Models of English text

W. J. Teahan, John G. Cleary[1]

Department of Computer Science, University of Waikato, New Zealand

The problem of constructing models of English text is considered. A number of applications of such models including cryptology, spelling correction and speech recognition are reviewed. The best current models for English text have been the result of research into compression. Not only is this an important application of such models but the amount of compression provides a measure of how well such models perform. Three main classes of models are considered: character based models, word based models, and models which use auxiliary information in the form of parts of speech. These models are compared in terms of their memory usage and compression.

1 Applying English text compression

It is shown in [21] how the best performed text compression scheme PPM [5, 6] can be used to implement novel solutions to language modelling problems including: cryptology; language identification; authorship attribution; spelling correction; and speech recognition. Three examples are given in Table 1. In the first example, compression is used to crack a simple substitution cipher (in this case, the letter *u* has been substituted by the letter *t*, the letter *p* by the letter *v*, and so on). The algorithm devised by Irvine [10] repeatedly rearranges the cipher text by exhaustively swapping letters of the alphabet in groups of 3 or 4. After each rearrangement, the text is compressed using PPM with an order 3 character based model of English trained on the Brown Corpus [8] and twenty English novels. The resultant size of the compressed text is used to reject or accept the particular arrangement of the letters. Irvine shows how the same technique can be applied to other more difficult cryptosystems.

The second example shows how to apply compression to spelling correction (in this case, the mis-spelt word is *acticities*, and the correct spelling is *activities*). First generate alternative spellings for a mis-spelt word by reversing one of four possible typing errors—a wrong letter, a missing letter, an extra letter, or two transposed letters. Next, compress each of these alternative spellings using a character based model trained on a large corpus of representative text. The preferred spelling is the one which reduces the overall codelength for the compressed text (including the surrounding context). In the example shown, we used an order 5 PPM model trained on the Brown Corpus, with the preferred spelling producing the lowest codelength.

The third example is for speech recognition. The traditional method splits speech recognition into two parts—an acoustic modeller and a language modeller [3]. We apply compression to the second part: the acoustic model first generates alternative homophonic text representations for each sound or groups of sounds (for example, the words *to*|*too*|*two*; and the phrase *ice cream*|*I scream*); the next step then compresses each alternative and chooses the one with the lowest codelength. In the example shown, we generated 24 alternatives for the phrase "*how to recognize speech*" using the homophones *to*|*too*|*two*, *recognize*|*wreck a nice*, and *beach*|*beech*|*peach*|*speech*, and then compressed them using an order 5 PPM model trained on the Brown Corpus. As shown in the example, the more plausible alternative produces the smallest codelength.

In all of these applications, effective performance requires that the model be able to predict as well as or better than a native English speaker. We have obtained

[1] email {wjt, jcleary}@waikato.ac.nz

application	model	sample text	(bits)
cryptology	*order 3 PPM model*	`tvaqzbrxsznfsxszxzfo...`	1777.1
	trained on the Brown	`rfet unao wioao a im...`	921.1
	Corpus plus twenty	`upot sean dinan a il...`	688.6
	English novels	`upon this basis i am...`	**312.2**
spelling	*order 5 PPM model*	... the actiocities too ...	70.60
correction	*trained on the Brown*	... the anticities too ...	65.80
	Corpus	... the acticities too ...	55.98
		... the activities too ...	**51.17**
speech	*order 5 PPM model*	how too wreck a nice beech	78.14
recognition	*trained on the Brown*	how to wreck a nice beach	70.07
	Corpus	how two recognize speech	55.24
		how to recognize speech	**48.53**

Figure 1: Applying English text compression

preliminary results indicating that small incremental improvements in the models significantly improve the application's effectiveness.

In the following sections, we consider a number of models of English and compare them using their ability to compress. Previous work on PPM based character models is reviewed first. In Section 3, we review word based models and present some new models that combine words with parts-of-speech (tags). Some implementation issues necessary to achieve memory efficiency for these models are then solved. Section 4 concludes by comparing the compression performance of these different models.

