

Text classification and segmentation using minimum cross-entropy

W. J. Teahan

School of Computing and Mathematical Sciences,
The Robert Gordon University, Aberdeen, Scotland
Email: *wjt@scms.rgu.ac.uk* Phone: +44 (0)1224 262737

Abstract

Several methods for classifying and segmenting text are described. These are based on ranking text sequences by their *cross-entropy* calculated using a fixed order character-based Markov model adapted from the PPM text compression algorithm. Experimental results show that the methods are a significant improvement over previously used methods in a number of areas. For example, text can be classified with a very high degree of accuracy by authorship, language, dialect and genre. Highly accurate text segmentation is also possible—the accuracy of the PPM-based Chinese word segmenter is close to 99% on Chinese news text; similarly, a PPM-based method of segmenting text by language achieves an accuracy of over 99%.

1 Introduction

Having a computer model that is able to predict and model natural language as well as a human is critical for cryptology, language identification, authorship attribution, text correction, and text segmentation. In this paper, it is shown how character based algorithms based on the text compression scheme PPM (for “prediction by partial match”) can be used to construct language models that can be applied successfully to each of these problems. Many of these applications require little or no knowledge of the text being analysed to work successfully.

The Markov modelling based approach (as used in PPM models) has been previously applied to many problems in natural language processing such as language identification (Dunning, 1994), optical character recognition (Chen, 1996), automatic spelling correction (Kukich, 1992; Kernighan, Church & Gale, 1990) and continuous speech recognition (Jelinek, 1990; Bahl, Jelinek & Mercer, 1983). The most successful types of language models are called n -gram models (which base the probability of a word on the preceding n words) and n -pos models (which base the probability on the preceding words and parts of speech). Models based on characters have also been tried, although they do not feature as prominently in the literature. They have been most vigorously applied to the domain of text compression (Teahan, 1998; Ristad & Thomas, 1995; Bell, Cleary & Witten, 1990) but also to other application domains, such as cryptography (Irvine, 1997), language identification (Ganeson & Sherman, 1993) and various applications for automatically correcting words in texts such as OCR and spell-checking (Kukich, 1992).

In this paper, a PPM model is applied to several problems in text classification and segmentation. Although the methods described here are not new, the novelty is in the adoption of the PPM model for the language model. Other language modelling approaches include hidden Markov modelling (Charniak, 1993; Jelinek, 1990), context-free grammars (Chen, 1996), decision trees (Bahl *et al.*, 1989), local rules (Brill, 1993), collocational matrices (Garside, Leech & Simpson, 1987) and neural networks (Schmid, 1994).

The performance of the PPM model in many applications is superior to previously published methods. For example, results described in Sections 5.2 and 5.3 show how a variant of the PPM

text compression algorithm can be applied to the problem of segmenting words in both Chinese and English text to achieve an accuracy of 99%. The use of the character based model required substantially less training text than other methods; for example, the 5 Mbyte Brown Corpus (Francis & Kučera, 1982) was found to perform better than a previously published word-based model (Ponte & Croft, 1996) trained on 1 Gbyte of text.

A PPM model has also been used successfully in several novel applications. Section 4.1 describes how PPM can be used to identify language dialects (such as American and British English) and the period for historical texts (such as Old and Middle English). Section 4.2 applies PPM to a famous problem in authorship attribution—that of determining the authorship of the Federalist papers—and arrives at the same conclusions as other studies on the subject (Mosteller & Wallace, 1984). Section 4.3 shows how PPM can classify text by genre (such as whether it is about politics, religion or sport). Section 5.4 demonstrates how PPM can locate different languages in text with a very high degree of accuracy.

The purpose of this paper is threefold:

- firstly, to show that a PPM model can be applied successfully to a wide range of applications in natural language processing;
- secondly, to show that in certain cases the performance of the PPM model is superior to other methods;
- and thirdly, to show how the PPM model can be applied to several novel applications.

This paper is organized as follows. A method of classifying text using minimum cross-entropy against multiple language models is discussed in the next section. One particular method of calculating the cross-entropy is then described using the text compression scheme PPM. Several applications of PPM-based text classification are explored: language and dialect identification, authorship ascription and classification by genre. Another class of text correction based algorithms are also described. These algorithms are then applied to two specific applications—word segmentation, and segmenting text by language.

2 Minimum cross-entropy as a text classifier

In information theory (Shannon, 1948), the fundamental coding theorem states that the lower bound to the average number of bits per symbol needed to encode a message (i.e., a sequence of text) is given by its *entropy*:

$$H(P) = - \sum_{i=1}^k p(x_i) \log p(x_i)$$

where the probabilities are independent and sum to 1 and there are k possible symbols with probability distribution $P = p(x_1), p(x_2), \dots, p(x_k)$. The entropy is a measure of how much uncertainty is involved in the selection of a symbol—the greater the entropy, the greater the uncertainty. It can also be considered a measure of the “information content” of the message—more probable messages convey less information than less probable ones. The above equation can be extended to the more general case for a language with probability distribution L :

$$H(L) = \lim_{m \rightarrow \infty} - \frac{1}{m} \sum p(x_1, x_2, \dots, x_m) \log p(x_1, x_2, \dots, x_m).$$

This is called the *entropy of a language* and can be considered to be the limit of the entropy as the length of the message gets very large. Usually the true probability distribution L is not known. However, an upper bound to $H(L)$ can still be obtained by using a model M as an approximation to L :

$$H(L, M) = - \sum p_M(x_1, x_2, \dots, x_m) \log p_M(x_1, x_2, \dots, x_m)$$

where the correct model is $p(x_1, x_2, \dots, x_m)$ and the probabilities are estimated by the model $p_M(x_1, x_2, \dots, x_m)$. $H(L, M)$ is called the *cross-entropy*, and is always greater than or equal to the entropy $H(L)$, as this is based on the best possible language model, the source itself:

$$H(L) \leq H(L, M).$$

Text compression can be used directly to estimate an upper bound to the entropy (Brown *et al.*, 1992). If $b_M(x_1x_2 \dots x_m)$ is the number of bits required to encode the string $x_1x_2 \dots x_m$ using model M , then

$$H(L, M) = \lim_{m \rightarrow \infty} \frac{1}{m} b_M(x_1x_2 \dots x_m)$$

where $H(L, M)$ is the number of bits per symbol required to encode a long sequence of text drawn from L .

The cross-entropy is relevant as it provides a measure of how well the approximate model is doing on the test text—the closer it is to $H(L)$, the less inaccurate the model is. It also gives a useful yardstick for comparing the accuracy of competing models. This is done by computing the cross-entropy for each model, and the model with the smallest cross-entropy is inferred to be the “best”. Information is extracted from the text by associating with each model a semantic label (such as whether it was trained on English text, written by William Shakespeare or about sport or religion) and then choosing the label associated with the “best” model to classify the text.

The next section discusses one particular method of calculating the cross-entropy based on the PPM text compression scheme. A description of other text compression schemes may be found in Bell, Cleary & Witten (1990).

