Text Mining Using HMM and PPM

A thesis

submitted in partial fulfilment

of the requirements for the degree

of

Master of Philosophy

at the

University of Waikato

by

Yingying Wen

Department of Computer Science



Hamilton, New Zealand

November 6, 2001

Abstract

Text mining involves the use of statistical and machine learning techniques to learn structural elements of text in order to search for useful information in previously unseen text. The need for these techniques have emerged out of the rapidly growing information era. Token identification is an important component of any text mining tool. The accomplishment of this task enhances the function of diverse applications involving searching for patterns in textual data.

Several different identification methods have been reported in the literature. HMMs and PPM models have been successfully used in language processing tasks. They have also been applied separately to learning-based token identification. Most of the existing systems are domain- and language-dependent.

In this thesis, we implement a system that bridges the two well known methods through words new to the identification model. The system is fully domain- and language-independent. No changes of code are necessary when applying to other domains or languages. The only thing required is an annotated corpus.

The system has been tested on two corpora and achieved an overall F-measure of 76.59% for TCC, and 69.02% for BIB. This is not as good as would be expected from a system which includes language-dependent components. However, our system is more generalized. The identification of date has the best result, 73% and 92% of correct tokens are identified respectively. The system also performs reasonably well on people's name with correct tokens of 68% for TCC, and 76% for BIB.

Acknowledgements

During the time of my MPhil. study, I have been so lucky to have had a huge amount of help in academic, financial and personal from a number of people.

First and foremost, I would like to thank my chief supervisor, Ian Witten, for acceptance and introducing me to the topic of text mining. I will never forget the time when Ian tried to explain and solve problems by examples. He is extremely good on explanation. I was also amazed by how much patience he had while reading through my draft. I felt so sorry for always forgetting to do a spell check.

Secondly, I would like to thank Malika Mahoui, one of my supervisors, for encouraging me academically and personally. I certainly enjoyed our conversations.

I would also like to thank Mark Apperley, head of the department, for providing me with a Graduate Assistant position and to Phil Treweek, Tim Elphick and Nilesh Kanji for help during this work.

My great thanks also go to Stuart Yeates for the original version of bibliography corpus and for helps in other aspects. I am indebted to the many helpful suggestions and comments made by Matt Jones, Katherine McGowan, John Cleary, Yong Wang and Margaret Brown.

I also owe many thanks to those who answered both my technical and language questions, those who discussed the project during our meetings, and those who encouraged me by saying "Finished?".

Finally, my deepest thanks go to my parents for their unconditional love, and to my family for the understanding.

This thesis would have been much different (or would not exist) without these people.

Contents

Ał	ostrac	:t	ii
Ac	cknow	vledgements	iii
Li	st of l	Figures	ix
Li	st of [Fables	X
1	Intr	oduction	1
	1.1	Text mining	2
	1.2	Text mining versus data mining	3
	1.3	Token identification	5
		1.3.1 Task and definition	6
		1.3.2 About tokens	7
		1.3.3 Applications	7
	1.4	Corpora	10
		1.4.1 TCC corpus	11
		1.4.2 Bibliography corpus	13
	1.5	Thesis statement 1	15
	1.6	Thesis outline	16
2	Rela	ated work	18
	2.1	Different methods	18
		2.1.1 Hand-crafted systems	20
		2.1.2 Decision trees	21

		2.1.3	Maximum entropy models	22
		2.1.4	НММs	23
		2.1.5	PPM models	25
	2.2	Unkno	wn token	26
	2.3	Model	smoothing	28
	2.4	Summ	ary	30
3	Info	rmation	n extraction with HMMs and PPM	32
	3.1	Hidden	n Markov model	33
		3.1.1	Model definition	33
		3.1.2	Applying HMMs to token identification	37
	3.2	Decod	ing	39
	3.3	Estima	ating the parameters of an HMM	43
	3.4	Trainii	ng the HMM	46
	3.5	The PI	PM model for text compression	48
		3.5.1	PPM model	48
		3.5.2	Estimating probabilities in PPM	52
	3.6	Unkno	wn word handling	53
		3.6.1	PPM models for unknown words	53
		3.6.2	Unified probability for unknown words	54
	3.7	Smoot	hing the probabilities	55
4	Exp	eriment	tal evaluation	57
	4.1	Measu	ring	58
	4.2	Applic	cation to the TCC corpus	61
	4.3	Defect	is in the corpus	63
		4.3.1	Ambiguities	64
		4.3.2	Omissions	65
		4.3.3	Errors	66
		4.3.4	Effect of the imperfect training data	67
	4.4	Applic	cation to the corrected TCC corpus	68

	4.5	Exploring impact of PPM	70
	4.6	Effect of the quantity of training data	72
	4.7	Effect of model smoothing method	74
	4.8	Application to bibliographies	75
5	Con	clusion	79
	5.1	Key findings	79
	5.2	Discussion	80
	5.3	Future work	82
Bi	bliog	caphy	84

List of Figures

1.1	A fragment of text documents.	5
1.2	Example of application in digital library.	8
1.3	Example of machine translation.	9
1.4	Example of TCC corpus. (a) A section of original text. (b) The	
	same section marked-up.	12
1.5	Generic data items extracted from one issue of The Computists'	
	Weekly	14
1.6	Example of BIB corpus. (a) A section of bibliography. (b) The	
	same section marked-up. (c) Modified marked-up version	15
31	Example of HMM	34
3.2	Example of partially connected HMM	35
3.3	Viterbi algorithm interpretation.	41
3.4	The result of the first two steps of the Viterbi algorithm running on	
	sequence <i>acba</i> .	42
3.5	The result of the third step of the Viterbi algorithm running on the	
	sequence <i>acba</i> .	43
3.6	The result of the final step of the Viterbi algorithm running on the	
	sequence <i>acba</i> .	44
3.7	The interpretation of generating sequence <i>acba</i>	44
3.8	A HMM obtained from the training data.	47
	č	
4.1	Examples of ambiguity. (a) Hand mark-up. (b) System mark-up	64

4.2	Examples of omission. (a) Tokens are left unmarked. (b) Tokens	
	are not marked up for all occurrences.	65
4.3	Examples of error. (a) Hand mark-up. (b) System mark-up	67
4.4	Detailed result for five TCC test files using the corrected corpus and	
	the model of HMM+PPM	69
4.5	Effect of the amount of training data	73
4.6	Detailed result for 100 bibliographic entries and 5 TCC issues	76
4.7	Example of marking error. (a) Hand mark-up. (b) System mark-up	77

List of Tables

A relational database with two relations	3
Rules induced by a data mining process from the data in Table 1.1	4
Token types with labels and examples	11
Token types in BIB corpus.	13
Statistics of the 10 most frequent words in the Brown Corpus	26
Percentage of word that occur 1 to 5 times in the Brown Corpus. $\ . \ .$	27
PPM model after processing the string <i>tobeornottobe</i>	50
Effect of context and current character with order-2 PPM model	51
Methods for zero frequency problems	52
Statistics of the training data in the TCC corpus	62
Result for five TCC test files using the model of HMM+PPM	63
Result for five TCC test files using the corrected corpus and the	
model of HMM+PPM.	68
Average results of 5 TCC files using PPM and unified probability	
for unknown words	71
Average results from different model smoothing methods	74
Average result for 100 bibliographic entries and 5 TCC issues	75
	A relational database with two relations

Chapter 1 Introduction

In today's information age, we have witnessed and experienced an ever increasing flood of information. The Internet makes available a tremendous amount of information, on an amazing variety of topics, that has been generated for human consumption. According to Lawrence and Giles (1999), 800 million web pages were publicly indexable in February 1999. However, today's number is far greater. For example the search engine Google claims 1, 346, 966, 000 web pages as of June 2001.¹

Unfortunately, the hundreds of millions of pages of information make it difficult to find information of interest to specific users or useful for particular purposes. The amount of text is simply too large to read and analyze easily. Furthermore, it changes constantly, and requires ongoing review and analysis if one wants to keep abreast of up-to-date information. Working in this ever-expanding sea of text becomes extremely difficult.

In the past, much technological effort has been focused on computer tools that improve the amount of storage and retrieval of information. This results in instant access to far more information than humans can possibly handle. We have all experienced frustration when surfing the web with traditional search engines. Immense

¹http://www.google.com

lists of documents are returned. This makes it extremely difficult for users to find information of interest.

1.1 Text mining

Text mining is about the use of statistical and machine learning techniques to learn structural elements of text in order to search for useful information in previously unseen text. It is an extension of data mining, which finds information in structured database, to the far less structured domain of free text. In other words, it looks for patterns in text while data mining looks for patterns in data (Witten and Frank, 2000). Text mining may be defined as the process of automatically analyzing text to extract information that is of interest to a particular user or useful for a particular purpose. It represents a new perspective to the common problem of finding relevant information.

Text mining is particularly relevant today, because the enormous amount of information is mostly available in text format. According to Tan (1999), approximately, 80% of information of an organization is stored in unstructured textual forms: reports, e-mails, meeting minutes and so on, although other forms do exist, such as structured databases, videos and images. Text mining is used to create an environment that helps make sense of information that is embedded in text documents, either within an organization or outside it. Example reasons for using text mining include: creating links between objects that mention the same event such as a person's name, extracting metadata for a modern digital library, uncovering a "narrative" in an unstructured mass of text, exploring how a market is evolving, and looking for more ideas or relations.

Previous work has proven that text mining is possible (e.g., Sekine, 1998; Borthwick et al., 1998; Bikel et al., 1999; Bray, 1999). Using text mining tools, people are able to explore items which consist of one or more words, such as a person's name and a

name of location, in a large collection of documents without having to look through a great number of files, and to understand the given text in order to extract useful information from it.

However, text mining is difficult, because text can be in many different styles, such as names of people, names of organization, names of locations, phone numbers, fax numbers, money amount, email addresses, web addresses, text tables, captions, lists, bibliographies, and more. This makes it difficult to design automated systems to extract information of interest.

1.2 Text mining versus data mining

Data mining works on structured data and extracts information for further use from such data. In other words, data mining finds patterns and associations between fields in a relational database—a set of relations. The result of data mining is a rule (or set of rules) that allows people to predict future values of variables; find new associations between events; or classify data into clusters of related values. Some data mining systems are limited to work with numerical data, while others use any structured data—including categorical, time series, and boolean data.

Person	Age	Sex	Income	Customer
Ann Smith	32	F	10 000	yes
Joan Gray	53	F	1 000 000	yes
Mary Blythe	27	F	20 000	no
Jane Brown	55	F	20 000	yes
Bob Smith	30	Μ	100 000	yes
Jack Brown	50	Μ	200 000	yes
Married-To				
Husband Wife				
Bob Smith	Ann Sn	nith		
Jack Brown	Jane Br	rown		

Potential-Customer

Table 1.1: A relational database with two relations.

Induced Rules

IF Income(Person) \geq 100 000 THEN Potential-Customer(Person)	
IF Sex(Person) = F AND Age(Person) \geq 32 THEN Potential-Customer(Person)
IF Married(Person, Spouse) AND Income(Person) ≥ 100 000	
THEN Potential-Customer(Spouse)	
IF Married(Person, Spouse) AND Potential-Customer(Person)	
THEN Potential-Customer(Spouse)	

Table 1.2: Rules induced by a data mining process from the data in Table 1.1.

For example, Table 1.1 (Dzeroski, 1996) is a relational database, containing two relations: Potential-Customer and Married-To. Rules induced by a data mining process are shown in Table 1.2 (Dzeroski, 1996). The first set of rules is to distinguish between persons who are potential customers and those who are not. The second set of rules defines the relation Potential-Customer in terms of itself and the relation Married-To. The process of rule extraction relies strongly on the structure of the relational database.

However, as mentioned above, a large portion of information appears in textual form—unstructured data. Therefore, techniques that extract information from such data become necessary. Text mining is analogous to data mining in that it uncovers relationships in information. However, unlike data mining, text mining works on information stored in a collection of text documents.

Text documents, the raw material of text mining, are unstructured, because they contain no predefined relationships between words or phrases when they are stored on a computer. Given the text document as shown in Figure 1.1, one can hardly find relations between names of people, "Guy Kawasaki", "William A. Law", "Hewlett" and "Packard", or dates, "12Jul98" and "14Jun98", and their surrounding words.² In contrast, relational databases have a schema that describes the meaning of the data source. However, text mining aims at finding all patterns of interest in such text data.

²Here and later, textual examples are all taken from the available corpora.

Somewhat related, there was some interesting commentary about Guy Kawasaki in the San Jose Mercury News "West" magazine insert on 12Jul98, following up on a 14Jun98 profile article. Kawasaki has written books about software evangelism (initially at Apple), and is now working a "garage.com" consultancy to help small companies grow and then get venture funding. Reader William A. Law notes the irony, that "Hewlett and Packard actually *built* something in their garage." Kawasaki is a salesman looking for a product he can market – preferably a success-bound company that doesn't need much evangelizing. He lives in a mansion in Atherton; drives a Mercedes, Suburban, and Land Rover; dines at Il Fornaio; and worries that he hasn't "hit the multimillion-dollar big time" yet and that he doesn't have time to be with his kids. Readers suggest that he sell the physical assets, move to a tract house in San Jose, and "kick back with the kids." He's had enough success for most people, and there's more to life than social networking to find the next big deal. Silicon Valley has too many people on this "self-indulgent treadmill."

Figure 1.1: A fragment of text documents.

In conclusion, text mining is similar to data mining in terms of dealing with large volumes of data, and both fall into the information discovery area. The difference between them is that text mining is looking for patterns in unstructured text data, whereas data mining extracts patterns from structured data. Data mining is more mature, while text mining is still in its infancy.

1.3 Token identification

Token identification is an important component of any text mining tool. It involves identifying certain kinds of terms in text, such as names of people and locations, whether or not these items occur in earlier text. A token consists of one or more words. The task of token identification is to match tokens to their semantic class—*type*. Example classes are name, location and organization.

Several researchers have reported token identification systems (e.g. Appelt et al., 1993; Sekine, 1998; Borthwick et al., 1998; Bikel et al., 1999; Bray, 1999), including hand-crafted systems and ones that use machine learning. Many of them rely on a specific domain, and extra work needs to be done to apply them to other domains or languages. The use of machine learning for token identification not only saves human development effort, but also affects retargetability and generality. Retargetability means that applying an existing technique to a new domain should not require code modification; at most, some feature modification is required. In terms of generality, the system should be able to handle a wider range of domains and be less language– specific.

1.3.1 Task and definition

Token identification involves finding, in unseen text, all instances of tokens whose type is determined by training text. It takes place in two steps: first find tokens of interest and then assign a type to each of them. Tokens of interest are distinguished by inserting start token and end token labels before and after each token, where the label itself represents the token type. This process is called mark-up.

The token identification system produces a single, unambiguous type for any relevant string in the text. The only insertions allowed during processing are start and end labels, which are placed in angle brackets. No extra symbols are to be inserted, like white spaces, carriage returns and punctuations.

The text after mark-up has the following form:

<token type>text-string</token type>.

For example:

<n>Michael Hucka</n> recommends the following ...

where n is a tag that represents a person's name.

The need to identify token types has two aspects: the identification of known tokens and the discovery of new tokens. Some systems as described in Chapter 2, rely on lists of tokens of different types. To create these lists requires significant effort. Many applications operate without such a resource. Therefore, tokens need to be discovered in the text and assigned the types they refer to.

1.3.2 About tokens

Among the different types of tokens, some are more important than others. The names of people, locations and organizations are particularly important for extraction systems. Typically, one wants to extract events, properties, and relations about some particular objects, and the objects are usually identified by their names.

Some token types are easy to identify while others are more difficult. For example, email addresses and URLs can be represented by a few simple patterns. On the other hand, although names are important, they present some difficult problems for identification. For example, token-type ambiguity is quite common in names, because places are named after famous people, such as "Washington D.C.", and organizations are named after their owners or locations, such as "Ford Motors" and "New York University". A naming convention is followed by most people, however, there is no restriction on what words may comprise a name. For people who enjoy having unconventional and eccentric names, any word can be part of a name. The name appearing in the following text fragment (McDonald, 1996) is just such a example,

... Her name was equally preposterous. April Wednesday, she called herself, and her press card bore this out.

1.3.3 Applications

Token identification is crucial for text mining and information extraction, and is also useful as a preprocessing step for other applications, such as digital libraries,

7



Figure 1.2: Example of application in digital library.

machine translation, information retrieval, and natural language processing.