2 Character based models

Previously reported results on the PPM text compression scheme show that it can predict English text almost as well as humans. In [20], Teahan and Cleary performed experiments on the same text that Claude E. Shannon used in a famous experiment to estimate the entropy of English [18]. Shannon's estimates were based on humans guessing the upcoming text, letter by letter. Teahan and Cleary used the PPM scheme to build a character based computer model, and found that performance was close to, and in some cases, superior to human based results.

PPM [5] has set the performance standard for lossless compression of text for over a decade. The PPM technique blends character context models of varying lengths to arrive at a final overall probability distribution for predicting upcoming characters in the text. The models are *adaptive*: the counts for each context are updated progressively throughout the text. In this way, the models adapt to the specific statistical properties of the text being compressed.

Experiments with English text show that order 5 character based PPM models perform well [19] although results in [4] suggest that higher orders may perform better using improved blending algorithms. Cleary *et al.* [6] report on another approach where the length of the context models are unbounded.

Performance of these models on English text can be substantially improved by training them on large amounts of text [20]. Text from several corpora (the Brown Corpus, the Lancaster-Oslo/Bergen (LOB) Corpus [12], ACL-DCI's Wall Street Journal [1], the complete works of Jane Austin and the King James Version of the Bible) were concatenated and used to construct a language model. Figure 2 shows the compression (in bits per character) achieved as the model was incrementally constructed. This is a measure of how well the model would have performed had it been frozen at

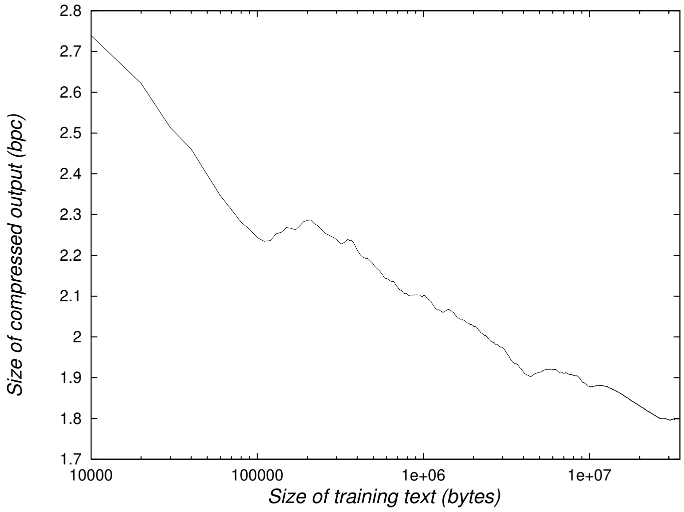


Figure 2: How compression varies with size of training text

that point. The compression generally decreases, with some increases when the text changes style, for example, when the Brown Corpus finishes and LOB Corpus starts. Out to 35 Mbytes, there is no indication of an end to the decrease. This opens the possibility of using the 15×10^9 words (75×10^9 bytes) of text available on the World Wide Web (as of late 1996) to build a model—if the trends of Figure 2 continue, this might give a model compressing below 1 bit per character.

One difficulty with such large models is the amount of memory needed to store them. Figure 3 shows that the memory size required is closely proportional to $x^{0.4}$ where x is the size of text compressed. Extrapolating from this, a model of a 15×10^9 word corpus would need roughly 560 Mbytes of memory—a feasible amount with current technology. This $x^{0.4}$ dependency of model size is reminiscent of the result in [17] that the size of the vocabulary is proportional to $x^{0.5}$ to $x^{0.6}$.

3 Word and tag based models

Word-based models use the previous few words in the text to predict the upcoming ones. These should provide faster compression than character based models because fewer symbols are being processed. A number of word-based text compression schemes have already been proposed [9] and we extend the approach taken in [15].

Another approach to modelling is to use parts-of-speech (tags) such as *noun*, *verb* and *adjective*. The idea is that knowing the tag of a word helps in predicting it. The advantage of using the tag is that it may have occurred many times previously and so a good representative sample of what is likely to follow it has been built up. By contrast, an individual word may have occurred only a small number of times. There are two major issues with using tags: first, the words in the text must have tags assigned to them somehow; and second, the tags need to be encoded in the models along with the text itself.

