

Data Compression problem: given a string S containing

n_1 occurrences of symbol s_1
 n_2 occurrences of symbol s_2
 $\dots$
 n_k occurrences of symbol s_k

k is the alphabet size

$n = n_1 + n_2 + \dots + n_k$ is the string length

we want to assign a codeword c_i to each s_i so that the total length is minimized; but the code must be **uniquely decodable**.

For example the code

a 0
b 1
c 00
d 01
e 10

is very compact but not uniquely decodable:

the sequence 00 could be the code for "aa" or for "c"

A code is prefix free if

no codeword is the prefix of another codeword.

During decoding as soon as we see a full codeword we can decode it, since it cannot be the prefix of another codeword. For this property such codes are called instantaneous.

Every instantaneous code is uniquely decodable, but not viceversa.

TABLE 5.1 Classes of Codes

X	Singular	Nonsingular, But Not Uniquely Decodable	Uniquely Decodable, But Not Instantaneous	Instantaneous
1	0	0	10	0
2	0	010	00	10
3	0	01	11	110
4	0	10	110	111

The code in the 3rd column is uniquely decodable but not instantaneous:

110000 -> 322

1100000-> 422

we need to count the 0s
to figure out if the
first symbol is 3 or 4

Kraft inequality: the codewords lengths $|c_i|$ of any prefix free or uniquely decodable code must satisfy

$$\sum_{i=1}^k 2^{-|c_i|} \leq 1$$

Example: Morse code

A ..	B -...	C -.-.	D -..	E .	F	G -.-.
H	I ..	J .-.-	K -.-	L -....	M --	N -.
O ---	P -....	Q -.-.-	R -..	S ...	T -	U ...
V	W -.-	X -.-.-	Y -.-.-	Z -....		

as a binary code it is not uniquely decodable: in practice it is a ternary code with a special symbol (pause) after each letter

Mathematical solution to the optimal coding problem

Given $n_1, \dots, n_k$, $n = \sum_{i=1}^k n_i$

find the optimal lengths $\ell_1, \dots, \ell_k$

minimizing the length of the encoded string: $\sum_{i=1}^k \ell_i n_i$

We want a prefix free (instantaneous) code so we must have

$$\sum_{i=1}^k 2^{-\ell_i} \leq 1$$

We get a minimization problem on the variables $\ell_1, \dots, \ell_k$

$$\min_{\ell_1, \dots, \ell_k} \sum_{i=1}^k \ell_i n_i, \quad \text{with the constraint} \quad \sum_{i=1}^k 2^{-\ell_i} \leq 1$$

If we could accept solutions with non integer lengths, we could compute the optimal lengths using Lagrange Multipliers:

$$F(\ell_1, \dots, \ell_k, \lambda) = \sum_i n_i \ell_i + \lambda \left(\sum_i 2^{-\ell_i} - 1 \right)$$

that give us the solution

$$\ell_i = -\log_2 \frac{n_i}{n}$$

We cannot design codes with non-integer lengths, but using integer lengths we cannot do better than the above lengths hence the total encoding will be at least

$$-\sum_i n_i \log_2 \frac{n_i}{n} = n \left(-\sum_i \frac{n_i}{n} \log_2 \frac{n_i}{n} \right)$$

The quantity $H = -\sum_i \frac{n_i}{n} \log_2 \frac{n_i}{n}$ is called the input **entropy**

and represents the optimal compression (in the above setting)

Entropy and Huffman Coding

Given $n_1, \dots, n_k$, $n = \sum_{i=1}^k n_i$

Huffman algorithm is a technique for building a binary prefix-free code with lengths $h_1, \dots, h_k$ such that

$$H \leq \left(\sum_{i=1}^k n_i h_i \right) / n \leq H + 1$$

Huffman algorithm provides actual codewords which provide an encoding which is optimal (among actual codewords) and at most n bits off the theoretical limit (valid for hypothetical non integer code lengths)

The above results generalize to the case of an infinite number of symbols for example when we want to assign a codeword to each positive integer.

For infinite codewords the generalized Kraft inequality states:

$$\sum_{i=1}^{\infty} 2^{-|c_i|} \leq 1$$

In addition, as in the finite case, if integer i appears with frequency (or probability) f_i then its codeword should have length $\ell_i = -\log_2(f_i)$

Codes for the positive integers:

Binary (not uniquely decodable!) $|B(i)| = 1 + \lfloor \log_2 i \rfloor$

1	2	3	9	33
1	10	11	1001	100001

Unary: $U(i) = 0^{i-1}1$ $|U(i)| = i$

1	2	3	9
1	01	001	000000001

Elias gamma: $\gamma(i) = U(|B(i)|)B^*(i)$ $|\gamma(i)| = 2\lfloor \log_2 i \rfloor + 1$

1	2	3	9	33
1	010	011	0001001	00000100001

In the integer coding schemes above the codewords are given, so we use the optimality results **backwards**.

For example the Elias Gamma encodes the integer i with $2\lfloor \log_2 i \rfloor + 1$ bits, the above analysis shows the code is optimal if symbol i appears with frequency

$$2^{-2\lfloor \log_2 i \rfloor - 1} \approx 2^{-\log_2 i^2 - 1} \approx \frac{1}{2i^2}$$

Elias delta code

$$\delta(i) = \gamma(|B(i)|)B^*(i) \quad |\delta(i)| \approx 2 \log_2 \log_2 i + \log_2 i + 1$$

4 -> 100 delta(4) = gamma(3)00 -> 01100

1	2	3	4	9
1	0100	0101	01100	00100001

Bad for small integers, competitive if you need to encode many large integers.

Optimal if symbol i appears with frequency

$$= 2^{-2 \log \log i + \log i + 1} \approx \frac{1}{2i(\log i)^2}$$