Data Compression

lrfk99

Version 2020.12.23

1 Reduce size of characters

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 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.

3 Prediction by Partial Matching (PPM)

Train model of context probabilites on Codes and Cryptography module lecture notes.

4 Context mixing

Pass

5 Burrows-Wheeler transform (BWT)

Pass

6 Lempel-Ziv Welch

Pass

7 Statistical compression I: Huffman coding

7.1 Prefix codes

Memoryless sources We have some data that we wish to encode. It could be anything: Spoken English, Data from a digital camera sensor, DNA string, etc.

We model our data as coming from a memoryless source X. We imagine that symbols are emitted at random according to the probability distribution of X. In other words, we view our data as a random string $X_1, X_2, \ldots$ over some alphabet $\mathcal{X}$. Our memoryless assumption is that those form a sequence of independent identically distributed (i.i.d.) random variables: $X_i \sim X$ for all i.

More concretely, for any $x \in \mathcal{X}$ and any i, the probability

$$\mathbb{P}(X_i = x)$$

is independent of i, and of all previous or future emitted symbols.

Note that this is not always a valid assumption. We will look into source modelling into more detail in the next lectures.

The coding problem We have a source emitting symbols in $\mathcal{X} = \{x_1, \ldots, x_n\}$ with respective probabilities $\{p_1, \ldots, p_n\}$.

Question: If $\mathcal{D}$ is an alphabet of D code symbols, how can we encode the source symbols using code words (finite strings of code symbols) as economically as possible?

Formally: a source code is a map $C : \mathcal{X} \to \mathcal{D}^*$ where $\mathcal{D}^*$ is the set of all finite strings of symbols in $\mathcal{D}$.

The words $C(x)$ are called the codewords, and the integers $|C(x)|$ (the length of $C(x)$) are the word lengths.

We can extend the code to messages as follows. A message is any finite string of source symbols $m = m_1 \ldots m_k \in \mathcal{X}^*$ and its encoding is the obvious concatenation

$$C(m) = C(m_1)C(m_2)\ldots C(m_k).$$

Prefix codes A code C is uniquely decodable (a.k.a. uniquely decipherable) if every finite string in $\mathcal{D}^*$ is the image of at most one message.

A prefix of a word $w = w_1 \ldots w_k \in \mathcal{D}^*$ is any word of the form $w_1 \ldots w_l$ for some $0 \leq l \leq k$ (for $l = 0$, we obtain the empty word). A code is prefix (a.k.a. instantaneous or prefix-free) if there are no two distinct source symbols $x, y \in \mathcal{X}$ such that $C(x)$ is a prefix of $C(y)$.

Theorem 7.1. *A prefix code is uniquely decodable.*

Proof. Let C be a prefix code, and let $w = C(m)$ for some message $m = m_1 \ldots m_k \in \mathcal{X}^*$. We give a decoding algorithm which, given w, determines m. Let $w = w_1 \ldots w_l$.

Let i be the smallest integer such that $w_1 \ldots w_i$ is a codeword, say $w_1 \ldots w_i = C(x)$. Then the $m_1 = x$. Indeed, if $m_1 = y \neq x$, then $C(x)$ is a prefix of $C(y)$, which is a contradiction. Then repeat this step, beginning with w_{i+1} and hence determining m_2, and so on until w is empty. □

Example 7.2. *Let* $\mathcal{X} = \{a, b, c, d, e\}$, $\mathcal{D} = \{0, 1\}$ *and*

$$\begin{aligned} C(a) &= 01 \\ C(b) &= 100 \\ C(c) &= 101 \\ C(d) &= 1101 \\ C(e) &= 1111. \end{aligned}$$

Suppose we need to decode the word $C(m) = w = 10010111011111100101$. *We proceed as follows. We read the word until we reach a codeword:*

$$\begin{aligned} w_1 &= 1 \\ w_1w_2 &= 10 \\ w_1w_2w_3 &= 100 = C(b). \end{aligned}$$

Therefore $m_1 = b$. *We continue until we reach a codeword:*

$$\begin{aligned} w_4 &= 1 \\ w_4w_5 &= 10 \\ w_4w_5w_6 &= 101 = C(c). \end{aligned}$$

Therefore $m_2 = c$. *And so on... Exercise* **??** *asks you to finish this simple example.*

7.2 Huffman codes

Compact codes Our main aim is to design codes where the typical length of messages is reduced dramatically. The basic idea is to assign short codewords to more frequent symbols and longer codewords to less frequent ones.