Horspool and Cormack [9] propose a simple first order state based model using just five parts of speech—*noun*, *verb*, *adjective*, *article* and *other*. For the models we are concerned with, however, we assume that the text has already been tagged using a much more comprehensive tagset (such as that used for the two tagged corpora shown in Figure 4) and we wish to explicitly encode and decode these tags along with the words. The traditional language modelling approaches (for example, used in speech recognition and machine translation) have been either word or part-of-

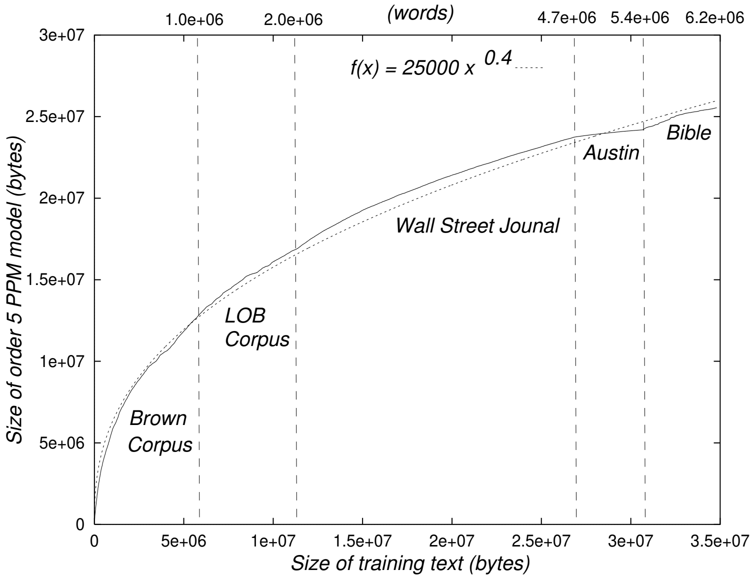


Figure 3: How the size of the PPM model varies with size of training text

LOB Corpus

words	*to*	*be*	*or*	*not*	*to*	*be*	*that*	*is*	*the*	*question*
tags	**TO**	**BE**	**CC**	**XNOT**	**TO**	**BE**	**DT**	**BEZ**	**ATI**	**NN**

ACL-DCI's Wall Street Journal

words	*to*	*be*	*or*	*not*	*to*	*be*	*that*	*is*	*the*	*question*
tags	**TO**	**VB**	**CC**	**RB**	**TO**	**VB**	**DT**	**VBZ**	**DT**	**NN**

Figure 4: Examples of tagged texts

speech based [2, 11, 13].

When compressing tagged text, it is necessary to include the tags as well as the text itself. This has the potential for increasing the size of the compressed text. However, the extra contextual information provided by the tags more than compensates for this and we will see that the total result is comparable to pure word based coding.

Both our word and tag based models exploit the blending mechanism of the PPM compression scheme. Higher order contexts are first tried, but if the next word has not been seen before in this context then a lower order context is used instead. So that the decoder knows which context to use, an "escape" symbol is transmitted to signal that the prediction should be done with a lower order context. Various escape algorithms are discussed in [22]. Our experiments with word and tag based models show that a variant of method X (dubbed method X1) performs best in most cases [19]. This method estimates the probability of the escape symbol as being proportional to the number of words in the context which have occurred only once. Performance of these models can also be improved by two simple mechanisms—the first, called *full exclusions*, excludes words already predicted by higher order contexts. The second, called *update exclusions* updates the counts only in contexts that actually make the prediction. Both full and update exclusions typically improve the compression by a few per cent, and our experiments with word and tag based

WW model	**WTW model**
$P(W_n \mid W_{n-1})$	$P(W_n \mid T_n W_{n-1})$
$\hookrightarrow P(W_n \mid)$	$\hookrightarrow P(W_n \mid T_n)$
$\hookrightarrow$ *character model*	$\hookrightarrow$ *character model*

Figure 5: Some models for predicting words

TTT model	**TTWT model**
$P(T_n \mid T_{n-1} T_{n-2})$	$P(T_n \mid T_{n-1} W_{n-1} T_{n-2})$
$\hookrightarrow P(T_n \mid T_{n-1})$	$\hookrightarrow P(T_n \mid T_{n-1} W_{n-1})$
$\hookrightarrow P(T_n \mid)$	$\hookrightarrow P(T_n \mid T_{n-1})$
$\hookrightarrow P_{eq}(T_n \mid)$	$\hookrightarrow P(T_n \mid)$
	$\hookrightarrow P_{eq}(T_n \mid)$

Figure 6: Some models for predicting tags

models bear this out.

