A Quick Introduction to Data Compression Through Learning

Francisco Claude

David R. Cheriton School of Computer Science
University of Waterloo, Canada
fclaude@cs.uwaterloo.ca

ABSTRACT

We present an introduction to data compression with an example of a simple technique based on Bayesian Inference, which achieves compression ratio similar to known compression programs in practice. We relate this method to known compression techniques. The main goal is to show data compression from a learning point of view and encourage further research on compression of biological sequences.

1 Introduction

Data compression aims at representing a sequence using as little space as possible. Compression algorithms can be roughly divided into two groups: dictionary based and statistical.

Dictionary based methods can be explained as representing the sequence, based on previously seen substrings. Some examples of such algorithms are LZ77 [1], LZ78 [2] and Re-Pair [3] [1].

Statistical compressors rely on predicting the next symbol to appear in the sequence, encoding this information and the predictive model in an efficient way. Classical examples are Huffman [4] and arithmetic coding [5]. In some cases, transforming the sequence allows to achieve better compression with simple methods, such an example is the Burrows-Wheeler transform [6, 7].

We first discuss some necessary background in Section 2; then, in Section 3, we show a simple and direct way to use Bayesian inference to compress data. In Section 4 we show experimental results obtained by the methods proposed in Section 3, aiming at giving some empirical feeling to the reader. Then, we discuss the connection of our proposal to existing work (Section 5), and finally we conclude mainly presenting a simple open problem (Section 6).

2 Data Compression

The efficiency of a compression method is usually measured by comparing the length of the resulting sequence with the entropy of the source.

DEFINITION 2.1 (ENTROPY [4]). *The entropy of a sequence S of length n, drawn from an alphabet Σ of size σ, is defined as*

$$H(S) = -\sum_{c\in\Sigma} P(c)\log P(c),$$

where $P(c)$ is the probability of seeing a symbol c in S.

If we consider each symbol independent from each other, the resulting value for the entropy is called zero-order entropy (H_0) and it is a lower bound on a symbol-based coder. This means that we can not compress a sequence S to less than $nH_0(S)$ if we give a single code to each symbol of the alphabet ignoring the context in which it appears. Another definition, very useful in practice, is the k-th order entropy [4], H_k, it considers contexts of length k and is defined as

$$H_k(S) = -\sum_{s\in\Sigma^k} P(s)\sum_{c\in\Sigma} P(c|s)\log P(c|s).$$

In a practical setting this definition is expressed using frequencies, $P(c) = C(c)/n$, where $C(c)$ is the number of times c appears in the sequence. This corresponds to the definition of *empirical entropy* [7].

A simple approach to achieve compression close to the k-th order entropy is to model the symbols distribution for each possible context of length k, assign zero-order codes to each model and then compress the sequence using

[1] The last two have a straightforward grammar-representation and are sometimes referred as grammar-based compressors.

them. This method is simple and seems that it could achieve a good compression ratio, but it hides a huge overhead in storing the models. There are σ^k contexts of length k that have to be stored, each of them with their corresponding model.

For the rest of this work we will assume the sequence we want to compress to be generated by a stationary Markov process in which each symbol depends only on the past k symbols seen. So, for our purposes, $H(S)$ and $H_k(S)$ are the same. We make use of the n-gram model used for natural language processing (NLP) [8].

It is clear that there will be a trade off between the complexity of the model, the space required to store it, and the compression ratio achieved. From the pure compression point of view, it is interesting to have a model that adapts itself over the part of the sequence already seen. This allows the encoder and the decoder to adapt their model in a similar way and by doing so encode and decode without storing much of the model (only the initial assumptions or priors). A compression method that works this way is Prediction by partial matching (PPM) [9]. The main idea is to adapt the model based on the context seen, and if that context has not been seen before, the (de)compressor tries to find a shorter context seen before in order to predict the next symbol.

In this work we will rely on Arithmetic compression (AC) [5]. Given a probability distribution of a set of sequences, AC maps the probability of a given sequence to a range in $[0,1)$, and allows to represent that sequence with a number in that range. In order to avoid the assumption of infinite precision, many authors (see [10] for further details) have showed how to handle the intervals using finite precision for the numbers and improve upon coding efficiency. The most important property of AC for our work is that the encoding process is separated from the probability model we use for the source, and thus it allows us to modify it as we learn from the sequence and not having to deal with the problems presented by other codification methods like Huffman [11] for this scenario. It has been shown that AC achieves compression close to the entropy of the sequence, in particular, we have the following theorem:

THEOREM 2.2 (ARITHMETIC CODING OPTIMALITY [5]). *Let S be a sequence of length n, drawn from an alphabet Σ of size σ. Consider L to be the length of encoding S using arithmetic coding (with the right probability distribution). Then, L satisfies*

$$|S|H(S) < L < |S|(H(S)+2).$$

There have been other approaches for estimating the probability of a symbol given the history seen in the sequence. In particular, an interesting work that plugs AC with their modeling system was presented by Davies and Moore [12], where they show how to train a Bayesian Network in order to estimate the probability distribution of the next symbol.

