Ensemble Prediction by Partial Matching

Byron Knoll
Computer Science 540 Course Project
Department of Computer Science
University of British Columbia

Abstract

Prediction by Partial Matching (PPM) is a lossless compression algorithm which consistently performs well on text compression benchmarks. This paper introduces a new PPM implementation called PPM-Ens which uses unbounded context lengths and ensemble voting to combine multiple contexts. The algorithm is evaluated on the Calgary corpus. The results indicate that combining multiple contexts leads to an improvement in the compression performance of PPM-Ens, although it does not outperform state of the art compression techniques.

1 Introduction

Prediction by Partial Matching (PPM) [1] is a lossless compression algorithm which consistently performs well on text compression benchmarks. There are a variety of PPM implementations with different performance properties. This paper introduces a new PPM implementation called PPM-Ens which uses ensemble voting to combine multiple contexts. Section 2 provides basic information about PPM and discusses some related compression techniques. Section 3 provides details about the PPM-Ens algorithm and how it was created. Section 4 provides additional background information on how compression performance can be empirically evaluated. Section 5 discusses how automated parameter tuning improves the performance of PPM-Ens. Section 6 evaluates the performance of PPM-Ens on a standard compression corpus. Finally, sections 7 and 8 discuss the results.

2 Background

An arbitrary data file can be considered as a sequence of characters in an alphabet. The characters could be bits, bytes, or some other set of characters (such as ASCII or Unicode characters). Data compression usually involves two stages. The first is creating a probability distribution for the prediction of each character. The second is to encode these probability distributions into a file using a coding scheme such as arithmetic coding [2] or Huffman coding [3]. PPM is concerned with the first task of generating a probability distribution for the prediction of the next character in a sequence.

Consider the alphabet of lower case English characters and the input sequence “abracadabra”. For each character in this string, PPM needs to create a probability distribution representing how likely the character is to occur. However, the only information it has to work with is the record of previous characters in the sequence. For the first character in the sequence, there is no prior information about what character is likely to occur, so assigning a uniform distribution is the optimal strategy. For the second character in the sequence, ‘a’ can be assigned a slightly higher probability because it has been observed once in the input history.

Consider the task of predicting the next character after the sequence “abracadabra”. One way to go about this prediction is to find the longest match in the input history which matches the most recent input. The most recent input is the character furthest to the right and the oldest input is the character

File	Size (KiB)	Description
bib	111.261	structured text (bibliography)
book1	768.771	text, novel
book2	610.856	formatted text, scientific
geo	102.400	geophysical data
news	377.109	formatted text, script with news
obj1	21.504	executable machine code
obj2	246.814	executable machine code
paper1	53.161	formatted text, scientific
paper2	82.199	formatted text, scientific
pic	513.216	image data (black and white)
progc	39.611	source code
progl	71.646	source code
progp	49.379	source code
trans	93.695	transcript terminal data

Table 1: File size and description of Calgary corpus files.

furthest to the left. In this case, the longest match is "abra" which occurs in the first and eighth positions. The string "dabra" is a longer context from the most recent input, but it doesn't match any other position in the input history. Based on the longest match, a good prediction for the next character in the sequence is simply the character immediately after the match in the input history. In this case, after the string "abra" was the character 'c' in the fifth position. Therefore 'c' is a good prediction for the next character.

Longer context matches can result in better predictions than shorter ones. This is because longer matches are less likely to occur by chance or due to noise in the data. Consider using a context length of one in the "abracadabra" example. This would involve making a prediction of the next character in the sequence based on the characters that occur immediately after 'a' in the input history. In this case, 'b' occurs twice, 'c' occurs once, and 'd' occurs once. Hence, 'b' can be assigned a higher probability than 'c' and 'd'.

PPM essentially creates probability distributions according to the method described above. Instead of generating the probability distribution entirely based on the longest context match, it blends the predictions of multiple context lengths and assigns a higher weight to longer matches. There are various techniques on how to go about blending different context lengths.

