Available online www.jsaer.com

***Journal of Scientific and Engineering Research*, 2018, 5(4):134-139**

Review Article

ISSN: 2394-2630
CODEN(USA): JSERBR

Noiseless Coding Techniques- A Review

Akram Abdul Maujood Dawood

Computer Engineering Department, Mosul University /Iraq

Abstract In this paper, the techniques of initial compression, which are free of noise, are reviewed, the source code, and the effects of the length of the code, its efficiency and the methods of re-pressing are discussed. Also, the so-called synchronous code and the effect of the property of the prefix are reviewed along with the length of the word. The random compression and its methods are also studied with examples.

Keywords lossless compression, Huffman coding, information theory, Entropy, Shannon coding, image compression

1. Introduction

In this paper, we are interested in presenting noiseless (lossless) coding. They are efficient codes that can be used to send data over a communication line that does not introduce noise (cause loss). Such codes can also be used to compress data for storage purposes. Here 'efficiency' refers only to the code itself, not to the amount of time or space required to do the encoding and decoding processes.

Source symbols → [Encoding] **(0|1) →** [Decoding] **→ Source symbols**

The average code-word length (message length), for events v from S, is

$$\sum_{x \text{ in } S} \{P(x) \,.\, |\text{code}(x)|\} \quad (1)$$

This cannot be less than the entropy, H. This lower bound is achieved if

$$|\text{code}(x)| = -\log_2(P(x)) \quad (2)$$

Different source coding techniques are desired in this paper. Some examples are also given. Shannon's source coding is contained in section 2 of this paper. Section 3 presents the dependence of such codes on entropy. The section 4 contains the different methods of entropy coding with some examples. Section 5 contains the run _length coding with an example. Finally, section 6 concludes this paper.

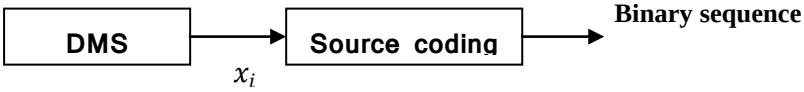
2. Source Coding

In information theory, Shannon's source coding theorem (or noiseless coding theorem) establishes the limits to possible data compression, and the operational meaning of the Shannon entropy. The source coding theorem shows that (in the limit, as the length of a stream of independent and identically-distributed random variable data tends to infinity) it is impossible to compress the data such that the code rate (average number of bits per symbol) is less than the Shannon entropy of the source, without it being virtually certain that information will be lost. However it is possible to get the code rate arbitrarily close to the Shannon entropy, with negligible probability of loss [1].

The source coding theorem for symbol codes places an upper and a lower bound on the minimal possible expected length of codeword as a function of the entropy of the input word (which is viewed as a random variable) and of the size of the target alphabet. If s a mapping from (a sequence of) symbols from an information

source to a sequence of alphabet symbols (usually bits) such that the source symbols can be exactly recovered from the binary bits (lossless source coding) or recovered within some distortion (lossy source coding). This is the concept behind data compression [1].

Source coding is a conversion of the o/p of a discrete memoryless source (DMS) into a sequence of binary symbols (binary code word).



$X = (x_1 \ldots . x_m)$

An objective of source coding is to minimize the average bit rate required for representing the source by reducing the redundancy of the information source.

A. Code Length and Code efficiency

The length of code word is the number of binary digits in the code word. The average code word length L, per source symbol is given by.

$$L = \sum_{i=1}^{m} p(x_i) n_i \quad (3)$$

The code efficiency ζ is defined as $\zeta = \frac{L_{min}}{L}$ where L_{min} is the min possible of L.

At $\zeta = 1$, the code is said to be efficient and The code redundancy r is defined as: $r = 1 - \zeta$

Source Coding Theorem: (Shannon's theorem)

$\zeta = \frac{H}{L}$, With $L_{min} = H$, $L \geq H$

B. Decoding

"A code is *uniquely decodable* if every finite string of bits represents at most one *sequence* of source text. i.e. There is no possibility of two different source texts giving the same coded string".

