD escape method avoiding full exclusion. Non-full exclusion is a substantial performance improvement over full-exclusion with marginal less of correctness.

The best first optimisation leads to substantial gain. It could be argued that the best first optimisation was an implementation detail rather than a true optimisation. It is, however, absent from the immediately preceding system, Teahan's TMT.

Hardware implementations of Viterbi search usually avoid the need for the best first optimisation by performing this step in parallel.

The maximum lookahead heuristic is used elsewhere and was shown to work in CEM to good effect. Unfortunately there is no apparent *a priori* method for selecting a maximum lookahead, other than by splitting a known-good corpus into a training corpus and a testing corpus. This technique is less effective once the Baum–Walsh algorithm has been used to adapt the model to a superset of the original corpus.

CEM also implements two novel heuristics, TagC and automatic tokenisation, to some advantage. Both are reliant on the consistency of the training data and are unlikely to be widely useful on uncurated diverse corpora. They also largely prune the search tree in ways that the best first optimisation also prunes effectively.

The state tying heuristic, which is widely used in voice-recognition systems, was found to have little effect on the search space, but reduced the size of the hidden Markov model by merging some of the underlying Markov models. If the semantics of tag nesting are changed, state tying is likely to be more effective. In either case, it reduces the number of Markov models, and proportionally reduces the required volume of training data. The use of state tying in this way, however, hampers the convergence towards consistent tagging in the marked up text, by making the Markov model that best matches a fragment accessible at multiple hidden states. This is likely to be a significant barrier to the incremental development of corpora using the system to improve the quality of the training text. It may be possible to enable state tying during training, and disable it during testing and re-estimation to restrict access to each Markov model to a single hidden state, thus standardising the tagging.

Four corpora were used in this thesis. Marking-up the Chinese text segmentation corpus was a task on which CEM achieved an F-measure of 98%, in the same range as other systems and better than TMT. The Reuters' corpus was used in conjunction with the Brill part of speech tagger, but CEM performed poorly on this classification task, because the PPM models in CEM have a linear context and lack super-adjacency, a key aspect of the Brill tagger and other part-of-speech taggers.

A detailed comparison of the performance of CEM and the similar TMT system on the Computists' corpus showed that TMT performed consistently better. The differences were shown to be related to both the modelling characters rather than words, and the search algorithm.

The fourth corpus was the bibliography corpus, which was used for entity extraction. CEM appeared to perform well, but the lack of a standard test corpus made comparison with other systems difficult.

CEM includes the Baum–Welch algorithm: this was successfully used to help adapt a model trained on one style of bibliography to markup a different style. In this thesis the Baum–Welch algorithm was evaluated using the edit-distance metric.

CEM can be applied to solve a significantly wider range of problems than the immediately preceding system (TMT), which could solve segmentation and classification problems but not entity extraction. CEM performed well at both the simple and complex ends of the computational spectrum. It was, however, not so well optimised for speed or memory consumption as TMT.

7.3 Impact of Unicode and Document Orientation

Use of Unicode solves many internationalisation issues, but not the unknown-character problem: the character level equivalent of the unknown word problem. It also provides a set of cross-language character classes on which word-level rules and models can be built. The character classes are similar in approach to the character classes from the C programming language, which have a long history of use in parsers.

Encoding metadata, as a CEM does, in a single hierarchical insertion of attribute-free XML tags, limits the classes of metadata that can be represented, in particular, overlapping structures and alternative interpretations of the same passage. There are interesting sets of metadata that fall into the excluded category, in particular: overlapping hierarchies such as physical and logical document structure, and metadata constructed from fragments scattered throughout the document text.

The view of the data and metadata as an annotated document rather than a collection of facts has a number of impacts on further use, even though metadata held in an external database could be processed to embed it in the document and *vice versa*. Firstly it makes the document more amenable to presentation as a metadata-enhanced document, such as in a digital library or an XML-based document repository. Secondly it makes the kinds of higher-level processing used in the later stages of many of the MUC systems harder, because these perform operations such as relational joins which have no direct equivalent in an annotated document. Thirdly it makes the metadata significantly less amenable to export for use in external systems, many of which expect relations of data. Fourthly document-centric, XML-native, databases allow queries on the annotated XML documents, including aspects of the documents which the querier might consider important which the metadata extractor might not. The best representation for inferred metadata is thus likely to be determined by the larger context and the intended uses of the metadata.

7.4 Limitations of CEM

CEM has two broad sets of limitations, those imposed by modelling and search techniques, and those due to the implementation of those techniques.

Attribute data CEM does not capture attribute data. For enumerable attributes, this can be mitigated by XML transformations which transform each possible combination of attributes in each tag to a separate tag. For continuous attributes this technique leads to an infinite number of tags. It is not clear how many continuous attributes occur in linguistic corpora, the author has seen continuous attributes in spoken linguistic corpora (particularly in the time dimension) but not in written linguistic corpora.

Differentiable tags Tags that do not have different character distributions, or whose character distributions PPM is unable to model, cannot be inserted. An extreme case of this might be the task of marking-up the prime-numbered digits in a decimal representation of π. While automating such a marking up is possible, doing it with Viterbi search and learnt PPM models is not. The author is aware of no linguistic corpora for which this is an issue.

Consistency Tags are assumed to be used consistently. This does not hold for many real-world situations, but curated textual corpora are becoming more common. There are also various tools such as jtidy[1] which regularise some aspects of HTML/XHTML.

These three limitations are shared with all directly comparable applications of searching using Markov models, including TMT and HTK. The second set of limitations are implementation-based, caused by choices made when building CEM.

Number of tags CEM has an upper bound on the number of Markov models and thus of tags modelled. The implementation represents tags using Unicode characters from the private use range \uE000–\uF8FF, of which 3 are reserved as special markers. While an order of magnitude greater than the number of tags appearing in commonly used markup such as XHTML, MathML and those appearing in this thesis, this limits the use of tag transformations as work-arounds for other limitations.

Nesting of tags CEM cannot represent tags nested directly within tags of the same type. This is currently impossible because in the search nodes only the tag is

[1] `http://jtidy.sourceforge.net/`

noted and not whether it is opening or closing. None of the corpora examined here displays such nesting and while it would be relatively easy to fix, it would involve an extra test in the inner loop of the search operation, slowing searching. An alternative to changing the implementation is to transform the text so that every odd-depth tag has a different name, and then use state-tying to tie the odd and even tags together. HTK supports models such as these, TMT does not.

Adaptive Models The PPM models implemented in CEM are not adaptive. This means that the Baum–Welch algorithm cannot be applied any finer than the document level, for example to allow intra-document learning. This is likely to be a problem when the re-estimation text contains relatively few but unusually large documents, allowing few re-estimation cycles. If the documents are internally homogeneous, it may be possible to overcome this by splitting them to increase the number of inter-document re-estimation cycles. Both HTK and TMT can be adaptive.