3 PPM: Prediction by partial match

PPM has set the performance standard in lossless compression of text throughout the past decade. The original algorithm was first published in 1984 by Cleary & Witten (1984b), and a series of improvements was described by Moffat, culminating in a careful implementation, called PPMC, which has become the benchmark version (Moffat, 1990). This still achieves results superior to virtually all other compression methods, despite many attempts to better it. Other methods such as those based on Ziv-Lempel coding (Ziv & Lempel, 1978; Ziv & Lempel, 1977) are more commonly used in practice, but their attractiveness lies in their relative speed rather than any superiority in compression—indeed, their compression performance generally falls below that of PPM in benchmark tests (Bell, Cleary & Witten, 1990).

Prediction by partial matching, or PPM, is a finite-context statistical modelling technique that can be viewed as blending together several fixed-order context models to predict the next character in the input sequence. Prediction probabilities for each context in the model are calculated from frequency counts which are updated adaptively; and the symbol that actually occurs is encoded relative to its predicted distribution using arithmetic coding (Moffat, Neal & Witten, 1995; Witten, Neal & Cleary, 1987). The maximum context length is a fixed constant, and it has been found that increasing it beyond about five does not generally improve compression (Cleary, Teahan & Witten, 1995; Moffat, 1990; Cleary & Witten, 1984b).

PPM is a character-based model which uses the last few characters in the input stream to predict the upcoming one. Models that condition their predictions on a few immediately preceding symbols are called “finite-context” models of order k , where k is the number of preceding symbols used. PPM employs a suite of fixed-order context models with different values of k , up to some pre-determined maximum, to predict upcoming characters.

Order $k = 2$			Order $k = 1$			Order $k = 0$			Order $k = -1$		
Predictions	c	p	Predictions	c	p	Predictions	c	p	Predictions	c	p
ab	→ r	2 $\frac{2}{3}$	a	→ b	2 $\frac{2}{7}$	→ a	5 $\frac{5}{16}$		→ A	1 $\frac{1}{ A }$	
	→ Esc	1 $\frac{1}{3}$		→ c	1 $\frac{1}{7}$	→ b	2 $\frac{2}{16}$				
				→ d	1 $\frac{1}{7}$	→ c	1 $\frac{1}{16}$				
ac	→ a	1 $\frac{1}{2}$		→ Esc	3 $\frac{3}{7}$	→ d	1 $\frac{1}{16}$				
	→ Esc	1 $\frac{1}{2}$	b	→ r	2 $\frac{2}{3}$	→ r	2 $\frac{2}{16}$				
				→ Esc	1 $\frac{1}{3}$	→ Esc	5 $\frac{5}{16}$				
ad	→ a	1 $\frac{1}{2}$									
	→ Esc	1 $\frac{1}{2}$	c	→ a	1 $\frac{1}{2}$						
				→ Esc	1 $\frac{1}{2}$						
br	→ a	2 $\frac{2}{3}$									
	→ Esc	1 $\frac{1}{3}$	d	→ a	1 $\frac{1}{2}$						
				→ Esc	1 $\frac{1}{2}$						
ca	→ d	1 $\frac{1}{2}$	r	→ a	2 $\frac{1}{3}$						
	→ Esc	1 $\frac{1}{2}$		→ Esc	1 $\frac{1}{3}$						
da	→ b	1 $\frac{1}{2}$									
	→ Esc	1 $\frac{1}{2}$									
ra	→ c	1 $\frac{1}{2}$									
	→ Esc	1 $\frac{1}{2}$									

Table 1: PPMC model after processing the string *abracadabra* (maximum order 2)

For each model, note is kept of all characters that have followed every length- k subsequence observed so far in the input, and the number of times that each has occurred. Prediction probabilities are calculated from these counts. The probabilities associated with each character that has followed the last k characters in the past are used to predict the upcoming character. Thus from each model, a separate predicted probability distribution is obtained.

These distributions are effectively combined into a single one, and arithmetic coding is used to encode the character that actually occurs, relative to that distribution. The combination is achieved through the use of “escape” probabilities. Recall that each model has a different value of k . The model with the largest k is, by default, the one used for coding. However, if a novel character is encountered in this context, which means that the context cannot be used for encoding it, an escape symbol is transmitted to signal the decoder to switch to the model with the next smaller value of k . The process continues until a model is reached in which the character is not novel, at which point it is encoded with respect to the distribution predicted by that model. To ensure that the process terminates, a model is assumed to be present below the lowest level, containing all characters in the coding alphabet. This mechanism effectively blends the different order models together in a proportion that depends on the values actually used for escape probabilities. Formally, given $S = c_1, c_2, \dots, c_m$, then

$$p(S) = \prod_{i=1}^m p'(c_i | c_{i-5}, c_{i-4}, c_{i-3}, c_{i-2}, c_{i-1})$$

where p' gives the probabilities returned by the order 5 PPM character model.

3.1 An example

As an illustration of the operation of PPM, Table 1 shows the state of the four models with $k = 2, 1, 0$, and -1 after the input string *abracadabra* has been processed. For each model, all previously occurring contexts are shown with their associated predictions, along with occurrence counts c and the probabilities p that are calculated from them. By convention,

character	probabilities encoded (without exclusions)	probabilities encoded (with exclusions)	codespace occupied
c	$\frac{1}{2}$	$\frac{1}{2}$	$-\log_2 \frac{1}{2} = 1$ bit
d	$\frac{1}{2}, \frac{1}{7}$	$\frac{1}{2}, \frac{1}{6}$	$-\log_2(\frac{1}{2} \cdot \frac{1}{6}) = 3.6$ bits
t	$\frac{1}{2}, \frac{3}{7}, \frac{5}{16}, \frac{1}{ A }$	$\frac{1}{2}, \frac{3}{6}, \frac{5}{12}, \frac{1}{ A -5}$	$-\log_2(\frac{1}{2} \cdot \frac{3}{6} \cdot \frac{5}{12} \cdot \frac{1}{251}) = 11.2$ bits

Table 2: Encodings for three sample characters using the model in Table 1

$k = -1$ designates the bottom-level model that predicts all characters equally; it gives them each an initial probability $\frac{1}{|A|}$ where A is the alphabet used.

Some policy must be adopted for choosing the probabilities to be associated with the escape events. There is no sound theoretical basis for any particular choice in the absence of some *a priori* assumption on the nature of the symbol source; some alternatives are evaluated in Witten & Bell (1991) and Bunton (1997). The method used in the example, commonly called “Method C,” gives a count to the escape event equal to the number of different symbols that have been seen in the context so far (Moffat, 1990); thus, for example, in the order-0 column of Table 1 the escape symbol receives a count of 5 because five different symbols have been seen in that context. Other methods for estimating the escape probability are described in Section 3.2.

Sample encodings using these models are shown in Table 2. As noted above, prediction proceeds from the highest-order model ($k = 2$). If the context successfully predicts the next character in the input sequence, the associated probability p is used to encode it. For example, if c followed the string *abracadabra*, the prediction $ra \rightarrow c$ would be used to encode it with a probability of $\frac{1}{2}$; that is, in one bit.

Suppose instead that the character following *abracadabra* was d . This is not predicted from the current $k = 2$ context ra . Consequently, an escape event occurs in context ra , which is coded with a probability of $\frac{1}{2}$, and then the $k = 1$ context a is used. This does predict the desired symbol through the prediction $a \rightarrow d$, with probability $\frac{1}{7}$. In fact, a more accurate estimate of the prediction probability in this context is obtained by noting that the character c cannot possibly occur, since if it did it would have been encoded at the $k = 2$ level. This mechanism, called *exclusion*, corrects the probability to $\frac{1}{6}$ as shown in the third column of Table 2. Finally, the total number of bits needed to encode the d can be calculated to be 3.6.