Digital library

Today's digital libraries, a burgeoning information organizing technology, normally keep thousands of documents. For instance, the New Zealand Digital Library ³ allows users to access collections by different ways, such as browsing by title and author. It would be ideal if names of people in a document could be identified, then

³http://nzdl.org

Name in English	Mrs. April Wednesday
Transliterated in Chinese	爱波柔-温斯得夫人
Converted to PinYin	Ai Bo Rou Wen Si De 夫人
Translated as regular words	四月-星期三夫人

Figure 1.3: Example of machine translation.

links could be inserted automatically to other documents that mention the same names, for example, documents written by the identified persons.

Figure 1.2 is an illustration of how token identification can be used in a digital library, featuring the New Zealand Digital Library. A person's name, "Charles Dickens", appears in a document shown in the rear image. Ideally, it would be linked to all documents written by the same person as shown in the front image.

By the same principle, links could be inserted to documents that mention the same organization, location, email address, URLs and so on. This makes browsing a digital library more flexible and convenient.

Machine translation

In the application of machine translation, token identification is used to create translations of unknown words or for disambiguation. For example, if a machine translation system encounters "Mrs. April Wednesday" in the input text and the supporting component, token identification, identifies that this string represents a person's name, it should not try to translate "April" and "Wednesday", but translate the preceding title to the appropriate personal title in the target language and leave the name itself intact.

However, if the target language is Chinese, the name can be transliterated or converted to PinYin, rather than being translated as regular words. Figure 1.3 shows the translation samples, where the transliterated name can be variant because the transliteration depends on the pronunciations in the original language. However, exactly matching pronunciations in Chinese do not always exist. The transliteration is then determined by the person in charge. In addition, tones in Chinese also result in different characters. The family name translated as regular words has never been used. And the name in PinYin is just another representation of the transliteration.

Information retrieval

For the same reason, an information retrieval system should not expand words in name "Hunter Wood", organization "Tiger Dictionary", location "Telegraph Hill" and so on, to all of their morphological variants.

Natural language processing

Creating an annotated corpus is a common requirement in natural language processing, and involves tedious labour. According to an experiment performed at BBN Systems and Technologies (Bikel et al., 1999), annotating a text of 650,000 words, which is about two-thirds the length of one edition of the Wall Street journal, takes an inexperienced annotator 27 days, and 16 days for an experienced one. Token identification can be used to annotate text automatically, making it easier to produce a labeled corpus. Conversely, a labeled corpus can be very useful in the development of token identification systems.

1.4 Corpora

Two different corpora have been used in the experiments described in this thesis. The first is based on The Computists' Weekly—formerly known as The Computists' Communique (TCC).⁴ This is an on-line weekly publication of Computists Interna-

⁴http://www.computists.com

Token type (label)	Examples
Dates/time periods (d)	August, 15Aug98
Email addresses (e)	amin@cse.unsw.edu.au
Fax numbers (f)	+44 161 275 6204 Fax
Phone numbers (h)	650-941-0336
Locations (1)	Beaverton, Quebec
Sums of money (m)	\$1K, \$100
People's names (n)	Randall B. Caldwell, Vernon Ehlers
Organizations (o)	NSF, Santa Fe Institute
Sources, journals, book series (s)	Genetic programming book series
URLs (u)	http://www.elsevier.nl/locate/parco

Table 1.3: Token types with labels and examples.

tional, a professional association for artificial intelligence, information science, and computer science researchers. It covers many topics: artificial intelligence, neural networks, genetic algorithms, machine learning, logic, fuzzy logic, natural language processing (NLP), machine translation, computational linguistics, information retrieval, expert systems and so on. A full-text indexing of issues from April 1991 to the present is available from the New Zealand Digital Library, where one can search for particular words that appear in the text, and access publications by title and by date as well.

The second corpus comes from The Collection of Computer Science Bibliographies.⁵ This is a collection of bibliographies of scientific literature in computer science from various sources, covering most aspects of computer science. The collection itself contains more than one million references, mostly from journal articles, conference papers and technical reports.

1.4.1 TCC corpus

The Computists' Weekly or TCC corpus is a collection of 38 issues selected randomly from an archive in the New Zealand Digital Library. Ten different types of tokens in the collection are manually marked up with XML (*eXtensible Mark-up*

⁵http://liinwww.ira.uka.de/bibliography/index.html

(a) Microsoft is planning a new campus in Mountain View, CA. "The best expertise today not only lives in Silicon Valley, it wants to stay living in Silicon Valley." [AP, 07Aug98. EduP.] (New facilities often mean job opportunities...) Mindjack is an online magazine about technology, culture, and technosocial issues, available monthly at <<http://www.mindjack.com>. [Donald Melanson] <<donald@mindjack.com>, newjour, 11Jun98.] (b) <o>Microsoft</o> is planning a new campus in <l>Mountain View, CA</l>. "The best expertise today not only lives in <1> Silicon Valley</1>, it wants to stay living in Silicon Valley." [<s>AP</s>, <d>07Aug98</d>. <s>EduP</s>.] (New facilities often mean job opportunities...) <s>Mindjack</s> is an online magazine about technology, culture, and technosocial issues, <d>available month</d>ly at <<<u>http://www.mindjack.com-</u>>. [<n>Donald Melanson</n> <<<e>donald@mindjack.com </e>>, <s>newjour</s>, <d>11Jun98</d>]

Figure 1.4: Example of TCC corpus. (a) A section of original text. (b) The same section marked-up.

language) style tags. They will also be referred as *classes*. Table 1.3 lists all the types with corresponds labels and examples for each type.

Figure 1.4 shows a fragment of the original text and the annotated corpus. Labels appear only at the beginning and end of tokens in the ten classes. In fact, every word corresponds to a class. Words that do not belong to any of the ten classes are called *plain text*, and they are left unmarked. Therefore, the class sequence of the first line in Figure 1.4b is:

where each pair of brackets, along with the label of class, corresponds to a word in the text.

There are two things in the figure that need to be mentioned. The first is that one appearance of "Silicon Valley" in the first fragment is not marked in the corpus. This is an error and will be discussed in Section 4.3. The second is the way that email address and URL are represented. As shown in the figure, these both start with two angled open brackets and end up with one angled close bracket in the original text. These brackets are retained in the corpus.

Although the TCC corpus is a free text collection, it contains many semi-structured items. Figure 1.5 (Witten and Frank, 2000) lists all tokens in just one issue of The Computists' Weekly.

1.4.2 Bibliography corpus

The bibliography or BIB corpus is a collection of 2400 bibliographies from The Collection of Computer Science Bibliographies. It has been formatted as free text, with tags placed around all the tokens. There are several types of tokens, such as last name, first name, title, date, year, pages and number in the marked-up text (Yeates et al., 2001).

Original marker	Token type (label)
publisher	publisher (b)
pages	page (g)
address	location (l)
title	title (t)
journal	source (s)
booktitle	
date	
month	date (d)
year	
name	
first	name (n)
last	
school	
organization	organization (o)
institution	

Table 1.4: Token types in BIB corpus.

Because the undertaken study does not consider hierarchical structure, the corpus has been modified. Figure 1.6 shows two entries in different formats, where (b) is the original marked-up records of (a), and (c) is the modified version in which all unrelated tags are deleted and the remainders are changed to their corresponding labels. Table 1.4 shows the changes from the original tags to the remaining labels.

People's names (n)	Dates/time periods (d)	Email a
Al Kamen	30Jul98	amin
Barbara Davies	31Jul98	bona
Bill Park	02Aug98	book
Bruce Sterling	04Aug98	cbd-s
Ed Royce	05Aug98	erric
Eric Bonabeau	07Aug98	espaa
Erricos John Kontoghiorghes	08Aug98	herm
Heather Wilson	09Aug98	koza
John Holland	10Aug98	koza
John R. Koza	11Aug98	kung
Kung-Kiu Lau	13Aug98	1.jain
Lakhmi C. Jain	14Aug98	mzer
Lashon Booker	15Aug98	rbcal
Lily Laws	August 18, 1998	ricos
Maria Zemankova	01Sep98	s.l.ro
Mark Sanford	15Sep98	scisti
Martyne Page	15Oct98	simo
Mike Cassidy	31Oct98	thcla
Po Bronson	10Nov98	tolk
Randall B. Caldwell	01Dec98	zorai
Robert L. Park	01Apr99	
Robert Tolksdorf	Nov97	Organi
Sherwood L. Boehlert	Jul98	ACM
Simon Taylor	Aug98	Aust
Sorin C. Istrail	Mar99	Bure
Stewart Robinson	August	CRC
Terry Labach	July	Case
Vernon Ehlers	Spring 1999	Frau
Zoran Obradovic	Spring 2000	Ida S
	1993-4	Kluv
Sums of money (m)	1999	NSF
\$1K	120 days	Nohi
\$24K	eight years	Oreg
\$60	eight-week	Perm
\$65K	end of 1999	Ranc
\$70	late 1999	Santa
\$78K	month	UOk
\$100	twelve-year period	UTre

Sources, journals, book series (s)

Autonomous Agents and Multi-Agent Systems Journal Commerce Business Daily (CBD) Computational Molecular Biology Series DAI-List ECOLOG-L Evolutionary Computation Journal Genetic Programming book series IRList International Series on Computational Intelligence J. of Complex Systems J. of Computational Intelligence in Finance (JCIF) J. of Symbolic Computation (JSC) J. of the Operational Research Society Parallel Computing Journal Pattern Analysis and Applications (PAA) QOTD SciAm TechWeb WHAT'S NEW Washington Post Wired comp.ai.alife comp.ai.doc-analysis.ocr comp.ai.genetic comp.ai.neural-nets comp.simulation dbworld sci.math.num-analysis sci.nanotech

Email addresses (e)

@cse.unsw.edu.au beau@santafe.edu er@mitre.org support@gpo.gov os.kontoghiorghes@info. unine.ch a@soc.plym.ac.uk nes@iway.fr @cs.stanford.edu @genetic-programming.org g-kiu@cs.man.ac.uk 1@unisa.edu.au manko@nsf.gov ldwell@delphi.com s@dcs.qmw.ac.uk bbinson@aston.ac.uk ra@frodo2.cs.sandia.gov n.taylor@brunel.ac.uk x00@ukcc.uky.edu @cs.tu-berlin.de n@eecs.wsu.edu

Organizations (o)

Austrian Research Inst. for AI Bureau of Labor Statistics CRC Press Case Western Reserve U. Fraunhofer CRCG Ida Sproul Hall Kluwer Academic Publishers NSF Nohital Systems Oregon Graduate Inst. Permanent Solutions Random House Santa Fe Institute UOklahoma UTrento

Locations (l) Beaverton Berkeley Britain Canada Cleveland Italy Montreal NM Norman Providence, RI Quebec Silicon Valley

the Valley **Phone numbers (p)** 650-941-0336 (703) 883-7609

Stanford

US Vienna

+44 161 275 5716 +44-1752-232 558

Fax numbers (f) 650-941-9430 fax +44 161 275 6204 Fax (703) 883-6435 fax +44-1752-232 540 fax

URLs (u) http://cbdnet.ac

http://cbdnet.access.gpo.gov/ http://ourworld.compuserve.com/homepages/ftpub/call.htm http://www.cs.gov/ac/interim/ http://www.cs.man.ac.uk/~kung-kiu/jsc http://www.cs.sandia.gov/~scistra/DAM http://www.cs.tu-berlin.de/~tolk/AAMAS-CfP.html http://www.eslevier.nl/locate/parco http://www.santafe.edu/~bonabeau http://www.soc.plym.ac.uk/soc/sameer/paa.htm http://www.wired.com/wired/5.11/es_hunt.html

Figure 1.5: Generic data items extracted from one issue of The Computists' Weekly.

(a)	[9] Case, K. M., and Monge, A. Explicitly time-dependent constants/symmetries of			
	the higher-order KP equations. Journal of Mathematical Physics 30, 6 June 1989,			
	1250-1253.			
	[10] Eich, M. H. Main memory database research directions. In Proc. Sixth Int'l			
	Workshop on Database Machines (Deauville, France, June 1989) p. 251.			
(b)	[9] <name><last>Case,</last> <first>K. M.,_</first></name> and <name>-</name>			
	<last>Monge,</last> <first>A.</first> <title>Explicitly time-</title>			
	dependent constants/symmetries of the higher-order KP equations. -			
	<journal>Journal of Mathematical Physics</journal> 30, 6 (<date><month>-</month></date>			
	June <year>1989</year>), <pages>1250-1253</pages> .			
	[10] <name><last>Eich,</last> <first>M. H.</first></name> <title>Main</title>			
	memory database research directions. In Proc. Sixth Int'l Workshop on			
	Database Machines (<address>Deauville, France,</address> <date><month>-</month></date>			
	June <year>1989</year>) p. <pages>251</pages> .			
(c)	[9] <n>Case, K. M.,</n> and <n>Monge, A.</n> <t>Explicitly time-</t>			
	dependent constants/symmetries of the higher-order KP equations.			
	Journal of Mathematical Physics 30, 6 (<d>June 1989</d>), <g>1250-</g>			
	1253.			
	[10] <n>Eich, M. H.</n> <t>Main memory database research directions.</t>			
	In Proc. Sixth Int'l Workshop on Database Machines (<1>Deauville, France, 1			
	<d>June 1989</d>), p. <g>251</g> .			

Figure 1.6: Example of BIB corpus. (a) A section of bibliography. (b) The same section marked-up. (c) Modified marked-up version.

1.5 Thesis statement

This thesis makes two claims:

- 1. A HMM-based token identification system can be fully domain- and languageindependent.
- 2. PPM models can be utilized to handle unknown words in a HMM-based token identification system.

Some token identification systems that have been described in the literature take text tagged by part-of-speech as their input (e.g. Sekine, 1998; Baluja et al., 1999). These systems perform syntactic and/or morphological analysis on all words, including capitalized ones, that are part of candidate tokens. Other systems keep a huge list of known tokens. Both of these kinds of systems depend on the domain

and the language. Although a system has been reported as language independent (Cucerzan and Yarowsky, 1999), it does not use HMMs. The first thesis statement claims that it is possible to have a purely domain- and language-independent system. This makes a system more flexible.

Unknown word handling is an essential component of any robust token identification system. The second statement claims that PPM—*prediction by partial matching*—models can be used in conjunction with HMMs—*hidden Markov models*—to deal with this problem.

A hidden Markov model is a finite-state automaton with stochastic state transitions and symbol emissions (Rabiner, 1989). Recent research has demonstrated the effectiveness of HMMs for token identification. But when a unknown word is encountered, there is no information in a pre-trained HMM. Different methods have been presented, but no one has ever used PPM models. This thesis bridges the gap between HMMs and PPM models.

1.6 Thesis outline

Chapter 2 surveys previous research related to this thesis. It is divided into four sections. Section 2.1 reviews the methods which have been recently used in the token identification area, including hand-crafted systems, decision trees, maximum entropy models, HMMs and PPM models, and discusses systems which are based on these methods. Section 2.2 poses the problem of unknown tokens and presents some solutions. Section 2.3 discusses techniques for model smoothing. As a summary of the chapter, Section 2.4 briefly describes the method used in the undertaken research and compares it with other systems.

Chapter 3 describes the methodology used in the thesis, and is partitioned into seven sections. Section 3.1 introduces hidden Markov models, the basis of the system. Token identification is to find out the class for each word in the input text. The

algorithm that solves this problem is described in Section 3.2. Section 3.3 details how the model is constructed and Section 3.4 describes what the model looks like. In Section 3.5, the PPM model is introduced. It is a character level model, and will be the back-off of word-level HMM model for unknown words. How PPM models are used to handle unknown words is discussed in Section 3.6. In order to explore the impact of PPM, another method that handles unknown words is also described in this section. Finally, Section 3.7 describes model smoothing methods used in the system.

Chapter 4 presents the experiments carried out using the algorithm described in Chapter 3, and answers the questions proposed in the thesis statements. Before presenting empirical results, Section 4.1 describes measurements of the results. The first series of experiments are done on the TCC corpus, and the results are presented in Section 4.2. After analyzing the results, Section 4.3 discusses the shortcomings of the corpus that affect both the quality of the model and the accuracy of the result. Section 4.4 presents the result using the corrected corpus and discusses in detail. In Section 4.5, the HMM using PPM for unknown words is compared to the one using unified probability. Section 4.6 investigates the effect of the quantity of the training data. In Section 4.7, the effect of different model smoothing methods is investigated. As two corpora are available, Section 4.8 presents the results of applying the system to BIB corpus, and discusses how the structure of the text influences the accuracy of the result.

Chapter 5 is the conclusion. Section 5.1 highlights the findings of the thesis, and the related discussions are presented in Section 5.2. What could possibly be done in the future is described in Section 5.3.

