Language Modelling

Candidate 8212R

Supervisor: Dr. David MacKay

May 12, 2003

Abstract

The aim of this project was to make an adaptive language model of text. Models at both the character and word level were investigated. These were all loosely based on the Prediction by Partial Match (PPM) scheme. Performance of these models was quantified by calculating the compression they could achieve on various representative texts. Two different character based schemes were investigated, both of which performed well in comparison to traditional schemes. The word based models implemented suggested an increase in performance, but would have to be made fully adaptive before quantitative comparisons could be made. 'Recency boosting' was investigated for a monogram model, and found to improve performance.

1 Introduction

The aim of this project was to make an adaptive language model of text. Explicitly, to write a program that would predict the probability of the next symbol as closely as possible:

$$P(\text{next symbol}|\text{document so far})$$

Where the form $P(a|b)$ means the probability of a occurring given b has occurred. The performance can be quantified by calculating the compressed length (the total information content) L, of representative files, as predicted by the language model.

$$L_{\text{compressed}} = \sum_{\text{all symbols}} log_2 \frac{1}{P(\text{actual symbol}|\text{document so far})} \text{ bits}$$

Although, for comparison between different sized text files, the average compressed length per character $\langle l \rangle = L/(\text{number of characters})$ is a more useful measure of performance, since it allows us to ignore file size effects.

1.1 Motivation

Why is it useful to have good language models? They have a wide array of applications [Teahan and Cleary, 1997], including:

1. Compression - The motivation for compressing text files is simple, it requires less resources to store or send the files than in their original uncompressed form. Measuring how well a document could be compressed is also a convenient way of quantifying performance of a language model, as we shall see in Section 2.1. The best performing text compression scheme is PPM [Cleary and Witten, 1984, Cleary et al, 1995] (see also Section 2.3).
2. Cryptography - Many codes are cracked by exploiting redundancy in language. By using a language model to compress text before it is encrypted, it can be made much harder to crack. Language models can also be used to crack some simple codes efficiently [Irvine, 1997].
3. Spell checking - Language models can be used to perform spell checking by generating alternative spellings for a misspelt word, and then seeing which one has the highest probability as predicted by the language model.
4. Speech and text recognition. The last few years have seen a great improvement in the performance of speech recognition programs, used primarily for automatic dictation of speech. Performance is directly related to the performance of the underlying language model. Similarly, text recognition works by using a language model to decide which possible interpretation of a sequence of characters is most probable.

Why an adaptive language Model? We could simply use a frequency table of the occurrence of letters in the English language, and use a Huffman (see, for example [Brigger]) code to perform compression. But this would perform badly if the model was used to compress text in a different language, or text written by someone who uses the letter with a different frequency. For example, *Gadsby* by Ernest Wright, is a novel of 50,000 words without a single instance of the letter 'e' [Wright, 1953].

An adaptive language model can be used in a much wider variety of contexts without needing to be altered. For example, if we were to use the language model in a text compression program, that one program could deal well with all files it is likely to see.

2 Theoretical Background

2.1 Why Compression?

Why is compression a good measure of performance? We have some probability distribution $P(\boldsymbol{x})$ over samples of text. We can use a language model to make

an approximation to this distribution $Q(\boldsymbol{x})$. A sample of text can be compressed close to its information content h, as given by $Q(\boldsymbol{x})$

$$h = log_2 \frac{1}{Q(\boldsymbol{x})} \text{ bits}$$

The average compressed length per character will be the sum of the information contents of the symbols, multiplied by their probability of occurrence, i. e.

$$\langle l \rangle = \sum_{x_i} P(x_i|\boldsymbol{x}) log_2 \frac{1}{Q(x_i|\boldsymbol{x})} \text{ bits per character}$$

This can be re-expressed as

$$\begin{aligned} \langle l \rangle &= \underbrace{\sum_{x_i} P(x_i|\boldsymbol{x}) log_2 \frac{1}{P(x_i|\boldsymbol{x})}}_{H(\boldsymbol{x})} + \underbrace{\sum_{x_i} P(x_i|\boldsymbol{x}) log_2 \frac{P(x_i|\boldsymbol{x})}{Q(x_i|\boldsymbol{x})}}_{D_{\mathrm{KL}}(P\|Q)} \\ &\equiv H(\boldsymbol{x}) + D_{\mathrm{KL}}(P\|Q) \end{aligned}$$