Streaming documents Documents are held entirely in memory rather than being streamed. Holding documents in memory consumes extra memory. While this was not a problem for corpora used in this work, which have reasonably short documents, it would prevent processing of large documents. Documents as large as 6MB (unmarked up size) have been successfully marked up. Document length is linearly related to this aspect of memory consumption. HTK allows documents to be streamed, TMT does not.

Document-at-once processing An entire XML document, rather than an XML fragment, must be marked up at once. The command line to interface CEM requires documents be read from the file system, one document per file. A Java interface allowing arbitrary XML nodes to be marked-up exists but is not used in the experiments presented here. Marking-up document fragments is important in interfacing CEM with other systems. Both HTK and TMT have interfaces allowing partial documents to be processed.

Integer overflow The PPM models implemented in CEM implicitly assume that none of their counters rolls over. This assumption holds unless more than $2^{31} - 1$ characters of training data (or combined training and re-estimation data) are seen. HTK overcomes this limit by encoding probabilities as floating-point numbers rather than as ratios of integers. TMT overcomes this limit using integers that are scaled prior to overflow. The latter could be worked into CEM.

CEM does not have a mode of operation which calculates the entropy of entire documents in each of the Markov models. This is used effectively by TMT for calculation of whole document metadata such as language and genre. Of these implementation limitations, only making the PPM models adaptive and removing the upper bound on the number of tags would require extensive redesign of CEM.

7.5 Problems Suitable for CEM and Text Augmentation

There are several broad indicators that metadata will be marked up well by CEM: it should be relatively fine-grained, at the character, word or phrase level; it should be discriminatable from the immediately surrounding text; there should be a training corpus which matches the testing text sufficiently well to build a model from (or text available to build such corpus from); if the testing text is changing with time, it should be changing sufficiently slowly that the model can be re-estimated to track the changes.

Segmentation problems that meet these requirements include the segmentation of languages written without spaces between words (i.e. Chinese, Japanese and

Thai) and locating potential hyphenation points in European languages (i.e. English, German and French). Classification problems that meet these requirements include part-of-speech tagging, finding proper nouns, email addresses, URLs, stock, cross-references and similar classes of textual entities. Entity-extraction problems that meet these requirements include marking-up bibliographies, title and frontis pages, email headers, standard forms and other highly-structured sections of text.

Parsing of many computer programming languages, including Scheme, Java and C, into an XML representation is an entity-extraction problem, although not one CEM is ideal for, because of the length of structures involved. Parsing of the Python language is not, and CEM is not capable of this task: the concept 'the same indentation as the previous line' cannot be learnt using PPM.

In all cases, higher-order reasoning based on the inferred metadata is beyond the ability of CEM. For example, while it can find proper nouns in English text, but it cannot be used to find equivalences between different nouns used for the same subject, because this requires reasoning about on non-adjacent values. Since this higher-order reasoning is an integral part of many systems used in the wild, CEM is unlikely to be a suitable drop-in replacement for many systems.

7.6 Training Corpora Sizes

The relative success of text augmentation on the Computists' corpus, with only 38 issues of 1200 words, shows that augmentation can be useful even when trained on relatively small volumes of text. Certainly this augmentation is of high enough quality to be used for transforming the document for presentation to end users. With F-measures as low as 55%, however, the augmented text should be used with care. In particular, the compilation of indexes and of extracted terms, in which recurring terms contribute less than singly-occurring terms should be avoided, as this emphasises errors, which tend to be unique, singly-occurring items.

Estimating the quantity of training text needed to produce results of a certain quality is challenging because of the many factors that influence this, but it seems apparent, supported by the experimental results in Chapter 6, that model discrimination is key. For example in the Computists' corpus, the easily-discriminated URL and email tags were augmented reliably, whereas the poorly-discriminated name, organisation and location tags were augmented poorly, despite considerably more examples being seen in training.

The incremental development of the Computists' corpus, together with an examination of the errors of text augmentation systems leading the correction of the training text, is likely to be particularly scalable, since it allows leveraging of work already completed to converge on a consistently marked up corpus. Unfortunately, incremental development may reveal flaws in the initial assumptions, which are unlikely to be rectifiable without considerable work.

The automated conversion of existing data and metadata into a corpus, as for the bibliography corpus, has the advantage that the metadata in existing data is presumably present for a reason, reflecting the use or meaning of the data. The conversion is automated, so if the conversion reveals issues it can be re-performed completely.

Automatic conversion is limited to those corpora for which a suitable data source can be found with suitable metadata, and those found are unlikely to be structured to allow for control of arbitrary variables of interest. The growth of curated repositories may increase the likelihood that a corpus already exists that can be converted, extended or developed to be suitable.

7.7 Original Contributions

A number of original contributions are made in this thesis. A system called 'Colloquial Entropy Markup' or CEM was designed and implemented. CEM builds a hidden Markov model from a corpus of marked-up XML documents and uses variants of Viterbi search to augment unmarked-up XML documents with tags in the marked-up XML documents.

Four corpora were used. The Reuters' and segmentation corpora required relatively little data preparation. The Computists' corpus was systematically re-marked-up. The bibliography corpus is a new corpus.

The following are the key novel aspects of the work presented in this thesis.

- Partitioning of tag insertion problems into a coherent taxonomy with three taxa (Section 2.1.2).
- Exploration of the relationship between PPM (Prediction by Partial Matching) models and Markov models (Section 3.3). Previously published as [164].
- Expansion of text augmentation to include nested tags (Chapter 4).
- The best first (Section 4.3.2) optimisation, the automatic tokenisation (Section 4.3.3), alphabet reduction (Section 4.3.4) and TagC (Section 4.3.6) heuristics.
- Detailed analysis of the search space size of tag insertion (Section 4.4). Earlier versions of this work were published as [162].
- Detailed analysis of the correctness measures for different types of tag insertion problems and research methodology (Section 2.3).
- Development of an entropy-based technique to determine whether tag-insertion errors are the result of a PPM modelling failure or of a searching failure (Section 2.3.4).
- A new extension of confusion matrices suitable for evaluating hierarchical many-class classification problems (Section 4.6.4).

7.8 Open Questions

There are a number of open questions not examined in this thesis:

1. Whether the conceptualisation of context used here (and elsewhere) is optimal. There is an alternative method for computing the context of the current character in a character stream. This was discovered during the experimental work for this thesis, but not explored. The context for *e* in …*<a>ab</a><b>cd</b><c>e*… can be 'collapsed' to …*<a/><b/>e*…. This could be achieved by substituting the character representing the transition into the tag for the entire tag. This approach is likely to be most successful where tag densities are highest, such as in part-of-speech tagging, where state-of-the-art systems take advantage of super-adjacency.

2. Whether adding a default tag with an uninitialised (untrained) model accessible from every context would remove the tendency to place high-entropy sequences in the model with the least training data.

3. Whether different escape methods would reduce the tendency to place high entropy sequences in the model with the least training data.

4. Whether a more universal similarity metric such as Kolmogorov complexity [85, 86] might be an appropriate measure for comparing sequences. This would move evaluation to a theoretical framework independent of any particular approach to solving the problem and resolve some of the complexities of evaluating performance.