Most PPM variants only consider perfect context matches. However, if the data is known to be noisy, in some applications there may be a benefit to allowing a certain number of errors in a context match. This has led to the development of an algorithm called Prediction by Partial Approximate Matching (PPAM) [4]. PPAM was developed to perform lossless image compression. The pixels of an image tend to contain more noise than some other domains, such as the characters in a text document. PPAM was shown to have superior compression performance compared to PPM for images.

It should be noted that although PPM performs well on text compression benchmarks, there are other state of the art algorithms which outperform it. One example of a compression benchmark is the Hutter Prize [5]. This is a contest to compress the first 100MiB of Wikipedia. An algorithm called PAQ [6] currently dominates the contest. PAQ is closely related to PPM, improving on it by combining contexts which are arbitrary functions of the input history. Another example of an algorithm which achieves state of the art cross entropy rates on other datasets is the stochastic memoizer [7].

3 Algorithm Development

The maximum context size of PPM is usually bounded in order to improve prediction accuracy and avoid exponential memory usage. A PPM implementation called PPM*C [8] demonstrates how unbounded length contexts can be used to improve prediction accuracy. PPM-Ens was created based on this work. It uses unbounded length contexts and ensemble voting to mix context models. Much of the development of PPM-Ens was influenced by empirical performance evaluations on data

from the Calgary corpus [9], a standard dataset used for comparing lossless compression algorithms. Table 1 gives a summary of the Calgary corpus files.

PPM-Ens has the advantage of linear memory usage (in terms of context length) instead of the exponential memory requirements of most PPM variants. However, the time complexity of PPM-Ens is quadratic in the length of the text which is slower than typical PPM implementations. PPM-Ens maintains a complete input history of characters it encounters and uses it for the prediction of future characters.

There seems to be no theoretical basis for a particular method of creating the probability distributions for character prediction using PPM [1]. The formula used to calculate the weights in PPM-Ens is based on a combination of results from previous papers and modifications based on empirical testing. For a particular context length, the following formula was used to determine the probability of a character x:

$$p(x) = \frac{c}{a} \times \left(\frac{a}{a+b} \right)^{param1}$$

a is the number of context matches, b is the cardinality of the set of characters encountered after the matches, and c is the number of times x occurs after each match. $param1$ is a tunable parameter.

The probability distributions for the different context lengths were combined together using a weighted average (ensemble voting). The weight w for a particular context length n was calculated using the following recursive function:

$$w(n) = \begin{cases} 1 & \text{if } n = maxLength, \\ param2 & \text{if } w(n+1) < param2, \\ param3 \times w(n+1) + (1 - param3) \times \frac{w(n+1) \times b}{a+b} & \text{otherwise} \end{cases}$$

$param2$ and $param3$ are tunable parameters (in the range of zero through one) and $maxLength$ is the length of the maximum context match. Finally, the resulting probability distribution over characters was normalized to sum to one.

PPM-Ens also uses ensemble voting to average over different types of contexts. The above formulas are used to calculate the probability of a character for each context. Contexts can be different functions of the input history. For example, instead of being the sequence of most recent characters, a context could be a sequence starting at the most recent character and skipping every second character encountered. For the string "abracadabra" this context would contain "arcdba". Similarly, other contexts can be created by skipping two out of every three characters, three out of every four characters, and so on. Even more contexts can be created by considering an offset from the most recent input character for the start of the context. For example, a context with an offset of one and skipping every second character in the string "abracadabra" would contain "baaar". Contexts which skip characters in the input sequence have the advantage of potentially finding longer matches in the input history. This is because a character which could have blocked a match for one context could be skipped by another context. This is why combining the information from multiple contexts can lead to a performance benefit for PPM-Ens.