Which demonstrates that the compressed length of the text is bounded below by its entropy $H(\boldsymbol{x})$, which will be the same whatever language model we use. The Kullback-Leibler divergence $D_{\mathrm{KL}}(P\|Q)$ is a measure of how close our estimated probability distribution $Q(\boldsymbol{x})$ is to the true probability distribution $P(\boldsymbol{x})$. It satisfies Gibbs' inequality which states $D_{\mathrm{KL}}(P\|Q) \geq 0$, with equality only when $P(\boldsymbol{x}) = Q(\boldsymbol{x})$. The closer $Q(\boldsymbol{x})$ is to $P(\boldsymbol{x})$, the smaller $D_{\mathrm{KL}}(P\|Q)$ is.

Therefore, the compressed size of the text gives us a measure of how closely our model fits reality, the smaller the file size, the better our model is.

2.2 *n*-graphs

In this scheme we approximate P(char|document) by P(char|last n-1 chars). This turns out to be a surprisingly effective approximation. For example, consider the case of bigraphs (2-graphs). In this case we use the approximation to P(char|document) of P(character|previous character). This is known as an order 2 model. Similarly, an order 5 model approximates the probabilities using P(char|previous 4 chars).

The problem with simply using a order-n scheme is one of sparsity of data. Say we are using an order 2 model, and we have so far seen the text string "an apple is a...". The only non-zero probabilities after having just seen the letter 'a' are P(n|a) and P(p|a). This is because 'n' and 'p' are the only letters we have ever seen that immediately follow 'a'. But this is not what we want. The probability of 'a' being followed by a space is not zero, we just haven't seen an occurrence of it yet. If we were to use an efficient coder, such as an arithmetic coder, to compress the output using these predictions, the string "an apple is a fruit" would have infinite length. Needless to say, this is not desirable behaviour for a compression program.

2.3 PPM - Prediction by Partial Match

PPM is a scheme than can be viewed as blending together predictions of different order n-graph models to obtain an estimate of the probability of a certain character occurring.

In its most general form, smoothing is expressed by

$$P = \eta P(\text{uniform}) + \mu P(\text{monograph}) + \rho P(\text{bigraph}) + \ldots$$

where η, μ, ρ, etc. are the smoothing coefficients.

This avoids the zero count problem by blending in a small amount of a uniform distribution, thus no count will be strictly zero. Also, lower order models may make better predictions near the start of the file, when the higher order counts are still sparse.

In practice, combination of the different order models is achieved by the use of 'escape' probabilities. By default, the highest order model is used for encoding. However, if a character occurs that has not been previously seen by that model, an escape symbol is transmitted to tell the decoder to switch to the next order model down.

This effectively blends the different order n-graph models in proportions that depend on the choice of values for the escape probabilities. In this report however, a more general approach is taken where the models are explicitly blended using choices of smoothing coefficients. This avoids the bit cost of transmitting escape symbols.

3 Implementation & Results

A C++ program was developed that could read in a text file and use a language model to assign probabilities to the next character/word as it traversed the document. The language model was updated as more text was seen, and the average compressed length per character was calculated and returned to the user. The language model was contained within its own function, so that it could be changed easily without affecting the rest of the program.

3.1 Data Storage

When using a simple order 2 model, frequency counts can be stored in a 2D array (Figure 1). When using higher order models however, this approach becomes unfeasible. For instance, using an order 5 model with a 40 character alphabet would require $40 \times 40 \times 40 \times 40 \times 40 \simeq 100$MB of RAM. While this figure is achievable by newer computers, it is hardly desirable, and simply increasing the character alphabet to 128 characters to allow compression of realistic ASCII

	a	b	c	d	e	f	g	h	i
a	0	0	0	0	0	0	0	0	0
b	0	5	0	0	0	0	0	0	0
c	0	3	0	0	0	0	0	3	0
d	0	0	0	0	0	0	0	0	0
e	0	1	0	0	0	0	0	1	0
f	0	0	0	0	0	0	0	1	0
g	0	0	0	0	0	0	0	0	1
h	0	0	0	0	0	0	0	0	0
i	0	0	0	0	0	0	0	0	0