If the next character were one that had never been encountered before, say t , escaping would take place repeatedly right down to the base level $k = -1$. Once this level is reached, all symbols are equiprobable—except that, through the exclusion device, there is no need to reserve probability space for symbols that already appear at higher levels. Assuming a 256-character alphabet, the t is coded with probability $\frac{1}{251}$ at the base level, leading to a total requirement of 11.2 bits including those needed to specify the three escapes. This example highlights how the probabilities allocated to different symbols can be either arbitrarily small or large. In comparison, an order 0 static model that assigns the same probability $\frac{1}{256}$ to each character in the alphabet requires 8 bits to encode each of them.

A further improvement called *update exclusion* (Moffat, 1990) is based on how the counts for each context are updated. When exclusions are being applied, only one context is performing the predictions. This suggests a modification to how the counts are updated—the counts are updated only if they are not predicted by any higher order context. Consequently, the counts more accurately reflect which symbols are likely to have been excluded by the higher order contexts. Bell, Cleary & Witten (1990) state that update exclusion generally improves compression by 2%, as well as reducing the time spent updating the counts.

3.2 Estimating the probabilities

Various methods have been proposed for computing the escape and symbol probabilities for the context models. The probabilities are based on the frequency counts of symbols that have followed the context each time it has occurred in the text. The most commonly used methods are Method C, proposed by Moffat (1990) and Method D by Howard (1993). Method C uses the number of times a novel symbol has occurred before (i.e., the number of types) as the basis of its probability. Let $c(\phi)$ be the number of times a particular context was followed by the symbol ϕ ; n be the number of tokens that have followed, i.e., the sum of the counts for all symbols, $\sum_{\phi} c(\phi)$; and t be the number of types i.e. the number of unique symbols. Then we have

$$e = \frac{t}{n + t} \quad \text{and} \quad p(\phi) = \frac{c(\phi)}{n + t}$$

where e is the escape probability, $p(\phi)$ is the probability for symbol ϕ , given a particular context model,

Method D is a minor modification to method C. Whenever a novel event occurs, rather than adding one to both the symbol and escape counts, $\frac{1}{2}$ is added instead:

$$e = \frac{t}{2n} \quad \text{and} \quad p(\phi) = \frac{2c(\phi) - 1}{2n}.$$

Other methods are described and their performance compared in Witten & Bell (1991).

The different implementations of PPM are usually identified by the escape method each of them uses. For example, PPMA stands for “prediction by partial match using escape method A”. Likewise, PPMC and PPMD use the escape methods C and D. The maximum order of the models may also be included; for example, PPMD5 is a fixed order 5 implementation of PPM that uses escape method D.

3.3 PPM-based language models

PPM character based language models have been applied successfully to many applications in natural language processing (Teahan, 1998) such as language identification and cryptography (Irvine, 1997), and various applications for automatically correcting words in texts such as OCR and word segmentation. Experiments with English text show that PPM models with an upper bound of five characters in their context perform competitively against other PPM models based on shorter or longer length contexts (Teahan, 1998). Performance of these models can be substantially improved by training them on large amounts of related text (i.e. text of the same language, authorship or genre), but even unrelated text has been found to work well.

PPM can also be used to compare the performance of human and computer based models. Teahan & Cleary (1996) have shown how PPM predicts English text almost as well as humans. They performed experiments on the same text that Claude E. Shannon used in a famous experiment to estimate the entropy of English (Shannon, 1951). Shannon’s estimates were based on humans guessing the upcoming text, letter by letter. Teahan and Cleary used the PPM scheme to build a computer based model, and found that performance was close to, and in some cases, superior to human results.

Katz’s (1987) backoff method (a method which is widespread in the language modelling literature) uses a similar blending mechanism to PPM to “smooth” the probabilities—in both methods, higher-order models are defined recursively in terms of lower-order models. The essential difference between the methods is how much weight is given to estimates from the lower order models. PPM weights the different order models using the escape mechanism. Unlike Katz’s method, all counts greater than 0 (rather than 5) are assigned the same weight, but each of these counts is effectively “discounted” to make room for the escape count, which is used for backing-off to a lower order model when a zero-count is encountered. The major differences

between PPM and Katz's method is two-fold: firstly, PPM excludes estimates from lower order models that have already been predicted by higher order models (using the full exclusion mechanism); and secondly, PPM's counts are incrementally updated on-line using the update exclusion mechanism.

4 PPM-based text classification

Text classification is the problem of assigning text to any of a set of pre-specified categories. It is useful in indexing documents for later retrieval, as a stage in natural language processing systems, for content analysis, and in many other roles (Lewis & Hayes, 1994).

For PPM-based classification applications, the test text is encoded using PPM models that have been pre-trained on text that is (hopefully) representative of the particular style of text being modelled. For example, the style could be a particular language (i.e. English, French or German); it could be a particular author (Jane Austen, Thomas Jefferson or William Shakespeare); or it could be a particular genre (politics, sport or religion). Results presented in the next few sections show that cross-entropy calculated in this manner can accurately distinguish language, dialect (i.e. British and American English) and authorship. Section 4.1 describes how PPM can be used to identify language dialects (such as American and British English) and the period for historical texts (such as Old and Middle English). Section 4.2 applies PPM to a famous problem in authorship attribution—that of determining the authorship of the Federalist papers—and arrives at the same conclusions as other studies on the subject (Mosteller & Wallace, 1984). Section 4.3 then investigates the more difficult problem of classifying by genre. A novel method of hierarchical classification is also presented which can significantly speed up classification without much loss of accuracy when there are a large number of classes.

4.1 Language and dialect identification

Language identification concerns the problem of identifying from examples of written or spoken language which language is being used. Dunning (1994) describes several statistical methods for identifying the language of small pieces of text (such as *e pruebas bioquimica* and *faits se sont produits*). Ganesan and Sherman (1993) adopt various statistical modelling methods to perform various language recognition problems such as: recognizing a known language; distinguishing a known language from uniform noise; and detecting a non-uniform unknown language. Cross-entropy calculated using PPM-based models may equally be applied to each of these problems as it provides the means to rank the performance of statistical models. The text being analysed is compressed using PPM character based language models trained on representative texts written in various languages. The most likely language is the one used to train the model that has the best compression performance.

Table 3 shows how this works for a sample of text taken from the Bible's *Book of Genesis*. Public domain versions of the Bible in six different languages were obtained—English, French, German, Italian, Latin and Spanish (*McFarlin Library Research Guides*, 1977). Sample texts of the first 10,000 characters found in the Book of Genesis were then compressed using order 5 PPM character models trained on the remaining Genesis text for the six different languages. The compression ratios are shown in the figure in bits per character for all the 36 permutations of training model and original sample text, and the best performed model for each sample text highlighted in bold font. Also shown are the compression ratios for the sample texts without training. In all cases, the best performed model is the one trained with the same language as the original text. Interestingly, the untrained model also consistently outperforms models trained on the other languages by about the same margin in each case (15%). The same process can also be used to identify the language period for historical texts such as Old, Middle and Early Modern English (Teahan, 1998).

Another interesting possibility is the problem of identifying the dialect for a sample text.