5. Whether certain textual strings (such as *References* on a line by itself) can be used as synchronisation points in a finite automata sense. This is likely to form part of the infrastructure integrating CEM into a possible digital library structure, which will need ways of detecting when it is appropriate to use various tools such as CEM.

6. Whether Teahan search or Viterbi search will perform better on certain classes of text-augmentation tasks.

All of these seem useful avenues of investigation, 1 and 4 being significantly more novel than 2 and 3. Issues 5 and 6 are likely to be directly and immediately relevant to a practical production system.

Bibliography

[1] Eugene Agichtein and Luis Gravano. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the Fifth ACM conference on Digital libraries*, pages 85–94, San Antonio, Texas, United States, 2000.

[2] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques and Tools*. Addison-Wesley, 1986.

[3] Rie Kubota Ando and Lillian Lee. Mostly unsupervised statistical segmentation of Japanese Kanji sequences. *Journal of Natural Language Engineering*, 9(2):127–149, August 2003.

[4] J. Anigbogu and A. Belaid. Hidden Markov models in text recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 9(6):925–958, 1995.

[5] Steven Atkin and Ryan Stansifer. A generalized mechanism for Unicode metadata. In *Proceedings of the Nineteenth International Unicode Conference*, San Jose, California, USA, 10–14 September 2001.

[6] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. ACM / Addison–Wesley, Massachusetts, USA, 1999.

[7] Amit Bagga. Analyzing the complexity of a domain with respect to an Information Extraction task. In *Proceedings of the Fifth Workshop on Very Large Corpora*, 1997.

[8] David Bainbridge, Dana McKay, Ian H. Witten, and Stefan Boddie. *Greenstone Digital Library Developer's Guide*. Digital Library Laboratory, University of Waikato, March 2004.

[9] Alex Bateman, Ewan Birney, Lorenzo Cerruti, Richard Durbin, Laurence Etwiller, Sean R. Eddy, Sam Griffiths-Jones, Kevin L. Howe, Mhairi Marshall, and Erik L. L. Sonnhammer. The Pfam protein families database. *Nucleic Acids Research*, 28:263–266, 2000.

[10] Leonard E. Baum. An inequality and associated maximisation technique in statistical estimation for probabilistic functions of a Markov process. *Inequalities*, 3:1–8, 1972. Not sighted.

[11] Leonard E. Baum, Ted Petrie, George Soules, and Norman Weiss. A maximisation techniques occurring in the statistical analysis of probabilistic functions of markov chains. *The Annals of Mathematical Statistics*, 41(1):164–171, 1970.

[12] Doug Beeferman, Adam Berger, and John D. Lafferty. Statistical models for text segmentation. *Machine Learning*, 34(1-3):177–210, 1999.

[13] Timothy C. Bell, John G. Cleary, and Ian H. Witten. *Text Compression*. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1990.

[14] Timothy C. Bell, Ian H. Witten, and John G. Cleary. Modeling for text compression. *ACM Computing Surveys*, 21(4):557–591, December 1989.

[15] Dominique Besagni, Abdel Belaïd, and Nelly Benet. A segmentation method for bibliographic references by contextual tagging of fields. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition*, page 384, Edinburgh, Scotland, August 2003.

[16] Alan W. Biermann and Amit Bagga. Analyzing the complexity of a domain with respect to an Information Extraction task. In *Proceedings of the Fifth Workshop on Very Large Corpora*, 1997.

[17] Daniel M. Bikel, Richard L. Schwartz, and Ralph M. Weischedel. An algorithm that learns what's in a name. *Machine Learning*, 34(1-3):211–231, 1999.

[18] Jeff A. Bilmes. A gentle tutorial on the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models. Technical Report ICSI-TR-97-021, University of Berkeley, Massachusetts, USA, 1997.

[19] Steven Bird, Peter Buneman, and Wang-Chiew Tan. Towards a query language for annotation graphs. In *Proceedings of the Second International Conference on Language Resources and Evaluation*, pages 807–814, Paris, France, 2000. European Language Resources Association.

[20] Steven Bird, Kazuaki Maeda, Xiaoyi Ma, Haejoong Lee, Beth Randall, and Salim Zayat. TableTrans, Multitrans, InterTrans and TreeTrans: Diverse tools built on the annotation graph toolkit. In *Proceedings of the Third International Conference on Language Resources and Evaluation*, Paris, France, 2002.

[21] Kurt Bollacker, Steve Lawrence, and C. Lee Giles. CiteSeer: An autonomous web agent for automatic retrieval and identification of interesting publications. In Katia P. Sycara and Michael Wooldridge, editors, *Proceedings of the Second International Conference on Autonomous Agents*, pages 116–123, New York, New York, USA, 1998. ACM Press.

[22] Kalina Bontcheva, Marin Dimitrov, Diana Maybard, Valentin Tablin, and Hamish Cunningham. Shallow methods for named entity coreference resolution. In *Conference annuelle sur le Traitement Automatique des Langues Naturelles*, Nancy, France, 24–27 June 2002.

[23] A. Borthwick, J. Sterling, E. Agichtein, and R. Grishman. Nyu: Description of the mene named entity system as used in MUC. In *Proceedings of the Seventh Message Understanding Conference (MUC-7)*, 1998.

[24] Bert Bos, Tantek Çelik, Ian Hickson, and Håkon Wium Lie. Cascading style sheets 2.1 specification. Technical report, World Wide Web Consortium (W3C), 25 February 2004.

[25] Tim Bray, Jean Paoli, and C. Michael Sperberg-McQueen. EXtensible Markup Language 1.0. Recommendation, The World Wide Web Consortium, 10 February 1998.

[26] Zane C. Bray. Using compression models for text mining. Master's thesis, Department of Computer Science, University of Waikato, Hamilton, New Zealand, July 1999.

[27] Alvis Brazma, Inge Jonassen, Ingvar Eidhammer, and David Gilbert. Approaches to the automatic discovery of patterns in biosequences. *Journal of Computational Biology*, 5(2):277–304, 1998.

[28] Eric Brill. A simple rule-based part-of-speech tagger. In *Proceedings of the Third Conference on Applied Natural Language Processing*, pages 152–155, Trento, Italy, 1992.

[29] Eric Brill. Some advances in transformation-based part of speech tagging. In *Proceedings of the Twelth National Conference on Artificial Intelligence*, pages 722–727, 1994.

[30] John Seely Brown and Paul Duguid. *The Social Life of Information*. Harvard Business School Press, Boston, Massachusetts, USA, March 2000. Excerpt published in First Monday 5:4 April 2000.
http://www.firstmonday.org/issues/issue5_4/brown_contents.html.

[31] M. Buckland and F. Gey. The relationship between recall and precision. *Journal of the American Society for Information Science*, 45(1):12–19, 1994.

[32] Jeffrey T. Chang, Hinrich Schütze, and Russ B. Altman. Creating an online dictionary of abbreviations from MEDLINE. *Journal of the American Medical Informatics Association*, 23 July 2002.