Figure 1: A 2D Array containing bigraph counts for a nine letter alphabet

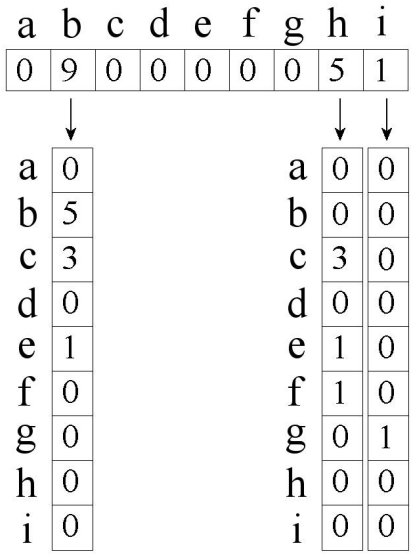

Figure 2: The practical data storage structure used in this report. Note how this structure stores the same information as in Fig 1, but using less memory.

(see, for example [Price]) text files results in an array of size $\sim$ 34GB, which is unfeasible for all but supercomputers.

For these higher order models, a more sophisticated way of storing the frequencies is required. We note that most of the frequency array will be empty, since many combinations of letters occur very rarely or not at all. This is the motivation behind a so-called 'trie' structure used by many PPM implementations. This is a type of tree structure for data storage, where only the branches that have actually occurred are added.

In this investigation, a more basic structure was used, where memory was simply not allocated to empty lines of the storage array. This was achieved by constructing the storage array out of many 1D arrays of length equal to the alphabet, and joined together using pointers (Figure 2) . By using this more dynamic approach to memory allocation, we may construct parts of the array only when needed. This is easier to program and faster than using a proper trie structure, although it uses more memory.

3.2 Simple Character based models

Some simple character-based models were developed, consisting largely of frequency tables for the different orders, and some way of combining these predictions using smoothing. Models up to order 5 were investigated. Initially a 40

character alphabet was used, consisting of the lower case letters of the alphabet, and common punctuation characters. A version using a full 128 character ASCII alphabet was also used to compare the performance of the models to other models in existence. The texts used for this comparison were the ASCII text files found in the Canterbury Corpus [Canterbury].

3.2.1 The Canterbury Corpus

Choosing representative text files is difficult [Arnold and Bell, 1997]. The Canterbury Corpus is a collection of files used for benchmarking compression programs. These files have been selected because they are considered representative. The ASCII text members of this group of files were used to test and benchmark the programs written for this report, in the hope that these will at least be more representative than an arbitrary choice by the author.

However, as all the files are quite small (less than 500kB), two larger files were used in some of the tests, namely *Emma* and *Pride and Prejudice*, two novels by Jane Austen. These were both approximately 0.8MB in size.

3.2.2 Smoothing

Taking the example of an order-3 model, there will be four smoothing coefficients:

$$P = \eta P(\text{uniform}) + \mu P(\text{monograph}) + \rho P(\text{bigraph}) + \nu P(\text{trigraph})$$

where η, μ, ρ, ν, are the smoothing coefficients to be determined. Given the coefficients could take any combination of values, finding the most suitable ones is a demanding task. This is especially true since the optimum coefficients will change with the amount of data seen. In fact each of the coefficients may be more accurately written in the form η(characters seen so far), which we will term $\eta(N)$.

3.2.3 Method 1 - An ad-hoc solution

A slightly ad-hoc solution to this was to set each of the coefficients to be some common multiple of the smaller ones, giving the more simple:

$$P = \frac{1}{Z}\left[\frac{P(\text{uniform})}{\lambda^3} + \frac{P(\text{monograph})}{\lambda^2} + \frac{P(\text{bigraph})}{\lambda} + \frac{P(\text{trigraph})}{1}\right]$$

Where λ is the smoothing coefficient, and Z is a normalising constant.

The motivation for using this form is that when a sizeable amount of data has been seen, one would expect better performance by giving the higher order models more weight than the lower order ones. By setting λ more than one, this can be easily achieved.