Original text ¹	Training text ²						
	(Compression ratio in bpc)						
	<i>Untrained</i>	<i>English</i>	<i>French</i>	<i>German</i>	<i>Italian</i>	<i>Latin</i>	<i>Spanish</i>
<i>English</i>	1.97	1.56	2.30	2.38	2.30	2.33	2.29
<i>French</i>	2.13	2.54	1.71	2.56	2.54	2.59	2.56
<i>German</i>	2.14	2.64	2.60	1.75	2.55	2.59	2.59
<i>Italian</i>	2.26	2.67	2.66	2.65	1.83	2.61	2.68
<i>Latin</i>	2.45	2.87	2.92	2.91	2.82	1.97	2.84
<i>Spanish</i>	2.19	2.57	2.61	2.61	2.57	2.57	1.75
Size of training text (chars.)		180,359	175,331	191,840	170,923	149,121	169,461

1. The “original text” is taken from the first 10,000 characters of the Book of Genesis.
2. The “training text” is all the remaining characters from the Book of Genesis for each of the six different languages.

Table 3: Identifying the text for six different languages.

The Brown and LOB (Johansson *et al.*, 1986) corpora represent diverse samples from two different dialects, American and British English; as such, they provide a means for rigorously testing this possibility. In an experiment, equal-size partitions of 100,000 characters were successively extracted from the two corpora, and then compressed using order 5 PPMD models trained on the remaining text from each (i.e. all the text in the corpora with just the partition deleted). Improvements in compression performance between the Brown-trained or LOB-trained models for each partition vary up to 3.6% for the Brown Corpus and 3.1% for the LOB Corpus. There were no cases with a decrease in performance, and only three cases had no difference in performance. These results show that the method chooses the correct classification (LOB-trained model performing better for “British” text, or Brown-trained model performing better for “American” text) in almost every case.

4.2 Authorship ascription

A similar process is possible for determining authorship. It is widely known that style and statistical properties differ markedly for different authors (Bogges, Argarwal & Davis, 1991). This can be exploited by training models using text written by each of the disputed authors, and then compressing each text in question using all these models. The hope is that the style of each author will be sufficiently different to clearly distinguish between them. Experiments with English texts show that this is true in most cases (Teahan, 1998).

A famous problem in authorship attribution concerns the Federalist Papers written by Alexander Hamilton, John Jay and James Madison. Between the years 1787–1788, eighty-five short essays addressed to the citizens of New York on the U.S. Constitution were published in newspapers under the pseudonym “Publius”. Of these papers, it is generally agreed that Jay wrote 5, Hamilton 43, Madison 14 with the authorship of the remaining 12 being in flat dispute between Hamilton and Madison.

Mosteller and Wallace (1984) determine odds of authorship for the papers by applying Bayes’ theorem to evidence based upon 30 common words (such as *as* and *upon*). Their work supports the claim made by historians (and by Madison himself) that Madison wrote the disputed papers.

Experiments with PPM character based computer models also supports this claim.* Each of the disputed papers was compressed using computer models trained on text taken from the non-disputed papers. The same models were also used to test papers with known authorship. To minimize the fluctuations due to differing periods, the papers tested are all in consecutive order from paper number 45 through to 66 (excluding paper number 64 written by John Jay).

*Similar results have been reported by Juola (1998) who used a different method of calculating the cross-entropy.

Disputed papers

No.	Madison (bpc)	Hamilton (bpc)	No.	Madison (bpc)	Hamilton (bpc)
49	1.79	1.93	55	1.86	1.97
50	1.92	2.07	56	1.70	1.85
51	1.72	1.88	57	1.85	1.98
52	1.78	1.94	58	1.80	1.93
53	1.79	1.93	62	1.83	1.84
54	1.73	1.89	63	1.82	1.82

*Papers known to have
been written by Madison**Papers known to have
been written by Hamilton*

No.	Madison (bpc)	Hamilton (bpc)	No.	Madison (bpc)	Hamilton (bpc)
44	1.76	1.90	59	1.82	1.88
45	1.67	1.81	60	1.78	1.71
46	1.78	1.85	61	1.78	1.73
47	1.70	1.81	65	1.87	1.82
48	1.85	1.99	66	1.80	1.75

Figure 1: Ascribing authorship to the Federalist Papers

Figure 1 lists the results obtained. Compression ratios for Madison are much lower for the disputed papers (excluding 62 and 63), supporting the claim that Madison was the primary author for those papers. For the non-disputed papers, only one paper (59) has a result which is contradictory. Mosteller and Wallace state that historians feel least settled about papers 62 and 63, which is reflected in the results. They draw attention to the difficulty of the task confronting researchers—both Hamilton and Madison had remarkably similar prose styles, which was no unique phenomenon as for most educated Americans of the late eighteenth century they employed “the same stylistic devices, the same standard phrases, and remarkably similar sentence structure.” These results are also surprising in that only a relatively small amount of training text (a few hundred kilobytes) was required to train the models.

4.3 Classification by genre

A preliminary investigation into the effectiveness of using PPM to classify by genre was performed using the `Newsgroups` data set (McCallum & Nigam, 1998). This data set contains 20,000 articles evenly divided among 20 Usenet discussion groups. This data set is particularly interesting as many of the newsgroup categories fall into confusable clusters: for example, three discuss religion (`soc.religion.christian`, `talk.religion.misc`, `alt.atheism`), three discuss politics (from the `talk.politics.*` hierarchy) and five are from the `comp.*` hierarchy. In an initial experiment, 20 separate order 5 PPMD models were trained on the text contained in the first 80% of the articles, and the remaining 20% of the articles were compressed against each of them to see which model performed best. The Usenet headers (including the subject, followup and newsgroup lines) were removed from each of the articles beforehand. The newsgroup category was correctly deduced in 82.1% of the articles, which compares favourably with the results reported by McCallum & Nigam who achieved 74% accuracy using a multi-variate Bernoulli event model, and 85% using a multinomial model using the same training/testing split of the data.

A confusion matrix of the errors made by the PPM-based classifier for ten of the newsgroups is shown in Table 4. As is typical for a PPM-based classifier, the leading diagonal correctly

	<i>aa</i>	<i>sc</i>	<i>se</i>	<i>sm</i>	<i>ss</i>	<i>src</i>	<i>tpg</i>	<i>tpme</i>	<i>tpmi</i>	<i>trm</i>
<i>aa</i>	149				2	4		1	2	40
<i>sc</i>		177	3	3	1	1	4		2	
<i>se</i>	2	3	136	13	4			1	3	
<i>sm</i>	2	1	4	165	3	1	2		6	2
<i>ss</i>		1	3	1	172		1	1	8	5
<i>src</i>						191	1			3
<i>tpg</i>	1			2	1	2	171		11	9
<i>tpme</i>	1					5		185	6	1
<i>tpmi</i>	1				2	1	19	12	139	26
<i>trm</i>	35	1			2	9	5	1	6	141

Table 4: Cross-entropy confusion matrix for the **Newsgroups** data set. Columns show results for each model trained on text taken from the respective newsgroup; rows show test results for each newsgroup based on articles not included in the training data. Only 10 (of the 20) newsgroups are shown. These are: alt.atheism (i.e.*aa*), sci.script, sci.electronics, sci.med, sci.space, soc.religion.christian, talk.politics.guns, talk.politics.mideast, talk.politics.misc and talk.religion.misc.

accounts for most of the classifications. For most of the newsgroups, there are only a few outliers, the major exception between the confusion between the `alt.atheism` and `talk.religion.misc` newsgroups, which seems a reasonable confusion to make, and which accounts for 11% of the overall mis-classifications. For many of the mis-classifications, the correct newsgroup category has been ranked second or third-best, with only a small difference in cross-entropy from the best. Choosing the best two improves accuracy to 91.5%, and the best three to 94.2%. However, each test article is now multiply classified, a penalty that some retrieval-based applications may be willing to pay for the increased accuracy. This penalty can, however, be significantly reduced by choosing to discard the second or third best categories if the difference in cross-entropy from the best is above some threshold. For example, examining the best three categories and choosing a difference threshold of as little as 0.2 bits per character in cross-entropy will discard nearly 62% of the multiple classifications while still ensuring a very high accuracy level of 90.9%.