PPM-Ens combines the information from eight different contexts. These are:

1. offset of zero and use every character
2. offset of one and use every character
3. offset of zero and use every second character
4. offset of one and use every second character
5. offset of zero and use every third character
6. offset of one and use every third character
7. offset of zero and use every fourth character
8. offset of one and use every fourth character

Each additional context causes PPM-Ens to take a constant factor longer to run (but does not affect its big O time complexity). Based on empirical testing, any additional contexts tend to cause an increase in compression performance of PPM-Ens. The reason the above eight contexts were chosen for PPM-Ens is because they provided a reasonable trade-off between running time and compression performance based on empirical evaluation on the Calgary corpus files.

Contexts were combined using a weighted average. The basic idea was to assign higher weights to contexts which have better predictive accuracy. The following sequence of formulas was used to determine the weight w for a particular context d:

1. $w(d) = 1 - \frac{crossEntropy(d)}{8}$
2. Normalize the weights over contexts so that they sum to one.
3. Set e to be the context with the lowest weight.
4. $w(d) = (w(d) - param4 \times w(e))^{param5 \times numContexts}$
5. Normalize the weights over contexts so that they sum to one.

The $crossEntropy$ function is a measure of the compression performance of a particular context, and is discussed in the following section. $numContexts$ is the total number of contexts being combined (which is eight for PPM-Ens) and $param4$ and $param5$ are tunable parameters. $param4$ is constrained to be between the values of zero through one. These formulas were constructed based on empirical testing on the Calgary corpus. When examining the weights assigned to the contexts after step 2, the values are very similar because there is not a large difference in the cross entropy rates between contexts. The purpose of steps 3-5 is to assign a higher weight to the best context and a lower weight to the worse contexts (essentially making the distribution less uniform).

4 Performance Metrics

One way of measuring compression performance is to use the file size of compressed data. However, file size is dependent on a particular type of coding scheme (such as arithmetic coding or Huffman coding). Since PPM is concerned with generating probability distributions for the prediction of characters, there are ways to measure its compression performance directly from these distributions.

There are three common metrics used to measure the performance based on the predicted probability distributions: cross entropy, perplexity, and prediction error. Cross entropy can be used to estimate the average number of bits needed to code each byte of the original data. For a sequence of N characters x_i, and a probability $p(x_i)$ assigned to each character by the prediction algorithm, the cross entropy can be defined as:

$$-\sum_{i=1}^{N} \frac{1}{N} log_2 p(x_i)$$

This gives the expected number of bits needed to code each character of the string. Another common metric used to compare text prediction algorithms is perplexity which can be defined as two to the power of cross entropy:

$$2^{-\sum_{i=1}^{N} \frac{1}{N} log_2 p(x_i)}$$

In 1991, a trigram model was used to estimate an upper bound on the cross entropy of English. The trigram model was used on a large corpus of one million English words to achieve a perplexity score of 247 per word, corresponding to a cross entropy of 7.95 bits per word or 1.75 bits per letter [10]. On this corpus, ASCII coding has a cross entropy of 8 bits per character, Huffman coding has 4.46, and the UNIX command `compress` has 4.43. On more specialized corpora it is possible to achieve lower perplexity scores than for more general corpora. For example, a word perplexity score of 96.9 was reported on the Associated Press corpus by the stochastic memoizer. This is significantly lower than the perplexity scores reported by competing approaches.

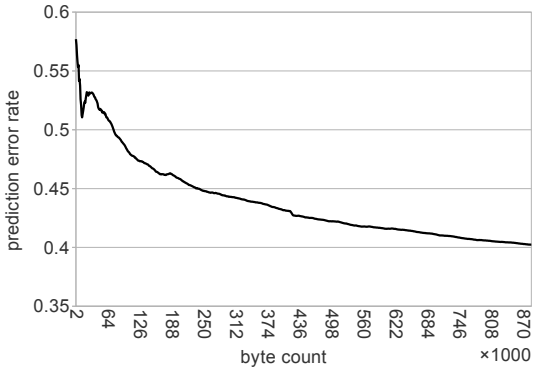

Figure 1: Average prediction error rate for each byte of the novel Twenty Thousand Leagues Under the Sea.