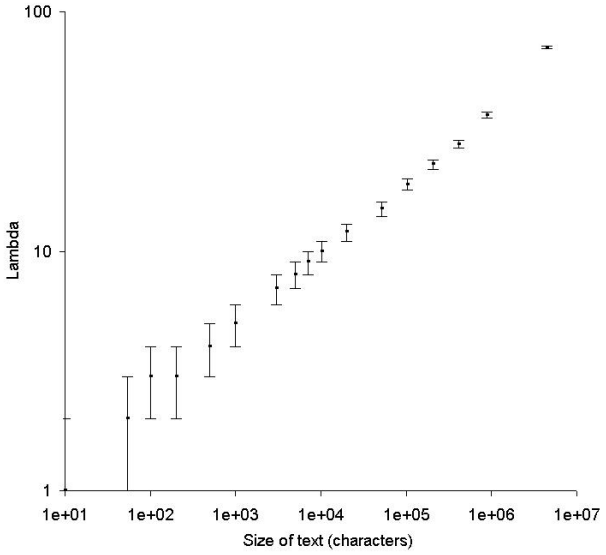

Figure 3: Graph of file size vs optimum λ for the novels of Jane Austen

When no data has been seen, we want the uniform distribution to have a greater weight than the others. This can also easily be achieved by setting λ less than one.

3.2.4 Determining λ

How do we find the value of λ that gives us the best compression? This can be done by running the program on a section of a text file, and then repeating this with different values of λ. This was done for lots of different sized sections of a text file containing the novels of Jane Austen. This resulted in a data set containing file sizes and optimum λs. The novels of Jane Austen were used as the files from the Canterbury Corpus were too small to allow investigation of text sizes over ~400kB.

The data set obtained is shown in Fig 3. Using this data set, we can estimate the form of $\lambda(N)$. The results appear to form a straight line on a log-log plot, so we assume the form

$$\lambda(N) = kN^c$$

By compressing representative files with various values of k and c, we can optimise these coefficients for compression. Strictly speaking this may impede the adaptive qualities of the model, but it was decided to continue this line of investigation to determine whether the optimum k and c are general or if they vary wildly from file to file.

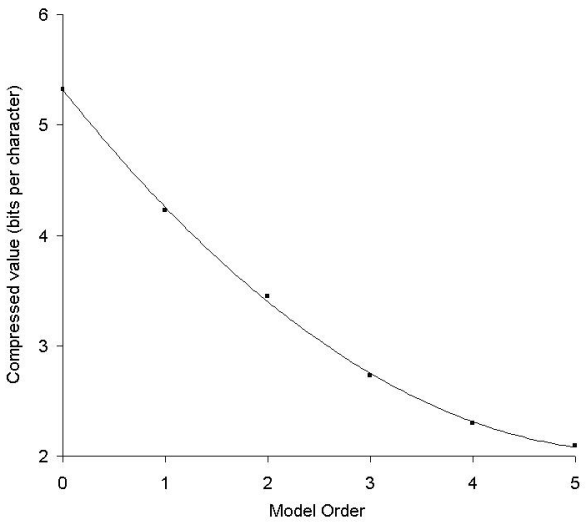

Figure 4: Compression results for different order character-based models

3.2.5 Results for different order models

Character based language models were implemented up to order 5. They were initially implemented using a 40 character alphabet. By compressing files from the Canterbury Corpus with different values of k and c, the optimum values for compression were determined. For this 40 character alphabet, the optimum form of lambda was found to be

$$\lambda(N) = 0.8N^{0.3}$$

The average compressed length per character for each order model is shown in Fig 4. The compressed values are expressed in units of bits per character, henceforth termed 'bpc'.

3.2.6 Comparison with traditional methods

A version of the order 5 model using a 128 character alphabet was implemented, in order to better compare performance with other programs. This allowed the program to handle any ASCII text input file. The optimum $\lambda(N)$ was found to be the same as for a 40 character alphabet, i.e. $k = 0.8$, $c = 0.3$. The program was used to compress text files from the Canterbury Corpus. The program's performance was compared to that of traditional methods published on the Canterbury Corpus website and is shown in Table 1.

In both the case of the 40 character and the 128 character alphabet, there was variation in optimum k and c for individual files. For the 128 character alphabet, the form of the optimum λs for individual files varied from $\lambda(N) = 0.6N^{0.32}$

Text	*Fivegraph*	*Best Reported*
alice.txt (152kB)	2.23	2.20
asyoulik.txt (125kB)	2.48	2.49
lcet.txt (427kB)	2.00	1.95
parlost.txt (481kB)	2.30	2.36
Average	**2.25**	**2.25**

Table 1: Compression performance of an order 5 model (Fivegraph) compared to traditional methods. Values are in bits per character.

to $\lambda(N) = 1.3N^{0.25}$. However, the difference in compression between encoding using the average optimum, and the optimum for an individual file, was ≤ 0.001 bpc. This difference was considered negligible.

