Bayesian Networks for Lossless Dataset Compression

Scott Davies
Carnegie Mellon University
scottd@cs.cmu.edu

Andrew Moore
Carnegie Mellon University
awm@cs.cmu.edu

Abstract

The recent explosion in research on probabilistic data mining algorithms such as Bayesian networks has been focussed primarily on their use in diagnostics, prediction and efficient inference. In this paper, we examine the use of Bayesian networks for a different purpose: lossless compression of large datasets. We present algorithms for automatically learning Bayesian networks and new structures called "Huffman networks" that model statistical relationships in the datasets, and algorithms for using these models to then compress the datasets. These algorithms often achieve significantly better compression ratios than achieved with common dictionary-based algorithms such those used by programs like ZIP.

1 Introduction

It has long been understood that even when confronted with a ten-gigabyte file containing data to be statistically analyzed, the actual information-theoretic amount of information in the file might be much less, perhaps merely a few hundred megabytes. This insight is currently most commonly used by data analysts to take high-dimensional real-valued datasets and reduce their dimensionality using principal components analysis, with little loss of meaningful information. This can turn an apparently intractably large data mining problem into an easy problem. PCA is applicable to real-valued data, and is usually a *lossy* form of compression: some information is lost in the transformation.

This paper is about dealing with large datasets with categorical (i.e. symbolic) values, and about using Bayesian network learning to discover interrelationships that allow very aggressive compression of the data. Furthermore, this compression can be completely lossless. The paper begins with an abridged overview of the relevant parts of compression and Bayesian network technology. We then discuss how to learn Bayesian networks that are useful for compression and how we use them with arithmetic coding and Huffman coding.

2 Background: Compression

Dictionary Techniques

Perhaps the most commonly used class of compression algorithms is the set of "dictionary techniques" used in general-purpose compression programs such as `gzip`. Dictionary-based algorithms maintain dictionaries containing sequences of source symbols. Whenever the source contains a symbol sequence that appears in the dictionary, that sequence's position in the dictionary is encoded rather than the individual symbols themselves. An example of a dictionary-based algorithm is the LZ77 algorithm [18] (employed by `gzip`), which uses a sliding window of previously encoded symbols as its dictionary. These algorithms can be shown to achieve asympotically optimal compression rates [17]; however, they may require the use of unmanageably large dictionaries in order to do so.

Huffman Coding

Given a small discrete set of source symbols and their associated probabilities, a simple greedy algorithm developed by David Huffman [7] can be used to find an optimal code with which to encode these source symbols on an individual basis. However, if the probability for one particular source symbol is very high (theoretically only needing a fraction of a bit), Huffman coding can be inefficient, as the code requires at least one bit for each source symbol encoded.

Arithmetic Coding

Arithmetic coding (developed by Rissanen [13] and Pasco [11]; see Witten, Neal, and Cleary [16] for a tutorial) allows sequences of symbols to be encoded nearly optimally (in the limit as k increases) without requiring the enumeration of all possible source code sequences of length k. Arithmetic coding effectively maps an entire sequence of source symbols to a real number between 0 and 1. The arithmetic encoder begins with a range $R = [0, 1)$. As each symbol in the source sequence is encoded, the current range R is subdivided into a partitions, where a is the number of possible values the symbol could have taken on; the size of each of these partitions is proportional to the probability of symbol taking on the corresponding value. The current range is then restricted to the partition corresponding to the source symbol being encoded. Once all symbols have been processed in this manner, the encoder outputs the binary representation of a number within the final subrange, to a sufficient precision to disambiguate the number from all numbers outside of this subrange.

Arithmetic coding achieves asymptotically optimal compression performance as the number of symbols in the encoded block tends to infinity, assuming that the model probability distributions used for encoding perfectly reflect the probability distributions inherent in the data. As described above, arithmetic coding appears to require the use of arbitrary-precision arithmetic operations in order to manipulate the current range R; However, it is possible to use limited-precision integer arithmetic to approximate "perfect" arithmetic coding with only a small loss in compression performance.

Both Huffman encoding and arithmetic coding require probabilistic models of the data they encode. We now discuss a class of probabilistic models particularly well-suited for modelling probability distributions over categorical datasets.