The representation of the two best performed models we have experimented with are shown in Figure 5. The order 1 word model first predicts the word using just the previous word. If the word is not predicted, then the model escapes down to an order 0 model. (In the diagram we use the symbol $\hookrightarrow$ to represent the escape process). If the word is not predicted at all, then a further escape is encoded down to a character based model, where each character in the word (including the space at the end) is encoded separately. All words encoded at this level are either unique (for the word based models) or have been tagged uniquely (for the tag based models), so in a sense can be regarded as the "vocabulary" of the text. This model will be referred to as the "WW" model.

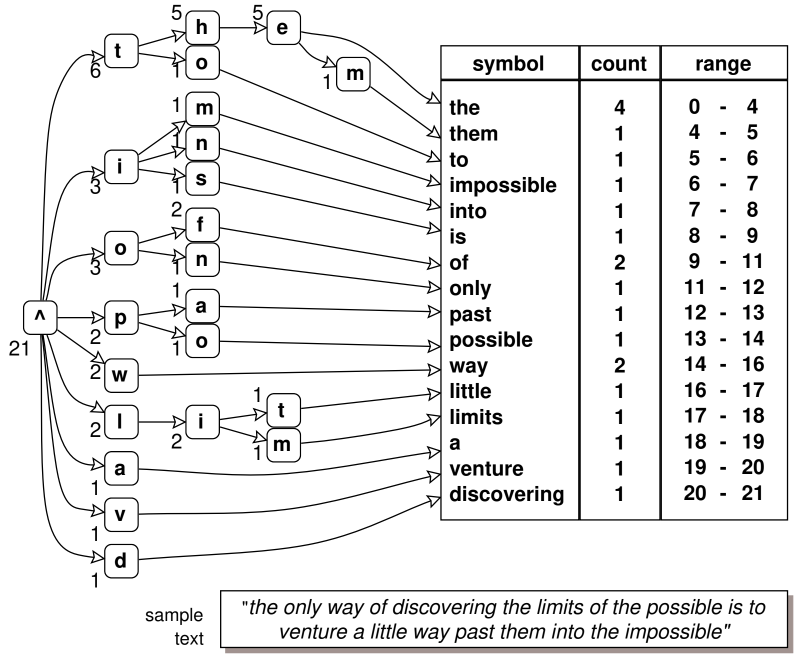
Two methods present themselves for encoding this vocabulary. The first method builds a PPM character model using all the previous characters in the text, and then uses this to predict the characters. The second method exploits the concept of update exclusions using a process we call *character exclusion*—the model is built using only the characters contained within the vocabulary itself. Experiments have shown that this method consistently outperforms the first.

The second model shown in Figure 5 first predicts the word using the current tag and the previous word. If this is unsuccessful, it tries based on the current tag only, otherwise it escapes down to the character based model. This model will be referred to as the "WTW" model.

Figure 6 shows two of the best models for encoding the tags. The current tag is first predicted using the previous two tags, then the model escapes and uses just the previous tag, then escapes again and predicts without any context. The first time a new tag is seen it will not have been recorded even in this model, so there is a final escape to a model which predicts all the tags with equal probabilities. This model is referred to as the "TTT" model. The second tag model predicts the current tag using the prior tag, the prior word and the tag preceding that. It then uses an escape hierarchy similar to the previous model. This model is referred to as "TTWT".