Chapter 2 Related work

Text mining is looking for patterns of interest in text, such as a person's name, a geographical name and time factors. It is of recent interest to many researchers, such as Merkl (1998), Tan (1999) and Bray (1999). The task we are reporting in text mining is to find such patterns and mark them up with pre-defined labels. We call the task *token identification*. Many existing systems are based on the context of the Message Understanding Conferences (MUCs) (Grishman and Sundheim, 1996), which have involved the evaluation of information extraction systems applied to a common task.

This chapter is structured as follows. Section 2.1 describes different approaches used in the token identification task by other people. Section 2.2 discusses the issue of unknown words—words that have not been seen in the training data and appear in the test text. In Section 2.3 we talk about model smoothing, the process of replacing the original elements in the model with modified ones.

2.1 Different methods

The token identification task discussed here is to automatically identify the boundaries of a variety of phrases of interest in raw text and mark them up with associated labels. The systems reported in the Message Understanding Conference are limited to the following tokens: person, organization, location, date, time, money and percent. For us, however, the token identification task has no restriction—tokens are defined by a system designer and could encompass any type of information that is of interest. Token identification is an important component of many tasks such as information extraction and retrieval, machine translation and so on, as described in Section 1.3.3. This section reviews previous research in token identification.

Several systems have been reported for token identification tasks. They are based on: hand-crafted regular expressions (Appelt et al., 1993; Grishman, 1997); large name lists (Iwanska et al., 1995); sophisticated rule-based approaches (Morgan et al., 1995) and learning algorithms (e.g., Sekine, 1998; Bennett et al., 1997; Baluja et al., 1999; Borthwick et al., 1998; Mikheev et al., 1999; Bikel et al., 1999; Seymore et al., 1999; Bray, 1999). The early systems used the first three approaches and rely on much manual work, although some recently-reported learning systems are entirely automatic (e.g. Stevenson and Gaizauskas, 2000). Such systems are normally domain-specific, can be extremely expensive to develop, and require large amounts of maintenance. Also, it is not clear how much work is needed to adapt them to other domains or languages. The automatic approach has an advantage over hand-crafted rules, but the advantages of rapid and easy adaptation must be considered when applying these techniques to different domains or languages.

We briefly describe regular expressions in hand-crafted systems (Section 2.1.1), and then focus on learning algorithms such as decision trees (Section 2.1.2), maximum entropy models (Section 2.1.3) and hidden Markov models (Section 2.1.4). Other learning algorithms such as Brill's transformation-based learning algorithm (1995) which has been used as described by Aberdeen *et al.* (1995), are not discussed in this thesis. However, a closely related token identification system, which uses the PPM model, is reviewed (Section 2.1.5).

2.1.1 Hand-crafted systems

Models used in hand-crafted systems are assembled by inspecting training data. Tokens are identified using a set of regular expressions. Generally the expressions are stated in terms of parts-of-speech, syntactic features and orthographic features such as capitalization. An example pattern in this type of system might be "If proper nouns follow a person's title, then the proper nouns are a person's name". There are several indicators that help to identify names. Most commonly, for example, personal names can be identified by a preceding title such as "Mr.", "Prof." and "Rep." as in the following:

Mr. Robert Smith Prof. Sophia Young Rep. Natalie Cowley

Some suffix words are also indicators of human names such as ordinals (e.g. 1st), Roman numerals and the words "Junior" and "Senior". Other indicators include middle initial, commonly used first names and royal titles such as "Queen" and "Prince". Many indicators of human names are common (but capitalized) words that can take a prepositional phrase, usually an "of" phrase. For example, "Professor" can immediately precede the name, such as "Professor Andrew Grishman", or it can be followed by a prepositional phrase to form a professional title followed by a human name, such as:

Professor of Computer Science Andrew Grishman

As with human names, there are a few words that indicate the beginning of a geographical name, such as "Lake Taupo" and "Mount Maunganui". There are also words that indicate the end of a location, such as "River", "Park", "Bay", "Domain" and "Garden". Some words can indicate either the beginning or end of a location, for example "North Cape" and "Cape Reinga". There are also prepositional phrases for location names like "City of Auckland". Many well-known places, of course, are obvious indicators of location.

Similarly, names of organization can usually be identified by their final token, such as "Inc." or "Co.". Some words that indicate the name of an organization may appear at the end or before the very end of the name, such as "Associates" or "Systems", while some can appear anywhere in the name, such as "Broadcasting", "Club" and "Bank".

An information extraction system described by Grishman (1997) uses this method to identify certain kind of tokens.

2.1.2 Decision trees

Decision trees are a way to represent rules underlying training data, with hierarchical sequential structures that recursively partition the data. Sekine (1998) and Bennett *et al.* (1997) both implemented their token identification systems using decision trees. Their decision trees are based on almost identical features, such as part-of-speech, character type information and special dictionaries. While the two systems are similar, there are, however, significant differences between them.

The system described by Bennett *et al.* (1997) has multiple decision trees. Each one decides whether a particular class starts or ends at the current word. The system makes more than one decision at each word, thus multiple tags could be assigned to a single word, or possibly incompatible tags for two consecutive words. They introduced two methods to force one tag per word. One is to use a distance score, which finds an opening and closing pair for each word, based mainly on distance information. The other is to use a priority scheme for tags, which chooses a class among different candidates based on a priority ordering over words. Parameters required by these methods must be adjusted before being applied to a new domain.

In contrast, Sekine's (1998) system uses only one decision tree to produce the prob-

abilities of information about a token. Multiple possibility problems are solved by a probabilistic method. Manual adjustment of parameters is unnecessary. This is a strong advantage over the approach reported by Bennett *et al.*, because it makes the system more retargetable. Sekine also found that the dictionaries used in his system are not very domain-dependent, and speculated that little modification to the dictionaries might be required when applying the system to a new domain. This is an advantage for any system that uses dictionaries.

Another system using decision trees is proposed by Baluja *et al.* (1999). Like the systems described by both Sekine and Bennett *et al.*, it uses a part-of-speech tagger, dictionary lookups, and word-level features, such as all-uppercase, initial-caps, single-character, and punctuation features. The experiments focused on identifying which features affect the final performance most. Three sets of experiments were performed: the use of each of the dictionary, part-of-speech and word-level knowl-edge sources independently; the use of all pairwise combinations of the sources; and the use of all three sources together. They found that although none of these features performs well independently, performance improves when the context is increased. The experiments indicated that word level features. Furthermore, adding the dictionary feature to a system that uses only the part-of-speech tagger and word-level features achieved only a slight improvement.

2.1.3 Maximum entropy models

Maximum Entropy is a general technique for estimating probability distributions from data. It is widely used for a variety of natural language tasks, such as part-of-speech tagging (Ratnaparkhi, 1996), language modeling (Rosenfeld, 1994) and text segmentation (Beeferman et al., 1999). It has been shown that maximum entropy performs well in these tasks. The underlying principle is that without any external knowledge, the distributions should be as uniform as possible—that is, have maximal entropy.

Borthwick *et al.* (1998) described a token identification system built around a maximum entropy framework. The system uses a variety of knowledge sources, such as orthographic, lexical, section and dictionary features, to make tagging decisions. For any particular class label, there are four states: *label_start*, *label_continue*, *label_end* and *label_unique*. The first three states are for the case that more than one consecutive words are identified as the same class. The fourth is for the case that only one word is identified in a particular class. In addition, there is a special label *other*, which indicates that the word is not part of a class. For example, the phrase "Jenny Bentley lives in Hamilton" is marked as "person_start, person_end, other, other, location_unique". One label is assigned to every word in the text. This approach is essentially the same as that described by Sekine (1998). Borthwick *et al.* use Viterbi's (1967) search algorithm to find the highest probability legal path. For example, label_end can only be assigned to a word that follows a word with either label_start or label_continue. The system is a purely statistical one, and contains no hand-generated patterns.

Another system for token identification that uses a maximum entropy model is reported by Mikheev *et al.* (1999). The model uses contextual features of tokens, for example the position of tokens in a sentence, whether they appear in lowercase in general, whether they were used in lowercase somewhere else in the same document and so on. This system makes decisions using the answers provided by the Maximum Entropy model.

2.1.4 HMMs

A hidden Markov model, or HMM, is a particular kind of probabilistic model based on a sequence of events—in terms of token identification, this represents sequential words in text. Although different approaches have been studied and implemented, the best-known token identification system that incorporates a machine learning component is based on hidden Markov models (Baluja et al., 1999). IdentiFinder (Bikel et al., 1999) is a well-known system. It uses a variant of a hidden Markov model to identify tokens like names, dates and numerical quantities. Each state of the HMM corresponds to a token class. There is a conditional state for "not a token class". Each individual word is assumed to be either part of some predetermined class or not part of any class. According to the definition of the task, one of the class labels or the label that represent "none of the classes" is assigned to every word. IdentiFinder uses word features, which are language-dependent, such as capitalization, numeric symbols and special characters, because they give good evidence for identifying tokens.

There are three components in the top-level model, one for generating a token class, one for generating the first word in a class, and one for generating words that are not the first in a class. The models used by IdentiFinder are constructed using the counts in the training data.

Nymble (Bikel et al., 1997), a token identification system reported about two years earlier than IdentiFinder and by the same authors, is similar to IdentiFinder in terms of algorithm or techniques except that the results are slightly different. When applied to English text, the F-measure (refer to Section 4.1) of Nymble is 93% and 91% for mixed case and upper case respectively, and 94.9% and 93.6% for Identi-Finder. The reason that causes these differences cannot be found in the papers.

An information extraction system is reported by Seymore *et al.* (1999). It extracts fields of interest from the headers of computer science research papers using an HMM. One state is assigned for each class, such as title, author, date and affiliation. Unlike IdentiFinder, this system considers automatically determining model structure from data. At the beginning, each word in the training data has its own state, which transitions to the state of the following word. Each state is associated with the class label of its word. Two merging techniques—*neighbor-merging* and *V-merging*—are used to form the final model. They used three sets of training data: labeled, unlabeled and distantly-labeled, which refers to data labeled for other purposes, but can be partially applied to the task.

2.1.5 PPM models

PPM, prediction by partial matching, is a language model that was developed in the field of text compression (Cleary and Witten, 1984). It uses the preceding context to predict the probability of the upcoming symbol. A detailed description of how PPM works is provided in Section 3.5.

A system using character-based PPM models to identify tokens in running text has been reported by Bray (1999) and Witten *et al.* (1999a). All tokens in labeled training data are grouped according to their type, such as names of people, dates, locations and email addresses. Words that are not part of any classes are grouped together and assigned a class—plain text. All tokens in each type are used to construct a PPM compression model. Thus each type has its own model.

The goal of identifying tokens is tantamount to deciding which class each word belongs to. Using compression models, the input text is compressed. A word is assigned the class whose PPM model compresses it most. For example, if the location model compresses a word the most, the location class is assigned to the word—that is, the word is identified as part of a geographical name. All words in the text are assigned classes that maximize the overall compression.

An experiment has been done to investigate how well tokens are compressed both in and out of the context of the surrounding text, namely how possible tokens are identified by the correct type (Bray, 1999). The result shows that although the total number of incorrectly identified tokens is increased by using surrounding text, the actual mis-identification is greatly decreased, while the other errors are caused by failure to identify a token as plain text instead of as part of a class. The overall compression is increased slightly.

Bray then applies a Viterbi-style algorithm to detect the underlying state sequence that gives the best compression of the unseen text, in order to decide where is the best place to start or end a model. The purpose is to insert labels in a way that maximizes compression of the entire text. The system is based on PPM character models. It is a fully automatic system which can be adapted to other domains or languages without changing. The only thing required is a labeled training corpus.

2.2 Unknown token

Unknown words are those that have not been encountered in training data. There are two kinds of unknown word: neologisms, and existing words that happen not to appear in the training data.

Neologisms, or novel words, are almost always found in free text. Lexicons can never contain all possible words because they are static in nature, whereas the real world never stops changing.

Zipf's law (Zipf, 1965), a theoretical model of word occurrence, gives a good explanation of encountering unknown words. It states that if words are ranked, in descending order of frequency, with the most frequent ranked 1, the second most frequent ranked 2, and so on, the product of rank (r) and frequency (f) roughly fits the relation:

$$r \times f = C \tag{2.1}$$

Rank (r)	Word	Frequency (f) in %	r*f
1	the	6.15	0.062
2	of	3.54	0.071
3	and	2.70	0.081
4	to	2.51	0.100
5	а	2.14	0.107
6	in	1.90	0.114
7	that	0.97	0.068
8	is	0.95	0.076
9	was	0.94	0.085
10	for	0.86	0.086

Table 2.1: Statistics of the 10 most frequent words in the Brown Corpus.
where C is a constant, which is estimated to be about 0.1 in English text (Teahan, 1998). For example, Table 2.1 (Bell et al., 1990) shows the word statistics of the 10 most frequent words in the Brown Corpus. The right-most column is the product of the rank and the frequency. They round to the constant 0.1. Zipf's law states that the probability of occurrence of words starts high and tapers off rapidly. Thus, a few occur very often while many others occur rarely. For example in English, words like "the", "of" and "to" occur frequently while other words are rare, such as "ubiquitous", although it means something that exists everywhere.

Table 2.2 (Teahan, 1998) gives the percentage of words that occur only 1 to 5 times in the Brown Corpus. It indicates that a large percentage of words occur very few times. Among the different words, about 71% occur five or fewer times, 53% occur two or fewer times, and 38% occur only once. The consequence is that increasing the amount of training text does not help much in avoiding unknown words. Moreover, typographical errors also result in apparently unknown words.

Number of occurrences (n)	Types occurring n times (%)
1	38.3
2	14.6
3	8.5
4	5.4
5	4.0

Table 2.2: Percentage of word that occur 1 to 5 times in the Brown Corpus.

Regardless of which approach is used to find and mark up tokens, the size of the lexicon is determined either by pre-formed name lists or by a dictionary obtained from the training data. Previously unseen words will constantly be encountered. Dealing with these unknown words is an essential component for any robust token identification system.

Some studies that focus on unknown words (Mikheev, 1997; Daciuk, 1999) have been reported. They are based on morphological analysis and are especially useful for part-of-speech tagging, although an extension to token identification seems possible. Bikel *et al.* (1999) and Seymore *et al.* (1999) both use an HMM and both assign a special token to unknown words. Any words in the test data that are not present in the vocabulary are mapped into this single "unknown-word" token. Bikel *et al.* split the training data in half and used each half to train the model separately; then they added the counts from the two models. This yielded an estimate of how often unknown tokens occur, and used all available training data. In the system described by Seymore *et al.*, the probability of an unknown word is calculated using absolute discounting and is assigned as a portion of the mass proportional to the fraction of singleton words observed only in the current class. More detail about absolute discounting is given in the next section.

2.3 Model smoothing

Model smoothing is the process of replacing the original counts with modified ones, so as to redistribute the probability mass from more commonly observed events to less frequent and unseen events. It is necessary because of the limitation of the training data. If we define the actual count of an event E (such as a word) to be c(E), then the modified count $\overline{c(E)}$ is

$$\overline{c(E)} = d(c(E))c(E) \tag{2.2}$$

where d(c(E)) is the weight applied to the original counts.

Several methods of smoothing are discussed in the literature. Chen and Goodman (1998) have empirically compared the most widely-used smoothing techniques developed by Jelinek and Mercer (1980), Katz (1987), Bell, Cleary, and Witten (1990), Ney, Essen, and Kneser (1994), and Kneser and Ney (1995), using a word model based on the Brown Corpus, the Wall Street Journal Corpus (WSJ), the North American Business news (NAB), the Switchboard Corpus, and the Broadcast News Corpus. Because the undertaken research uses HMMs, we focus on smoothing tech-

niques that are used in similar models.

Thede and Harper (1999) describe the smoothing method used in their Second-order HMM model. The method takes into account lower order information. The model is smoothed using a logarithm function to calculate smoothing coefficients from the number of occurrences of all order events within the training data. It can be viewed as a variant of Jelinek-Mercer smoothing method (Jelinek and Mercer, 1980) with the difference that Thede and Harper calculate the smoothing coefficients for each event while Jelinek-Mercer smoothing requires the bucketing of interpolation coefficients according to the total number of counts of the corresponding history. It is unnecessary to use held-out data to optimize parameters, and thus computationally inexpensive. Jelinek-Mercer method, however, is a far more complicated procedure as the Baum-Welch algorithm is used.