A number of enhancements can be made to further improve the the PPM-based classifier. By replacing sequences of non-alphanumeric characters in the data with a single space, the same accuracy can be achieved, but the PPM models are significantly smaller, and take less time to train. Hierarchical classification is also possible. The advantages of this are that classification is much faster (since categories in other branches of the tree do not have to be searched) and less memory is required during classification. McCallum *et al.* (1998) have also shown that text classification can be significantly improved by taking advantage of a hierarchy of classes. For the hierarchical PPM-based classifier, a decision tree of models is used. For higher nodes in the tree, PPM models are built from a concatenation of the training text used for categories that occur at the leafs of the sub-tree. Each node’s models are used to decide which branch to take after encoding the sample text and finding which model compresses it best. Branching is continued from the root until a leaf of the tree is reached. The category is the one used to train the PPM model in the leaf that compresses the sample text best. For example, using a two-level hierarchy on the **Newsgroups** data set by first pre-classifying into five “super”-categories (`comp.*`, `rec.*`, `sci.*`, `talk.*` and the rest) before classifying them individually has only a marginal effect on the original overall accuracy (down to 81.4% from 82.1%).

Biber (1988) uses factor analysis to identify six dimensions of variation on the basis of linguistic co-occurrence patterns across 23 spoken and written genres. An interesting possibility for future work in this area is to compare the effectiveness of the PPM based classifier at distinguishing these variations.

5 PPM-based text correction and segmentation

An essential component of many applications in natural language processing is a language modeler able to correct errors in the text being processed. For optical character recognition (OCR), poor scanning quality or extraneous pixels in the image may cause one or more characters to be mis-recognized, while for spelling correction, two characters may be transposed, or a character may be inadvertently inserted or missed out.

This section[†] describes a method for correcting English and other natural language texts based on a noisy channel model using PPM for the language modelling component and cross-entropy as the means to rank alternative re-arrangements of text sequences.

A common framework for the statistical modelling of natural language is based on a theory developed by Shannon (1948) at AT&T Bell Laboratories to model a noisy communication channel such as a telephone line. The adoption of the noisy channel model (also referred to as the *source-channel* model) was pioneered by the IBM Research Group at Yorktown Heights, New York, who applied it to the problem of continuous speech recognition (Bahl, Jelinek & Mercer, 1983; Jelinek, Bahl & Mercer, 1975). Since then the model has been applied to the problems of machine translation (Brown *et al.*, 1990), automatic spelling correction (Kernighan, Church & Gale, 1990), and many other applications such as part of speech tagging, optical character recognition (OCR) and handwriting recognition (Chen, 1996, page 6).

With this theory, the idea is to use two models—one a model of “good” text and the other a model of the types of errors that can occur in the original text, sometimes referred to as a *confusion* model. For word segmentation, the problem is one of insertion of spaces back into text where they have been elided. Even more complex problems are the extraction of text from other data such as OCR data or speech. The principle is the same—two models are used. One model is of the confusion between the original glyphs or phonemes and the text, together with a model of the good text.

More formally, a sequence of text, S , is sent into a communications channel and the noisy text, O , comes out at the end. For example, in spelling correction, the noisy text corresponds to output from a typist who makes spelling errors, and in machine translation, it corresponds to text in another language. The problem is to recover the original input text from the output text. This can be done by hypothesizing all possible input texts, S , and then selecting the most probable input \hat{S} given the output O :

$$\hat{S} = \arg \max_S p(S|O).$$

Applying Bayes’ theorem, we can rewrite this as:

$$\hat{S} = \arg \max_S \frac{p(S)p(O|S)}{p(O)} = \arg \max_S p(S)p(O|S).$$

Hence, the most probable sequence depends both on the *prior* probability $p(S)$ the sequence S will occur, and on the *observation* (or channel) probability $p(O|S)$ that the output O will be observed given that the sequence S has occurred. The latter depends on the application—for example, in speech recognition, the output *two* might be observed given the input *too*; in spelling correction, the output *percieve* might be likely if the input was *perceive*. The prior probability $p(S)$ is usually not available, so a model is used instead.

Applying this model to the problem of PPM based text correction, we wish to find the sequence of text, \hat{S} , that maximizes the prior and observation probabilities:

$$\hat{S} = \arg \max_S p(S)p(O|S).$$

[†]An extended abstract of part of the work in this section was first published in Teahan *et al.* (1998). For a survey of many other techniques devised for automatically correcting words in text, see Kukich (1992).

The prior probability $p(S)$ is estimated by training a PPM model on English text. Given $S = c_1, c_2, \dots, c_m$, then:

$$p(S) = \prod_{i=1}^m p'(c_i | c_{i-5}, c_{i-4}, c_{i-3}, c_{i-2}, c_{i-1})$$

where p' gives the probabilities returned by the order 5 PPM character model.

The observation probability $p(O|S)$ is also estimated from training data—this requires two training texts: the first consists of typical output from the application containing a representative sampling of errors, and the second the corrected text. From this we can estimate the error probability—for example, in an OCR application, the probability that the letter c was mistaken for the letter e , or that the letters lc were mistaken for the letter k . Formally, given the OCR text $O = t_1 \circ t_2 \circ \dots \circ t_m$, where \circ denotes concatenation, and each of the t_i is a transformed character sequence, then

$$p(O|S) = \prod_{i=1}^m p''(t_i | c_i)$$

where $p''(t|c)$ is the probability computed from a confusion table that the OCR sequence t was reported in place of the true character c .

In order to find the most probable correct text given the observed (incorrect) input text, a variation of the Viterbi dynamic programming method (Viterbi, 1967) can be used. The Viterbi algorithm is commonly used in hidden Markov models, for example in part of speech tagging programs, to assign the most likely sequence of tags to words in a sentence (Charniak, 1993). The key idea behind the Viterbi algorithm is that at any point in the search we need only retain, for all the possible active contexts, the sequence of text with the minimum entropy, with poorer performing alternatives discarded. Thus, with a Markov based model where the maximum length of the contexts are fixed (as with PPM models), the number of alternatives being searched is kept within manageable limits.

The next section explores several methods for evaluating models used for text correction and segmentation. Section 5.2 explores how a correction-based algorithm can be applied to the specific problem of word segmentation. Results with segmenting English text are described and compared with further results with Chinese text[‡]. The Viterbi algorithm is also discussed in more detail along with alternative algorithms that use heuristic search methods rather than exhaustive search.