The program performance was found to equal the best reported performance on the website. These 'best performance' results were by an implementation of PPM. There are some recently developed programs whose performance is better than those listed on the website [Bloom, 1998], but these use more complex computational models.

3.2.7 Method 2 - An alternative approach

A different approach was also tried, whereby the different order models were blended in a different way. Considering the bigraph model for a moment, we define a few new terms. $P(i|j)$ is the probability that character x_i occurs given that character x_j has just been seen. f_i is the normalised probability of character x_i occurring, as predicted by the monograph model. We define the bigraph count, i.e. the number of times x_i has occurred straight after x_j, as $F_{i|j}$. We can smooth the two different order models using

$$P(i|j) = \frac{F_{i|j} + \alpha f_i}{\sum_{i'} F_{i'|j} + \alpha f_{i'}}$$

This effectively allocates a specific smoothing coefficient to each character. This scheme was implemented using an order 5 model, where each order was combined with the one lower than it in the form above.

There are several motivations for using this form. It has been found that by having different smoothing coefficients for different groups of letters, performance can be improved [Teahan, 1998]. It is a similar form to that used in PPMC, an implementation of PPM [Teahan, 1998].

It is also the same form predicted by taking a Bayesian approach to the smoothing using a Dirichlet prior [MacKay and Peto, 1995], but using marginal statistics directly rather than the number of contexts.

A 128 character alphabet was used for this implementation, so that its results could be more easily compared to both the above approach and to the results reported on the Canterbury Corpus website.

An advantage of this approach was that there was only one parameter α, to set.

Once again, the best value for this parameter was determined by compressing the text files using different values of α. For the files from the Canterbury Corpus, an optimum α of 12 was found.

This gave an average compression of 2.27 bpc over the test files. Compare this with the result of the first method, i.e. 2.25 bpc. This alternative method of smoothing performs almost as well as the first method, but has only one parameter that needs to be set.

As with the first method, there was some variation in optimum α between the individual files. optimum αs ranged from 9 to 14 for the individual files from the Canterbury Corpus. Once again, it was found that using the best average α instead incurred only a negligible reduction in performance. In this case a difference of $\sim$0.003 bpc.

3.2.8 German

Out of interest, both approaches mentioned above were applied to some texts in German rather than English, to see if similar results were obtained. Four texts were chosen to have a total length similar to the Canterbury Corpus files. These were obtained from the Project Gutenberg website [Gutenberg], and are listed in Appendix A. An order 5 model with a 128 character alphabet was used in both approaches.

In the case of the first method with $\lambda(N) = kN^c$, it was found that the optimum form of λ was

$$\lambda(N) = 0.9N^{0.28}$$

The variation in k and c for individual files was similar to that found for English, ranging from $\lambda(N) = 0.6N^{0.32}$ to $\lambda(N) = 1.5N^{0.35}$. Again, these fluctuations made little impact to the compression ability of the average λ, typically the same as for English, although in the case of one file it was as large as $\sim$0.03 bpc.

In the case of the alternative method described in Section 3.2.7 it was found that the optimum α was 15. The optimum αs for individual files ranged from 10 to 16. Using the overall optimum α made a difference of only $\sim$0.001 bpc.

Comparing the performance of the two models on German texts, the first method averaged 2.17 bpc, whereas the alternative method of smoothing averaged 2.15 bpc.

The performance of the models over both languages averaged to 2.21 bpc in both cases. The first method performed better on English but worse on German than the second, alternative method.

Text	*Fivegraph*	*Uniform*	*Monogram*	*Bigram*
emma.28ch (845kB)	1.85	2.51	1.79	1.74
pandp.28ch (658kB)	1.86	2.40	1.75	1.70
alice.28ch (134kB)	2.07	2.31	1.82	1.76
asyoulik.28ch (117kB)	2.28	2.34	1.88	1.85
lcet.28ch (386kB)	1.86	2.05	1.61	1.56
parlost.28ch (443kB)	2.20	2.42	1.89	1.89
Average	**2.02**	**2.34**	**1.79**	**1.75**

Table 2: Performance of various word-based models, plus and order 5 character based model (Fivegraph). All programs used a 28 character alphabet. Values are in bits per character.