[33] Eugene Charniak. *Statistical Language Learning*. MIT Press, Cambridge, Massachusetts, USA, 1993.

[34] Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. In Arivind Joshi and Martha Palmer, editors, *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pages 310–318. Morgan Kaufmann, 1996.

[35] Nancy A. Chinchor. Overview of MUC-7/MET-2. In *Proceedings of the Seventh Message Understanding Conference*, April 1998.

[36] John G. Cleary and William J. Teahan. An open interface for probabilistic models of text. In James A. Storer and Martin Cohn, editors, *Proceedings of the Data Compression Conference*, 1999.

[37] Hamish Cunningham, Yorick Wilks, and Robert J. Gaizauskas. Gate—a general architecture for text engineering. In *Sixteenth International Conference on Computational Linguistics*, volume 2, pages 1057–1060, Denmark, August 1996.

[38] Richard Curtis, Ben Elton, and John Lloyd. *Blackadder: the Whole Damn Dynasty*. Michael Joseph, London, England, 1998.

[39] Doug Cutting, Julian Kupiec, Jan Pedersen, and Penelope Sibun. A practical part-of-speech tagger. In Oliviero Stock, editor, *Proceedings of the Third Conference on Applied Natural Language Processing*, pages 133–140, Trento, Italy, April 1992.

[40] Doug Cutting and Jan Pedersen. *The Xerox Part-of-Speech Tagger*. Xerox Palo Alto Research Center, Palo Alto, California, USA, v1.0 edition, 12 April 1993.

[41] E. B. Dynkin. *Markov Processes*. Springer, New York, 1965. Translated into English by J. Fabius, V. Greenberg, A. Maitra and G. Majone.

[42] Tom Emerson. Segmentation of Chinese text. *MultiLingual Computing & Technology*, 12(38), February 2003.

[43] David C. Fallside. *XML Schema*. The World Wide Web Consortium (W3C), 2 May 2001.

[44] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. RFC 2068: HyperText Transfer Protocol — HTTP 1.1, January 1997.

[45] Dayne Freitag. Multistrategy learning for information extraction. In *Proceedings of the Fifthteenth International Conference on Machine Learning*, pages 161–169, San Francisco, California, USA, 1998. Morgan Kaufmann.

[46] Dayne Freitag and Andrew McCallum. Information extraction with HMM structures learned by stochastic optimization. In *Proceedings of the American Association for Artificial Intelligence Conference*, pages 584–589, 2000.

[47] Dayne Freitag and Andrew Kachites McCallum. Information extraction with HMMs and shrinkage. In *Proceedings of the American Association for Artificial Intelligence Workshop on Machine Learning for Information Extraction*, 1999.

[48] Betty Furrie. *Understanding MARC Bibliographic: Machine-Readable Cataloging*. Cataloging Distribution Service, Library of Congress, fifth edition, 1990.

[49] R. Gaizauskas and Y. Wilks. Information extraction: Beyond document retrieval. *Journal of Documentation*, 54(1):70–105, 1998.

[50] Xianping Ge, Wanda Pratt, and Padhraic Smyth. Discovering Chinese words from unsegmented text. In *Research and Development in Information Retrieval*, pages 271–272, 1999.

[51] Charles F. Goldfarb. *The SGML Handbook*. Oxford, 1990.

[52] James Gosling, Bill Joy, and Guy Steele. *The Java Language Specification*. Addison–Wesley, Massachusetts, USA, 1996.

[53] Donna K. Harman. Overview of the first text retrieval conference. In *Proceedings of the First Text REtrieval Conference*, pages 1–20, Gaithersburg, Maryland, USA, 4–6 November 1992. National Institute of Standards and Technology.

[54] Donna K. Harman. Overview of the fourth text retrieval conference. In *Proceedings of the Fourth Text REtrieval Conference*, pages 1–24, Gaithersburg, Maryland, USA, 1–3 November 1995. National Institute of Standards and Technology.

[55] M. Hearst. Untangling text data mining. In *Proceedings of the Thirty seventh Annual Meeting of the Association for Computational Linguistics*, 1999.

[56] Chris Heegard and Stephen B. Wicker. *Turbo Coding*. Kluwer Academic Publishers, 1999.

[57] Paul Glor Howard. *The Design and Analysis of Efficient Lossless Data Compression Systems*. PhD thesis, Department of Computer Science, Brown University, June 1993.

[58] Akira Ishikawa. A functional operator-based morphological analysis of Japanese. In *14th International Conference of Applications of Prolog*, Tokyo, Japan, 20–22 October 2001.

[59] ISO-9899—Harmonized standard for the C programming language, 1990.

[60] Frederick Jelinek. *Statistical Methods for Speech Recognition*. MIT Press, Boston, Massachusetts, USA, 1998.

[61] Thorsten Joachims. A probabilistic analysis of the Rocchio algorithm with TF/IDF for text categorization. In Douglas H. Fisher, editor, *Proceedings of Fourteenth International Conference on Machine Learning*, pages 143–151, Nashville, USA, 1997. Morgan Kaufmann.

[62] Thorsten Joachims. Text categorization with support vector machines: learning with many relevant features. In Claire Nédellec and Céline Rouveirol, editors, *Proceedings of Tenth European Conference on Machine Learning*, pages 137–142, Chemnitz, Germany, 1998. Springer.

[63] Rolf Johannesson and Kamil Sh. Zigangirov. *Fundamentals of Convolutional Coding*. Wiley, 1999.

[64] S. Johansson, G. Leech, and H. Goodluck. Manual of information to accompany the Lancaster-Oslo-Bergen corpus of british english for use with digital computers. Technical report, Bergen: Norwegian Computing Center for the Humanities, 1978.

[65] Karen Sparck Jones and C. J. van Rijshergen. Report on the need for and provision of an "ideal" information retrieval test collection. Technical report, Cambridge University Computer Laboratory, December 1975.

[66] Jussi Karlgren and Douglass Cutting. Recognizing text genres with simple metrics using discriminant analysis. In *Proceedings of the Fifteenth International Conference on Computational Linguistics*, volume II, pages 1071–1075, Kyoto, Japan, 1994.

[67] Slava M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics Speech and Signal Processing*, 35(3):400–401, March 1997.

[68] D. Khmelev and William J. Teahan. A repetition-based measure for verification of text collections and for text categorization. In *Proceedings of the 26th Annual International ACM SIGIR Conference (SIGIR),* Toronto, Canada, 28 July–1 August 2003.

[69] Y Khmelev and William J. Teahan. A repetition-based measure for verification of text collections and for text categorization. In *26th Annual International ACM Special Interest Group on Information Retrieval Conference (SIGIR),* Toronto, Canada, 28 July–1 August 2003.

[70] Jeffrey H. Kingston. *Algorithms and Data Structures—Design, Correctness, Analysis*. Addison–Wesley, Massachusetts, USA, 1990.

[71] Donald E. Knuth. *The Art of Computer Programming*, volume 1, Fundamental Algorithms. Addison–Wesley, first edition, 1968.