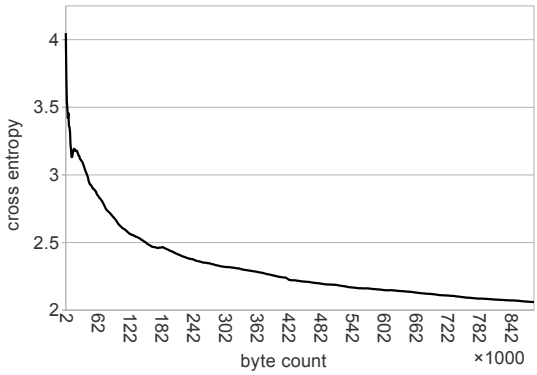

Figure 2: Cross entropy for each byte of the novel Twenty Thousand Leagues Under the Sea.

Finally, a third common metric used is prediction error. For each character x_i a prediction can be made based on the character assigned the highest probability. The prediction error is simply the total of the number of incorrect predictions divided by N.

Figure 1 shows the average prediction error rate of PPM-Ens for each byte of the novel Twenty Thousand Leagues Under the Sea (Jules Verne, 1870). Figure 2 shows the cross entropy rates for the same data. It can be noted that the shape of the two curves are very similar. Both curves have a bump near byte number 50,000 which could indicate a section of the novel which was particularly difficult to compress.

All of the experiments in this report were run on a computer with the following specifications:

- CPU: Intel(R) Core(TM)2 Quad CPU Q6600 @ 2.40GHz
- Memory: 3.2 GiB
- OS: Ubuntu 9.10 (2.6.31-20-generic)
- Java(TM) SE Runtime Environment version 1.6.0_16

5 Parameter Tuning

PPM-Ens has five parameters which are used to determine the weights for ensemble voting. All five parameters are double precision floating-point numbers. $param2$, $param3$, and $param4$ are constrained to be between the values of zero and one, while the other two can be any value. The parameters are not independent which means changing the value of one parameter might change what the optimal values are for the other four. In addition, the parameters have different optimal values for different types of data. In a scenario in which we know a priori that the compressor will be used for natural language data, the parameters can be tuned based on a training set of text

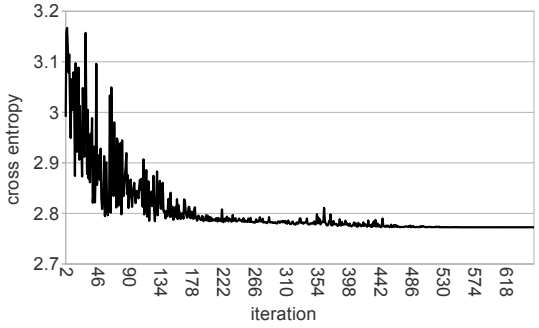

Figure 3: Cross entropy for the file 'bib' using CMA-ES for parameter tuning.

documents. The five PPM-Ens parameters only have an effect on compression performance and have no impact on memory usage or runtime of the algorithm.

Initially the five PPM-Ens parameters were set using manual tuning. The manual tuning was done by optimizing the cross entropy rate of the 'bib' file in the Calgary corpus. It was performed using approximate independent ternary searches on the five parameters. The final parameter values used were 2, 0.0001, 0.2, 0.999, and 1 for the first through fifth parameters respectively.

Automated parameter tuning was performed using Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [11]. CMA-ES is known to be effective at optimizing a small number of continuous parameters. In addition, CMA-ES does not require the use of user supplied meta-parameters. A Java implementation of CMA-ES was used from Hansen's webpage (http://www.lri.fr/∼hansen/cmaes_inmatlab.html). In order to use CMA-ES for tuning the PPM-Ens parameters, two Java functions needed to be created. The first was to define the feasible region for the parameters. The second was to determine the value of the objective function for a particular assignment of parameter values. This objective function was implemented as the resulting cross entropy when PPM-Ens was run to compress a particular file.