C. Synchronizable Codes

A code is *comma-free* if it is impossible to get one out-of-frame code-word in the middle of an encoded string. In the early days of molecular biology it was briefly suspected that the genetic code (DNA, amino acids) was comma-free, but this turned out *not* to be the case. A code has a stagger limit of 'k' if the synchronization of all windows of length k+1 or greater is apparent without any more context B. Neveln [2] suggests synchronizability and not comma freeness as the requirement of an early genetic code. The genetic code *is* very nearly a Gray code, i.e. an error in the (en | de) coding (DNA) will probably either be *silent* (code for the same amino acid), or code for a chemically similar amino acid. The code seems to have evolved to minimize the effect of mutations and misreading [3].

D. Prefix Property

Another desirable property for a code to have is the *prefix property*; No code word is a prefix of any other code word. A code having the prefix property is uniquely decodable, but not necessarily .The prefix property allows us to identify the end of the first (and second etc.) code word in a data stream immediately, and is required to be able to decode a stream of data unambiguously *without look-ahead*. While the prefix property seems to be quite strict, some fundamental results show that such codes are all that we need to consider for the purposes of inference [4],[5].

Example

Recalling the biased four sided dice, P(a)=1/2, P(c)=1/4, P(g)=P(t)=1/8, the following is an efficient code for data generated by the dice:

a:'0'	**1?**		
	c:'10'	**11?**	
		g:'110'	**t:'111'**
½	¼	**1/8**	**1/8**

This is a particularly simple example of a Huffman code. You will notice that the probabilities 1/2, 1/4, 1/8, 1/8, were deliberately chosen so that their (- $\log_2$ s) correspond exactly to convenient code word lengths; things do not always work out so easily. When they do, code words can be allocated by using a binary tree.

3. Codes and Entropy

Suppose we have events with probabilities p1, p2 ..., p_N and there is a code for these events with code-word length l_1 ,l_2 ,….l_N where $\sum i = 1 \ldots . N \{2_i^{-1}\}$.The average code-word length will be $\sum i = 1 \ldots . N \quad P_i l_i$. We saw under entropy, that this expression is minimized when l_i = -log2(pi). Such values will not be integers in general, but if we had a code with code-word lengths of ceiling (-log2(pi)), then it would have an average code-word length less than the entropy plus one, < H+1.

3.1. Kraft Inequality

There is a prefix code with code-words of lengths l_1, l_2, ...,l_N iff $\sum_{i=l..N}\{2^{-l}{}_i\} \leq 1$.There must be a maximum code-word length, say max. Consider the full binary tree of depth max with edges labeled (0|1), e.g. below if max=3 (root at the left):

	1	2	3
root	1	11	111
			110
		10	101
			100
	0	01	011
			010
		00	001
			000

Every code-word must appear somewhere in the tree and because of the prefix property no other code-word can be a descendant or an ancestor of it in the tree. There are 2^k bit-strings of length k. A bit-string of length max, has probability 2^{-max}. The probability of a shorter bit-string is the sum of the probabilities of the max-bit strings below it. The sum of the probabilities of the code-words cannot exceed 1.0

3.2. Theorem

There exists a prefix code with code-words of length l1, l2, ...,lN, iff there is a uniquely decodable code with these lengths. So it is reasonable to insist on the use of prefix codes because if there is any uniquely decodable code then there is a prefix code [6].

4. Run–Length Coding

Instead of encoding the consecutive symbols ,its more efficient to encode the run_ length and the value that these consecutive symbols commonly share, where run mean the repetition of a symbol and run_length means number of repeated symbols (L).Adopted in JPEG (Multi – level image coding), binary document coding, and many other coding standards. in RLC the sequence of image elements along a scan line (row) $x_1, x_2, \ldots \ldots, x_n$ is mapped into asequence of pairs $(g_1, l_1),(g_2, l_2),\ldots\ldots\ldots\ldots (g_k, l_k)$ where g_i denoted the gray level and l_i denotes the length of the i_{th} run.the RLC works by counting the number of adjacent pixels with the same gray – level values. This count is called the run-length L and the pair of gray level G. It is worth mentioning RLC technique is effective with image containing a small number of gray levels.