[72] Anders Krogh, Michael Brown, I. Saira Mian, Kimmen Sjolander, and David Haussler. Hidden Markov models in computational biology: Applications to protein modeling. Technical Report UCSC-CRL-93-32, University of California at Santa Cruz, 1993.

[73] Karen Kukich. Techniques for automatically correcting words in text. *ACM Computing Surveys*, 24(4):377–439, December 1992.

[74] Nicholas Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118:15–68, 1999.

[75] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence

data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289. Morgan Kaufmann, 2001.

[76] Corrin Lakeland. Part of speech tagging in statistical parsing. In M. Jason-Smith, A. Renaud, and T. Wright, editors, *Proceedings of the New Zealand Computer Science Research Students' Conference*, pages 138–139, April 2001.

[77] Leslie Lamport. *LaTeX: A Document Preparation System: User's Guide and Reference Manual*. Addison–Wesley, Massachusetts, USA, 1986.

[78] Christophe Laprun, Jonathan G. Fiscus, Sylvain Pajot, and John Garofolo. A practical introduction to ATLAS. In *Human Language Technology*, San Diego, California, USA, 24–27 March 2002.

[79] Steve Lawrence and C. Lee Giles. Searching the World Wide Web. *Science*, 280(5360):98–100, 1998.

[80] Steve Lawrence, C. Lee Giles, and Kurt Bollacker. Digital libraries and autonomous citation indexing. *IEEE Computer*, 32(6):67–71, 1999.

[81] Steve Lawrence and Lee Giles. Accessibility of information on the web. *Nature*, 400:107–109, 1999.

[82] T. R. Leek. Information extraction using hidden Markov models. Master's thesis, University of California, San Diego, USA, 1997.

[83] Douglas B. Lenat. CYC: A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, 1995.

[84] D. D. Lewis. Evaluating Text Categorization. In *Proceedings of Speech and Natural Language Workshop*, pages 312–318. Morgan Kaufmann, 1991.

[85] Ming Li, Xin Chen, Xin Li, Bin Ma, and Paul Vitányi. The similarity metric. In *Proceedings of the Fourteenth Annual Symposium on Discrete Algorithms*, pages 863–872, Baltimore, MD, 12–14 January 2003. ACM/SIAM.

[86] Ming Li and Paul M. B. Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, Berlin, Germany, 1993.

[87] Hokon Wium Lie and Bert Bos. *Cascading Style Sheets, level 1*. The World Wide Web Consortium (W3C), 17 December 1996.

[88] Jean loup Gailly. *gzip v1.0*, 1993. UNIX Manual page.

[89] Cláudio L. Lucchesi and Tomasz Kowaltowski. Applications of finite automata representing large vocabularies. *Software—Practice and Experience*, 23(1):15–300, January 1993.

[90] Helmut Lucke. Which stochastic models allow Baum-Welch training. *IEEE Transactions on Signal Processing*, 44(11):2746–2756, November 1996.

[91] Hans Peter Luhn. A statistical approach to mechanised encoding and searching of literary information. *IBM Journal*, pages 309–317, October 1957. Presented at the American Chemical Society meeting in Miami, April 8 1957.

[92] Hans Peter Luhn. *Modern Trends in Documentation*, chapter Auto-Encoding of Documents for Information Retrieval Systems, pages 45–58. Pergamon Press, London, England, 1959.

[93] Clifford Lynch. The battle to define the future of the book in the digital world. *First Monday*, 6(6), June 2001.

[94] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1994.

[95] Diana Maynard, Kalina Bontcheva, Horacio Saggion, Hamish Cunningham, and Oana Hamza. Using a text engineering framework to build an extendable and portable IE-based summarisation system. In *Proceedings of the ACL Workshop on Text Summarisation*, 2002.

[96] Andrew McCallum, Dayne Freitag, and Fernando Pereira. Maximum entropy Markov models for information extraction and segmentation. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 591–598. Morgan Kaufmann, 2000.

[97] Katherine J. McGowan. Efficient phrase hierarchy inference. Master's thesis, University of Waikato, Hamilton, New Zealand, 2002.

[98] Geoffrey J. McLachlan and Thriyambakam Krishnan. *The EM Algorithm and Extensions*. Probability and Statistics. John Wiley & Sons, Indianapolis, Indiana, USA, 1996.

[99] John G. McMahon and F. Jack Smith. A review of statistical language processing techniques. *Artificial Intelligence Review*, 12(5):347–391, 1998.

[100] Rodger J. McNab, Ian H. Witten, and S. J. Boddie. A distributed digital library architecture incorporating different index styles. In *Forum on Research and Technology Advances in Digital Libraries*, pages 36–45, Santa Barbara, California, 1998. IEEE Computer Society Press, Los Alamitos.

[101] F. Mittelbach, M. Goossens, Braams, Carlisle, and Rowley. *The LaTeX Companion*. Addison–Wesley, second edition, 2004.

[102] Alistair Moffat, Timothy C. Bell, and Ian H. Witten. Lossless compression for text and images. *International Journal of High-Speed Electronics*, 8(1):179–231, 1997. Special issue on Signal Compression.

[103] Alistar Moffat. Lossless compression. *The Computer Journal*, 40(2/3):65–66, 1997. Special Issue on Lossless Compression, Editorial.

[104] Mehryar Mohri and Michael Riley. Weighted determinization and minimization for large vocabulary speech recognition. In *Proceedings of the Fifth European Conference on Speech and Communication Technology*, pages 131–134, Rhodes, Greece, 1997.

[105] Manijya Rao Muddamalle. Natural language versus controlled vocabulary in information retrieval: A case study in soil mechanics. *Journal of the American Society for Information Science*, 49(10):881–887, 1998.

[106] Un Yong Nahm, Mikhail Bilenko, and Raymond J. Mooney. Two approaches to handling noisy variation in text mining. In *International Conference on Machine Learning Text Learning Workshop*, pages 18–27, Sydney, Australia, July 2002.

[107] Theodor H. Nelson. *Computer Lib*. Microsoft Press, Redmond, Washington, 1987. 'Dream Machines' by the same author in the same volume.

[108] Network Development and MARC Standards Office, Library of Congress. *MARC 21 Format for Bibliographic Data: Field List*, 1999 english edition, 1999.

[109] Craig G. Nevill-Manning, T. Reed, and Ian H. Witten. Extracting text from PostScript. Working Paper 97/10, Department of Computer Science, University of Waikato, April 1998.

[110] David M. Nichols, Kirsten Thomson, and Stuart A. Yeates. Usability and open-source software development. In Elizabeth Kemp, Chris Phillips, Kinshuk, and John Haynes, editors, *Symposium on Computer Human Interaction*, pages 49–54, Palmerston North, New Zealand, 6 July 2001. ACM SIGCHI NZ.

[111] Joseph Z. Nitecki. *Metalibrarianship: A Model for Intellectual Foundations of Library Information Science*. Published by Joanne Twining Williams at the Texas Woman's University, 1993. Volume 1 of the Nitecki Trilogy.