IdentiFinder (Bikel et al., 1999) uses several back-off models, schemes used for events that have not been encountered in the training data, and assigns a variable weighting to each one. If λ is the weight used for direct estimation of probabilities, then $(1 - \lambda)$ is the weight of the back-off model. λ is based on a function of the number of times the given event occurs:

$$\lambda = \left(1 - \frac{\operatorname{old} c(Y)}{c(Y)}\right) \frac{1}{1 + \frac{\operatorname{type of } Y}{c(Y)}}$$
(2.3)

where "old c(Y)" is the sample size of the model from which they back off. The expression of λ has two factors. It is the second that does the real work of smoothing. For example, consider only the second factor. Suppose that "Susan Smith" has occurred twice in the training data, "Susan Robert" four times, and the word "Susan" has not been seen anywhere else in the person's name class. When calculating the probability of the word "Smith" given that the model is in the name class, the bi-gram probability

would back off to the unigram probability

Pr(("Smith", initCap) | person's name)

with a weight of 1/4, where *initCap* is the feature of the word, which indicates that the word has a capitalized initial letter. The reason is that there are two different words that follow "Susan", and the total number of occurrences of "Susan" being the first word in a bi-gram is six. Thus a weight of 1/(1 + 2/6) = 3/4 is assigned to the bi-gram probability, and a weight of 1 - 3/4 = 1/4 is for the back-off model.

A further back off is

Pr("Smith" | person's name) × Pr(initCap | person's name)

and the final one is

$$\frac{1}{|V|} \times \frac{1}{\text{number of features}}$$

where |V| is the size of alphabet. Similarly, the policy described above applies to two other models, name class model and first-word model.

Absolute discounting is a common method of smoothing. It defines

$$d(r) = (r-b)/r$$
 (2.4)

in (2.2), where r is the original count here. This has been applied on the emission estimates in the system described by Seymore *et al.* (Seymore et al., 1999). How they choose the constant b is not mentioned in the paper. This is, of course, equivalent to simply subtracting the constant b from each count.

2.4 Summary

The system described in this thesis is based on an HMM. Each state represents a token class and emits different words. Words unknown to the model are handled by

PPM models. In order to label a new text with token classes, words in the text are treated as observations and the Viterbi search algorithm is used to recover the most-likely, hidden state sequence—a label sequence that is associated with the word sequence in the task.

Three of the systems discussed previously in this chapter have a close relation to the one presented in this thesis. The first one is IdentiFinder (Bikel et al., 1999). It is the same as the system described in this thesis in model construction—using counts of word occurrences in the labeled training data; label assignment—only one label, either a class label or the label that indicates a given word is none of the classes; and search algorithm—the Viterbi algorithm is used. Aside from these identical factors, the two systems differ in some significant points. IdentiFinder uses multiple HMMs and each word has its own state with emission probability 1, which is unlike the common or traditional HMMs. It also uses word features. The methods used to handle the unknown words and to smooth the model are quite different.

The second one is the system which extracts fields from the headers of computer science research papers described by Seymore *et al.* (1999). Their system and the one described in this thesis are same in label assignment, search algorithm, and both use only one HMM. However, the two systems use different kinds of training data and different methods of unknown word handling and model smoothing.

The third is the one that uses PPM model as described in Section 2.1.5. The system itself is, in fact, not very similar to the described system in terms of the main techniques, but the way it uses PPM models is the same, whereby each token class has a PPM model, and the same training corpus is used.

Chapter 3 Information extraction with HMMs and PPM

One of the thesis statement in Section 1.5 claims that PPM models can be used to handle the problem of unknown words in HMM-based token identification. PPM is a statistical language model in which a certain number of previous symbols predict the probability of occurrence of the next one. A character-level PPM model contains accurate information on the statistics of characters seen in the past, which encourages its use to estimate the probabilities of unknown words. This chapter introduces both HMMs and PPM, and shows how PPM can be used to predict the occurrence of unknown words.

Section 3.1 describes the basic theory of hidden Markov models and how they are used in the task. For a general introduction to HMMs, refer to Rabiner (1989) and Poritz (1988). Section 3.2 reviews the Viterbi algorithm, which is used to find the hidden state sequence. Section 3.3 describes how the HMM parameters are obtained from the training corpus. Section 3.4 is about training. Section 3.5 reviews the PPM compression model and its zero-frequency problem. In Section 3.6, we talk about how to deal with unknown words. This is where PPM is introduced into the HMM model. Finally, model smoothing is discussed in Section 3.7.

3.1 Hidden Markov model

A hidden Markov model is a finite-state automaton with stochastic state transitions and symbol emissions (Rabiner, 1989). It is a particular model based on a sequence of events, and consists of a set of states and a set of output symbols. The automaton generates a sequence of symbols by starting from the initial state, transitioning to a new state, emitting an output symbol, transitioning to another state, emitting another symbol, and so on, until the final state is reached and the last symbol is emitted.

3.1.1 Model definition

For each member of the set of states, $S = \{S_1, S_2, ..., S_N\}$, there are two probability distributions. One governs the outgoing state transitions, which indicates how likely another state is to follow; the other governs the emission of symbols in the observation vocabulary $V = \{V_1, V_2, ..., V_M\}$, which indicates how likely a symbol is to be generated in the particular state. N and M are the number of states and number of symbols respectively.

We assume that time is discrete, and the model transitions between states at each time unit. In the case of a first-order Markov model, which is used in the undertaken research, the probability of moving from state S_i to state S_j is stored in the *state transition matrix*, $A = \{a_{ij}\}$, where:

$$a_{ij} = \Pr[q_t = S_j | q_{t-1} = S_i], \qquad 1 \le i, j \le N.$$
(3.1)

In this and future equations, t refers to the time instant, q_t is the variable that records the state assignment to the t^{th} symbol, and S_j , the j^{th} member of the set of possible states, is the assigned value. In other words, the probability of being in the current state is determined by the previous state.

When the HMM moves between states, it emits an output symbol after each transi-

tion. Exactly which output symbol is emitted depends on the *output symbol distribu*tion B, which defines the probability of emitting a particular symbol in a particular state. For first-order HMM, B is a two dimensional matrix defined as $B = \{b_j(k)\}$, where:

$$b_j(k) = \Pr[o_t = V_k | q_t = S_j], \qquad 1 \le j \le N, \quad 1 \le k \le M.$$
 (3.2)

Here, o_t is the variable that records the t^{th} symbol emission, and V_k , the k^{th} member of the observation vocabulary, is the emitted symbol.

To complete the model we need an initial probability distribution $\pi = {\pi_i}$ over states, where:

$$\pi_i = \Pr[q_1 = S_i], \qquad 1 \le i \le N.$$
 (3.3)

The entire model can be described as $\lambda = \{A, B, \pi\}$. It is based on two independence assumptions—Markov assumptions. The first assumption is that the next state is based on the current state only. The second assumption is that the appearance of a symbol is independent of the preceding or succeeding state. While these independence assumptions are often not valid, they appear to work reasonably well in practice.



Figure 3.1: Example of HMM.

Figure 3.1 is an example of a three state first-order HMM. From the figure, state S_1 transitions to state S_2 with probability 0.5 and to state S_3 with probability 0.3. The probability of staying in the same state is 0.2. Notice that each state can be reached

from any other state in only one step. This is a so-called *fully connected* model. However, models obtained from training data (refer to Section 1.4) to represent certain circumstances are not originally in this style.



Figure 3.2: Example of partially connected HMM.

Because of the assumptions, distribution B is state-specific, and generally it is multinomial, that is, each state generates more than one symbol and the probability of generating a symbol differs from state to state.

Figure 3.2 shows a partially connected HMM model with four states, $S = \{S_1, S_2, S_3, S_4\}$ and their transitions. It is more like a model attained in practice. Its state transition probability A written in a matrix is:

$$A = \begin{bmatrix} 0.2 & 0.0 & 0.0 & 0.8 \\ 0.6 & 0.0 & 0.1 & 0.3 \\ 0.7 & 0.2 & 0.0 & 0.1 \\ 0.0 & 0.3 & 0.4 & 0.3 \end{bmatrix}.$$
 (3.4)

Assume that there are four observation symbols, $V = \{a, b, c, d\}$, and the symbol emission distribution B is:

$$B = \begin{bmatrix} 0.36 & 0.23 & 0.12 & 0.29 \\ 0.65 & 0.00 & 0.25 & 0.10 \\ 0.74 & 0.26 & 0.00 & 0.00 \\ 0.00 & 0.13 & 0.44 & 0.43 \end{bmatrix}.$$
 (3.5)

Each state emits a symbol with different probability, for example, state S_1 emits symbol a with probability 0.36, S_3 emits symbol a with probability 0.74, and S_4 is unable to emit a. If the four states have equal likelihood at the beginning, then

$$\pi = \{0.25, 0.25, 0.25, 0.25\}.$$
(3.6)

A hidden Markov model probabilistically connects the observations to the state transitions in a system. There are three basic problems that can be solved by an HMM:

- 1. the probability that model λ generates a particular observation sequence O,
- 2. the most likely state sequence the model went through in generating the observed sequence of symbols,
- 3. a set of re-estimation formulas for iteratively updating the HMM parameters given several observation sequences as training data, in order to maximize the probability of the sequences being generated by the model.

The first one is described below, the second one is discussed in detail in Section 3.2, and the third one, also known as Baum-Welch algorithm, is mentioned in Section 2.3 and Section 3.3.

For a given state sequence, the first problem, the probability that the model generates a particular sequence of symbols, is computed by multiplying the probability of being in a state by the probability of generating a certain symbol in the state. For instance, given the model in Figure 3.2 and the sequence $S_2S_1S_4S_3S_4S_2$, the probability that the sequence generates *acdbbc* is computed as follows. We start from the initial stage. The probability that the start state is S_2 is 0.25, and the probability of generating symbol *a* in state S_2 is 0.65, so the probability of being in S_2 and generating symbol *a* is 0.25×0.65 . Then we move on to the second symbol. The probability of reaching state S_1 from S_2 is 0.6 and the probability of generating symbol *c* from state S_1 is 0.12, so the combined probability is 0.6×0.12 . We move on to the next symbol and compute the combined probability of being in the state and generating a specific symbol in the same way until the last symbol is processed. Finally, the required probability is obtained by multiplying all these combined probabilities $(0.25 \times 0.65)(0.6 \times 0.12)(0.8 \times 0.43)(0.4 \times 0.26)(0.1 \times 0.13)(0.3 \times 0.25) =$ 4.08×10^{-7} .

The word *hidden* in the term of hidden Markov model refers to the state transition sequence which is hidden from the observer. It is revealed only through the observed symbol sequence, and may never be known with certainty.

3.1.2 Applying HMMs to token identification

For the token identification application discussed in Section 1.3, the observation sequence is a sequence of words in text. The symbols emitted in each state are words, and the HMM is a word-level model.

In the system described in this thesis, each sequence corresponds to a sentence in text, and each state corresponds to a type of token that the program will identify and mark up. Example token class include people's names, geographical locations, monetary amounts and e-mail addresses. Each type of token will be marked in the text by a unique tag. N, the number of states in the model, is the number of different token classes, and is determined by the training data. Because the system uses a word-level HMM model, M, the size of the output vocabulary, is the number of different words that appear in the training data.

The matrix A gives the probability that the current word belongs to a particular token type given that the previous word belongs to a particular token type as well. We also call it the *contextual probability*. Distribution B is the probability of the same words being seen in a particular token class. It is token-dependent: different token classes have different probabilities for a certain word. B is also called the *lexical probability*. The initial distribution π is the probability that each type of token starts a sentence.

For instance, in the following sentence, a fragment in annotated version of The Computists' Weekly, 1998,

<o>Polytechnic University</o> in <l>Brooklyn</l> will get <m>\$190M</m> from the <n>Othmer</n> estate, about four times the school's previous endowment.

the sequence of four words "Polytechnic", "University", "in" and "Brooklyn" contributes to the four-element class sequence $\langle o \rangle \langle o \rangle \langle p \rangle \langle l \rangle$ (see Section 1.4.1). It contributes the probabilities of transitioning from organization (*o*) to organization, state *o* to *o*; organization to plain text (*p*), *o* to *p*; and plain text to location (*l*), *p* to *l*. Thus probabilities are given by the elements of matrix *A*. The words themselves will be counted as the appearances in the corresponding token class to make up the elements of matrix *B*, for example, words "Polytechnic" and "University" labeled as organization would increase the counts for their occurences in this class. "Polytechnic" as part of organization would also increase the probability of token $\langle o \rangle$ starting a sentence.

Recall from Section 1.3 that the token identification task is as follows: given a sequence of words, identify the appropriate tokens and mark them up with predefined labels, given that a model has already been constructed.

It is assumed that each individual word belongs to a class. Consider names of particular types: people's names and location. Whether a word is a name or not is random with an estimable probability. For example, the word "Washington" could refer to a person, or it could refer to a location. The probability of being a person or a location can be estimated from a training corpus in which every type of name has been labelled. Because each state in the HMM has its own emission distribution, stated in Section 3.1.1, the probability is conditioned not only on the word, but on the state that the model is in at that moment. For example, if "Washington" follows a preceding title like "president" and "Mr." or a common name "Bob", it is likely to be a person, while if it is followed by "D.C.", it is likely to be a location.

As mentioned above, words and token class refer to observation symbols and states in the HMM. Finding tokens in an sequence of words means finding the state sequence that underlies the input. This is just the second of the three basic problems of HMM. Given the observation sequence and the model, how do we choose a corresponding state sequence which is optimal in some meaningful sense (i.e., that best "explains" the observations) (Rabiner, 1989)? The solution is discussed in the following section.

3.2 Decoding

Let us assume that an HMM model has been constructed for a particular kind of sequence, and we are presented with a new example of such a sequence, $O = o_1, o_2, ..., o_T$. The problem of finding the most likely state sequence $Q = q_1, q_2, ..., q_T$ that produces the given symbol sequence is called *decoding*. There are several possible ways of solving this problem. A commonly used method is the *Viterbi* algorithm (Ryan and Nudd, 1993; Viterbi, 1967; Forney, 1973), which is to recover the state sequence that has the highest probability of having produced the given observation sequence. For the sake of computation, the following variable is introduced:

$$\delta_t(j) = \max_{q_1, q_2, \dots, q_{t-1}} \Pr[q_1 q_2 \dots q_{t-1}, \ o_1 o_2 \dots o_{t-1}, \ q_t = S_j, o_t].$$
(3.7)

This is the probability of the best sequence ending at state S_j with a transition going from state S_i to state S_j for symbol o_t over all state assignments. It gives the highest possible probability that the partial observation sequence and state sequence up to time t can have. By induction, it is easy to observe the following recursive relationship:

$$\delta_{t+1}(j) = [\max_{1 \le i \le N} \delta_t(i)a_{ij}]b_j(o_{t+1}), \qquad 1 \le j \le N, \quad 1 \le t \le T - 1.$$
(3.8)

The initial condition is

$$\delta_1(j) = \pi_j b_j(o_1), \qquad 1 \le j \le N.$$
 (3.9)

The most important point is that because of the Markov assumptions, sequences that end in the same state can be collapsed together since the next state depends only on the previous state in the sequence.

In summary, the algorithm starts by calculating $\delta_1(j)$ for $1 \leq j \leq N$. Then it uses the recursive relationship (3.8) to calculate the following $\delta_t(j)$ until t = T to retrieve the optimal state sequence. It is necessary to keep track of the best state sequence found so far for each possible ending state. At the end, the final state j^* is found by

$$j^* = \arg \max_{1 \le j \le N} \delta_T(j).$$
(3.10)

Starting from the pointer to this state, the algorithm backtracks through the optimal sequence, and the path that is produced represents the required sequence of states.

The Viterbi algorithm can be written as:

$$Q^* = \arg\max_{q_1...q_T \in Q^T} \prod_{t=1}^T \Pr[q_t = S_j | q_{t-1} = S_i] \Pr[o_t = V_k | q_t = S_j],$$
(3.11)

where Q^* is the optimal state sequence that maximizes the Viterbi scores, and $\Pr[q_t = S_j | q_{t-1} = S_i]$ and $\Pr[o_t = V_k | q_t = S_j]$ are the state transition probability and symbol emission probability respectively, as defined above.

The entire algorithm is illustrated in Figure 3.3, where nodes represent the states of the HMM at each time instant t for $1 \le t \le T$. N = 4 in the figure.

To find the most likely sequence, the algorithm starts at the initial stage and moves forward through the observations one at a time, finding the most likely sequence for each ending state. In other words, it calculates the probability that each state generates symbol o_1 by using the initial condition (3.9), then finds the N = 4 best



Figure 3.3: Viterbi algorithm interpretation.

sequences for the two observations o_1o_2 : the best ending with o_2 in S_1 , the best in S_2 , the best in S_3 , until the best in S_N . This information is used to find the N = 4 best sequences for $o_1o_2o_3$, each one ending in a different state. This process is repeated until all the observations are accounted for. The solid arrows represent those best sequences. Finally the maximum probability at the last stage is chosen, and the required state sequence is recovered by backtracking the path indicated by heavier arrows.