3.1 A trie-based data structure for fixed order models

Designing an efficient data structure for these models that maintain the cumulative frequency counts required for arithmetic coding is a non-trivial task [16]. A simple array of counts is sufficient when encoding with small alphabets. For word and tag models where each word is a different symbol in the "alphabet" a more sophisticated



symbol	count	range
the	4	0 - 4
them	1	4 - 5
to	1	5 - 6
impossible	1	6 - 7
into	1	7 - 8
is	1	8 - 9
of	2	9 - 11
only	1	11 - 12
past	1	12 - 13
possible	1	13 - 14
way	2	14 - 16
little	1	16 - 17
limits	1	17 - 18
a	1	18 - 19
venture	1	19 - 20
discovering	1	20 - 21

Figure 7: Trie data structure for storing cumulative frequencies

data structure is required. Indeed, for word based models, the size of the alphabet is essentially unbounded, with new words being added continually.

Fenwick [7] has proposed a *binary indexed tree* data structure for maintaining the cumulative frequency counts. As well as being very compact (requiring just n words of storage for an n-symbol alphabet, compared to $3n$ for an array-based implementation), it takes $\Theta(\log n)$ time per symbol for each of the three main operations shown above (compared to $\Theta(n)$ for an array).

We propose a trie based data structure for large alphabets that has the potential for even faster performance, with search and update times proportional to the character length of the symbol being searched for. Although not as compact as Fenwick's approach, it ameliorates this by allocating space only when required, and there is no pre-set bound on the alphabet size. A diagrammatic representation of the data structure is given in Figure 7. The trie is constructed using the individual characters of the words rather than the words themselves. Each node contains a cumulative count of the counts for its descendants, and a pointer is maintained for each leaf node to a record containing the symbol and its count (shown in the table on the right). The cumulative range for each symbol is shown for illustration purposes only and is not stored as it can be determined from the symbol's frequency and from the sum of the cumulative counts along the path to the leaf node. The time required to perform search and update operations for arithmetic coding with this structure is proportional to the length of the symbol [14].

This data structure can be applied to implementing mixed word and tag models. A second trie structure is used to store each word or tag context as a string of characters (tags are represented by unique character codes). Stored at the leaves of this second trie structure are the tries that contain the cumulative frequencies associated with the context in the same manner as in Figure 7. Also stored is the number of singletons required to estimate the escape probability for each context.

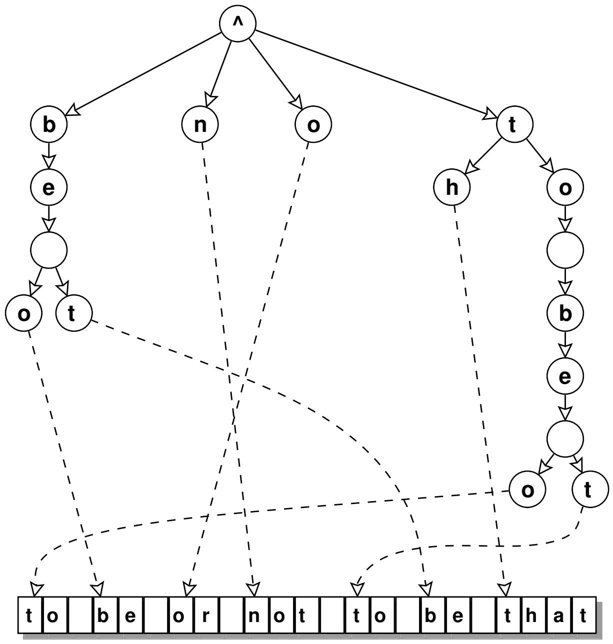


Figure 8: Word context trie for the string *"to be or not to be that"*

3.2 Data structures for models of unbounded order

Processing large input files with this data structure can rapidly exhaust all the available on-line memory. One solution is to adapt the context trie data structure used for the PPM* text compression scheme [6]. This structure provides an efficient means for working with context models of unbounded length, with storage requirements proportional to the size of the input string. Figure 8 illustrates how the structure can be adapted to word based models for the string *"to be or not to be that"*. The context trie is constructed in the same manner as in [6] using the space character to mark the end of each word. For word based models, access is only required to those character contexts that start words—all other contexts (*"hat"* and *"ot"* for example) are ignored.