[112] Library of Congress. *Library of Congress Classification Outline*. Library of Congress, 1990.

[113] David D. Palmer and John D. Burger. Chinese word segmentation and information retrieval. In *Proceedings of the Symposium on Cross-Language Text and Speech Retrieval*. American Association for Artificial Intelligence Conference, 1997.

[114] Steven Pemberton, Daniel Austin, Jonny Axelsson, Tantek Celik, Doug Dominiak, Herman Elenbaas, Beth Epperson, Masayasu Ishikawa, Shin'ichi Matsui, Shane McCarron, Ann Navarro, Subramanian Peruvemba, Rob Relyea, Sebastian Schnitzenbaumer, and Peter Stark. *XHTML—The Extensible HyperText Markup Language—A Reformulation of HTML 4 in XML 1.0*. The World Wide Web Consortium, 1.0 edition, August 2002.

[115] Fuchun Peng, Fangfang Feng, and Andrew McCallum. Chinese segmentation and new word detection using conditional random fields. In *Proceedings of The 20th International Conference on Computational Linguistics*, Geneva, Switzerland, 23–27 August 2004.

[116] Fuchun Peng and Dale Schuurmans. Self-supervised Chinese word segmentation. *Lecture Notes in Computer Science*, 2189:238–248, 2001.

[117] Jay M. Ponte and W. Bruce Croft. USeg: a retargetable word segmentation procedure for information retrieval. In *Symposium on Document Analysis and Information Retrieval*, 1996.

[118] Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, February 1989.

[119] Martin Rajman and Romaric Besanon. Text mining knowledge extraction from unstructured textual data. In *Proceedings of the Sixth Conference of International Federation of Classification Societies*, Rome, Italy, 1998.

[120] R. Rosenfeld. A maximum entropy approach to adaptive statistical language modeling. *Computer, Speech and Language*, 10:187– 228, 1996.

[121] M. S. Ryan and G. R. Nudd. The Viterbi algorithm. Warwick Research Report RR238, Computer Science, University of Warwick, Coventry, England, 12 February 1993.

[122] Gerard Salton. *Automatic Text Processing: The transformation, Analysis, and Retrieval of Information by Computer*. Addison–Wesley, Massachusetts, USA, 1989.

[123] Dieter Scheffner and Johann-Christoph Freytag. The xml query execution engine (xee). Technical Report hub-ib-157, Humboldt University Berlin, Berlin, Germany 2001.

[124] Julian Seward. *bzip2 v1.0*. UNIX Manual page.

[125] Kristie Seymore, Andrew McCallum, and Ronald Rosenfeld. Learning hidden Markov model structure for information extraction. In *Proceedings of the Sixteenth National Conference on Articial Intelligence: Workshop on Machine Learning for Information Extraction*, pages 37–42, Orlando, FL, 1999.

[126] Claude Elwood Shannon and Warren Weaver. *The Mathematical Theory of Communication*. The University of Illinois Press, Urbana, Illinois, USA, 1964.

[127] Dmitry Shkarin. Ppm: One step to practicality. In James A. Storer and Martin Cohn, editors, *Proceedings of the 12th Data Compression Conference*, pages 202–210. IEEE Press, 2002.

[128] John Simpson, editor. *Oxford English Dictionary (Online edition)*. Oxford University Press, Oxford, England, 2002.

[129] Tony C. Smith. *N-gram Models of Agreement in Language*. PhD thesis, Computer Science Department, University of Waikato, Hamilton, New Zealand, 2001.

[130] C.M. Sperberg-McQueen and Lou Burnard. *Guidelines for Electronic Text Encoding and Interchange*. Association for Computers and the Humanities Association for Computational Linguistics and Association for Literary and Linguistic Computing, Chicago, USA and Oxford, England.

[131] James A. Storer and Martin Cohn, editors. *Data Compression Conference*, Snowbird, Utah, USA, 28–30 March 2000. IEEE.

[132] James A. Storer and Martin Cohn, editors. *Data Compression Conference*, Snowbird, Utah, USA, 27–29 March 2001. IEEE.

[133] William J. Teahan. *Modelling English Text*. PhD thesis, Department of Computer Science, University of Waikato, Hamilton, New Zealand, May 1998.

[134] William J. Teahan. An improved interface for probabilistic models of text. Technical report, School of Computer and Mathematical Sciences, The Robert Gordon University, 2000.

[135] William J. Teahan and John G. Cleary. Tag based models of English text. In Storer and Cohn [132], page 582.

[136] William J. Teahan and D. J. Harper. Combining PPM models using a text mining approach. In Storer and Cohn [131], pages 153–162.

[137] William J. Teahan, Yingying Wen, Roger McNab, and Ian H. Witten. A compression-based algorithm for Chinese word segmentation. *Computational Linguistics*, 26(3):375–393, September 2000.

[138] The Unicode Consortium. *The Unicode Standard—Worldwide Character Encoding*. Addison-Wesley, 1992.

[139] Michael B. Twidale, David M. Nichols, and Chris D. Paice. Browsing is a collaborative process. *Information Processing and Management*, 33(6):761–783, 1997.

[140] Andrew J. Viterbi. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Transactions on Information Theory*, 13:260–269, 1967.

[141] Andrew J. Viterbi and J. K. Omura. *Principles of Digital Communication and Coding*. McGraw–Hill, 1979.

[142] Ellen M. Voorhees and Donna K. Harman. Overview of the fifth text retrieval conference. In *Proceedings of the Fifth Text REtrieval Conference*, pages 1–28, Gaithersburg, Maryland, USA, 20–22 November 1996. National Institute of Standards and Technology.

[143] Ellen M. Voorhees and Donna K. Harman. Overview of the tenth text retrieval conference. In *Proceedings of the Tenth Text REtrieval Conference*, pages 1–17, Gaithersburg, Maryland, USA, 13–16 November 2001. National Institute of Standards and Technology.

[144] Yingying Wen. Text mining using HMM and PPM. Master's thesis, Computer Science, University of Waikato, Hamilton, New Zealand, July 2001.

[145] Ian H. Witten. Applications of lossless compression in adaptive text mining. In *Proc 2000 Conference on Information Sciences and Systems 2*, pages 13–18, Princeton, USA, March 2000.

[146] Ian H. Witten and Timothy C. Bell. The zero-frequency problem: Estimating the probablitiies of novel events on adaptive text compression. *IEEE Transaction of Information Theory*, 37(4):1085–1094, 1991.

[147] Ian H. Witten and Stefan Boddie. *Greenstone Digital Library Users Guide*. Digital Library Laboratory, University of Waikato, Hamilton, New Zealand, 2003.

[148] Ian H. Witten, Zane Bray, Malika Mahoui, and William J. Teahan. Using language models for generic entity extraction. In *Proceedings of the International Conference on Machine Learning Workshop on Text Mining*, 1999.

[149] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, San Francisco, 1999.

[150] Ian H. Witten, Rodger J. McNab, Steve Jones, Mark Apperley, David Bainbridge, and Sally Jo Cunningham. Managing complexity in a distibuted digital library. *IEEE Computer*, 32(2):6, Feburary 1999.