More formally, the average length (a.k.a. expected length) of the code is

$$L(C) = \mathbb{E}(|C(X)|) = \sum_{x \in \mathcal{X}} |C(x)| \mathbb{P}(X = x).$$

A code is compact (for a given source X) if it is uniquely decodable and it minimises the average length of codewords over all uniquely decodable codes.

Theorem 7.3. *A uniquely decodable code with prescribed word lengths exists if and only if a prefix code with the same word lengths exists.*

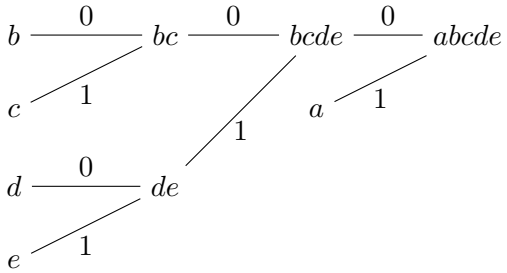
Corollary 7.4. *For any source X, there is a compact prefix code for X.*

Binary Huffman code The key is to construct a tree where the leaves correspond to the symbols in $\mathcal{X}$ and the paths from the root to the leaves give the codewords.

The tree is constructed iteratively. Suppose $\mathcal{X} = \{x_1, \ldots, x_n\}$ with $p_1 \geq p_2 \geq \cdots \geq p_{n-1} \geq p_n$. Then merge x_{n-1} and x_n into a new symbol, say $x_{n-1,n}$ with probability $p_{n-1} + p_n$, and let x_{n-1} and x_n be the children of $x_{n-1,n}$ on the tree. Label the edges from $x_{n-1,n}$ to its children as 0 and 1, respectively. Repeat for the new source $X^{(1)} = \{x_1, \ldots, x_{n-2}, x_{n-1,n}\}$ (making sure to order the symbols in non-decreasing probability). Repeat until the final source $X^{(n-1)}$ only has one symbol left with probability 1; that symbol is the root of the tree.

Once the tree is built, read off the labels on the path from the root to a leaf to get the corresponding codeword.

Example 7.5. *Let X with respective probabilities $a : 0.4, b : 0.2, c : 0.15, d : 0.15, e : 0.1$.*



The code is then

$$\begin{aligned} C(a) &= 1 \\ C(b) &= 000 \\ C(c) &= 001 \\ C(d) &= 010 \\ C(e) &= 011 \end{aligned}$$

The average length is then 2.2 bits per symbol.

Note that there is no need for general tie-breaking rules. Indeed, different merges may yield different codes, and maybe even different code lengths, but always the same expected length. Similarly, the assignment of 0 or 1 does not change the code lengths.

Huffman codes are compact Clearly, Huffman codes are prefix. The proof that they are compact is by induction on the number of symbols and omitted. It can be found in [2, Section 5.8].

Non-binary Huffman codes Huffman codes can be extended to non-binary alphabets: If we have an alphabet of D characters, we group the D least likely symbols at each stage of reducing the source. When expanding the code we append each of the D characters to one of the least likely symbols' codewords.

We must end up with exactly D symbols in the final source, so we may need to pad the original source up to $D + k(D-1)$ by adding symbols of probability 0.

7.3 See further

Codes and Automata The mathematical theory of uniquely decodable codes is reviewed in [1], where they are simply referred to as codes. The language generated by a prefix code can be recognised by very a simple deterministic finite automaton; in fact, the relation between codes and automata is very deep and explored throughout the book. Note that this book hardly talks about data compression!

Canonical Huffman codes As we shall see in Exercise **??**, there can be several different Huffman trees for the same source. However, there is always a so-called canonical Huffman tree (and hence code) with a special shape that can be easily computed; see [3, 3.2.2].

Adaptive Huffman coding Huffman coding is based on a source X with given probabilities. In general, the probability of an element is computed by its relative frequency in the message; for instance, if the message has 100 characters, 34 of them are "e", then the probability of "e" is 34%. Computing those probabilities then requires scanning the whole document before building the tree. Adaptive Huffman coding, on the other hand, builds the Huffman tree as the document is scanned, making small updates (if any) each time a new character is scanned.

References

[1] Jean Berstel, Dominique Perrin, and Christophe Reutenauer. *Codes and Automata*, volume 129 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, second edition, 2010.

[2] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley, second edition, 2006.

[3] Khalid Sayood. *Introduction to Data Compression*. Morgan Kaufmann, fourth edition, 2012.