5.1 Evaluating experimental results

Many models are evaluated in the literature by the two measures *recall* and *precision*. In word segmentation, for example, when comparing two strings they can only differ by the number of spaces present. The prediction of spaces is determined by three classes: correct space prediction a , spurious space prediction b , and missed space c . Recall is defined as $recall = 100 \times a / (a + c)$ and precision is defined as $precision = 100 \times a / (a + b)$. A perfect model will have recall and precision of 100%.

A more general measure is available when evaluating text correction models, and the processed text can be compared directly with the original (correct) text. The difference between these two texts can be determined by the *edit distance* between them (Cormen, Leiserson, & Rivest, 1990). The edit distance between two strings, x and y , is defined to be the minimum transformation sequence that converts x into y . The transformation operations are: delete a character, insert a new character, and change a character into another. For example, assuming equal costs for each transformation, the edit distance between *Eloplcinsons* and *Hopkinsons* is

[‡]PPM-based Chinese word segmentation is explored more fully in Teahan *et. al.* (2000).

<i>Original text:</i>	the unit of New York-based Loews Corp that makes Kent cigarettes stopped using crocidolite in its Micronite cigarette filters in 1956.
<i>With spaces removed:</i>	theunitofNewYork-basedLoewsCorpthatmakesKentcigarettesstoppedusingcrocidoliteinitsMicronitecigarettefiltersin1956.
<i>Ponte & Croft's method:</i>	the unit of New York-based Loews Corp that makes Kent cigarettes stopped using c roc id o lite inits Micron it e cigarette filters in 1956.
<i>PPM-based method:</i>	the unit of New York-based LoewsCorp that makes Kent cigarettes stopped using croc idolite in its Micronite cigarette filters in 1956.

Table 5: Segmenting words in English text

4. A sequence of operations that performs this transformation is: delete E , change l to H , delete l , change c to k . For word segmentation, the edit distance equals the sum of spurious space prediction b plus missed space c .

The *edit distance accuracy* is defined to be $100 - 100 \times e/m$, where e is the edit distance between the corrected and original text, and m is the number of characters in the original text.

5.2 PPM based word segmentation

English has an alphabetic orthography and word spacing, unlike other languages such as Chinese and Japanese, so it is relatively easy to adopt the “word” as a basic unit by using blank space and various punctuation marks as delimiters.

Word segmentation is an important task required for applications that do not start with an orthographic representation, such as speech recognition, or the automated transcription of Morse code. Ponte & Croft (1996) introduced a method (USeg) for predicting space positions and examined its performance using a 500 Kbyte extract from the Wall Street Journal. USeg is a word-based model that was trained on 1 Gigabyte of text, and produced a recall of 93.54% and precision of 90.03%.

Teahan *et al.* (1998) describe how a fixed order PPM model can be used to correct sequences of English text with a high degree of accuracy (over 99%). They applied their algorithm to the problem of segmenting words in English text, and showed that their method was a significant improvement over previously used methods. They also applied a similar technique as a post-processing stage after pages were recognized by a state-of-the-art commercial OCR system. It was shown that the accuracy of the OCR system could be increased from 96.3% to 96.9%, a decrease of over 20 errors per page. These results were obtained by training an order 5 PPMD language model on the text in the million-word (5.6 Mbyte) Brown Corpus (Francis & Kučera, 1982).

Using a PPM character-based method on the same corpus as Ponte & Croft produced both recall and precision rates of 99.52%, with an edit distance accuracy of 99.04%. This improvement over Ponte and Croft’s results used only a small fraction of their training text (the 5.6 Mbyte Brown Corpus compared to USeg’s 1 Gbyte).

Table 5 shows an example used by Ponte & Croft with the addition of the predictions made by PPM. The improvements that the PPM model provides are evident in this small example. Although the word *Micronite* does not occur in the Brown Corpus, the word was correctly segmented using PPM. Likewise, *inits* was correctly segmented into *in* and *its*. In this example, PPM made two mistakes. The space in *Loews Corp* was not predicted because *LoewsCorp* required 54.3 bits to encode the text while the original required 55.0 bits. Similarly, an extra

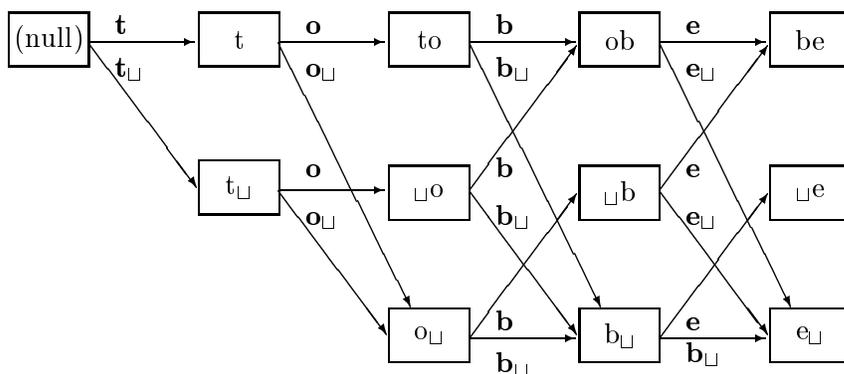


Figure 2: Word segmentation of the phrase *tobe*

space was added in *crocidolite* because the space reduced the number of bits to encode it from 58.7 to 55.3.

The PPM method consisted of finding the most probable correct segmentation of the text by first generating alternative segmentations of the text by inserting spaces after each letter and then searching for the best (i.e. most probable) segmentation. In their paper, Teahan & Cleary stated that they used a variation of the Viterbi dynamic programming method (Viterbi, 1967) to find the most probable segmentation. The Viterbi algorithm guarantees that the best possible segmentation will be found by using a trellis-based search—all possible segmentation search paths are extended at the same time and the poorer performing alternatives that lead to the same conditioning context are discarded. One of the drawbacks of the algorithm is that it is an exhaustive search rather than a selective one, unlike sequential decoding algorithms commonly used in convolutional coding which rely on search heuristics to prune the search space (Anderson & Mohan, 1984).

Figure 2 provides an illustration of how the algorithm works—it shows the trellis that is constructed when the phrase *tobe* is segmented. For this example the length of the conditioning context (i.e. the order of the PPM model) has been arbitrarily fixed at length 2. Hence each state in the diagram (the boxes) is labelled by two characters that represent the context (except for the initial few states where the context is of length 0 or 1). The transitions between the states have been labelled by two possibilities—either the character at the current position in the input phrase, or the current character with a space attached. Since the order of the model is fixed at 2, the maximum number of states at each input position will never exceed 3, since there are only 3 possible segmentations (i.e. the last two characters, a space followed by the last character, or the last character followed by a space). All possible segmentations are represented in the diagram since there is always a binary split after each state—for example, the preferred segmentation is found by following the path through the following transitions: t , o_{\perp} , b , e_{\perp} .

Later investigation revealed, however, that the method Teahan *et al.* used is not the Viterbi algorithm—it is, in fact, a variation of the sequential decoding algorithm called the *stack algorithm*—an ordered list of the best paths in a segmentation tree is maintained, and only the best path in terms of some metric is extended while the worst paths are deleted. The metric used in this case was the compression codelength of the segmentation sequence in terms of the order 5 PPMD model.

