Data Compression - Report

lrfk99

Version 2021.01.03

1 Dictionary

Symbols are words, coded as 1 or 2 bytes, usually as a preprocessing step. Use bytes rather than characters. It would be possible to use only characters in the encoded file to represent either a number or letter or special character, but it would be much more efficient to use bytes to store information since these have the same size as a character (8 bits) but they can be much more useful and versatile for encoding a numerical value as well which would otherwise require 8 bits per digit. Since standard ASCII has 256 possible characters and so requires 8 bits to store each character, we can reduce this number in our compression if we never see some of those characters, e.g. non-english characters which clearly will not be used in an english .tex document. As such we could use just 7 bits to encode each character if the document only uses a maximum of 128 different characters. The number of bits used to encode each character should be stated at the start of the encoded file such that the decoder can correctly decode the file.

2 Prediction by Partial Matching (PPM)

Lectures. Order-n e.g. o0,o1,o2,... - symbols are bytes, modeled by frequency distribution in context of last n bytes order-n, modeled in longest context matched, but dropping to lower orders for byte counts of 0. Since the .tex file will be typeset in English, it is possible to predict the next character, given the previous character(s). As such, a statistical model of english text is appropriate to use. This statistical model can be 'trained' on lots of english text such as Alice in Wonderland, and other .tex documents such as the lecture notes for this module. When using PPM, there are three methods for assigning frequencies to the escape symbol. The most appropriate method for this scenario is Method C since it takes into account the fact that some contexts can be followed by virtually any other character by giving the escape symbol an appropriate count, whilst not reducing the count of other symbols. I considered using Dynamic Markov compression, which uses predictive arithmetic coding similar to prediction by partial matching (PPM), except that the input is predicted one bit at a time rather than one byte at a time Bits modeled by PPM

3 Context mixing

See Further. bits modeled by combining predictions of independent models The cutting edge in lossless compression in terms of compression ratio is achieved by combining the next-symbol predictions of two or more statistical models, producing more accurate predictions than any model individually.

4 Lempel-Ziv (LZ)

Lectures. Symbols are strings. LZ77 - repeated strings are coded by offset and length of previous occurence LZ Welch - repeats are coded as indexes into dynamically built dictionary Reduced Offset LZ - LZW with multiple small dictionaries selected by context LZ predictive - ROLZ with dictionary size of 1

5 Burrows-Wheeler Transform (BWT)

The Burrows-Wheeler Transform will rearrange a character string into runs of similar characters. This makes it very easy to use move-to-front transform and then run-length encoding. Since in this scenario we are able to scan the entire file before encoding it, The BWT improves the efficiency of the compression algorithm without storing any extra data other than the position of the first original character. we can use the Burrows-Wheeler transform (BWT) converts a list of symbols into a much more structured list. Order-n e.g. o0,o1,o2,... - symbols are bytes, modeled by frequency distribution in context of last n bytes Symbol Ranking - Order-n, modeled by time since last seen. Burrows-Wheeler Transform - Bytes are sorted by context, then modeled by order-0 Symbol Ranking

6 Move-to-front transform

The move-to-front (MTF) transform allows for encoding a stream of bytes, and performs particularly well on sorted byte streams.

7 Run-length encoding

Run-length encoding (RLE) is where the actual compression will happen, after the data has been transformed twice by BWT and MTF.

[1]

References

[1] Matt Mahoney. Large text compression benchmark, 2020.