For instance, take the model described in Figure 3.2 (Section 3.1.1, Page 35) and assume that probabilities not in Figure 3.2—zero elements in the matrix A (3.4)—have a value of 0.0001. Suppose the input sequence is *acba*. The algorithm operates as follows. The first step is performed in the initialization phase using (3.9). Due to the emission probability of Figure 3.2, only entries for S_1 , S_2 , S_3 are non-zero. So the most likely sequence of one state ending in S_1 to generate *a* has a probability of $\pi_1 b_1(a) = 0.25 \times 0.36 = 0.09$, whereas the most likely one ending in S_2 has probability of $0.25 \times 0.65 = 0.1625$, and 0.185 and 0 for one ending in S_3 and S_4 respectively. The result of the first step of the algorithm is shown at the left-hand side of Figure 3.4.



Figure 3.4: The result of the first two steps of the Viterbi algorithm running on sequence *acba*.

The second step of the algorithm extends the sequences one symbol and keeps track of the best sequence. Equation (3.8) is used to compute the most likely sequence of two states ending in each state. For example, the probability of generating acending in state S_1 is computed as follows:

$$\begin{split} \delta_2(1) &= [\max_{1 \le i \le N} \delta_1(i) a_{i1}] b_1(o_2) \\ &= \max[\delta_1(1) a_{11}, \delta_1(2) a_{21}, \delta_1(3) a_{31}] b_1(c) \\ &= \max[0.09 \times 0.2, 0.1625 \times 0.6, 0.185 \times 0.7] \times 0.12 \\ &= 0.185 \times 0.7 \times 0.12 \\ &= 0.0159 \end{split}$$

The most likely two-state sequence that generates ac and ends in S_1 has probability 0.0159, namely the sequence S_3S_1 . The most likely one ending in S_2 has probability 0.0093, namely sequence S_3S_2 . The most likely one ending in S_4 has probability 0.0317, namely sequence S_1S_4 . There are no transitions to S_3 because in the emission distribution B (3.5) (Section 3.1.1, Page 35), $b_3(o_2) = b_3(c) = 0$. The right-hand side of Figure 3.4 is the result of the second step with a sequence



Figure 3.5: The result of the third step of the Viterbi algorithm running on the sequence *acba*.

of two states. The solid arrows represent the best sequences up to each state. The computation runs recursively in the same way until the last symbol has been processed.

Figure 3.5 illustrates the result after processing the third symbol, and Figure 3.6 illustrates the final result. The only difference between the values shown in Figure 3.6 and the actual computation is that the values are rounded in the figure. The maximum probability sequence ends in state S_3 . The heavier arrows in Figure 3.6 indicate the required state sequence $S_3S_1S_4S_3$.

A more explicit interpretation of the result is shown in Figure 3.7, where horizontal arrows indicate the underlying transition of states and vertical arrows indicate the emission of symbols in each state.

3.3 Estimating the parameters of an HMM

Once the model structure is determined, the next problem is to estimate the model parameters for the state transition probabilities *A* and state-specific lexical distri-



Figure 3.6: The result of the final step of the Viterbi algorithm running on the sequence *acba*.



Figure 3.7: The interpretation of generating sequence *acba*.

bution B given a set of training data. Generally, there are two kinds of methods: unsupervised and supervised.

For unsupervised learning, the training data is untagged—no labels are inserted into it. A and B can be estimated by applying the Baum-Welch algorithm (Baum, 1972). Given the initial parameters, this algorithm adjusts model parameters iteratively to maximize the likelihood of untagged data. However, because there can be different possible results, the corpus must be correctly analyzed before being used for parameter estimation. It requires great effort to analyze a large corpus manually. The result is also sensitive to the initial parameters, because the maximization is local.

Supervised learning uses tagged training data—that is, sequences of words with the target words already marked up with associated labels. The information required to construct the model can be obtained by recognizing the labels. But to obtain such a

corpus requires a large amount of effort to tag tokens manually.

While both methods involve some manual effort, analyzing the corpus may require more expertise than tagging. On the other hand, training data can be tagged by applying an automatic tagger to the raw material and checking the result manually. Also, the unsupervised method creates more complex models than the supervised one. Previous work has shown that supervised methods have been applied quite successfully to the task (Bikel et al., 1997).

In the undertaken research, labeled training corpora are used; thus learning is supervised. The original state transition probability and symbol emission distribution are calculated in a straightforward manner by using the ratio of counts, events/samplesize or words/vocabulary:

$$a_{ij} = \frac{Count(S_i \to S_j)}{Count(S_i)}$$
(3.12)

and

$$b_j(k) = \frac{Count(V_k \uparrow S_j)}{Count(S_j)},$$
(3.13)

where $Count(S_i \rightarrow S_j)$ is the frequency of occurrence of the two consecutive classes S_i and S_j , $Count(S_i)$ and $Count(S_j)$ are the frequencies of occurrences of class S_i and S_j respectively, and $Count(V_k \uparrow S_j)$ is the frequency of occurrence of word V_k in the class of S_j .

Let us take the sentence given in Section 3.1.2 (Page 38) as an example. It is repeated for easy reference:

<o>Polytechnic University</o> in <l>Brooklyn</l> will get <m>\$190M</m> from the <n>Othmer</n> estate, about four times the school's previous endowment.

The corresponding state sequence is shown below:

where each tag corresponds to a word. For example, the first four tags correspond to the first four words—"Polytechnic", "University", "in", and "Brooklyn".

Let *i* represent class $\langle o \rangle$ and *j* represent class $\langle p \rangle$. $\langle o \rangle$ occurred twice, once followed by $\langle o \rangle$ and the other time followed by $\langle p \rangle$. So $a_{ii} = 1/2$ and $a_{ij} = 1/2$. $\langle p \rangle$ appeared thirteen times, nine times followed by itself. So $a_{jj} = 9/13$. There are twelve different words in class $\langle p \rangle$, and the word *the* occurred twice. If we denote the word *the* by *k*, then $b_j(k) = 2/13$.

As mentioned in Section 3.1.1, we still need the initial probability vector π to complete the model. Two different methods are used to construct π .

The first is to estimate π from the training data, as with a_{ij} and $b_j(k)$:

$$\pi_j = \frac{Start \to Count(S_j)}{Count(Start)},\tag{3.14}$$

where Count(Start) is the total number of classes that start a sentence, that is, the number of sentences, and $Count(Start \rightarrow S_j)$ is the number of times that class S_j starts a sentence.

An alternative method is to assign each class equal probability, therefore $\pi_j = 1/N$, where N is the number of token class. This method has been applied to the model given in Figure 3.2.

3.4 Training the HMM

Before constructing the model, the training data is split into sentences. The system processes the training data in two passes. The first pass counts the number of token classes, N, the number of different words, M, and the vocabularies for each token type. The second pass counts the number of events and calculates the A and B

matrices.



Figure 3.8: A HMM obtained from the training data.

For illustration, Figure 3.8 is an example of an HMM obtained from part of the training data. It is annotated with token labels explained in Section 1.4.1, and with transition probabilities. The figure shows that it is possible for words in the plain text (p) class to follow words in any other token classes, and these words can also be followed by words in any other class except email (e). It is reasonable that words in all token classes can be surrounded by plain text. The location class (l), monetary class (m) and URL (u) class have no direct relationship between each other. They never appear one after another and always have tokens in other classes between them. This is understandable from the grammatical point of view. Probabilities from plain text to some other token class, such as date, source and location, are very low. This is not because the events are rare but because they are overwhelmed by plain text words, which makes the denominator bigger and results in smaller numbers.

The figure indicates that there are no tokens in fax and phone classes in this particular set of training data, because classes, along with vocabularies, depend on the training data.

The issue of data sparseness with respect to the estimates A and B is addressed in

Section 3.7

3.5 The PPM model for text compression

PPM is a predictive language model that was developed in the field of text compression (Cleary and Witten, 1984). On files of English text it typically achieves compression rates of about 2.2 bits per character (Moffat et al., 1997), which saves over 70% of the original data space. Such a big saving can be of great use in the practical world, where the amount of information is increasing tremendously. Though many other compression techniques exist, PPM has become a benchmark in the compression community (Witten et al., 1999b). It has been widely used in language processing tasks, such as character level language modeling (Teahan, 1998; Bray, 1999).

The PPM text compression model can be used together with HMM to identify tokens in text. In this application we use the prediction probabilities generated by PPM, but we do not make any use of the actual compressed result. This section describes the principle of PPM and how to determine symbol and escape probabilities. How the model is constructed and used in the task will be described in Section 3.6.

3.5.1 PPM model

Models that take a few immediately preceding symbols into account to make a prediction are called *finite-context* models of order m, where m is the number of preceding symbols used (Witten et al., 1999c). The PPM technique uses finite-context models of characters (Cleary and Witten, 1984). It is a so-called character-level model. It uses the last few characters in the input string to predict the upcoming one. By considering such a context, each character can be better predicted. The prediction is done by using the counts of occurrences of each context. The probabilities associated with each character that has followed the context are used to predict the probability for the upcoming character.

PPM uses fixed-order context models with different values of m, up to a predetermined maximum. The maximum number is a given constant, which we call the *order* of the model. The bigger the order, the more information is considered. But increasing the order does not guarantee better compression, because the contexts become rarer as the order grows. Studies have found that increasing the order beyond about five does not generally improve compression (Cleary and Witten, 1984; Moffat, 1990; Cleary et al., 1995).

Several orders are blended together in PPM to obtain a good probability estimate for the current character. The prediction starts with a given maximum order m and checks the occurrence of the order m context. If the order m context has occurred with the upcoming character following it, the corresponding counts are used to predict the probability. If the context has not been seen in the past, the model then uses the order m - 1 context.

Consider the case where the context has occurred but never followed by the upcoming character. This is called the *zero-frequency* situation (Witten and Bell, 1991) the character will be predicted using a zero count. In this case, a special transmission called *escape* is used to drop the model down one order, and the order m - 1model is used to make the prediction. The problem of what the escape probability should be will be discussed in Section 3.5.2.

Another possible situation is that the character has never occurred in the past—an unknown character. Then even order 0 cannot be used. This is another instance of the zero-frequency problem. The model then escapes down to a bottom-level model, order -1, that predicts all characters equally.

To illustrate the PPM modeling technique, Table 3.1 (Teahan et al., 2000) shows the four models with order 2, 1, 0 and -1 after the string *tobeornottobe* has been processed.

		Order	2				Order	1		Order 0
P	redicti	ion	С	p	Р	redicti	ion	С	р	Prediction c
be	\rightarrow \rightarrow	0 esc r	1 1	1/2 1/2	b	\rightarrow \rightarrow	e esc	2 1	3/4 1/4 1/2	$ \begin{array}{cccc} \rightarrow & b & 2 \\ \rightarrow & e & 2 \\ \rightarrow & p & 1 \end{array} $
0	\rightarrow	esc	1	1/2	C	\rightarrow	esc	1	1/2	\rightarrow 0 4
no	\rightarrow \rightarrow	t esc	1	1/2 1/2	n	\rightarrow \rightarrow	o esc	1	$\frac{1/2}{1/2}$	\rightarrow r 1 \rightarrow t 3
ob	\rightarrow \rightarrow	e esc	2	3/4 1/4	0	\rightarrow \rightarrow	b r	2 1	3/8 1/8	$\rightarrow esc 6$
or		n esc	1 1	1/2 1/2			t esc	1 3	1/8 3/8	Order –1
ot	$\stackrel{\rightarrow}{\rightarrow}$	t esc	1 1	1/2 1/2	r		n esc	1 1	$\frac{1/2}{1/2}$	\rightarrow A 1
rn		0 esc	1 1	$\frac{1}{2}$ $\frac{1}{2}$	t		o t	2 1	1/2 1/6	
to	\rightarrow \rightarrow	b esc	2	3/4 1/4		\rightarrow	esc	2	1/3	
tt	$\stackrel{\prime}{\rightarrow}$	o esc	1 1	1/2 1/2						

Table 3.1: PPM model after processing the string tobeornottobe.

p

3/26 3/26 1/267/26 1/26 5/26 3/13

1/|A|

In this illustration the maximum model order is 2. For each model, all previously occurring contexts are shown with their associated predictions, along with occurrence counts c and probabilities p. The probabilities are determined from the counts using escape method D that will be discussed below. In the table, the esc is for escape and |A| is the size of the alphabet. It is this that determines the probability for each unknown character.

The model in Table 3.1 is used as follows. Suppose the character following tobeorn ottobe is o. Since the order-2 context is be, and the upcoming symbol has already been seen once in this context, the order-2 model is used and the probability is 1/2. If the next character, instead of o, were t, this has not been seen in the current order. Consequently an order-2 escape probability of 1/2 is used and the context is truncated to the order-1 context e. Again it has not been seen in this context, so an order-1 escape probability of 1/2 is used and the context is truncated once more to the null context, corresponding to order 0. Finally the character t is predicted with a probability of 5/26. Thus the prediction of t is done in three steps, using order 2 to order 0 context respectively, with a probability of $1/2 \times 1/2 \times 5/26$. If the upcoming character had been x instead of t, a final level of escape to order -1

would have occurred with a probability of 3/13, and x would be predicted with a probability of 1/256 (assuming that the alphabet |A| = 256).

		Context				
		tobeor	nottobe	nottobeorto		
		Probability	Model used	Probability	Model used	
Upcoming	o ½	1⁄2	Order 2	¹ /2× ¹ /2×5/6	Order 0	
character	t	¹ / ₂ × ¹ / ₂ ×5/26	Order 0	1/2×1/6	Order 1	

Table 3.2: Effect of context and current character with order-2 PPM model.

The probabilities predicted by PPM are based on the occurrences of the prior context and the characters that have followed each context every time the context has occurred in the training text. Table 3.2 shows how the previous context being processed and current character affect the result in terms of the order of model and probabilities by using the same prior contexts to predict different characters and vice versa.

In the case where the upcoming character is o, for the string *tobeornottobe*, the preceding context is *be* (using an order-2 model), which has been seen once and followed by the same character o. Hence o can be predicted using the current order of model with probability 1/2. For the string *nottobeorto*, although the previous context *to* occurred before, it is followed by a different character from the upcoming one. The model is forced to move down to the lower order. This continues until o is finally predicted using the order-0 model, and the total probability is $1/2 \times 1/2 \times 5/6$, where two 1/2 are for escapes from higher-order model to lower ones. The same principle applies when the upcoming character is *t*. The orders of model used for prediction of the strings *tobeornottobe* and *nottobeorto* are 0 and 1 with probabilities of $1/2 \times 1/2 \times 5/26$ and $1/2 \times 1/6$ respectively.

3.5.2 Estimating probabilities in PPM

If the context has never occurred in the processed text—the zero-frequency situation, PPM uses an escape to drop the model down to a lower order. The problem of how to deal with zero frequencies reduces to the problem of calculating the escape probability. This issue has been discussed by researchers (e.g., Cleary and Witten, 1984; Moffat, 1990; Howard, 1993) and several different methods have been proposed. Table 3.3 shows four different methods used in the experiments.

Let us define $p(\phi)$ to be the probability of symbol ϕ , a character in this task; *e* the probability of escape; $c(\phi)$ the count of a particular context followed by the character ϕ ; *t* the distinct number of characters that have followed a particular context; and *n* the number of times a context has appeared.

Method	Escape probability	Symbol probability
А	$e = \frac{1}{n+1}$	$p_i(\phi) = \frac{c_i(\phi)}{n+1}$
В	$e = \frac{t}{n}$	$p_i(\phi) = \frac{c_i(\phi) - 1}{n}$
С	$e = \frac{t}{n+t}$	$p_i(\phi) = \frac{c_i(\phi)}{n+t}$
D	$e = \frac{t}{2n}$	$p_i(\phi) = \frac{2c_i(\phi) - 1}{2n}$

Table 3.3: Methods for zero frequency problems.

In Table 3.3, methods A and B were proposed by Cleary and Witten (1984). Method A simply adds one count to each occurrence: the more the context has occurred, the smaller the escape probability. Method B does not count contexts that have appeared only once. By doing this, unusual contexts are ignored. Method C was introduced by Moffat (1990), and bases the probability on the number of types of context. It is similar to method B except that characters are predicted immediately. Method D, a variant of method C, was proposed by Howard (1993). It adds 1/2 count to both character and escape, instead of one as in method C. The four methods also refer to

the name of PPM followed by the capital letter of the method itself, such as PPMA for method A.

3.6 Unknown word handling

One of the main goals of token identification is to choose the correct label in cases where a word can have more than one label assignment. Additionally, a system must deal with words that have not been encountered in the training data, and so are not found in the lexicon.