[151] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes — Compressing and Indexing Documents and Images*. Morgan Kaufmann, 2nd edition, 1999.

[152] Ian H. Witten, R. M. Neal, and John G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, June 1987.

[153] Ian H. Witten, Craig G. Nevill-Manning, Rodger J. McNab, and Sally Jo Cunningham. A public library based on full-text retrieval. *Communications of the Association for Computing Machinery*, 41(4):71–75, April 1998.

[154] Lauren Wood. Document object model (dom) level 1 specification. Technical report, World Wide Web Consortium (W3C), 1 October 1998.

[155] World Wide Web Consortium. *XSL Transformations (XSLT)*, version 1.0 edition, 16 November 1999.

[156] The World Wide Web Consortium (W3C). *RDF/XML Syntax Specification*, 10 February 2004.

[157] Dekai Wu and Pascale Fung. Improving chinese tokenization with linguistic filters on statistical lexical acquisition. In *Proceedings of the Fourth ACL Conference on Applied Natural Language Processing*, Stuttgart, Germany, 13–15 October 1994.

[158] Stuart Yeates. Automatic extraction of acronyms from text. In Stuart Yeates, editor, *Third New Zealand Computer Science Research Students' Conference*, pages 117–124, Hamilton, New Zealand, April 1999. University of Waikato.

[159] Stuart Yeates. *Colloquial Entropy Markup (CEM) Documentation*. University of Waikato, javadoc package edition, 2005.

[160] Stuart Yeates, David Bainbridge, and Ian H. Witten. Using compression to identify acronyms in text. In Storer and Cohn [131], page 582. A longer version of this appears as [161].

[161] Stuart Yeates, David Bainbridge, and Ian H. Witten. Using compression to identify acronyms in text. Working Paper 00/01, Department of Computer Science, University of Waikato, Hamilton, New Zealand, January 2000. A short version of this appears as [160].

[162] Stuart Yeates and Ian H. Witten. On tag insertion and its complexity. In Ah-Hwee Tan and Philip Yu, editors, *International Workshop on Text and Web Mining: Pacific Rim International Conference on Artificial Intelligence 2000*, pages 52–63, Melbourne, Australia, 28 August 2000.

[163] Stuart Yeates, Ian H. Witten, and David Bainbridge. Tag insertion complexity. In Storer and Cohn [132], pages 243–252.

[164] Stuart A. Yeates. The relationship between hidden markov models and prediction by partial matching models. In *Eighth Annual New Zealand Engineering and Technology Postgraduate Conference*, Hamilton, New Zealand, 30–31 August 2001. University of Waikato.

[165] Jeonghee Yi and Neel Sundaresan. Mining the web for acronyms using the duality of patterns and relations. In *Proceedings of the Second International Workshop on on Web Information and Data Management*, pages 48–52, Kansas City, Missouri, USA, November 2–6 1999. ACM.

[166] Steve J. Young. The HTK Hidden Markov model ToolKit: Design and philosophy. Technical Report CUED/F-INFENG/TR.152, Department of Engineering, Cambridge University, September 1994.

[167] Shilong Yu, Shuanhu Bai, and Paul Wu. Description of the Kent Ridge digital labs system used for muc-7. In *Proceedings of the Fifth Workshop on Very Large Corpora*, 1997.

[168] Xiaodan Zhu, Mu Li, Jianfeng Gao, and Chang-Ning Huang. Single character chinese named entity recognition. In *Second SIGHAN Workshop on Chinese Language Processing*, Sapporo, Japan, 7–12 July 2003.

[169] George Kingsley Zipf. *Human behavior and the principle of least effort; An introduction to human ecology*. Addison–Wesley, Massachusetts, USA, 1949. Republished 1965.

Appendix A
Corpora Samples

This appendix contains samples from each of the corpora discussed in Chapter 5 and used throughout this thesis. For reasons of space, documents have been abbreviated, an ellipsis marks a point at which content has been removed. All documents are presented after preparation rather than in the state in which they were received.

A.1 Computists' Corpus

The following is an issue from the Computists' corpus. The corpus is described in Section 5.1.

```
<issue>____________________________________________________________
AI Vol. 8, No. 1.1
IS <d>January 6, 1998</d>
CS <s>THE COMPUTISTS' COMMUNIQUE</s>
"Careers beyond programming."
1&gt;&gt; <o>NSF</o>news.
2&gt;&gt; Other funding.
3&gt;&gt; Career jobs.
____________________________________________________________
In the beginning the Universe was created. This has made
a lot of people very angry and been widely regarded as a bad move.
– <n>Douglas Adams</n>. [<s>QotD</s>, <d>16Oct97</d>.]
Greetings, Computists!
The <s>Computists' Communique</s>will now arrive three times
<d>per week</d>, on Tuesdays, Wednesdays, and Thursdays. Issues
will be shorter, for easy reading, and may vary a bit in length.
Part of each Wednesday issue will be a table of contents for
<d>that day</d>'s CAJ jobs digest. (You can request the digest issue
if you see an interesting opportunity.) I'm reducing the number
of <d>publication weeks</d>to 40 (or 120 issues!), to give me more time
for Web work and other activities. That means there will be
about <d>one week</d><d>each month</d>with no <s>Communique</s>s, usually
with
a holiday or at the end of <d>the month</d>. All to serve you better,
of course, but do get in touch with me if you have suggestions
about the changes.
Membership fees will hold steady at <d>last year</d>'s level,
but with a new "departmental rate" for groups of up to five
```

*participants. At <m>$195</m>per year (or half of that outside the <l>US</l>),
it should be attractive to lab directors and other group leaders.
(Please circulate copies of the <s>Communique</s>to the appropriate
people. They can write to me or visit &lt;<u>http://www.computists.com</u>&gt;
to check out the service.) Members may offer <d>two-month</d>
free trials to friends, or <d>three-month</d>free trials
(excluding their own dues) for groups.
My wife <n>Lily</n>will be taking over some of the renewal
billing communication, and will be getting in touch with you.
The captain is on holiday, but his "cool job of <d>the week</d>"
should return in <d>a week</d>or two. (Sometimes he just doesn't
find a cool enough job.) We're taking care of business,
so have a fun and prosperous <d>new year</d>!
1&gt;&gt; <o>NSF</o>news:
<o>NSF</o>'s Awards for the Integration of Research and Education
at Baccalaureate Institutions program will make 10-20 awards of
up to <m>$500K</m>. Eligibility is restricted to Carnegie Classification
Baccalaureate I and II institutions and Specialized Technical
institutions that award only baccalaureate degrees. Deadlines
are <d>04Feb98</d>for letters of intent, <d>17Mar98</d>for preliminary
applications, and <d>17Jun98</d>for full applications.
&lt;<u>http://www.nsf.gov/od/osti</u>&gt;. [<s>grants</s>, <d>23Dec97</d>.]
<o>NSF</o>'s CISE and ENG directorates have a Combined
Research-Curriculum Development (CRCD) Program to support
dynamic, relevant engineering and CS/IS education.
<d>31Mar98</d>deadline.
&lt;<u>http://www.nsf.gov/cgi-bin/getpub?nsf9838</u>&gt;.
[<n>Maria Zemankova</n>&lt;<e>mzemanko@nsf.gov</e>&gt;, <s>dbworld</s>,
<d>30Dec97</d>.]
...
I have been poor and I have been rich. Rich is better.
– <n>Sophie Tucker</n>, American singer. [<s>DailyQuote</s>,
<d>02Jan98</d>.]
</issue>*