Example

Given the following 8*8, 4 – bit image, find the RLC of the image?

$$\begin{pmatrix} 5 & 5 & 5 & 5 & 10 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 & 0 & 0 \\ 5 & 5 & 5 & 4 & 4 & 4 & 0 & 0 \\ 5 & 5 & 5 & 10 & 10 & 9 & 9 & 10 \\ 10 & 10 & 5 & 5 & 12 & 12 & 0 & 0 \\ 10 & 10 & 10 & 10 & 12 & 12 & 12 & 12 \\ 0 & 0 & 0 & 10 & 10 & 10 & 10 & 10 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Solution:

First row: (5,4) , (10 ,4)
Second row: (0,3) , (10,3),(0,2)
Third row: (5,3) , (4,3),(0,2)
Fourth row: (5,3) , (10,2),(9,2),(10,1)
Fifth row: (10,2) , (5,2),(12,2),(0,2)
Sixth row: (10,4),(12,4)
Seventh row: (0,3),(10,5)
Eighth row (0,8)
These numbers are stored or transmitted in RLC compressed form as:
5,4,10,4,0,3,10,3,0,2,5,3,4,3,0,2,5,3,10,2,9,2,10,1,10,2,5,2,12,2,0,2,10,4,12,4,0,3,
10,5, 0,8
The RLC used for binary image is called basic RLC. there are many ways to implement the basic RLC. One method used is the horizontal RLC which count along the rows. In this method a convention for the first number in the row should be define as "0 " or "1".

5. Entropy Coding Methods and Examples

It is a design of variable length code such that its average code word length L approaches the entropy. There are three main types of this coding. They are listed here.

A. Huffman coding

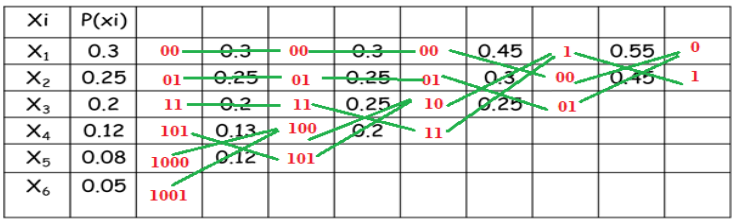
In computer science and information theory, a Huffman code is a particular type of optimal prefix code that is commonly used for lossless data compression. The process of finding and/or using such a code proceeds by means of Huffman coding, an algorithm developed by David A. Huffman while he was a Ph.D. student at MIT, and published in the 1952 paper "A Method for the Construction of Minimum-Redundancy Codes". The output from Huffman's algorithm can be viewed as a variable-length code table for encoding a source symbol (such as a character in a file). The algorithm derives this table from the estimated probability or frequency of occurrence (weight) for each possible value of the source symbol. As in other entropy encoding methods, more common symbols are generally represented using fewer bits than less common symbols. Huffman's method can be efficiently implemented, finding a code in linear time to the number of input weights if these weights are sorted. However, although optimal among methods encoding symbols separately, Huffman coding is not always optimal among all compression methods [7].

Huffman Coding Algorithm

1. List the source symbols in order of decreasing prob.
2. Combine the probabilities of the two symbols having the lowest probabilities, and recorder the resultant probabilities. The same procedure is repeated unit there are two ordered probabilities remaining.
3. Start encoding with the last reduction, which consist of exactly two ordered probabilities. Assign 0 as the first digit in the codeword for all the source symbols associated with the first probability, Assign 1 to the second probability.
4. Now go back and assign 0 and 1 to the second digit for the two probabilities that were combined in the previous reduction step, retaining all assignments made in step 3.
5. Keep regressing this way until the first column is reached.

Huffman Coding Example

For X= {x_1 , x_2 , x_3 , x_4, x_5} with $p(x_i)$=0.3,0.25,0.2 ,0.12,0.08 ,0.05, respectively. Find H, L, ζ, and r.



H =2.36 bit/sample, L =2.38 bit/sample, ζ=0.991, r =0.009.

B. Shannon Coding

In the field of data compression, Shannon coding, named after its creator, Claude Shannon, is a lossless (noiseless) data compression technique for constructing a prefix code based on a set of symbols and their probabilities (estimated or measured). It is suboptimal in the sense that it does not achieve the lowest possible expected code word length like Huffman coding.