3 Background: Bayesian networks

Suppose we have a dataset D in which each record $I_1, \ldots, I_k$ consists of a set of values assigned to a set of variables $X_1, \ldots, X_n$. Now suppose we wish to model D with a probability distribution $P(X_1, X_2, \ldots, X_n)$ that we may use to calculate the probability that a randomly selected record I from D will have any specific value v_1 assigned to X_1, any specific value v_2 assigned to X_2, and so forth. Naturally, the most accurate "model" of D would be an enormous lookup table specifying how many records in D have any given set of assignments of values to X_1 through X_N; however, such a model would typically be useless for compression, since it would usually require as much space as D itself. What kinds of probabilistic models might be useful for compression?

Bayesian networks [12], also commonly known as belief networks, are a popular class of probabilistic models that work well in conjunction with compression, although they are primarily used for data analysis and decision-making. A Bayesian network consists of a directed acyclic graph (or "DAG") in which each vertex corresponds to a variable, plus a probability distribution $P(X_i|\Pi_{X_i})$ for each variable X_i, where Π_{X_i} is the set of $X_i^{\prime}s$ parents in the DAG. Given such a Bayesian network, the joint probability distribution over all variables $X_1, \ldots, X_n$ is then calculated as

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P(X_i \mid \Pi_{X_i}).$$

3.1 Learning Bayesian networks

Given a dataset D, we would like to automatically learn a Bayesian network B that accurately models the probability distributions in D with a small number of network parameters (i.e., entries in the probability tables associated with the variables). If there are no missing values or hidden variables in D — that is, if the data is "complete" — then if we are given B's structure, it is trivial to fill in B's probability tables optimally: namely, we simply use the empirical distributions appearing in D. However, even with complete data, the problem of finding the best network structure is NP-hard [2]. Learning a Bayesian network is thus typically done by using a search procedure to find a network B that maximizes (or at least hopefully comes close to maximizing) a scoring function $C(B, D)$. A popular scoring function is the *Bayesian Information Criterion* (BIC) [15],

$$C(B, D) = \log P(D \mid B) - |B| * 0.5 \log k$$

where $|B|$ is the number of parameters (probabilities) stored in the net and k is the number of records in the dataset.

4 Using Bayesian networks with arithmetic coding

Maximizing BIC corresponds directly to minimizing the number of bits required to store both (1) the parameters of the network B to a reasonable level of precision and (2) an efficient encoding (such as arithmetic encoding) of D using the probability distribution entailed by B. Thus, the BIC is naturally suited for finding Bayesian networks that are good for compression. This "minimum description length" (or MDL) approach has also been used for learning Bayesian networks in cases where compression is not necessarily the primary objective [8].

Bayesian networks are straightforward to use with arithmetic coding. To encode a record I of the dataset with a Bayesian network B, one treats each of the values in I as an individual "source symbol". These values are passed to the arithmetic encoder in an order consistent with a topological sort of B's vertices. This way, the decoder will have already decoded the values of any given variable X_i's parent variables by the time it needs to decode the value of X_i, and can use the appropriate entry in X_i's probability table to determine the probability distribution of values for X_i.

For the experimental results in this section, we use two algorithms for learning Bayesian networks. The first algorithm uses a form of stochastic hillclimbing over possible network structures using the Bayesian Information Criterion as its scoring function. AD-Trees [10] are used to speed up this search by decreasing the amount of time necessary to calculate the dataset statistics required for the search.

The second algorithm takes two sweeps through the dataset. In the first sweep, the mutual information between all pairs of variables is computed. Using this information, as well as BIC-like penalty terms for the number of parameters required, the algorithm uses a greedy heuristic algorithm to add arcs to the empty Bayesian network in order to arrive at a network in which each node has at most c parents. A second sweep is then made over the dataset to fill in the probability tables of the resulting network. See [4] for further details. This algorithm is somewhat similar to an algorithm previously used by Sahami for classification [14]. In the special case where c is 1, this algorithm reduces to a penalized version of Chow and Liu's dependency-tree algorithm [3], and is provably optimal. While the network chosen by this greedy algorithm won't generally be as accurate as one found via a more thorough search, this algorithm has the advantage of being more computationally feasible on datasets with many records or attributes.