Figure 9 plots the memory requirements for this structure and compares it with word models of order 1 and 2 models and an order 5 character based PPM (using text from the LOB Corpus). The figure shows that the PPM* method provides substantial savings in on-line memory requirements for word models of order greater than 1. However, all of the word and tag models require substantially more memory than the character based model. Implementing tag based models with unbounded order is much more difficult, especially for models more complicated than the simple ones described in this paper. For further details refer to [19].

4 Compression results

Results for the best performed of the models are summarized below. We were able to obtain two tagged corpora for our experiments—the LOB Corpus and the Wall Street Journal. Table 1 compares how well the models compress these texts. (All experiments are with texts converted to 27 character English—26 letters plus space). Compression ratios are shown in bits per character (bpc). Table 2 lists the cost in bits per tag (bptag) of encoding the tags for the tag/word model. More detailed results for these and other models are given in [19].

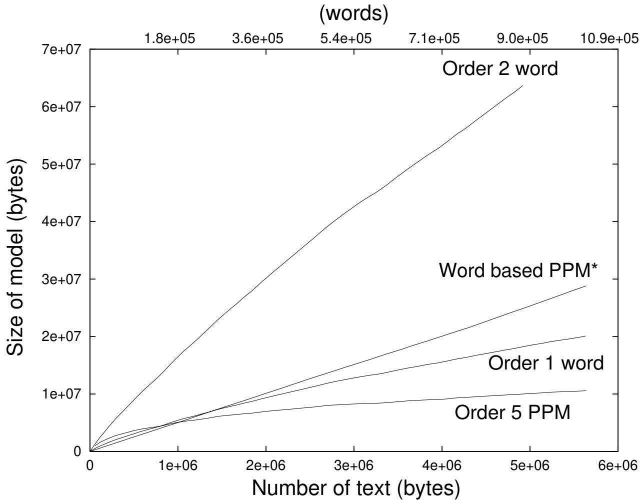


Figure 9: Memory requirements for the models (LOB Corpus)

tagged text	size of text (chars)	Order 5 PPM (bpc)	WW (bpc)	WTW + TTWT (bpc)
LOB Corpus	5636660	1.860	1.783	1.782
Wall Street Journal	15398849	1.602	1.539	1.547

Table 1: How well the models compress the tagged texts

The results show that performance of the word and tag based methods are comparable, both being better then the order 5 PPM[2] character based method by 3 to 4%. This is surprising as the tag-based model has to encode *both* the tags and the words—the tags are produced by the decoding process at no extra cost. We have not attempted to optimize the performance of the tag based models. The tags we used were designed primarily for linguistic purposes, and further gains should be possible by optimizing the tagset to improve compression performance.

Update exclusions were used in all these models and improve performance by about one per cent. Full exclusions similarly gave an improvement of one per cent. The best performed escape mechanism for the word and tag models is method X1. Experiments also show that separately encoding the vocabulary with an order 4 PPM character model performs best. Further experiments using the context trie based methods described in section 3.2 show that performance degrades with higher order models. We have not yet investigated this trade-off with much larger training texts. Also, it is unclear how the addition of the blending mechanisms described in [4] will affect these results.

In principle, the character based models, particularly the unbounded context PPM* models, should be able to perform at least as well as the word models as they have the same or more information available to them. Clearly more work is needed on computing predictions from PPM* models to extract this additional information.

Experiments with other non-tagged texts are shown in Table 3. In all cases, the results show that word based methods outperform the character based ones, with improvement ranging from 2 to 4%. Future experiments will compare the performance

[2] This method uses escape method D and bigram encoding as described in [20].

tagged text	tag types	tag tokens (words)	TTT (bptag)	TTWT (bptag)
LOB Corpus	154	1021049	3.708	3.647
Wall Street Journal	104	2614956	3.089	2.756

Table 2: How well the models compress the tags

untagged text	size of text (chars)	Order 5 PPM (bpc)	WW (bpc)	Improve-ment (%)
Dumas Malone's *Jefferson and His Time*	6448777	1.513	1.455	3.83
Complete works of Jane Austin	3872447	1.603	1.532	4.43
Brown Corpus	5766822	1.874	1.797	4.11
King James Bible	4139727	1.464	1.439	1.71

Table 3: Comparing word and character based models

of various taggers at automatically tagging these texts with different tagsets against the word and character based results.