3.3 Word based models

The C++ program was then altered to deal with a whole word as a single token, rather than just one character. For simplicity, a 28 character alphabet was used, the lower case alphabet plus full stop and space. All words were modelled as either being followed by a full stop or a space. In order to know how many words there will be, the program looks ahead in the file and counts them, then goes back to the beginning and models the file.

This is in some sense 'cheating', as we would not have the information about the number of words if we were doing truly adaptive modelling. However, the aim of this section was to take a brief look at word based models and to see what sort of compression they could achieve, so we will lessen our strict criteria to allow this.

3.3.1 Results for word based models

Monogram and bigram models were created, and compared with a model that assigns a uniform probability to each word. The smoothing was done using the method described in Section 3.2.7. The optimum α for the bigrams was found to be $\sim$200. The models were also compared with a version of the 5-graph character program that used a 28 character alphabet. The results are shown in Table 2.

3.4 Recency Boosting

In order to make a brief investigation into the effects of mid-range correlations, the monogram model described above was taken and altered in order to 'recency boost' words. This was done by multiplying the probabilities of any word that has occurred in the last M words by a boosting factor. The probabilities are then renormalised so that they still sum to one. By testing various values, it was found that using $P_{\text{new}}(x) = 2P_{\text{old}}(x)$ when word occurred in last 200 words

Text	*Monogram*	*MonogramRB*	*Bigram*
emma.28ch (845kB)	1.79	1.79	1.74
pandp.28ch (658kB)	1.75	1.75	1.70
alice.28ch (134kB)	1.82	1.80	1.76
asyoulik.28ch (117kB)	1.88	1.85	1.85
lcet.28ch (386kB)	1.61	1.59	1.56
parlost.28ch (443kB)	1.89	1.88	1.89
Average	**1.79**	**1.78**	**1.75**

Table 3: Performance of a recency boosted monogram model (MonogramRB) compared with monogram and bigram models. Values are in bits per character.

gave the best overall results. Results are shown in Table 3.

4 Discussion

Both the character and word models discussed in this report made use of a very small set of texts to set coefficients and test compression. This is more because of the difficulty in determining what is a 'typical' text file more than any lack of text files available. Project Gutenberg [Gutenberg] alone has over 6,000 texts freely available in ASCII text format. However, rather than make arbitrary choices of what the author believed to be representative files, only those from the Canterbury Corpus were used.

As mentioned on the Canterbury Corpus website, "The files were chosen because their results on existing compression algorithms are 'typical', and so it is hoped this will also be true for new methods.". It therefore seemed sensible to use these files, both because they were considered typical, and because new algorithms used them for benchmarking.

For German, there was no 'Canterbury Corpus' freely available, so an arbitrary choice of files had to be made (see Appendix A).

4.1 Data Storage

Models of order higher than five were not implemented, mainly due to time constraints. The memory usage of the program using a 128 character alphabet is approximately 50MB for a medium sized text file. This could be greatly reduced by changing the program to use a proper 'trie' structure. This would have the form of a tree or linked list. The reduced memory usage would allow higher order models to be implemented that require only moderate amounts of RAM.

4.2 Character based models

It would be interesting to implement higher order models to see if their performance continued to improve, as the trend in Figure 4 implies.

With practical implementations of PPM using escape characters, models of order ≥ 5 actually perform worse than order 5 models [Teahan, 1998]. This is because the higher order models will need to see a lot of data before they can make reasonable predictions. Until then, the sparsity of these higher order models will cause the transmission of many escape characters, which will incur a bit penalty in the final file.

There is no reason why this should occur in the models discussed in this report. We would expect an order 7 character model to perform even better than the results reported in Section 3.2.6.

4.2.1 The form of λ

The optimum λ for use in method one was found to have the general form

$$\lambda(N) = kN^c$$

This form is the same one as is found for growth of vocabulary size with number of words seen [Salton, 1988]. Equivalently, it is the form found for the growth of a 'trie' structure with number of characters/words seen. Given that the probability of seeing a new combination of characters/words is directly related to this, it seems unlikely that this form of λ is coincidental.