One of the best compression methods, considering compression ratio, is PPM [9]. The main idea is to model the source and predict the next symbol based on the symbols seen previously. Usually, we have to define a model for predicting the next symbol and this model is application-dependent [13]. The classical implementation works in a similar way to the two examples shown in this survey, we comment on this in Section 5.

3 Simple Statistical Compressor

In general, the n-gram model for NLP is used over words. In our case, we will apply this over symbol. The same ideas are valid for word-based compressors, which in practice have shown to achieve better compression ratios for natural language [14].

Let $S = s_1 s_2 \dots s_n$ be a sequence of length n, drawn from an alphabet $\Sigma = \{1, 2, \dots, \sigma\}$ of size σ. We will consider contexts of length k. Our goal is to estimate $P(s_j | s_{j-1} s_{j-2} \dots s_{j-k})$. For estimating the probability of $P(s_j | s_{j-1} s_{j-2} \dots s_{j-k})$, we will use a very simple maximum likelihood estimator [8, 15], where we model $P(s_j s_{j-1} \dots s_{j-k})$ as $C(s_j s_{j-1} \dots s_{j-k})/(N+B)$; N is the number of training instances, B is the number of possible classifications for the training text, and $C(s)$ corresponds to the frequency of s. For estimating the probability $P(s_j | s_{j-1} s_{j-2} \dots s_{j-k})$ we have:

$$P(s_j | s_{j-1} s_{j-2} \dots s_{j-k}) = \frac{P(s_j s_{j-1} s_{j-2} \dots s_{j-k})}{P(s_{j-1} s_{j-2} \dots s_{j-k})} = \frac{C(s_j s_{j-1} s_{j-2} \dots s_{j-k})}{C(s_{j-1} s_{j-2} \dots s_{j-k})}$$

This estimation has problems with zero frequencies, the probability of seeing a new symbol after a context is zero. For fixing this issue, we will use two different methods:

- M0: Laplace Law's results by adding one to the frequency count, obtaining

$$P_{Lap}(s_1 s_2 \dots s_n) = \frac{C(s_1 s_2 \dots s_n) + 1}{N + B}.$$

By using this, we actually obtain the value of the Bayesian estimator that assumes a uniform prior over the symbols in the sequence [8].

- M1: As a second approach, we will consider a special symbol representing an unseen symbol of the alphabet. For the coding purposes, each time we see a symbol whose probability is zero, we emit this special symbol and then we encode the symbol using $\lceil \log \sigma \rceil$ bits. Then we update the frequency counters and the probability distribution for the context.

For both methods, as in general with AC, we add a special symbol to the sequence that represents the end of the string, which is the symbol that tells the decoder when to stop decoding.

4 Experimental Setup

We compare the two models with AC[2] (M0 and M1) against the compression ratios reported in Pizza&Chili[3] for `bzip2`, `gzip`, and `ppmdi`. `Bzip2` is based on the Burrows-Wheeler Transform [6], `gzip` is based on the Lempel-Ziv family [1, 2], and `ppmdi` is an implementation of PPM [9]. In this test we consider the file `dna.50MB`. `Bzip2` achieves 25.98%; `gzip` 27.05%; `ppmdi` achieves 23.84%; finally M0 and M1 achieve 23.84% and 23.82% respectively. This shows that in this case the methods are competitive with general purpose compressors.

The second test considers the file `Hemo`[4]. We compare the result of compressing this file against compressors designed for the specific task of compressing biological data, using the results provided in [16]. `XM` [17][5] achieves 7.64%, `dna2` [18] achieves 9.65%, and `Comrad` [19] achieves 12.68%. The best lines achieved by M0 and M1 are 17.75% and 17.18%. As a guideline, gzip and bzip achieve 10.81% and 10.77% respectively.

Next, we study the effect of *k*, the length of contexts, in the compression. Figure 1 (left and middle) shows the compression ratio, as a fraction of the text, for different *k*s. It can be seen that the function described reaches a minimum and then it starts growing. The problem here is that when we divide the training set into too many bins, which is the case for large contexts, the number of elements hitting the bins are few and the prediction looses. With more data it would be possible to achieve better results for larger contexts, but it is given by a trade off on how much we can collect from the sequence we are compressing. As an example to illustrate this, we consider the DNA file and generate 3 prefixes of it: 10, 30 and 50MB. Each prefix is compressed with different *k*s and we evaluate the compression ratio achieved. Figure 1 (right) shows the results. As it can be seen, the prediction for larger *k*s improves in the larger files, and the longer the file, the better the compression tends to be.