The parameter values of PPM-Ens were optimized based on the cross entropy of the first 10,000 bytes of the 'bib' file in the Calgary corpus. The tuning took approximately 12 hours to run (wall clock time). The program was manually terminated when the cross entropy reached a relatively stable point of convergence. Manual tuning for the 'bib' file was performed in approximately two hours of time, so it was significantly faster than using CMA-ES. Figure 5 summarizes the results. Each iteration refers to an evaluation of the objective function (cross entropy of 'bib') for a particular parameter configuration. The figure demonstrates how initially there is a high variance in the cross entropy rates but as the number of iterations increases the cross entropy rates converge to a single value. The experiment was terminated after 657 iterations. The final parameter values were 1.65897, 0.0027, 0.02187, 0.8471, and 1.47531 for the first through fifth parameters respectively. The final cross entropy rate was 2.77265. This is slightly better than the cross entropy rate of 2.81789 that was obtained using the manually tuned values.

6 Results

The performance of PPM-Ens was evaluated on the Calgary corpus. Table 2 summarizes the evaluation results. PPM-Orig refers to PPM-Ens except with just a single context (offset of zero and using every character). PPM-Orig had an average cross entropy of 2.28. This is better than the 2.34 achieved by PPM*C. PPM-Ens outperformed PPM-Orig with an average cross entropy rate of 2.23. However, another PPM variant called cPPMII-64 [12] outperforms all of these algorithms with a cross entropy of 2.04.

Using the Wilcoxon signed-rank test [13], we can calculated whether there is a significant performance difference between PPM-Orig and PPM*C. Performing this test results in a p-value of 0.001618, indicating that there is a significant difference between the cross entropy rates of these algorithms. Similarly, there is also a significant performance difference between PPM-Ens and PPM-Orig (p=0.003603), and cPPMII-SE and PPM-Ens (p=0.01423).

File	PPM*C	PPM-Orig	PPM-Ens	cPPMII-64
bib	1.91	1.87	1.82	1.68
book1	2.40	2.28	2.3	2.14
book2	2.02	1.94	1.93	1.78
geo	4.83	4.72	4.45	4.16
news	2.42	2.36	2.33	2.14
obj1	4.00	3.92	3.87	3.50
obj2	2.43	2.42	2.28	2.11
paper1	2.37	2.33	2.31	2.14
paper2	2.36	2.28	2.28	2.12
pic	0.85	0.80	0.78	0.70
progc	2.40	2.36	2.33	2.16
progl	1.67	1.63	1.59	1.39
progp	1.62	1.62	1.60	1.39
trans	1.45	1.42	1.37	1.17
average	2.34	2.28	2.23	2.04

Table 2: Cross entropy rates on Calgary corpus files.

The performance of PPM-Ens was also evaluated on the first 10 million bytes of the Hutter prize data (contained text from Wikipedia). The reason only a subset of the data was used is due to the long running time of PPM-Ens. The final cross entropy rate was 1.93392. For comparison, the PAQ program paq8l compressed the same 10 million bytes to a file size of 1981983 bytes. This corresponds to 1.58559 bits per byte which is significantly better than the cross entropy rate of PPM-Ens.

7 Discussion and Future Work

The results from evaluating PPM-Ens on the Calgary corpus indicate that the use of multiple contexts leads to an improvement in compression performance. PAQ uses a similar strategy of combining multiple contexts. The cost of this improvement in compression performance is an increase in running time by a constant factor. Since memory usage in PPM-Ens is dominated by storing the input history, the use of additional contexts does not have a significant impact on memory usage. For determining the optimal number of contexts to use, a trade-off needs to be made between running time and compression performance. This trade-off is application specific and depends on the type and size of data that is being compressed.

PPM-Ens uses relatively simple contexts based on skipping characters in the input history. Further performance improvement might be achieved by combining the predictions from more advanced models. For example, the predictions created by an algorithm like PPAM could lead to better performance on image data. PAQ uses this technique to combine the information from multiple specialized models for particular types of data.