In the stack algorithm’s ordered list, paths vary in length. For deleting the worst paths, the traditional method is that the list is set to some maximum length, and all the worst paths longer than this length are deleted. For the PPM word segmenter, another variation was used instead.

Algorithm	Average edit distance	Average accuracy(%)	Time (secs)	Average # Nodes
Viterbi	438	98.91	109	968814
Teahan <i>et al.</i>	440	98.90	94	970176
Stack (size=25)	483	98.80	212	1016780
Stack (size=50)	475	98.82	422	1104970

Table 6: Segmenting words in Chinese text

All the paths in the ordered list except the first one were deleted if the length of the path plus m was still less than the length of the first's path, where m is the maximum order of the PPM model. The reasoning behind this pruning heuristic is that it is extremely unlikely (at least for natural language sequences) that the bad path could ever perform better than the current best path as it still needs to encode a future m characters despite being already worse in codelength.

One further pruning method was also found to significantly improve execution speed. As for the Viterbi algorithm, poorer performing search paths that lead to the same conditioning context are discarded. However, unlike the Viterbi algorithm, search paths vary in length, so only poorer performing search paths with *both* the same length and conditioning context are discarded.

5.3 Word segmentation of Chinese text

Word segmentation of Asian languages such as Chinese and Japanese, in contrast to English, is a much more interesting problem since they are written without word delimitation. This poses a problem for a number of applications, such as information retrieval and compression, where better performance is achieved if the application can apply some automatic method to find where the words in the text occur.

Experiments with segmenting Chinese text were performed using Guo Jin's Mandarin Chinese PH corpus of pre-segmented text containing almost two and a half million words of newspaper stories from the Xinhua news agency of PR China written between January 1990 and March 1991 (Hockenmaier & Brew, 1998).

In the experiments, an order 5 PPMD model was trained on the second half of the corpus. This was used to segment part of the first half which was split into fifty sub-sections each containing 10,000 words starting from the beginning of the corpus. Word delimiters were removed from each of the fifty sub-sections prior to performing the segmentation experiments. The results obtained are shown in Table 6. The experiments were repeated with the second half of the corpus using the start of the corpus for training instead with similar results.

As shown in the table's leftmost column, four different methods were used to segment each of the test subsections—the Viterbi algorithm, the method used by Teahan *et al.*, and two variants of the stack decoding algorithm, one with a maximum stack depth of 25, the other with 50. The next column in the table shows the average *edit distance* observed between the segmented test sub-sections of the corpus and their corresponding correct form obtained from the pre-segmented corpus. The edit distance can be used as a measure of the accuracy of the segmentation process (Teahan *et al.*, 1998). The third column converts these figures into an average edit distance accuracy based on the average size of text contained in each of the test sub-sections. The fourth column lists the average CPU time (in seconds) it took to segment the test sub-sections on a Pentium II 200 MHz processor with 128 MB RAM running Red Hat 5.2 Linux. The final column lists the average number of nodes stored in the paths tree constructed as the algorithms were executed. This figure can be used to gauge the memory consumption of each of the algorithms.

From the results it can be seen that the best method in terms of application accuracy is the Viterbi algorithm. This is not surprising as the method guarantees that the most probable

segmentation is found unlike the other methods which rely on search heuristics instead to prune the search space. However, the difference in accuracy between the four methods is small, ranging from 98.8 % to 98.9% in accuracy, although the average number of errors covered a much wider range, from an edit distance of 438 up to 483. The method used by Teahan *et al.* is the nearest to that of the Viterbi algorithm in terms of the number of errors made, but is slightly faster in speed while consuming slightly more memory resources. In comparison, the performance of the stack algorithm using either of the stack depths is much worse, taking over twice or four times as long in execution speed and also requiring increased memory resources.

5.4 Language segmentation

Language segmentation concerns the problem of identifying where the boundaries occur between one or more languages in text. This is of particular importance for many applications, especially in information retrieval. The problem is slightly different from the language identification problem tackled in Section 4.1—there we were interested in assigning a single categorization to the whole text document; here we are interested in finding if (and where) multiple languages exist in the document.

A similar approach to that taken for word segmentation can be used to solve this problem, although instead of a single PPM model, multiple models trained on representative text for each language are used. Like for word segmentation, all alternative segmentations of the text are generated and the best possible segmentation is found by application of the Viterbi algorithm. The segmentation is achieved by switching from using the current PPM language model to any of the alternative language models for each character in the test sequence. This is preceded by a special terminating character to signal the end of the prior segmentation and reset the context to null for the new model.

Experimental results show that this method has extremely high levels of accuracy. Using the same data as that used in Section 4.1, each of the Bible texts were split into testing data containing the last 1000 words or so[§], with the training data containing all the prior text. This training data was then used to train separate PPMD models for the six different languages. The testing data was split into 50 samples of 120 words each, with each sample containing six interleaved sub-samples of 20 words extracted consecutively for each language. The segmentation method was then tried on these samples—out of the 34049 total characters contained in them, only 180 were mis-classified, an edit-distance accuracy of 99.5%. Interestingly, this was achieved with no notion of a word boundary; if the segmentation was performed only at word boundaries, a further 72 of the errors would be corrected, with the added benefit of reducing the search space by a factor determined by the average word length.

It is interesting to speculate how a similar approach for authorship and genre segmentation might perform (that is, finding the boundaries between co-authored text or text with multiple genres). Although experiments using suitable data have yet to be performed in this area, it is hoped that similar accuracy levels to those reported in Sections 4.2 and 4.3 can be achieved.

6 Summary

One well-performed text compression scheme, PPM, has been described. PPM can be viewed as blending together several fixed-order context models to predict the next character in the input sequence.

A number of PPM-based techniques for text classification and segmentation have been presented. These can be applied successfully to a wide range of problems in natural language processing including cryptology, language identification, authorship attribution, classifying by genre, OCR spelling correction, and word segmentation. Witten *et al.* (1999a, 1999b) apply the same approach to text mining—they show how a broad range of patterns such as names, dates,

[§]The text was split at either a sentence or verse boundary, so each sample was slightly more than 1000 words.

email addresses and URLs can be located in text. In many cases, applications built on these techniques achieve state-of-the-art performance or close to it.

Not only do these models simplify many aspects of statistical language modelling, they require much less training. They also have the potential for performance comparable to, if not better than, more traditional word based methods used in statistical language modelling schemes. These models are severely handicapped by the fact that they have no understanding (in the way that humans might understand them) of the text being analysed. The process is simply one of *translation* or *composition*—the results are achieved simply by predictions made from prior sequences of characters in the text. Despite this, they are capable of performing certain tasks (such as decrypting cyphers, ascribing authorship and identifying language dialects) that humans find difficult.

It is an open question how well this technology will perform at other applications in natural language processing and text mining. Three promising areas are in part of speech tagging, word sense disambiguation and syntactic parsing. For the latter, however, Knight (1999) notes that “no one has yet been able to extract even *somewhat accurate* syntactic parses from raw text databases.”

7 Acknowledgments

I am very grateful to John Cleary, Ian Witten, Stuart Inglis and fellow researchers at Waikato University in New Zealand who have provided valuable advice and assistance on many aspects of this work.