4.1 Data reordering and Dynamic Bayesian networks

In applications in which we will only wish to scan through the dataset sequentially, we can take advantage of potential correlations between the ith record and $i+1$th record to obtain further compression. We use *Dynamic Bayesian Networks* [5] to seek out and learn such correlations and exploit them in a tighter encoding. These networks are learned with a greedy algorithm similar to the greedy Bayesian network-learning algorithm described in the previous section. Even more aggressive compression can be obtained when we are performing a data mining task that is indifferent to the order in which records are presented. In that case we can deliberately pre-sort the records to induce inter-record correlations where none existed originally and save even more space. Again, see [4] for further details.

4.2 Experimental Results

In this section, we examine the effectiveness of learning Bayesian networks in order to perform compression with arithmetic coding on real datasets. In conjunction with the Bayesian network learning algorithms discussed above, we used a limited-precision arithmetic coding library written by Carpinelli *et al.* [1] based on a paper by Moffat *et al.* [9].

We compare the compression performance of arithmetic coding with Bayesian networks (using the best of the two algorithms described above) and Dynamic Bayesian networks with the performance of `gzip` and `bzip2`, two popular compression utilities, on four datasets. Each dataset is compressed both in its natural ordering and in an order in which the items have been lexigraphically sorted. (See [4] for details.)

	Census	Banking	Astro1	Astro2
# dataset items	142500	6370	900000	3.08 M
# variables	12	142	27	49
variable arity	2-12	2-100	2-16	2-16
Uncomp. binary	536 KB	416 KB	11.8 MB	53.1 MB
gzip	294 KB	345 KB	6.9 MB	35.6 MB
bzip2	220 KB	273 KB	5.6 MB	27.9 MB
Bayes Net	169 KB	166 KB	4.2 MB	23.9 MB
Dyn. Bayes Net	171 KB	166 KB	2.6 MB	16.1 MB
Sort + gzip	36 KB	343 KB	6.5 MB	34.0 MB
Sort + bzip2	58 KB	272 KB	5.5 MB	27.7 MB
Sort + Dyn. BN	19 KB	163 KB	2.5 MB	17.1 MB

Depending on which dataset is being compressed and whether this dataset has been sorted, compression using Dynamic Bayesian networks in conjunction with arithmetic encoding was able to produce files that were 40-60% smaller than produced by `gzip`, and 20-60% smaller than produced by `bzip2`. Sorting the datasets sometimes increased compression performance — dramatically so in the case of the Census dataset.

5 Huffman networks

The arithmetic coding-based algorithms described above provide excellent compression ratios. However, arithmetic coding is somewhat computationally expensive; also, random access is impossible within a sequence of bits encoded with a single application of arithmetic coding, since there is no well-defined bit position where the encoding of one variable ends and another begins.

In contrast, Huffman coding uses relatively inexpensive lookup operations to perform encoding and decoding, and each coded value naturally has a well-defined start and end position in the resulting bitstream. This makes Huffman-based coding attractive for applications requiring faster decompression and/or random access. However, as mentioned previously, Huffman-based decoding provides poor compression performance when applied to probability distributions in which some values are very probable. It is possible to group variables together to overcome this problem, but then the tables required for encoding and decoding can become prohibitively large if too many variables are placed in one group. Additionally, if one variable is highly correlated with many other variables, it may help to have the value of that variable change the coding schemes associated with several variable groups, *without* that variable's value actually being coded in the compressed representations of all of the groups it influences.

We address these issues by using a hybrid Bayesian network — referred to hereafter as a *Huffman network* for convenience — in which each node actually models a *group* of variables in the dataset rather than an individual variable. Each group of variables is Huffman coded as a single unit. For example, if a group contains three binary variables, then that group is Huffman coded as if it were a single variable taking on eight possible values; each of these eight values is assigned a probability equal to the joint probability of the corresponding combination of values for the original three binary variables.