Shannon-Fano Coding Algorithm

1. List the source symbols in order of decreasing prob.
2. Partition the set into 2 sets that are close to equal-probability as possible ; and assign 0 to the upper set and 1 to the Lower set.
3. Continue this process, each time partitioning the sets with as nearly equal probability as possible until further partitioning is not possible.

Shannon-Fano Coding Example

For X= {x_1 , x_2 , x_3 , x_4, x_5, x_6} with *p(xi)*=0.3,0.25,0.2 ,0.12,0.08 ,0.05. Find H,L ,r and ζ.

Xi	P(xi)					Code
X_1	0.3	0	0			00
X_2	0.25	0	1			01
X_3	0.2	1	0			10
X_4	0.12	1	1	0		110
X_5	0.08	1	1	1	0	1110
X_6	0.05	1	1	1	1	1111

H =2.36 bit/sample, L =2.38 bit/sample, ζ =0.991, r =0.009.

C- Arithmetic Coding

Arithmetic coding works by dividing the interval [0, 1), *i.e.*, {x | x≥0 & x <1}, into smaller and smaller intervals which represent the successive symbols of an input sequence. It can be thought of as an extension of prefix coding where the interval boundaries lie neatly between adjacent powers of 1/2. e.g., The (trivial) prefix code for false and true {F,T} where *P*(F)=*P*(T)=½. See the following example:

1/2 F	1/2 **T**			1
	1/4 F	1/4 **T**		2
		1/8 **F**	1/8 T	3
		1/16 F	1/16 **T**	4
0	1/2		1	

Note that $1/2 = 0.5_{10} = 0.1_2$, $1/4 = 0.25_{10} = 0.01_2$, $1/8 = 0.125_{10} = 0.001_2$, etc.

In arithmetic coding, probabilities of events need not be neat powers of 1/2. e.g., given a 2-symbol example where *P*(F)=1/8, *P*(T)=7/8 in position one and *P*(F)=*P*(T)=1/2 thereafter. (The probabilities are allowed to vary with position in the sequence, so long as encoder and decoder both know them or can calculate them.). See the following example:

1/8 F	7/8 **T**			P(F)=1/8
1/4 F	5/8 **T**			P(F)=1/2
	1/4 **F**	3/8 T		P(F)=1/2
	1/8 F	1/8 **T**		P(F)=1/2
0	1/2		1	

Note that the second-level partition is 1/4:5/8 not the ideal 7/16:7/16.

As shown above, arithmetic coding is very efficient and extensions of it can get arbitrarily close to the entropy of the data. In some sense it has "solved" the coding problem, making it clear that compression = data modeling + coding. The (predicted) probabilities of the events can come from anywhere provided that the encoder and

decoder agree on them; they can for example be fixed, or come from an order-k Markov model, or be estimated from statistics gathered from the data previously coded.

6. Conclusion

Different source coding methods have been described in this review. The concentration has been focused on the noiseless types of source coding. The effects of code length and its efficiency has been also reviewed. The impact of prefix property on the code length has been demonstrated. Many of the entropy coding methods have been presented also in this review and many examples have been illustrated.

References

[1]. D. Taubman, M. Marcellin, *JPEG2000 Image Compression Fundamentals Standards and Practice*, Springer science + business media, LLC 2012.
[2]. B. Neveln, *Comma-Free and Synchnronizable Codes*, J. Theor. Biol., 144, pp. 209-212, 1990.
[3]. R. Swanson, *A Unifying Concept for the Amino Acid Code*, Bull. Math. Biol., 46(2), pp.187-203, 1984.
[4]. K. Krippendorff, *Mathematical Theory of Communication*, S. W. Littlejohn & K. A. Foss (Eds.), Los Angeles, 2009.
[5]. A. J. Viterbi and J. K. Omura, *Principles of Digital Communication and Coding*, Dover publications, INC Mineola , New York, 2013.
[6]. *Welsh Codes and Cryptography*, OUP, 1988.
[7]. D.A. Huffman, *A Method for the Construction of Minimum-Redundancy Codes*, Proceedings of the I.R.E., pp 1098–1102, September 1952.