The fact that one form of λ performs so well on a range of different files is surprising. One would perhaps expect that such an arbitrary form of lambda would not have widespread applicability, but this is indeed the case. Using

$$\lambda(N) = 0.8N^{0.3}$$

achieves compression within 0.001 bpc of the optimum (using this method) for any particular text file in the Canterbury Corpus.

4.2.2 Method 1 - Comparison with traditional methods

Given the simplicity of this rather ad-hoc smoothing model, and the ease of its implementation, it performs very well in comparison to the traditional methods, equalling the best performance reported on the Canterbury Corpus website.

For two of the files from the Canterbury Corpus, this ad-hoc smoothing method outperformed the best reported compression results. These were the two files that could be compressed least, presumably either because they had higher entropies, or just that their structure was less suited to this type of modelling. It would be interesting to investigate further the reasons for this, but this was not possible due to the time constraints of the project.

4.2.3 Method 2 - The alternative approach

The performance of method two was not quite as good as the first method on English text (2.27 bpc as opposed to 2.25 bpc), possibly because it could not be as highly tuned as the first model through only having one parameter.

4.2.4 Applicability to other languages

I do not believe that the slightly better performance of method two is due to 'overtuning' in method one. By 'overtuning' I mean tweaking k and c to achieve the best performance on the Canterbury Corpus files, but to the detriment of general performance.

If the slightly better performance of method two was due to the overtuning of method one, we should find that by using the German texts to set the optimum k and c, we could equal the performance of the alternative method.

This is not the case however. Indeed, the difference in compression when the optimum parameters for English rather than German are used in compressing the German texts is found to be negligible. The compression difference in this case is of order 5×10^{-5} bpc.

So the slightly better performance of method two is due to a factor other than overtuning. This may be due to the properties of the files chosen, which in the case of the German texts were rather arbitrarily chosen by the author. Alternatively, it may be due to some property of the second method that makes it more suitable to the properties of the German language.

4.3 Word based models

Word based models were found to outperform the character based models, as is to be expected from using whole words as tokens rather than individual characters. The results in Tables 2 & 3 should be viewed in the knowledge that we have looked ahead to determine which words will appear before compressing the files. A fully adaptive word model would have to be constructed before rigorous comparisons could be made. Sadly, this was not possible within the time frame of this project.

4.4 Recency boosting

By recency boosting words that had occurred in the last 200 words, an average improvement of 0.01 bpc was seen. In some texts it made a larger improvement, but no files compressed worse when using it. Given it is fairly easy to implement, it makes a viable addition to this type of language model.

It would be interesting to exploit more mid-range effects, but this was not

possible due to time constraints. For example I would have liked to try a scheme whereby a count was kept of the average 'distance' between the occurrences of the same character. This could be blended with the various order n-gram models to give some additional prediction as to when that character would be seen again. For instance, English will have some average sentence length. By keeping a track of how many characters occur between each full stop, it would be possible to augment the probability of a full stop occurring at any point using these predictions.

4.5 Future Directions

Since language modelling is such a large subject, there is plenty of scope for future directions. If more time had been available, it would have been interesting to look at some of the following ideas.

1. Less ad-hoc smoothing methods. It would be interesting to construct a character based language model using the Dirichlet prior and having the form as described in [MacKay and Peto, 1995]. This is similar to the method described in section 3.2.7, but using context information in the place of the explicit predictions of the lower order model.

2. Order changing adaptively to text content. Rather than smoothing with coefficients, an approach could be taken similar to the practical implementations of PPM, where only the predictions of one order is used. Rather than transmitting escape characters to the decompressor however, some adaptive switching inside the program could be implemented. This could be done by looking at which order model compressed the last ten characters best, and then switching to that model. This could be done inside the program so that no escape characters would need to be included in the file, hopefully allowing the construction of a text compressor where orders greater than five continue to improve the file size.

3. Training. This report has not mentioned training the model on one text before modelling another, for two reasons. First, this spoils the adaptivity of the model, as training on a file makes the language model take the form of that file's statistics, which may be completely different to those of the file we are about to model. Secondly, finding good typical texts to use for training is problematic. In the brief experiments with training that were done whilst investigating character based models, training was found to improve or worsen performance depending on the text chosen. Plus it made the model less adaptive, as the new counts from the file being modelled were swamped by the counts from the training text.