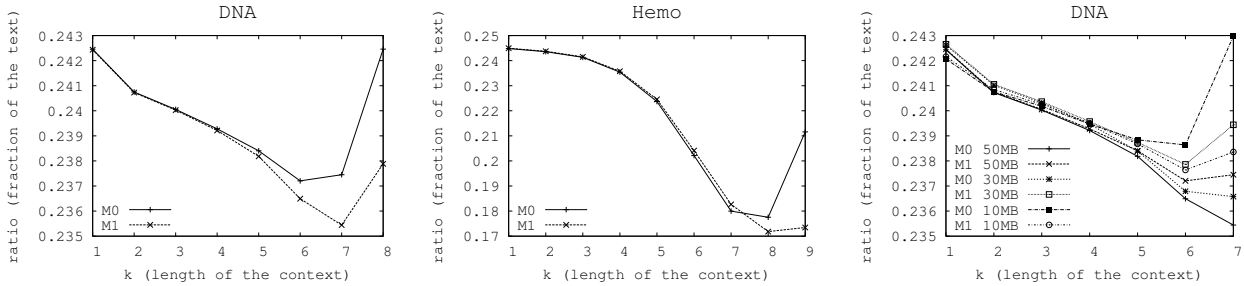

Figure 1. Compression ratio, as fraction of the text, for DNA and Hemo using methods M0 and M1. The leftmost picture shows that the methods achieve better compression as they see more examples.

5 Connection to Other Compression Methods

There is direct connection between the method explored in this work and PPM [9]. The main difference is how PPM handles the non-existing contexts or zero frequencies. The solutions used by PPM is quite similar, if it finds a context that has not been seen before, it tries with a shorter one. It keeps doing this until reaching the point of a

[2] Using the implementation of [5].

[3] http://pizzachili.dcc.uchile.cl/

[4] http://ww2.cs.mu.oz.au/∼kuruppu/comrad/hemoglobin.fa.gz

[5] Which uses a much more sophisticated modeler in a similar approach, keeping many modelers and combining them by the use of learning algorithms. In general the compression of biological data is hard because contexts of length *k* are not a good predictor.

context of length 0 or finding a context that allows to model the next symbol. This seems to be an alternative to the previous model that considers more information for modeling. If a short sequence is very rare, it can be used as a context instead of trying to embed this rare sequence inside a larger context, where it is very likely that we will not find many training examples and thus lose prediction capabilities.

All these methods, the ones presented here, PPM, etc., are just different methods for predicting the next symbol in a sequence plugged with arithmetic coding. This means, we are just trying to find a good alternative for the modeler. In the case of DNA and protein sequences this has been proven to be a very challenging problem.

6 Conclusions and Open Problems

We showed a simple relationship between learning/inference and data compression. This connection is not new, it was previously stated by Cover and Thomas [4], where they relate a good data compressor with a good gambler. In principle the idea is the same, but in our case we limited our selfs to Bayesian gamblers.

The option of exploring more general or powerful estimation methods and applications-dependent priors is an exciting path to work further in this topic. Another attractive direction is based on the convergence of the model to the real distribution. A common measure to quantify the closeness between two probability distributions is the Kullback-Leibler distance. In the strict sense it is not a distance, but it is always positive and evaluates to 0 only if the two probability distributions are the same.

DEFINITION 6.1 (KULLBACK-LEIBLER DISTANCE[4]). *The Kullback-Leibler (KL) Distance between two probability distributions $P(x \in X)$ and $Q(x \in X)$ is defined as:*

$$D(P||Q) = \sum_{x \in X} P(x) \frac{P(x)}{Q(x)}$$

It would be really interesting to see how fast can we approach the real probability distribution of the symbols. Given this result, and the following theorem, we could give a bound on the compression achieved by our method.

THEOREM 6.2 (WRONG CODE[4]). *The expected length under $P(x)$ of the code assignment with lengths $l(x) = \left\lceil \frac{1}{Q(x)} \right\rceil$ satisfies:*

$$H(P) + D(P||Q) < E_P l(x) < H(P) + D(P||Q) + 1$$

Finally, another path to explore is to build a transformation based on the estimations made. The main idea is that when processing position $\ell+1$, we have seen $s_1 s_2 \ldots s_\ell$ and we can try to predict $s_{\ell+1}$. If we write down the number of trials required to predict this next symbol. We can recreate the original sequence from this one by running a decoder that does the inverse process. If our predictor is good, we will have a sequence biased on to small numbers, and thus we could aim at compressing it very well. This could lead to similar results as the Burrows-Wheeler transform, where simpler compression methods applied over the transformation achieve competitive results.