The lexicon for the HMM is built during training, so the model contains all words. It also contains all the counts that are needed to calculate the probabilities in (3.11). If an unknown word is encountered during decoding, there is no entry in the model. The emission probability in the state transition matrix B is unknown. To ensure that the process continues and works in a proper way, some policy must be adopted to estimate the probability that the current state will emit the given unknown word.

To make an entry for unknown words, a special token is assigned that matches all unknown words, regardless of what they are. The question arises how to calculate the probability for this special token. Different methods have been discussed in Section 2.2. In this section, this section discusses two methods used in the system. One relies on a PPM model; the other uses a unified probability.

3.6.1 PPM models for unknown words

As described in Section 3.5.1, a PPM model is a finite-context character-based model that uses the last few characters in the input stream to predict the upcoming one. In the system, PPM models are used to provide HMM with a unique probability for each individual unknown word. In other words, the word-level HMM drops down to the character-level PPM model when unknown words are encoun-

tered. In this way we bridge the two models. The probability provided by PPM is more relevant to the unknown word itself than the unified probability discussed in Section 3.6.2.

A PPM model is constructed for each token class, using all tokens in a class in the training data. For example, in Figure 1.5 (Section1.4.1), words in the location column are used to construct a PPM model for location, and words in the people's name column are used to construct a PPM model for names.

The PPM models are now available to deal with unknown words. Whenever a word that has not been encountered in training is seen, and is therefore not contained in the lexicon, the value of $b_j(k)$ is assigned the probability that is predicted by an appropriate PPM model, when computing (3.8) and (3.9) to find out the most likely path.

Since there are several PPM models and each one predicts a probability, it is necessary to determine which is the appropriate one. In the experiments, the largest probability was used. All calculations are done on the fly.

3.6.2 Unified probability for unknown words

Another way of handling unknown words is to provide a unified probability that governs every such word in a class, regardless of what it is. This idea is based on the method of Bikel *et al.* (1999) mentioned in Section 2.2.

The training data is divided into two parts, and the training procedure is accomplished in two steps. The first step, which is called *training*, is to train the model using one half of the training data. The second, which is called *updating*, uses the other half of the training data to update the model that is obtained from the first step.

As mentioned in Section 3.1.1, the output symbol distribution is state-specific. The probability of generating a particular symbol depends on the state. Because of this, it is necessary to seek a method of assigning each state its own probability for the

unknown token.

During updating, the dictionary stores an extra item—the token classes that have been associated with each word. Every time a word is encountered, the system checks it in the dictionary along with the token class. There are three possibilities: not in the dictionary, in the dictionary but not associated with the class, and in the dictionary and associated with the same class. In the first case, the word is added to the dictionary, and associated with the token class to which it belongs, and the count for the unknown token of the corresponding state is updated. In the second case, the new token class is stored with the word, and the count of the corresponding unknown token is updated. In the third case, only the word's count is updated.

In this way, we estimate how often an unknown word will occur in each class. Only the first occurrence of a new word is taken into account when estimating the probability of the unknown token.

3.7 Smoothing the probabilities

Smoothing is a technique for adjusting probability estimates that have been obtained from the training data. Smoothing is necessary when data is sparse. It is especially important for handling the zero frequency problem, which is ubiquitous in models constructed by learning.

Zero frequencies occur in both contextual and lexical probabilities. For example, in the model described in Section 3.4, because of the lack of training data, there is no relationship between some classes. Therefore the probability that the model transitions between the corresponding states is zero. The zero frequency problem occurs more often in lexical estimation because of the nature of text: most words are plain text. The probabilities that these words are generated in other classes are zero.

In the illustration of the Viterbi algorithm on page 41 (Section 3.2), a small constant 0.0001 is assigned to zeros in the contextual probability matrix A (zeros in the lexical probability matrix B are left unchanged). This does not seem to be reasonable in general. It is hard to say how small the number should be without taking other counts into consideration. The value 0.0001 is small enough in the example of Figure 3.2, but not for a model with actual probabilities that are smaller than 0.0001.

Rather than using a fixed small number, we looked for a method that depended on the number of occurrences of existing events, such as token classes and words. The existing model smoothing methods are discussed in Section 2.3. To our knowledge, there has been no systematic investigation of smoothing approaches for token identification. However, smoothing methods proposed in other research areas, such as language modeling (Chen and Goodman, 1998) and part-of-speech identification (Thede and Harper, 1999), can be adapted.

The experiments use the four escape methods in PPM (described in Section 3.5.2) as smoothing methods. The results are discussed in Section 4.7.

Chapter 4 Experimental evaluation

The HMM-based token identification model described in Chapter 3 has been evaluated in several different ways. This chapter presents and discusses the experimental results.

Before describing experiments to assess the system, three standard measurements that are commonly used in this field are described in Section 4.1. For the purpose of easy analysis, a stand-alone measurement is also introduced. In order to see how the system worked, it was first applied to the TCC corpus described in Section 1.4. The results are presented in Section 4.2. Section 4.3 discusses errors in the corpus. Because the errors affect both the model and the final results, the corpus was corrected and the experiments run again on the new version. The corresponding results, along with discussion, are presented in Section 4.4. In order to explore the impact of PPM, Section 4.5 compares different unknown word handling methods. The empirical experiments indicate that combining PPM with an HMM-based identification model for unknown words results in better performance than that using a unified probability. Section 4.6 investigates the effect on performance of the size of the models, which is determined by the amount of training data. Increasing the amount of training data results in better identification. Although the performance continues to improve as the size of the training data grows, for the available corpus, using 25 issues of TCC newsletters to train the model seems to be acceptable. Section 4.7 investigates the effect of different model smoothing methods. Finally, in Section 4.8, the system is evaluated on bibliographies. The results are better than for TCC text because of the characteristic structure of bibliographic text.

4.1 Measuring

Three standard measures, *recall*, *precision* and *F-measure* (Van Rijsbergen, 1979; Lewis, 1995), along with *error-rate* are used to evaluate the accuracy of the token identification system. They are calculated by using the corresponding manually marked-up fragment in the training corpus as the gold standard. For easy reference, let us call this gold standard *hand mark-up*. To define them, the following terms are used:

N	Number of tokens occurring in the standard text;
с	Number of tokens correctly marked up by the system;
e	Number of tokens incorrectly marked up by the system;
n = c + e	Number of tokens marked up by the system.

The measures take into account two aspects of the mark-up: the label itself, and the boundary where the label is inserted. A token is considered to be correctly marked up when both label and boundaries are correct. For example

The board has been begging and bribing $\langle n \rangle$ Steve Jobs $\langle n \rangle$ to stay on, but he hasn't accepted yet.

"Steve Jobs" is correctly marked as a person's name and it contributes one count to *c*.

Recall and precision are widely used to assess the quality of an information retrieval system in terms of how many of the relevant documents are retrieved (recall) and how many of the retrieved documents are relevant (precision). In the token identification task, recall is the proportion of the correct tokens which are actually identified by the system, while precision is the proportion of tokens identified by the system which are correct. They are written as:

$$\operatorname{Recall} = \frac{c}{N},\tag{4.1}$$

$$Precision = \frac{c}{n}.$$
 (4.2)

The two measures do not always provide an adequate evaluation because there are some extreme situations where one of them is very small while the other is large. For example, if the system identifies few tokens compared to the number of Nand they are all correct, recall will be very small whereas precision is 100%. It is better to have a measure that yields a high score only when recall and precision are balanced. A widely used measure is the F-measure (Van Rijsbergen, 1979; Lewis, 1995):

$$F\text{-measure} = \frac{(\beta^2 + 1) \times \text{Precision} \times \text{Recall}}{\beta^2 \times \text{Precision} + \text{Recall}}$$
(4.3)

where values of β between 0 and ∞ give varying weights to recall and precision. In this thesis, $\beta = 1$, gives equal importance to recall and precision, therefore, F-measure is the harmonic mean of recall and precision:

$$F\text{-measure} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}.$$
 (4.4)

The measure of error-rate is used just for easy analysis of the result. It is defined as

$$\text{Error-rate} = \frac{e}{N}.$$
 (4.5)

This is normally used on its own as an overall indicator of the quality of identification. However, it can give misleading results—an extreme condition is where the system only identifies a single word, leading to a very small error-rate of 1/Ndespite the fact that all tokens but one remain unidentified. If the system marks up the same number of tokens as the hand mark-up, recall and precision both become equal to one minus the error-rate. A perfect identification system will have an error-rate of zero, and recall and precision of 100%.

During the checking procedure, labels and boundaries are scored separately at half a count each. In one case, a token may be marked with the correct label but has one of its boundaries displaced. Examples are given below with the correct mark-ups in bold:

Happens every **few** <d>**years**</d>, and sometimes does result in program cuts.

Happens <d>every **few years**</d>, and sometimes does result in program cuts.

<o>The **Bureau of Labor Statistics**</o> characterized the computer field as having "strong long-term growth trends."

In these cases, the correct label along with one correct boundary contributes a half count to c, the number of tokens correctly marked up by the system, and the other incorrect boundary contributes a half count to e, the number of tokens incorrectly marked up by the system against the hand mark-up. This policy does not always apply to each case. For example, the following are considered as errors:

<s>NY Times, 19Jan98. EduP</s>.

The Mining Company has a *<*u*>***college admissions guide at** *<<*http

:// collegeapps.miningco.com</u>>>, including tips on interviewing.

In the first example, not only is one of the boundaries at the wrong position, parts of the text included, which are shown in bold, should be marked as other classes—date and source respectively. In the second example, too many extra words (in bold) are included, in other words the wrong boundary is too far from its correct position. Only one word displacements are accepted. The correct mark-up for these two is shown below:

<s>NY Times</s>, <d>19Jan98</d>. <s>EduP</s>.

The Mining Company has a college admissions guide at <<<u>http:// collegeapps.miningco.com</u>>>, including tips on interviewing.

The second situation in which the mark-up contributes a half count to c is where the boundaries are correct but with the wrong label. For example:

And the company is definitely getting better at the <o>Washington </o> power game.

<o>Statistical Computing & Graphics</o> is a joint newsletter of the Statistical Computing and Statistical Graphics Sections of the American Statistical Association, distributed to members three times a year.

Here and later in this chapter all labels that are not discussed are suppressed for readability. Instead of organization, "Washington" here should be marked up as a location according to the hand mark-up. Similarly, "Statistical Computing & Graphics" in the second example is a source rather than an organization.

These measures are calculated manually by comparing the result of identification against the hand mark-up in the corpus.

4.2 Application to the TCC corpus

The first series of evaluations used the TCC corpus. The corpus was first split into sentences because the system processes input sentence by sentence. As mentioned in Section 1.4.1, the available TCC corpus contains 38 issues of the newsletter. 25 issues, which are randomly selected, are used as the training set.

The statistics of the training data are shown in Table 4.1. The number of words depends on exactly how a "word" is defined. Besides spaces, back slashes in URLs and dots in email addresses are also considered as word delimiters. This increases

Token type	Number of words	Percentage of words
dates (d)	1089	2.9%
email addresses (e)	1302	3.5%
fax numbers (f)	35	0.1%
phone numbers(h)	62	0.2%
locations (l)	559	1.5%
sums of money (m)	157	0.4%
names (n)	1151	3.1%
organizations (o)	814	2.2%
plain text	28919	77.7%
sources (s)	1054	2.8%
URLs (u)	2083	5.6%
total	37225	100.0%

Table 4.1: Statistics of the training data in the TCC corpus.

the number of words in the email and URL classes. Based on this definition, about 78% of words in the training data are plain text. Except for URL and email address, the name class has the most words (3.1%), and date and source rank second (2.9%) and third (2.8%) respectively. The percentage of words in these three classes, 8.8%, is just double that in the fax, phone, location, sum of money and organization classes—4.4%. The table shows that few words are labeled as fax or phone—taken together, these two classes account for only 0.3% of words. The more words in a class that occur in the training data, the more likely it is that tokens in that class will be identified correctly in new text.

To see how the system works, 5 of the remaining issues are used as the test set with all the tags removed. PPM models are used for the unknown words and the model is smoothed using PPMA, which is described in Section 3.5.2.

Table 4.2 shows the results for each of the five test files in terms of recall, precision, F-measure and error-rate. It also gives the average values of these measures.

In the table, the value of precision is almost always higher than that of recall. This indicates that the system marks less tokens than it should do. Consequently, some tokens are either missed out—left unmarked, or merged into another token by the system.
file	Recall (%)	Precision (%)	F-measure (%)	Error-rate (%)
test1	60.47	66.38	63.29	30.63
test2	59.00	55.66	57.28	47.00
test3	58.33	61.76	60.00	36.11
test4	54.39	62.00	57.94	33.33
test5	60.67	62.76	61.69	36.00
Average	58.57	61.71	60.04	36.61

Table 4.2: Result for five TCC test files using the model of HMM+PPM.

The errors can be divided into two different types:

- 1. correctly marked up tokens that are mistakenly not labeled or labeled incorrectly in the hand mark-up;
- 2. incorrectly marked up tokens.

The first type is called *false positive*. Inspect the result of file test2 in Table 4.2, the low precision compared to recall is caused by a large number of false positives–24 false positives out of a total of 106 marked-up tokens. Several aspects that affect the accuracy of the results are discussed in the following section.

4.3 Defects in the corpus

The identification system is based on two components: the corpus and the algorithm. Errors in either affect the final results. Because the corpus is annotated by hand, one hundred percent accuracy is hard to achieve. Errors can arise through human negligence or mistake, and also through differences in judgment that happen between individuals. This section presents the shortcomings of the corpus that affect the accuracy of the result.

As mentioned in Section 4.1, measurements are done by comparing the result of the system with the hand mark-up in the corpus, which is pre-marked manually. The

(a)	<l>U.S.</l> News has a ".edu Colleges and Careers Center" with lots of infor-					
	mation about colleges and campus life, graduate schools, and beyond.					
	<l>Stanford</l> Testing Systems offers free SAT skill-building exercises at					
	< <http: www.testprep.com="">>.</http:>					
(b)	<s>U.S. News</s> has a ".edu Colleges and Careers Center" with lots of infor-					
(b)	<s>U.S. News</s> has a ".edu Colleges and Careers Center" with lots of infor- mation about colleges and campus life, graduate schools, and beyond.					
(b)	<s>U.S. News</s> has a ".edu Colleges and Careers Center" with lots of infor- mation about colleges and campus life, graduate schools, and beyond. <o>Stanford Testing Systems</o> offers free SAT skill-building exercises at					

Figure 4.1: Examples of ambiguity. (a) Hand mark-up. (b) System mark-up.

marking differences between these two that count as errors during evaluation are discussed in terms of ambiguity, omission and error.

As mentioned in Section 4.1, both the training data and the hand mark-up, the marked-up version of test data, are part of the corpus and selected randomly. So problems in the hand mark-up exist in the training set as well. This reduces the performance of the system, because the model is trained on imperfect data.

4.3.1 Ambiguities

Marking ambiguities are found during experiments. Figure 4.1 shows two examples. Figure 4.1a shows how the tokens are marked in the hand mark-up and Figure 4.1b shows the results from the identification system. "U.S." can stand alone as a location, though more commonly people use "US" instead. But in the context of this sentence, "U.S. News" is more likely to be a multi-word token in the source class. Moreover, "U.S. News" has occurred as part of a source in the training data.

Similarly, in the training data, "Stanford" has been seen several times as a location. But based on semantics, identifying "Stanford Testing Systems" in the test file as an organization is more reasonable than "Stanford" alone as a location in the hand mark-up. The system, however, does not include any semantic processing.

It has also been found that all occurrences of "weekly" and "monthly" are stemmed when marking up by hand, as shown in the following examples:

(a)	IEEE Internet Computing is soliciting articles.
	Now in its second year, it has the 2nd largest circulation of the IEEE Com-
	puter Society optional publications.
	Integrating multiple overlapping metadata standards; J. of the American So-
	ciety for Information Science (JASIS).
	Queen's U.
(b)	4> <1>Silicon Valley 1 jobs:
	Martyne Page' has returned to Canada after a year of writing a Silicon Valley
	column for a French-Canadian newspaper in Quebec.
	(<1>Mount Arlington, NJ 1 /Paris, France): MS/PhD French/English
	computational linguist with NN, ML, NLP.
	(Mount Arlington, NJ): MS/PhD center director in computational linguistics,
	NN, NLP.
	That, and that <n>Steve Jobs</n> kept undermining his relationship with
	the board.
	The board has been begging and bribing Steve Jobs to stay on, but he hasn't
	accepted yet.
	The <1>US 1 created 20K new computer services jobs in Jul98, plus 3K
	in computer manufacturing, out of just 66K new US jobs total.