4. Scaling of frequency counts. The above training problem could be helped to some extent by dividing the counts by some factor every so often, so that whilst the statistics were still fairly accurate, the recent statistics were given more importance. This would in effect be a form of the recency boosting discussed in Section 3.4.

5. Recency boosting of nouns. Rather than recency boosting all words, perhaps boosting only nouns would give better performance. In texts such as newspaper articles, it can be the case that one person's name is used heavily in one paragraph, and then a different person's name is used extensively in the next. Recency boosting the nouns would exploit this.

5 Conclusions

1. Character based models of text were created using n-graphs and smoothing.
2. Two different smoothing schemes were investigated, both of which performed well in comparison to results obtained from traditional algorithms.
3. The smoothing coefficients in both schemes were found to be similar for all files, allowing typical values to be used without impeding adaptivity.
4. Word based models were also implemented, and suggested an increase in performance, but would have to be made fully adaptive before quantitative comparisons could be made.
5. Recency boosting was investigated as an example of exploiting mid range effects, and was found to increase performance.

A Text files used in this report

As mentioned in Section 3.2.1, finding representative text files to test compression is difficult. The following files from the Canterbury Corpus [Canterbury] were used.

1. alice.txt - *Alice's Adventures in Wonderland*, a story by Lewis Carroll (1832-1898).
2. asyoulik.txt - *As You Like It*, a play by William Shakespeare (1564-1616).
3. lcet.txt - A piece of technical writing.
4. parlost.txt - *Paradise Lost*, a poem by John Milton (1608-1674).

Also, in order to test compression on longer files, *Emma* and *Pride and Prejudice*, two novels by Jane Austen, were used. These were named emma.txt and pandp.txt respectively. These were not used in setting optimum coefficients however, as the novels were chosen rather arbitrarily. However, to determine the *form* of λ in Section 3.2.4, the complete works of Jane Austen were used, a text file of some four megabytes.

For testing the models on German, it was necessary to select files myself. Four files were chosen from Project Gutenberg [Gutenberg] to have approximately the same total length as the Canterbury Corpus files.

1. hldnde.txt - *Helden*, by George Bernard Shaw (1856-1950).
2. wsde.txt - German translation of *The Merchant of Venice*, by William Shakespeare (1564-1616).
3. oberde.txt - *Oberon*, by Christoph Martin Wieland (1733-1813).
4. faust.txt - *Faust*, by Johann Wolfgang von Goethe (1749-1832).

When compressing using the word based models, a program was written to change the texts into 28 character format. Therefore in the results tables for the word based models, the files are of the form '.28ch'.

B Code

Over forty programs were written to produce the results in this report. The most relevant examples are available in electronic form at:

http://www.burrow.org.uk/langmod/

References

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

[Cleary and Witten, 1984] 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 et al, 1995] Cleary, J.G. et al. (1995) "Unbounded length contexts for PPM" *Proceedings DCC'95*. IEE Computer Society Press, 52-61.

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

[Brigger] An explanation of Huffman coding can be found at http://ltswww.epfl.ch/pub_files/brigger/thesis_html/node93.html

[Wright, 1953] Wright, E.V. (1953) *Gadsby*. Also available online at http://www.spinelessbooks.com/gadsby/

[Price] Price, J. "ASCII Chart and other resources" at http://www.jimprice.com/jim-asc.htm/

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

[Canterbury] The Canterbury Corpus. http://www.corpus.canterbury.ac.nz

[Arnold and Bell, 1997] Arnold, R. and Bell, T. (1997) "A corpus for the evaluation of lossless compression algorithms" submitted to DCC'97.

[Bloom, 1998] Bloom, C. (1998) "PPMZ". Published on the web at http://www.cbloom.com/papers/

[Teahan, 1998] Teahan, W.J. (1998) *Modelling English Text.* PhD thesis, Univ. of Waikato, N.Z.

[MacKay and Peto, 1995] MacKay, D.J.C. and Peto, L. (1995) "A hierarchical Dirichlet Language model" Natural Language Engineering **1**(3):1-19.

[Gutenberg] Project Gutenberg, started by Michael Hart in 1971. Over 6,000 books available in ASCII text format. http://www.gutenberg.net/