It is interesting to notice that the transformation is very similar to the idea used originally by Shannon to estimate the entropy (upper bound) of the English language [20, 21]. The estimation was later improved by Cover and King [22], where they approach the problem as a gambling question, which helps to estimate a probability for the next symbol and not only the order of possible symbols for the next position. An interesting discussion about this can be found in the work presented by Teahan and Cleary [23].

Acknowledgements

The author would like to thank Gonzalo Navarro and Pascal Poupart for their useful comments.

References

[1] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory* **23**(3), pp. 337–343, 1977.

[2] J. Ziv and A. Lempel, "Compression of individual sequences via variable length coding," *IEEE Transactions on Information Theory* **24**(5), pp. 530–536, 1978.

[3] J. Larsson and A. Moffat, "Off-line dictionary-based compression," *Proc. IEEE* **88**(11), pp. 1722–1732, 2000.

[4] T. Cover and J. Thomas, *Elements of information theory*, John Wiley and Sons, Inc., 1991.

[5] M. C. E. Bodden and J. Kneis, "Arithmetic coding revealed - a guided tour from theory to praxis," Tech. Rep. SABLE-TR-2007-5, Sable Research Group, School of Computer Science, McGill University, Montréal, Québec, Canada, May 2007.

[6] M. Burrows and D. Wheeler, "A block sorting lossless data compression algorithm," Tech. Rep. Technical Report 124, Digital Equipment Corporation, 1994.

[7] G. Manzini, "An analysis of the Burrows–Wheeler transform," *Journal of the ACM* **48**(3), pp. 407–430, 2001.

[8] C. D. Manning and H. Schtze, *Foundations of Statistical Natural Language Processing*, The MIT Press, June 1999.

[9] J. Cleary and I. Witten, "Data compression using adaptive coding and partial string matching," *IEEE Transactions on Communications* **32**, pp. 396–402, Apr 1984.

[10] I. H. Witten, A. Moffat, and T. C. Bell, *Managing Gigabytes: Compressing and Indexing Documents and Images*, Morgan Kaufmann Publishers, 1999.

[11] D. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the I.R.E.* **40**(9), pp. 1090–1101, 1952.

[12] S. Davies and A. Moore, "Bayesian networks for lossless dataset compression," in *Proceedings of the Fifth International Conference on Knowledge Discovery in Databases*, pp. 387–391, AAAI Press, 1999.

[13] D. Mackay, *Information Theory, Inference & Learning Algorithms*, Cambridge University Press, June 2002.

[14] J. Adiego and P. de la Fuente, "On the use of words as source alphabet symbols in ppm," in *DCC '06: Proceedings of the Data Compression Conference*, p. 435, IEEE Computer Society, (Washington, DC, USA), 2006.

[15] F. Peng and D. Schuurmans, *Combining Naive Bayes and n-Gram Language Models for Text Classification*, vol. 2633, January 2003.

[16] F. Claude, A. Fariña, M. Martínez-Prieto, and G. Navarro, "Compressed *q*-gram indexing for highly repetitive biological sequences," in *Proc. 10th IEEE Conference on Bioinformatics and Bioengineering (BIBE)*, 2010. To appear.

[17] M. Cao, T. Dix, L. Allison, and C. Mears, "A simple statistical algorithm for biological sequence compression," in *Proc. DCC*, pp. 43–52, 2007.

[18] G. Manzini and M. Rastrero, "A simple and fast DNA compression algorithm," *Soft. Pract. Exper.* **34**, pp. 1397–1411, 2004.

[19] S. Kuruppu, B. Beresford-Smith, T. Conway, and J. Zobel, "Repetition-based compression of large DNA datasets," in *Proc. 13th International Conference on Computational Molecular Biology (RECOMB)*, 2009. Poster.

[20] C. E. Shannon, *A Mathematical Theory of Communication*, CSLI Publications, 1948.

[21] C. E. Shannon, "Prediction and entropy of printed English," *The Bell System Technical Journal* **30**, pp. 50–64, 1951.

[22] T. M. Cover and R. C. King, "A convergent gambling estimate of the entropy of English," *IEEE Transactions on Information Theory* **24**, pp. 413–421, July 1978.

[23] W. J. Teahan and J. G. Cleary, "The entropy of english using ppm-based models," in *DCC '96: Proceedings of the Conference on Data Compression*, p. 53, IEEE Computer Society, (Washington, DC, USA), 1996.