Figure 4.2: Examples of omission. (a) Tokens are left unmarked. (b) Tokens are not marked up for all occurrences.

<d>new week</d>ly <d>available month</d>ly

These examples show disagreement in human judgment. The person who marked the corpus might have followed his/her rules which are not incorrect. The system did the right thing as well. However, the results are considered as errors because they differ from that in the hand mark-up.

4.3.2 Omissions

The labels of some tokens are accidently missed out in the hand mark-up. There are two cases: tokens that obviously belong to some classes are left unmarked as shown in Figure 4.2a, and tokens that are not marked up for all occurrences as shown in Figure 4.2b.

In Figure 4.2a, the text in bold belongs to some token class, for example, "IEEE

Internet Computing" is a source and "American Society for Information Science (JASIS)" is an organization. All of them are marked up correctly by the system. However, they are considered as errors when measuring because the measure is against the imperfect hand mark-up.

Figure 4.2b shows four samples where a token, in bold, is marked somewhere while the same token is left unmarked in other places. For example in the first part of Figure 4.2b, "Silicon Valley" is marked as a location in one sentence and in the other sentence it is left unmarked. For the same reason as above, this increases the number of errors because the system marks both up as locations. The same situation occurs for the next two samples in Figure 4.2b. Typically, the last sample in Figure 4.2b presents the failure in the same sentence.

Moreover, some tokens are marked in different ways. For example "Educom Update" appears in three different ways:

Educom Update <s>Educom</s> Update <s>Educom Update</s>

where the last one is the right one. During measuring, whether the mark-up is either correct, partially correct or false depends on what it is in the hand mark-up at the same position.

4.3.3 Errors

Other errors have been found in the hand mark-up as shown in Figure 4.3. Figure 4.3a shows two errors that occurred in the hand mark-up. It is obvious that both "ISTA" and "CASA" are acronyms of the preceding organizations. Although the system did not mark them correctly either, as shown in Figure 4.3b, at least it identifies the tokens as the right class. These kind of errors in the hand mark-up may affect system performance elsewhere.

(a)	Int. Science & Technology Associates (<1>ISTA 1 ; Philadelphia): BS
	linguists for Japanese patent machine translation.
	Center for Adaptive Systems Applications (<1>CASA 1 ; Los Alamos,
	NM): scientists in data mining, adaptive computing for financial modeling or
	fraud detection.
(b)	Int. <o>Science & Technology Associates (ISTA</o> ; Philadelphia): BS
	linguists for Japanese patent machine translation.
	<pre><o>Center</o> for Adaptive Systems Applications (CASA; Los Alamos,</pre>
	NM): scientists in data mining, adaptive computing for financial modeling or
	fraud detection.

Figure 4.3: Examples of error. (a) Hand mark-up. (b) System mark-up.

4.3.4 Effect of the imperfect training data

The above types of failure occurred in the randomly selected test set in the corpus. They occur in the entire corpus as well. For example, in the training set, "US" occurs 68 times: 18 times as a location by itself, 6 times as part of an organization and 44 times it was left unmarked. This makes the models ambiguous and therefore affects the performance of the system because the identification system is based on the model.

For example, in the first test file, "US" occurs three times in the hand mark-up, once as a location and twice as plain text, but the system identifies it as plain text for all three occurrences. According to Table 4.1, the probability of generating "US" in the location class is higher than that in plain text. However, besides symbol emission probability, the hidden Markov model involves another probability—state transition probability. The large amount of text means that the state that represents plain text has a higher chance of being chosen as the current state. If all occurrences of "US" had been marked correctly, there would be only a small chance for it to be considered as plain text.

An additional experiment was performed to investigate this. For the same training data, location labels were added for all occurrences of previously unlabeled "US" that are likely to stand as a location. Other occurrences were part of a class which the token belongs to, for example "<o>US Air Force</o>" and "<o>US Dept. of

Education (DoED</o>". The model was then trained using the modified version. By applying the model on the same test file, the result shows that all three occurrences are marked up as location. Two of them are fully correct and the other has one boundary incorrect, "<1>The US</1>".

4.4 Application to the corrected TCC corpus

Because the models and therefore the performance of the system are based on the corpus, the quality of the corpus is important. According to the shortcomings described in Section 4.3 and further examination, the original TCC corpus was corrected by hand. The experiment described in Section 4.2 was run again using the corrected version of the corpus. The same training and test splits were used as in the original run.

File	Recall (%)	Precision (%)	F-measure (%)	Error-rate (%)
test1	64.86	72.37	68.41	24 76
test2	65.67	74.58	69.84	22.39
test3	68.72	78.34	73.21	18.99
test4	60.96	74.79	67.17	20.55
test5	66.26	66.67	66.46	33.13
Average	65.29	73.35	69.02	23.96

Table 4.3: Result for five TCC test files using the corrected corpus and the model of HMM+PPM.

Table 4.3 shows the results for the corrected corpus. On average, recall and precision are improved by 6.72% and 11.64% respectively, which results in an 8.98% improvement over the original corpus for F-measure. At the same time, error-rate drops down by 12.65%.

However, there were still some errors left by accident. For example, not all appearances of "Washington Post" and "Scott Thurm" were marked up as source and name respectively. And all mark-ups with word stemming were left unchanged.

In more detail, Figure 4.4 presents the proportion of correctly marked tokens (dark



Token type

Figure 4.4: Detailed result for five TCC test files using the corrected corpus and the model of HMM+PPM.

gray) and errors (light gray) over the corresponding numbers of tokens in the hand mark-up for each class and overall.

The figure shows that organization class has a very poor result, with only about 20% of correct tokens identified. This is probably due to the lower percentage of words in this class in the training data. As mentioned before, the more words in a class that occur in the training data, the more likely it is that tokens in the same class can be identified in new text. The proportion of words in classes such as fax number and sums of money are also small, however, they have special so-called indicators, for example "fax" and "\$". This increases the performance for these classes. However, a name of an organization is more likely to comprise diverse words, such as:

<o>Bell Labs</o> <o>John Benjamins</o> <o>Holland Academic Graphics</o> The percentage of correct tokens in date, email and fax classes are about the same. However, the system is more successful on date class because the error rate is very low. This is understandable because date has much less ambiguity than email address and fax number. Most tokens in date class are distinguishable in both words and format. Errors happen in special cases such as "30-<d>40 years</d> "and word stemming. But they are rare. On the other hand, email addresses can be mixed up with URLs due to the definition of a word. Some sources such as "<s>comp.ai.genetic</s>" also increase the error in email address and URL classes. There is no distinction between fax number and phone number except the word "fax/Fax", which is a strong indicator to distinguish fax number from any other classes.

It is interesting to notice that a token which is a money amount is either marked up correctly or left unmarked. Due to the format of the token in this particular class, there is no marking ambiguities.

Generally speaking, name of a person is relatively harder to identify than other classes. The figure shows that it gets the second best result for both the correct ratio and error ratio. As we have noticed, there are some names that occur repeatedly. For example, "Ken", "Bill Park" and "Brandon" have occurred several times in the entire corpus and none of them has been found to be identified incorrectly. However, most of the names are not in this special case.

4.5 Exploring impact of PPM

One of the thesis statements claims that PPM models can be used to handle unknown words for an HMM-based token identification system. The previous experiments have shown the results. How it works compared to other method is still a question. This section presents the results of the experiments which were run by using both PPM models and unified probability for unknown words.

The same 25 TCC issues of training data and 5 issues of test data are used. The experiment was run twice. The probability of every unknown token is provided by PPM models the first time and set to a unified value for each token class as described in Section 3.6.2 the second time. To estimate how often unknown tokens occur in each class, the 25 issues of training data were divided into two parts, 13 issues for training and 12 issues for updating.

model	Recall (%)	Precision (%)	F-measure (%)	Error-rate (%)
PPM+HMM	65.29	73.35	69.02	23.96
Unified+HMM	60.73	69.94	65.00	25.24

Table 4.4: Average results of 5 TCC files using PPM and unified probability for unknown words.

Table 4.4 shows the average results in terms of recall, precision, F-measure and error-rate. On close examination of the results, the differences are mainly due to the number of fully correct marked-up tokens. With PPM models, more tokens are marked up fully correctly, which means both label and boundaries are correct. For example, the system failed in the following cases when the unified probability is used,

<s>Al Kamen, Washington Post</s> <e>fr</e> <s>comp.ai.genetic, 05Aug98</s> <s>AWAD, 05Aug98</s>

but it performed correctly with PPM models,

<n>Al Kamen</n>, <s>Washington Post</s> <e>hermes@iway.fr</e> <s>comp.ai.genetic</s>, <d>05Aug98</d> <s>AWAD</s>, <d>05Aug98</d>

The results using PPM models are better overall, but in some cases, mark-up is

improved when using the unified probability. For example, the system identified the following URL correctly,

<u>http://www.elsevier.nl/locate/parco</u>

while only part of the URL is marked up when using PPM models,

<u>http://www.elsevier.nl</u>

4.6 Effect of the quantity of training data

For any learning system, the effect of the amount of the training data is an important issue, especially for a supervised learning system. It determines how much human work is required to prepare the data in order to get acceptable performance. In the previous experiments, 25 randomly selected issues were used to train the model. Whether the size of the training data is reasonable is also a question that needs to be addressed.

For the TCC corpus, experiments have been done with different amounts of training data. Eighteen models were trained with successively larger amounts of data starting from 2 issues of the newsletter, with each training set an extension of the preceding one with two more issues. The eighteen models were evaluated on one TCC issue, which is not included in any of the training sets.

Figure 4.5 presents the results from the different models. The lines marked with circular and cross points correspond to the value of precision and recall respectively. The solid line with no points corresponds to error-rate. The dashed line corresponds to F-measure.

Starting with two issues of training data, the performance of the system is improved as more training data is used. For precision and recall, the significant improvement happens before the point of 12 issues of training data. Both of the values keep



Figure 4.5: Effect of the amount of training data.

increasing, but at a more steady rate after this point. They appear to be more stable after the point of 24 issues of training data. For the error-rate, the value fluctuates. It increases while more training data is added until 10 issues of training data. This is because there is not enough training data yet. Therefore, more errors occurred although many more tokens are identified correctly as shown by both precision and recall. It starts to decrease after this point except for a fluctuation at the point of 22 issues. It then levels off after this point.

Overall, Figure 4.5 implies that the performance of the system is improved in terms of standard measurements and practice by using more training data. After 24 issues, although lines are more flattened, adding training data still improves the values of precision and recall. It is assumed that with sufficient data, the least amount of training data which gives reasonable performance will be more apparent. This point exists because more ambiguities are introduced when more training data is used, even if correctly annotated. For example, a token may appear in different classes, which affects the ability of the system to make the final identification decision. Balancing the improvement and errors caused by ambiguities, there must be an amount after which the entire performance of the system tends to be stable. However, for the available amount of corpus, 25 issues for training seems to be a good choice from the figure.

4.7 Effect of model smoothing method

Different smoothing methods have been described in Section 3.5.2. In the previous experiments, Method A has been used. This section investigates how the other methods perform, and whether using Method A is a good decision. The same 25 issues of TCC in the corpus are used to train the model. Three issues of TCC, which are randomly selected from the remainder are tested with four smoothing methods: Method A, Method B, Method C and Method D. They are defined in Section3.5.2, and originally for PPM models. Table 4.5 shows the average results.

Smoothing	Recall (%)	Precision (%)	F-measure (%)	Error-rate (%)
Method A	64.48	75.83	69.70	20.58
Method B	54.51	71.89	61.93	21.39
Method C	52.84	77.11	62.61	15.78
Method D	58.12	76.09	65.80	17.99

Table 4.5: Average results from different model smoothing methods.

From the table, Method A has the highest value of recall while precision and errorrate are worse than for some of the other methods. By inspecting the results, it is apparent that the system with Method A always marks up more tokens than the others, which increases both the number of correct tokens and errors.

Method C has the highest value of precision but the lowest value of recall. This is caused by a lower number of tokens being identified. This benefits precision a lot.

Recall from the definition of precision and recall in Section 4.1, a higher value of precision means that among the identified tokens, more of them are correct. And a higher value of recall indicates that more correct tokens are identified by the system. Ideally, a method which results in both the highest precision and recall is a good choice. However, for the situation here, Method A is the best choice overall at a little expense of error-rate. Method D can be an alternative because the differences compared to the best one in both precision and recall are small, and the error-rate is also lower.

Method B and Method C perform comparatively worse than other two. As above, Method B marks up more tokens either correct or incorrect. This results in larger recall but smaller precision. Considering the larger error-rate of Method B, Method C is more acceptable than Method B on average.

To sum up, the choice of smoothing method can be made by the value of F-measure with consideration of error-rate.

4.8 Application to bibliographies

Another thesis statement claims that the system is fully domain- and languageindependent. The domain-independence is evaluated by applying the system on a bibliography corpus without any changes.

The text in bibliographies is more structured than that in the TCC newsletter, which the model takes advantage of. A bibliography entry contains name, title and date. Most of them have page number(s), and some provide organization, publisher, location and source. In the experiments, the BIB corpus of 2400 references were selected randomly from the bibliography collection mentioned in Section 1.4.2. The model is trained on 2200 entries, and the experiments are done by running the system on 100 of the remaining references with labels removed.

Corpus	Recall (%)	Precision (%)	F-measure (%)	Error-rate (%)
BIB	72.02	81.78	76.59	16.05
TCC	65.29	73.35	69.02	23.96

Table 4.6: Average result for 100 bibliographic entries and 5 TCC issues.

Table 4.6 shows the average value of recall, precision, F-measure and error-rate for 100 bibliographic entries. It also shows the average result of 5 TCC issues for comparison. Overall, the system was successfully applied on the new text. The figure indicates that the performance is 7.57% better in F-measure than that of TCC.

Figure 4.6 shows the proportion of correctly marked tokens (dark gray) and errors



Token type

Figure 4.6: Detailed result for 100 bibliographic entries and 5 TCC issues.

(light gray) over the corresponding numbers of tokens in the hand mark-up for each class and overall.

In the figure, the combined values of correct and error for publisher, date and source class end up more than 100%. This is because the system marks up more tokens than that in the corresponding classes in the hand mark-up.

The figure shows that not a single token is marked up as an organization whether correct or not. In fact, there are 4 organizations in the test data. 3 of them are marked up incorrectly in the hand mark-up and one is left unmarked. Figure 4.7 gives an example, which shows the contrast between the output of the system and what in the hand mark-up. The other two have the exactly same problem—only labels are different, and it is still acceptable.

Tokens in date class have the best identification result. More than 90% of date are correctly marked up, and among the mistakes, some of them are because the system marks up date like

Bodnarchuk, R., and Bunt, R. A synthetic workload model for a distributed (a) systems file server. In Proc. 1991 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems ACM SIGMETRICS Performance Evaluation Review (San Diego, California, USA, May 21- 24 1991), Univ. of Saskatchewan, p. 50. Published as Proc. 1991 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems ACM SIGMETRICS Performance Evaluation Review, volume 19, number 1. Bodnarchuk, R., and Bunt, R. A synthetic workload model for a distributed (b) systems file server. In Proc. 1991 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems ACM SIGMETRICS Performance Evaluation Review (San Diego, California, USA, May 21- 24 1991), <o>Univ. of Saskatchewan</o>, p. 50. Published as Proc. 1991 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems ACM SIGMETRICS Performance Evaluation Review, volume 19, number 1.

Figure 4.7: Example of marking error. (a) Hand mark-up. (b) System mark-up.

<d>May 21- 24 1991</d>

as

$$< d > May < /d > 21 - 24 < d > 1991 < /d >$$
.

However, it is understandable regardless of the hand mark-up. Some of the errors are false positive.

Many tokens in the page class are missed out. Out of the total number of 72, almost all of those that have only one number like "p. 25" in the references are left unmarked. This is assumed to be due to ambiguities, because a similar number format occurs many times in plain text, such as the number of volume and series.

The system marks up only 17 locations out of 40, and 9 of them are errors. It is interesting that 8 of the errors are false positive, such as

<d>San Mateo, CA</d> <d>Amsterdam</d> <d>Atlantic City</d> It is noticeable that about two third of names are correctly identified. It is the structure of the text that improves the performance on name class, which is comparatively harder than some other classes.

Compare Figure 4.6 to Figure 4.4 (Section 4.4), there are five classes in common: date, location, name, organization and source. It is obvious that the system performs very poor on both location and organization, especially the latter. One of the reasons for this is the lower number of words in these classes in the training data. And another one is ambiguity. For example, a place may be named after a famous person, and an organization may be named after its owner or location.