The automated parameter tuning on PPM-Ens also lead to an improvement in performance. However, this was at the cost of a long optimization process. Parameter tuning is also sensitive to the type of data that it is trained on. The objective function of the optimization algorithm that is used for the parameter tuning needs to use a representative sample of the type of data that the compressor will be used for. In certain applications it may be hard to find training data which represents the intended use of the compression algorithm. For example, if a compression algorithm is being designed to work on a large range of data types, the amount of training data needed may be infeasibly large. One potential solution to this problem is the use of adaptive parameter tuning during the run of the compression algorithm. This way the parameters could be used to adapt to the type of data that is currently being compressed. This would be an interesting area for future work.

Figure 1 shows how the prediction error rate for a novel changes sequentially from the beginning of the text to the end using PPM-Ens. The prediction error at the end of the text was 41.066%. As expected, the graph is indicative that the prediction error roughly converges towards a horizontal asymptote as more data is encountered. This information could be useful for determining a limit on the size of the history that needs to be stored. By limiting the history size, the time complexity of

PPM-Ens is reduced from $O(N^2)$ to $O(N)$ and the memory usage is reduced from $O(N)$ to $O(1)$ where N is the length of the data. The prediction error rate will not be substantially affected when a bound on the history is imposed if the bound is sufficiently large. One simple way of enforcing the memory bound is to use a sliding window and discard everything before a certain point in history. An improvement upon this naïve approach is to remove portions of the history which are rarely matched. This would be another interesting modification for future work.

8 Conclusions

Although the experiments in this paper focussed on PPM-Ens, the results can apply to other PPM implementations as well. For example, any PPM implementation which uses parameters could benefit from automated parameter tuning in certain applications. The main result of this paper indicates that combining multiple contexts can improve compression performance. Existing PPM implementations could be easily modified to make use of additional contexts. This modification can be made by using the exact same prediction algorithms but on different portions of the input data. Finally, combining the probability distributions created from different types of predictive compression algorithms using ensemble voting could also lead to an improvement in compression performance.

References

[1] Cleary, J. & Witten, I. (1984) Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, Vol. 32 (4), pp 396-402.

[2] Witten, I. H., Neal, R., & Cleary, J. G. (1987) Arithmetic coding for data compression. *Comm. ACM*, vol. 30 pp. 520-540.

[3] Huffman, D. A. (1952) A method for the construction of minimum-redundancy codes. *Proc. Inst. Electr. Radio Eng*. 40, 9 (Sept. 1952), pp. 1098-1101.

[4] Zhang, Y., & Adjeroh, D. A. (2008) Prediction by partial approximate matching for lossless image compression. *IEEE Trans. Image Process*. 17 (6), pp 924-935.

[5] Hutter, M. (2006) The human knowledge compression prize. http://prize.hutter1.net

[6] Mahoney, M. (2005) Adaptive weighing of context models for lossless data compression. Florida Tech. Technical Report, CS-2005-16.

[7] Wood, F., Archambeau, C., Gasthaus, J., James, L., & Teh, Y.W. (2009) A stochastic memoizer for sequence data. In ICML 09: Proceedings of the 26th Annual International Conference on Machine Learning, pp 1129-1136.

[8] Cleary, J. & Teahan, W. (1997) Unbounded length contexts for PPM. *Comput. J.* 40, 2/3, pp 67-75.

[9] Bell, T., Cleary, J., & Witten, I. (1990) Text Compression. Prentice-Hall, Englewood Cliffs, NJ.

[10] Brown, P., Della Pietra, S., Della Pietra, V., Lai, J., & Mercer, R. An estimate of an upper bound for the entropy of English. *Computational Linguistics*, 18(1), pp 31-40.

[11] Hansen, N. & Ostermeier, A. (2001) Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation*, 9(2), pp 159-195.

[12] Shkarin, D. (2002) PPM: One step to practicality. In Data Compression Conference, pp 202-212.

[13] Wilcoxon, F. (1945) Individual comparisons by ranking methods. *Biometrics*, 1, pp 80-83.