In order to take into consideration dependencies between variables residing in different groups, we allow the probability distribution over the possible values for each group to be conditioned on the values of other variables. For example, in Figure 1A, six variables have been placed into three groups. The joint probability distribution of all the variables in Group 3 (namely, variables x_2 and x_6) is conditioned on the values of variables x_3, x_4, and x_5. This conditioning is represented in the graph by arcs from x_3, x_4, and x_5 to Group 3. Assuming all the variables are binary, this means that Group 3 requires eight Huffman trees — one for each possible combination of values to x_3, x_4 and x_5. Each of these trees then has four leaves — one for each possible combination of x_2 and x_6. Note, however, that Group 3 is *not* conditioned on the value of x_1, despite the fact that x_1 is in the same group as x_4 and x_5. This added flexibility can help in certain situtations — for example, if x_2 and x_6 are independent of x_1 given x_4 and x_5, then conditioning Group 3 on the value of x_1 would double the number of Huffmann trees required by Group 3 without increasing Group 3's coding efficiency.

The Huffman network can be thought of as a Bayesian network over the original variables in which all variables in the same group are completely connected (e.g., Figure 1B). This representation tells us what dependencies between variables are being modelled by the coding scheme associated with the Huffman network. At the same time, the Huffman network can be thought of as a Bayesian network over the groups themselves (e.g., Figure 1C), where an arc exists from group G to group G' if and only if an arc exists from at least one variable in G to the group G' in the Huffman network. This view summarizes how the coding groups in the Huffman network are connected, thus telling us which groups of variables need to be decoded before other groups can be decoded.

We use a given Huffman network to perform compression as follows. First, we take one pass through the dataset to compute contingency tables for each of the groups in the network. The contingency table for a given group with a set of variables V and set of conditioning variables P counts how many times each possible combination of values for $V \bigcup P$ occurs in the dataset. These contingency tables are represented sparsely so that combinations that never actually occur in the dataset are never explicitly represented. Once these these contingency tables are computed, they are transformed into Huffman trees (which can subsequently be transformed into tables for encoding purposes). When compressing a file, we encode the Huffman trees at the beginning of the file (we omit the details), and then encode all of the records. Each record is encoded group by group in an order consistent with a topological sort of the groups in the network. Decompression is performed in an analogous manner.

5.1 Learning Huffman networks

The problem of automatically finding effective Huffman networks to use for compression is very similar to the problem of finding maximum-BIC Bayesian networks, and is almost certainly at least as difficult. Therefore, as in the case of learning Bayesian networks, we must rely on heuristic search techniques. We have not yet extensively explored possible search algorithms for finding good Huffman networks; however, we have implemented a relatively simple multiple-start stochastic hillclimbing algorithm. At each step during a hillclimbing run, the search algorithm considers adding an arc from a randomly selected variable to a randomly selected group (or removing the arc if one already exists), or moving a variable from its current group to a randomly selected group. If the change under consideration would create a cycle in the Huffman network, then it is immediately rejected and another change is randomly considered. Otherwise, the algorithm evaluates the resulting network and compares its estimated compression performance to the estimated compression performance of the current working network. A good network minimizes the total number of bits required to: (1) encode the network itself, and (2) encode the data with the network. Both of these terms can be estimated accurately from the Huffman trees associated with the group nodes if we have computed them.

5.2 Experimental Results

We use multiple-restart hillclimbing over Huffman networks in order to find good coding networks for the three datasets previously examined. Once the algorithm settled on a "good" network, we measured the network's performance both in terms of compressed file size and in terms of how fast they were able to perform decompression on encoded representations of the data in memory. Speed is measured in terms of the number of bits of uncompressed data that can be decoded per second on a 450 MHz Pentium II.

	Arithmetic coding	Huffman network coding
Census	169 KB, 0.54 MB/sec	171 KB, 1.9 MB/sec
Banking	166 KB, 0.49 MB/sec	179 KB, 1.1 MB/sec
Astro1	4.2 MB, 0.50 MB/sec	4.3 MB, 2.2 MB/sec
Astro2	23.9 MB, 0.31 MB/sec	24.1 MB, 1.3 MB/sec

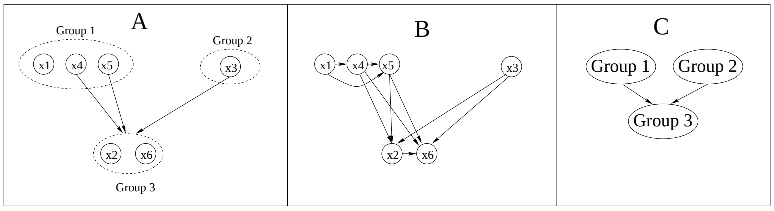

Figure 1: An example Huffman network (A), along with its corresponding variable-based (B) and group-based (C) Bayesian networks