On comparison, the system works well for tokens in date class with about 92% and 73% correct tokens identified for text in bibliographies and TCC newsletters respectively. Undeniably, the format of date in the text contributes a lot to the result.

The result for people's name is also good on both TCC and BIB text, although not the best among all the classes. It has been discussed (in Section 1.3.2) that names of people are not easy to identify correctly. This result indicates that the system performs well on this class.

As we have not got any proper corpora in languages other than English, no experiments were done to evaluate the language-independence. However, since the system does not comprise any language-dependent components, there should be no problem once a suitable corpus is available.

Chapter 5 Conclusion

This thesis makes two claims: that hidden Markov models can be successfully used to develop a domain- and language-independent token identification system, and that PPM models can be used in conjunction with HMMs to provide the probability for every unknown word in this application. The technique is domain independent in that it uses no domain or language-specific techniques, but good results cannot of course be guaranteed in any domain. The corresponding algorithms and evaluations have been described in Chapter 3 and Chapter 4 respectively. Section 5.1 highlights what has been done in the thesis. In Section 5.2, we briefly discuss each of the achievements. As with any research thesis, this is certainly not the end of the study. There are always problems that remain unsolved or just arise at the end. Section 5.3 presents some directions for future work.

5.1 Key findings

- PPM models can be used to provide probability for an unknown word for a HMM-based token identification system.
- PPM for unknown words performs better than the unified probability.

- The system can be successfully applied to a different corpus without any changes in the code. This makes the system more generalized.
- The amount of training data affects the performance of the system, increasing the amount leads to a better result.
- Shortcomings in the corpus degrade the performance of the system.
- Using PPM's escape Method A as the smoothing method gives the best identification result compared to other methods.

5.2 Discussion

The token identification system developed in this thesis has two motivations. One is to bridge HMMs and PPM models in order to handle unknown words in the input text. It is based on the success of HMMs and PPM models in previous applications (e.g. Bikel et al., 1999; Seymore et al., 1999; Bray, 1999; Witten et al., 1999a).

Starting from this motivation, we first showed that the idea of bridging HMM and PPM is feasible (Section 4.2). The system gets an F-measure of 69.02% when applied on the TCC corpus. It is not as good as would be expected from a system which includes language-dependent components. However, our system is more generalized.

As we know, there is always a trade-off between generality and accuracy. The more specific the application, the higher accuracy a system provides. The undertaken research emphasizes retargetability and generality, therefore, resulting in a lower accuracy. An obvious point where the system is degraded is the lack of a language-dependent component. For example, the model of IdentiFinder (Bikel et al., 1999) contains a word-feature component, such as the format of a numeric number, capitalization, initial of a person's name and the abbreviation of a source or organization. This can be a very small set. However, the model could be improved by these fea-

tures. For instance, the feature of a number helps the system to detect whether it belongs to date, monetary amount, phone/fax number or other. Overall, this system could be efficient on applications which prefer generalization. For applications which ask for higher accuracy, it is not the best choice.

The performance of the identification system with PPM for unknown words is superior to that with a unified probability (Section 4.5). The result is improved by about 4% in terms of F-measure. This is what we expected because the PPM models are character level, therefore, the information provided is more context related.

The system has been tested on another corpus, bibliography corpus, without any changes in the code. It has been found that structure in the text improves the performance of the system. A bibliography entry is so-called semi-structured. For example, it always starts with the author(s), followed by a title, and many sources start with the preposition "In". On average, the value of F-measure is 7.57% higher than that of TCC corpus.

Five classes are in common in TCC and BIB corpus. The system performs the best on date class, with 92% and 73% correct tokens identified in the two corpora. The performance on name class is also good, with 68% and 76% correct tokens identified. On the other hand, it is very poor on organization with an F-measure of 20% for TCC, while none of the organizations is identified for BIB.

We investigated the effect of the quantity of the training data (Section 4.6). Although more data leads to better identification, 24 issues of TCC newsletters achieved an acceptable result. Adding 50% more training data, making it 36 issues, only resulted in an increase of 1.37% in terms of F-measure.

The performance of the system is greatly affected by the quality of the corpus. The evaluation using TCC corpus has been done two times because some obvious errors in the corpus are found after the first evaluation. These errors affect the performance of the system. Therefore, the corpus was corrected manually, and same experiment was run again. As presented in Section 4.4, recall, precision and F-measure are

improved by 6.72%, 11.64% and 8.98%, respectively, by using the correct version of the corpus. And error-rate is improved by 12.65%.

Several smoothing methods have been tested (Section 4.7). The best performance of the system can be obtained by choosing Method A (Section 3.3), one of the escape methods of PPM model, as the smoothing method for the model, while Method D can be used alternatively. Method B and Method C are not recommended, especially the former because it not only marks less tokens, but there are more errors among the tokens.

5.3 Future work

We have contributed to the research area of text mining by developing a domainand language-independent token identification system; and by a novel method for unknown words, combining two well-known language models. However, there is still much work on the path that needs to be done to consolidate and extend the work in this thesis.

First of all, we would like to have a large and correct corpus by which the performance of the system can be evaluated. As the corpus is marked up manually, even if negligence and mistakes can be eliminated, differences in judgment between individuals would still exist. However, a larger amount of data makes the result more stable. We would also like to have more corpora in a wider range of domains, and in different languages, to further test the domain- and language-independence of the system.

As the system is word-based, the definition of a word plays an important role in the performance of the system. In the thesis, space and dot and back slash in email addresses and URLs are used as word delimiters. Other symbols, such as dash and apostrophe, are part of a word. The system will be more adaptable if these nonalphabetic symbols could be handled flexibly. Unknown word handling is an important part for any identification system, especially when training data is insufficient. In this thesis, we have used PPM models to provide the probability for unknown words. The largest probability predicted by a particular PPM model is used. In this way, the PPM model does not affect the state which the HMM is currently in. An obvious extension for the work in the thesis is that when the system encounters an unknown word, the HMM is forced to the corresponding state that represents the word best by providing the largest probability.

Another extension which we believe has potential is to increase the order of the HMM. Second-order HMMs have been successfully used for part-of-speech tagging (Thede and Harper, 1999). How it works for token identification deserves further exploration. However, it is assumed that more advanced or complicated smoothing methods will be necessary because the higher order will certainly cause more serious zero-frequency problems.

Bibliography

- Aberdeen, J., Burger, J., Day, D., Hirschman, L., Robinson, P. and Vilain, M. (1995). MITRE: Description of the alembic system as used for MUC-6. In *Proceedings of the Sixth Message Understanding Conference (MUC-6)* (pp. 141–155). Columbia, MD: Morgan Kaufmann Publishers.
- Appelt, D., Hobbs, J., Bear, J., Israel, D. and Tyson, M. (1993). FASTUS: A finitestate processor for information extraction from real-world text. In *Proceedings* of the 13th International Joint Conference on Artificial Intelligence (IJCAI-93) (pp. 1172–1178). Chambery, France.
- Baluja, S., Mittal, V. O. and Sukthankar, R. (1999). Applying machine learning for high performance named-entity extraction. In *Proceedings of the Conference of the Pacific Association for Computational Linguistics* (pp. 365–378). Waterloo, CA.
- Baum, L. E. (1972). An inequality and associated maximization technique in statistical estimation for probabilistic functions of a Markov process. *Inequalities*, 3, 1–8.
- Beeferman, D., Berger, A. and Lafferty, J. (1999). Statistical models for text segmentation. *Machine learning, special issue on Natural Language Processing*, 34(1-3), 177–210.
- Bell, T., Cleary, J. and Witten, I. (1990). *Text Compression*. Englewood Cliffs, NJ: Prentice-Hall.

- Bennett, S. W., Aone, C. and Lovell, C. (1997). Learning to tag multilingual texts through observation. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing* (pp. 109–116). Providence, Rhode Island.
- Bikel, D. M., Miller, S., Schwartz, R. and Weischedel, R. (1997). Nymble: a highperformance learning namefinder. In *Proceedings of the Fifth Conference on Applied Natural Language Processing (ANLP-97)* (pp. 194–201).
- Bikel, D. M., Schwartz, R. and Weischedel, R. M. (1999). An algorithm that learns what's in a name. *Machine Learning Journal*, *34*, 211–231.
- Borthwick, A., Sterling, J., Agichtein, E. and Grishman, R. (1998). Exploiting diverse knowledge sources via maximum entropy in named entity recognition. In *Proceedings of the Sixth Workshop on Very Large Corpora*. Montreal, Canada.
- Bray, Z. C. (1999). Using compression models for text mining. Master's thesis, Department of Computer Science, University of Waikato, Private Bag 3105, Hamilton, New Zealand.
- Brill, E. (1995). Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics*, 21(4), 543–566.
- Chen, S. F. and Goodman, J. (1998). An empirical study of smoothing techniques for language modeling. Technical Report TR-10-98, Harvard University.
- Cleary, J. G., Teahan, W. J. and Witten, I. H. (1995). Unbounded length contexts for PPM. In Stover, J. A. and Cohn, M. (Eds.), *Proceedings DCC'95*. IEEE Computer Society Press.
- Cleary, J. G. and Witten, I. H. (1984). Data compression using adaptive coding and partial string matching. *IEEE Trans on Communications*, *32*(*4*), 396–402.
- Cucerzan, S. and Yarowsky, D. (1999). Language independent named entity recognition combining morphological and contextual evidence. In *Proceedings of*

the 1999 Joint SIGDAT Conference on Empirical Methods in NLP and Very Large Corpora (pp. 90–99). College Park, MD.

- Daciuk, J. (1999). Treatment of unknown words. In *Workshop on Implementing Automata WIA'99*. Potsdam, Germany.
- Dzeroski, S. (1996). Inductive logic programming and knowledge discovery in databases. Advances in Knowledge Discovery and Data Mining (pp. 117–152). AAAI Press / The MIT Press.
- Forney, J. G. D. (1973). The Viterbi algorithm. *Proceedings of the IEEE*, 61(3), 268–278.
- Grishman, R. (1997). Information extraction: Techniques and challenges. In Information Extraction (International Summer School SCIE-97). Springer-Verlag: Maria Teresa Pazienza.
- Grishman, R. and Sundheim, B. (1996). Message understanding conference 6: A brief history. In Proceedings of the 16th International Conference on Computational Linguistics. Copenhage.
- Howard, P. G. (1993). *The design and analysis of efficient lossless data compression systems*. PhD thesis, Brown University, Providence, RI.
- Iwanska, L., Croll, M., Yoon, T. and Adams., M. (1995). Wayne State University: Description of the UNO natural language processing system as used for MUC-6. In *Proceedings of the Sixth Message Understanding Conference (MUC-6)*. Columbia, MD: NIST, Morgan Kaufmann Publishers.
- Jelinek, F. and Mercer, R. L. (1980). Interpolated estimation of Markov source parameters from sparse data. In *Proceedings of the Workshop on Pattern Recognition in Practice*. Amsterdam, The Netherlands: North-Holland.
- Katz, S. M. (1987). 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.

- Kneser, R. and Ney, H. (1995). Improved backing-off for m-gram language modeling. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, Volume 1 (pp. 181–184). Detroit, Michigan.
- Lawrence, S. and Giles, C. L. (1999). Accessibility of information on the web. *Nature*, 400(6740), 107–109.
- Lewis, D. D. (1995). Evaluating and optimizing autonomous text classification systems. In Proceedings of the Eighteenth Annual International ACM Special Interest Group on Information Retrieval (pp. 246–254).
- McDonald, D. D. (1996). Internal and external evidence in the identification and semantic categorization of proper names. In B. Boguraev and J. Pustejovsky (Eds.), *Corpus Processing for Lexical Aquisition* (pp. 21–39). Cambridge, MA: The MIT Press.
- Merkl, D. (1998). Text data mining. In A Handbook of Natural Language Processing: Techniques and Applications for the Processing of Language as Text. New York: Marcel Dekker.
- Mikheev, A. (1997). Automatic rule induction for unknown-word guessing. *Computational Linguistics*, 23(3), 405–423.
- Mikheev, A., Moens, M. and Grover, C. (1999). Named entity recognition without gazetteers. In *Proceedings of EACL*. Bergen, Norway.
- Moffat, A. (1990). Implementing the PPM data compression scheme. *IEEE Trans*actions on Communication, 38(11), 1917–1921.
- Moffat, A., Bell, T. C. and Witten, I. H. (1997). Lossless compression for text and images. *International Journal of High Speed Electronics and Systems*, 8(1), 179–231.
- Morgan, R., Garigliano, R., Callaghan, P., Poria, S., Smith, M., Urbanowicz, A., Collingham, R., Costantino, M., Cooper, C. and the LOLITA Group (1995).

University of Durham: Description of the LOLITA system as used for MUC6. In *Proceedings of the Sixth Message Understanding Conference (MUC-6)*.
Columbia, MD: NIST, Morgan Kaufmann Publishers.

- Ney, H., Essen, U. and Kneser, R. (1994). On structuring probabilistic dependencies in stochastic language modeling. *Computer Speech and Language*, 8(1), 1–38.
- Poritz, A. B. (1988). Hidden Markov models: A guided tour. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '88)* (pp. 7–13).
- Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2), 257–286.
- Ratnaparkhi, A. (1996). A maximum entropy model for part-of-speech tagging. In Brill, E. and Church, K. (Eds.), *Proceedings of the Conference on Empirical Methods in Natural Language Processing* (pp. 133–142). University of Pennsylvania, Philadelphia, Pa.
- Rosenfeld, R. (1994). Adaptive Statistical Language Modeling: A Maximum Entropy Approach. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.
- Ryan, M. S. and Nudd, G. R. (1993). The Viterbi algorithm. Research Report CS-RR-238, Department of Computer Science, University of Warwick, Coventry, UK.
- Sekine, S. (1998). NYU: Description of the Japanese NE system used for MET-2. In *Proceedings of MUC-7 1998*.
- Seymore, K., McCallum, A. and Rosenfeld., R. (1999). Learning hidden Markov model structure for information extraction. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence: Workshop on Machine Learning for Information Extraction* (pp. 37–42). Orlando, FL.

- Stevenson, M. and Gaizauskas, R. (2000). Using corpus-derived name lists for named entity recognition. In Proceedings of the Sixth Conference on Applied Natural Language Processing and First Conference of the North American Chapter of the Association for Computational Linguistics (pp. 290–296.). Seattle.
- Tan, A.-H. (1999). Text mining: Promises and challenges. In South East Asia Regional Computer Confederation (SEARCC'99). Westin Stamford Hote, Singapore.
- Teahan, W. J. (1998). *Modelling English Text*. PhD thesis, Department of Computer Science, University of Waikato, Private Bag 3105, Hamilton, New Zealand.
- Teahan, W. J., Wen, Y., McNab, R. and Witten, I. H. (2000). A compression-based algorithm for Chinese word segmentation. *Computational Linguistics*, 26(3), 375–393.
- Thede, S. M. and Harper, M. P. (1999). A second-order hidden Markov model for part-of-speech tagging. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, Association for Computational Linguistics (pp. 175–182).
- Van Rijsbergen, C. J. (1979). *Information Retrieval* (Second Ed.). London: Butterworths.
- Viterbi, A. J. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theroy*, *IT-13(2)*, 260–269.
- Witten, I., Bray, Z., Mahoui, M. and Teahan, W. (1999a). Text mining: a new frontier for lossless compression. In *Proceedings Data Compression Conference* (DCC'99) (pp. 198–207). Snowbird, Utah.
- Witten, I. H. and Bell, T. C. (1991). The zero-frequency problem: Estimating the

probabilities of novel events in adaptive text compression. *IEEE Transactions* on *Information Theory*, *37*(*4*), 1085–1093.

- Witten, I. H., Bray, Z., Mahoui, M. and Teahan, W. J. (1999b). Using language models for generic entity extraction. In *ICML-99 Workshop on Machine learning in text data analysis*. Stockholm, Sweden.
- Witten, I. H. and Frank, E. (2000). Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. San Francisco, CA.: Morgan Kaufmann Publishers.
- Witten, I. H., Moffat, A. and Bell, T. C. (1999c). Managing Gigabytes-Compressing and Indexing Documents and Images (2 Ed.). ISBN 1-55860-570-3. San Francisco, California: Morgan Kaufmann Publishers.
- Yeates, S., Witten, I. H. and Bainbridge, D. (2001). Tag insertion complexity. In Storer, J. A. and Cohn, M. (Eds.), *Proceedings of Data Compression Conference* (pp. 243–252). Snowbird, Utah, US.
- Zipf, G. K. (1965). Human Behavior and the Principle of Least Effort: an Introduction to Human Ecology (Facsimile of 1949 edition Ed.). Hafner Publishing Company.