8 References

- Anderson, J.B. and Mohan, S. (1984) “Sequential coding algorithms: A survey and cost analysis.” *IEEE Transactions on Communications*, **32**(2), 169–176.
- Bahl, L.R., Brown, P.F., deSouza, P.V. & Mercer, R.L. (1989) “A tree-based statistical language model for natural language speech recognition.” *IEEE Transactions on Acoustics, Speech and Signal Processing*, **37**: 1001–1008.
- Bahl, L.R., Jelinek, F. & Mercer, R.L. (1983) “A maximum likelihood approach to continuous speech recognition.” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **2**: 179–190.
- Brown, P.F., Della Pietra, S.A., Della Pietra, V.J., Lai, J.C. & Mercer, R.L. (1992) “An estimate of an upper bound for the entropy of English” in *Computational Linguistics*, **18**(1):31–40.
- Bell, T.C., Cleary, J.G. & Witten, I.H. (1990) *Text compression*. Prentice Hall, New Jersey.
- Bogges, L., Agarwal, R. & Davis, R. (1991) “Disambiguation of prepositional phrases in automatically labelled technical text.” *AAAI’91*, pages 155–159.
- Brill, E. (1993) *A corpus-based approach to language learning*. Ph.D. thesis, University of Pennsylvania.
- Charniak, E. (1983) *Statistical language learning*. MIT Press, Cambridge, Massachusetts.
- Chen, S.F. (1996) *Building probabilistic models for natural language*. D.Phil. thesis, Harvard University.
- Cleary, J.G. and Witten, I.H. (1984) “Data compression using adaptive coding and partial string matching.” *IEEE Transactions on Communications*, **32**(4), 396–402.
- Cleary, J.G. & Teahan, W.J. (1997) “Unbounded length contexts for PPM.” *Computer Journal*, **40**(2/3): 67–75.
- Cormen, T.H., Leiserson, C.E. and Rivest, R.L. 1990. *Introduction to algorithms*. MIT Press, Cambridge, Mass.
- Dunning, T. (1994) “Statistical identification of language.” Technical report 94–273, Computing Research Laboratory, New Mexico State University.
- Francis, W.N. & Kučera, H. (1982) *Frequency analysis of English usage: lexicon and grammar*. Houghton Mifflin, Boston.

- Ganeson, R. & Sherman, A.T. (1993) "Statistical techniques for language recognition: an introduction and guide for cryptanalysts." *Cryptologia*, **17**(4): 321–366.
- Garside, R., Leech, G. & Sampson, G. (editors) (1987) *The computational analysis of English*. London. Longman.
- Hockenmaier, J. & Brew, C. 1998. "Error-driven learning of Chinese word segmentation" in *12th Pacific Conference on Language and Information*, edited by Guo, J., Lua, K.T. & Xu, J., Singapore, Chinese and Oriental Languages Processing Society, 218–229.
- Howard, P.G. *The design and analysis of efficient lossless data compression systems*. Ph.D. thesis, Brown University, Providence, Rhode Island.
- Irvine, S.A. (1997) *Compression and Cryptology*. D.Phil. thesis, University of Waikato, N.Z.
- Jelinek, F. (1990) "Self-organized language modeling for speech recognition" in *Readings in speech recognition*, edited by Weibel, A. & Lee, K., pages 450–506. Morgan Kaufman Publishers, Inc.
- Johansson, S., Atwell, E., Garside, R. & Leech, G. (1986) *The tagged LOB Corpus*. Norwegian Computing Centre for the Humanities, Bergen.
- Juola, P. (1998) "What Can We Do With Small Corpora? Document Categorization Via Cross-Entropy" in *Proceedings of the Workshop on Similarity and Categorization*.
- Katz, S.M. (1987) "Estimation of probabilities from sparse data for the language model component of a speech recognizer." *IEEE Trans. on Acoustics, Speech, and Signal Processing*, **35**(3):400–401.
- Kernighan, M.D., Church, K.W. & Gale, W.A. (1990) "A spelling correction program based on a noisy channel model" in *Proceedings of the Thirteenth International Conference on Computational Linguistics*, pages 205–210.
- Knight, K. (1999) "Mining on-line text" in *Communications of the ACM*, **42**(11): 58–61.
- Kukich, K. (1992) "Techniques for automatically correcting words in text." *ACM Computing Surveys*, **24**(4):377–439.
- Lewis, D.D. & Hayes, P.J. (1994) "Guest Editorial" in *ACM Transactions on Information Systems*, **12**(3): 231.
- McCallum, A. & Nigam, K. (1998) "A comparison of event models for Naive Bayes text classification in *AAAI-98: Workshop on learning for text categorization*.
- McCallum, A., Rosenfeld, R., Mitchell, T. & Ng, A.Y. (1998) "Improving text classification by shrinkage in a hierarchy of classes" in *ICML-98: Proc. of the 15th International conference*.
- McFarlin Library Research Guides*. (1997) <<http://www.lib.utulsa.edu/guides/bible2.htm>>
- Moffat, A. (1990) "Implementing the PPM data compression scheme." *IEEE Transactions on Communications*, **38**(11): 1917–1921.
- Mosteller, F. & Wallace, D.L. (1984) *Applied Bayesian and classical inference: the case of the Federalist papers*. Springer-Verlag, New York.
- Ponte, J.M. & Croft, W.B. (1996) "USeg: A retargetable word segmentation procedure for information retrieval." *Firth Annual Symposium on Document Analysis and Information Retrieval*. Las Vegas, Nevada.
- Ristad, E.S. & Thomas, R.G. (1995) "New techniques for context modeling." *Proceedings of the 33rd Annual Meeting of the ACL*, Cambridge, Massachusetts, June 27–30.
- Schmid, H. (1994) "Part-of-speech tagging with neural networks" in *Proceedings of the International Conference on Computational Linguistics, 1994*.
- Shannon, C.E. (1948) "A mathematical theory of communication." *Bell System Technical Journal*, **27**: 379–423, 623–656.
- Shannon, C.E. (1951) "Prediction and entropy of printed English." *Bell System Technical Journal*, pages 50–64.
- Teahan, W.J. (1998) *Modelling English text*. D.Phil. thesis, Univ. of Waikato, N.Z.

Teahan, W.J. & Cleary, J.G. (1996) "The entropy of English using PPM-based models" in *Proceedings DCC'96*, edited by Storer, J.A. & Cohn, M., IEEE Computer Society Press.

Teahan, W.J., Inglis, S., Cleary, J.G. & Holmes, G. (1998) "Correcting English text using PPM models" in *Proceedings DCC'98*, edited by Storer, J.A. & Cohn, M., IEEE Computer Society Press.

Teahan, W.J., Wen, Y.Y., McNabb, R. & Witten, I.H. (2000) "A compression-based algorithm for Chinese word segmentation," *Computational Linguistics*, to appear.

Viterbi, A.J. (1967) "Error bounds for convolutional codes and an asymptotically optimal decoding algorithm," *IEEE Transactions on Information Theory*, **13**, 260–269.

Witten, I.H., Bray, Z., Mahoui, M. & Teahan, W.J. (1999a) "Text mining: A new frontier for lossless compression" in *Proceedings DCC'99*, edited by Storer, J.A. & Cohn, M., IEEE Computer Society Press.

Witten, I.H., Bray, Z., Mahoui, M. & Teahan, W.J. (1999b) "Using language models for generic entity extraction" in *ICML-99 Workshop: Machine learning in text data analysis*.