As one might expect, the Huffman-based coding performed slightly (1 to 8%) worse than arithmetic coding in terms of file size, since arithmetic coding does not have to output integral numbers of bits for individual variables or groups of variables as the Huffman-based coding does. However, the Huffman-based decoding was two to four times faster than arithmetic decoding, and still achieved significantly smaller files than those achieved by the dictionary-based approaches (see section 4). While this is still much slower than the decoding speed of `gzip` and `bzip2`, there are further optimizations to be performed that will be examined in [4].

6 Concluding remarks

Related Work

Automatically-learned Bayesian networks have been used previously in conjunction with arithmetic encoding in recent research by Brendan Frey [6]. Rather than learning the structure of the Bayesian network, Frey uses a fixed network structure in which each node has many parents; the probability of each node given its parents is paramaterized using logistic regression. This approach has disadvantages compared to the approaches we describe here, although it has some advantages as well. In particular, Frey's approach may be more difficult to apply to nonbinary datasets. Additionally, Frey's use of hidden variables necessitates the use of a complex "bits-back" coding scheme in order to achieve decent compression rates, and this scheme may prevent fine-grained random access to the data. Comparisons with Frey's approach will be performed in future research.

Conclusion

This paper has proposed that Bayesian networks, in addition to their other virtues, may be an important tool for dataset compression. Experimental results show that excellent compression ratios are obtainable with arithmetic coding in conjunction with Bayesian networks. The Huffman networks introduced here provide nearly the same compression ratios, but with significantly less computational time.

References

[1] J. Carpinelli, A. Moffat, R. Neal, W. Salamonsen, L. Stuiver, and I. Witten. *Word, Character, and Bit Based Compression Using Arithmetic Coding*. Available for download at ftp://munnari.oz.au/pub/arith_coder/, 1995.

[2] D. Chickering. Learning Bayesian networks is NP-complete. In *Learning from Data*, pages 121–130. Springer-Verlag, 1996.

[3] C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, IT-14:462–467, 1968.

[4] S. Davies and A. Moore. Bayesian Networks for Lossless Dataset Compression. Technical Report in Progress, CMU School of Computer Science, 1999.

[5] T. Dean and K. Kanazawa. Probabilistic temporal reasoning. In *AAAI-88 Proceedings*, pages 524–528, 1988.

[6] B. J. Frey. *Graphical Models for Machine Learning and Digital Communication*. MIT Press, 1998.

[7] D. A. Huffman. A Method for the Construction of Minimum Redundancy Codes. In *Proceedings of the IRE*, volume 40, pages 1098–1101, 1951.

[8] W. Lam and F. Bacchus. Learning Bayesian belief networks: an approach based on the MDL principle. *Computational Intelligence*, 10:269–293, 1994.

[9] A. Moffat, R. Neal, and I. H. Witten. Arithmetic Coding Revisited. In *Proceedings of the IEEE Data Compression Conference*, March 1995.

[10] A. W. Moore and M. S. Lee. Cached Sufficient Statistics for Efficient Machine Learning with Large Datasets. *Journal of Artificial Intelligence Research*, 8, 1998.

[11] R. Pasco. Source Coding Algorithms for Fast Data Compression. Ph.D. Thesis, Stanford University, 1976.

[12] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan-Kaufmann, 1988.

[13] J. J. Rissanen. Generalized Kraft inequality and arithmetic coding. *IBM Journal of Research and Development*, 20:198–203, May 1976.

[14] M. Sahami. Learning Limited Dependence Bayesian Classifiers. In *KDD-96: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 335–338. AAAI Press, 1996.

[15] G. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 6:461–464, 1978.

[16] I. H. Witten, R. M. Neal, and J. G. Cleary. Arithmetic coding for data compression. *Communications of the Association for Computing Machinery*, 30:520–540, June 1987.

[17] J. Ziv. Coding theorems for individual sequences. *IEEE Transactions on Information Theory*, 24:389–394, 1978.

[18] J. Ziv and A. Lempel. A Universal Algorithm for Data Compression. *IEEE Transactions on Information Theory*, 23(3):337–343, May 1977.