A.2 Bibliography Corpus

The following is a bibliography from the bibliography corpus. The corpus is described in Section 5.2.

```
<bibliography xsi:schemaLocation="http://www.greenstone.org"
filename="/research/say1/bib/tmp_bib_files/graphics/2748.bib"><plain><bibproc>
<p></p>References
<p></p>[1] <bibbody><article><author><name><first>Till
B.</first><last>Anders</last></name>and
<name><first>Wolfgang</first><last>Jachmann.</last></name></author><title>Cross
sections with polarized spin-1over2 particles in terms of helicity
amplitudes.</title><journal><emphasis>Journal of Mathematical
Physics,</emphasis></journal><volume>24</volume>(<number>12</number>):<pages>2847-
2854,</pages><date><month>December</month><year>1983</year></date>.</article></bibbody>
<p></p>[2] <bibbody><article><author><name><first>V.
G.</first><last>Bagrov,</last></name><name><first>V.
V.</first><last>Belov,</last></name>and <name><first>I.
M.</first><last>Ternov.</last></name></author><title>Quasiclassical
trajectory-coherent states of a particle in an arbitrary electromagnetic
field.</title><journal><emphasis>Journal of Mathematical
Physics,</emphasis></journal><volume>24</volume>(<number>12</number>):<pages>2855-

2859,</pages><date><month>December</month><year>1983</year></date>.</article></bibbody>
. . .
<p></p>[25] <bibbody><article><author><name><first>W.
M.</first><last>Zheng.</last></name></author><title>The Darboux
transformation and solvable double-well potential models for Schrodinger
equations.</title><journal><emphasis>Journal of Mathematical
Physics,</emphasis></journal><volume>25</volume>(<number>1</number>):<pages>88-
90,</pages><date><month>January</month><year>1984</year></date>.</article></bibbody>
<p></p>Page <pagematter>2 </pagematter>
</bibproc></plain></bibliography>
```

If the output is indented to show the full structure, it appears as:

```
<bibliography xsi:schemaLocation=”http://www.greenstone.org” filename=”/research/say1/bib/tmp_bib_files/graphics/2748.bib”>

 <plain>
  <bibproc>

    <p> </p> References

    <p> </p> [1]
    <bibbody>
      <article>
        <author>
      <name>
        <first> Till B.</first>
        <last> Anders</last>
      </name>  and
      <name>
        <first> Wolfgang</first>
        <last> Jachmann.</last>
      </name>
    </author>
    <title> Cross
sections with polarized spin-1over2 particles in
terms of helicity amplitudes.</title>
    <journal>
      <emphasis> Journal of Mathematical
Physics,</emphasis>
    </journal>
    <volume> 24</volume> (
    <number> 12</number> ):
    <pages> 2847-2854,</pages>
    <date>
      <month> December</month>

      <year> 1983</year>
    </date> .</article>
  </bibbody>

  <p> </p> [2]
  <bibbody>
   <article>
    <author>
      <name>
        <first> V. G.</first>
        <last> Bagrov,</last>
      </name>
      <name>
        <first> V. V.</first>
        <last> Belov,</last>
      </name>  and
      <name>
        <first> I. M.</first>
        <last> Ternov.</last>
      </name>
    </author>

    <title> Quasiclassical trajectory-coherent states of
a particle in an arbitrary electromagnetic field.</title>

    <journal>
      <emphasis> Journal of Mathematical Physics,</emphasis>
```

```
          </journal>
          <volume> 24</volume> (
          <number> 12</number> ):
          <pages> 2855-
2859,</pages>
          <date>
            <month> December</month>
            <year> 1983</year>
          </date> .</article>
      </bibbody>


...

      <p> </p> [25]
      <bibbody>
        <article>
          <author>
            <name>
              <first> W. M.</first>
              <last> Zheng.</last>
            </name>
          </author>
          <title> The Darboux transformation
and solvable double-well potential models for
Schrodinger equations.</title>
          <journal>
            <emphasis> Journal of Mathematical
Physics,</emphasis>
          </journal>
          <volume> 25</volume> (
          <number> 1</number> ):
          <pages> 88-90,</pages>
          <date>
            <month> January</month>
            <year> 1984</year>
          </date> .</article>
      </bibbody>

      <p> </p> Page
      <pagematter> 2

</pagematter>
</bibproc>
  </plain>
</bibliography>
```

A.3 Segmentation Corpus

The following is a single document from the segmentation corpus. Whitespace appearing here is a side-effect of layout, the only whitespace in the original file is a terminal EOL. The corpus is described in Section 5.3.

```
<document>
<word> &#20551;</word> <word> &#26230;&#21326;</word> <word>
&#39277;&#24215;</word> <word> &#20030;&#34892;</word> <word>
&#39041;&#22870;</word> <word> &#20856;&#31036;</word> <word>
&#65292;&#21040;&#24213;</word> <word> &#30495;&#30456;</word> <word>
&#22914;&#20309;</word> <word> &#21602;</word> <word>
&#65311;&#19968;</word> <word> &#12289;</word> <word>
&#36164;&#26684;</word> <word>
&#65306;&#19969;&#32903;&#20013;</word> <word>
&#38498;&#22763;</word> <word> &#21363;</word> <word> &#22240;</word>
<word> &#39318;&#20808;</word>
...
</document>
```

A.4 Reuters’ Corpus

The following is a single document from the Reuters’ corpus. The corpus itself is described in Section 5.4.

<document> <NNP> PDCP</NNP> <NNP> Development</NNP> <NNP> Bank</NNP> <VBD> said</VBD> <IN> on</IN> <NNP> Thursday</NNP> <PRPSTRING> its</PRPSTRING> <NN> board</NN> <VBD> approved</VBD> <DT> the</DT> <NN> issue</NN> <IN> of</IN> <CD> one</CD> <CD> billion</CD> <NN> pesos’</NN> <JJ> worth</JJ> <IN> of</IN> <JJ> convertible</JJ> <JJ> preferred</JJ> <CD> shares.</CD>

<DT> The</DT> <NNS> proceeds</NNS> <IN> of</IN> <DT> the</DT> <NN> issue</NN> <MD> will</MD> <VB> fund</VB> <NN> lending</NN> <NN> operations,</NN> <NN> computerisation,</NN> <CC> and</CC> <VBG> refurbishing</VBG> <IN> of</IN> <NN> branch</NN> <NN> offices,</NN> <PRP> it</PRP> <JJ> said.</JJ>

...

</document>