5 Summary and conclusions

A number of models have been investigated for compressing English text. Contrary to previous results, models based on parts-of-speech (tags) can perform as well as word based models. The character based models perform slightly less well than the word or tag/word models. However, they have some advantages. They are much more economical of memory space; also, they can be applied to a wider range of problems in natural language processing such as cryptology and spell-checking, and also simplify many aspects of statistical language modelling for other areas (such as speech recognition). Clearly techniques to improve the performance of the character based models should be well rewarded.

Despite the foregoing, there are classes of information that are not taken into account by these models. There is evidence, for example, that in English text there are words which show strong recency effects [13]. If a word occurs once then there is a high probability that it will occur again soon. It is unclear, at this time, what models and estimation techniques can effectively take advantage of such information. However, by far the best technique for obtaining better models is to use longer training texts. This forces emphasis on economy of memory usage.

References

[1] ACL-DCI (1991) Association for Computational Linguistics Data Collection Initiative CD-ROM 1.

[2] Brown, P.F., Della Pietra, V.J., deSouza, P.V., Lai, J.C. and Mercer, R.L. (1992) “Class-based n-gram models of natural language”. *Comp. Linguistics*, **18**(4):467–479.

[3] Bahl, L.R., Jelinek, F. (1983) “A maximum likelihood approach to continuous speech recognition”. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2:179–190.

[4] Bunton, S. (1996) *On-line stochastic processes in data compression.* PhD thesis, University of Washington.

[5] 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.

[6] Cleary, J.G., Teahan, W.J. and Witten, I.H. (1995) "Unbounded length contexts for PPM," *Proceedings DCC'95.* IEEE Computer Society Press, 52–61.

[7] Fenwick, P. (1994) "A new data structure for cumulative probability tables". *Software—Practice and Experience*, **24**(3), 327–336.

[8] Francis, W.N. and Kučera, H. (1982) *Frequency analysis of English usage: Lexicon and grammar.* Houghton Mifflin, Boston.

[9] Horspool, R.N. and Cormack, G.V. (1992) "Constructing word-based text compression algorithms". *Proceedings DCC'92.* IEEE Computer Society Press, 62–71.

[10] Irvine. S.A. (1997) *Compression and Cryptology.* DPhil thesis, Univ. of Waikato, N.Z.

[11] Jelinek, F. (1990) "Self-organized language modeling for speech recognition". **In** A. Waibel and K. Lee, editors, *Readings in speech recognition*, 450–506. Morgan Kaufmann Publishers Inc.

[12] Johansson, S., Atwell, E., Garside, R., and Leech, G. (1986) *The tagged LOB Corpus.* Norwegian Computing Centre, Bergen.

[13] Kuhn, R. and De Mori, R. (1990) "A cache-based natural language model for speech recognition". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **12**(6), 570–583.

[14] Lewis, H.R. and Denenberg, L.. (1991) *Data structures and their algorithms.* Harper Collins Publishers.

[15] Moffat, A. (1989) "Word based text compression". *Software—Practice and Experience*, **19**(2), 185–198.

[16] Moffat, A., Neal, R. and Witten, I.H. (1995) "Arithmetic coding revisited," *Proceedins DCC'95.* IEEE Computer Society Press.

[17] Salton, G. (1988) *Automatic text processing.* Addison-Wesley Pub. Co.

[18] Shannon, C.E. (1951) "Prediction and entropy of printed English". *Bell System Technical Journal*, 50–64.

[19] Teahan, W.J. (1997) *Modelling English text.* DPhil thesis, Univ. of Waikato, N.Z.

[20] Teahan, W.J. and Cleary, J.G. (1996) "The entropy of English using PPM-based models," *Proceedings DCC'96.* IEEE Society Press, 53–62.

[21] Teahan, W.J. and Cleary, J.G. (1997) "Applying compression to natural language processing," Submitted to ANLP'97.

[22] 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 Informaton Theory*, **37**(4), 1085–1094.