Lempel-Ziv Compression

EE5143: Information Theory

Feb 15, 2024

Introduction

Motivation

Problems with Huffman-coding

- We need to know a probability distribution P *a priori*
- Big block of text $\rightarrow$ we could count $\#$ of times a character occurs, giving us P
- Is *offline*
- Needs two passes through the text:
  - for finding P
  - for the encoding itself
- Does not work if we have a data-stream coming in

LZ Algorithms

- LZ algorithms are *online*
- *Universal Code:* Regardless of the input probability distribution, it performs nearly as well as the optimum source code for that distribution
- Used in `gzip`, `gif` (Graphics Interchange Format)
- We will see:
  - **LZ77** (1977): the **sliding-window** version
  - **LZ78** (1978): the **tree-structured** version

LZ78 Algorithm

LZ78 Algorithm

Algorithm 1: LZ78 algorithm

$p = 0$
dictionary dict $\leftarrow \{\}$
$F \leftarrow$ ""
while $p \neq$ *end-of-file* **do**
    c is the character at position p
    if $F + c$ *in dict* **then**
        $p \leftarrow p + 1$
        $F \leftarrow F + c$
    else
        Output: $\langle dict[F], c \rangle$
        $F \leftarrow$ ""
    end
end

The info-theory of LZ78

- Consider the original string of length n
- Assume we broke it up into $c(n)$ phrases (pieces)

Figure 1: Partitioning of the string

- Each phrase is broken up into:
  - a reference to a previous phrase ($\log_2 c(n)$)
  - a letter of our alphabet ($\log_2 |\mathcal{X}|$)

$$\text{Number of bits used } \leq c(n)(\log_2 c(n) + \log_2 |\mathcal{X}|)$$

Upper Bound on the number of bits used

- Assume all the elements of the dictionary are similar sized, of size k

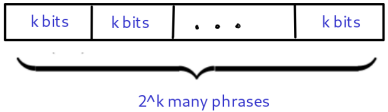

Figure 2: Partitioning of the string

- $2^k k \approx n$
- Also, $c(n) = 2^k$
- $\implies$ Number of bits used $\leq c(n)\log_2(c(n)) + c(n) = \boxed{n + O(\text{small})}$

Outline of Proof of Optimality of LZ78

Consider we have some sequence of iid rvs: $X = (X_1, X_2, \ldots, X_n)$

- Assume $x \to (y_1, y_2, \ldots, y_{c(n)})$, with Y's being the encoded phrases
- Probability that this sequence occurs, $P(X) = \prod_{i=1}^{n} P(X_i = x_i)$

$$P(X) = \prod_{i=1}^{c(n)} P(y_i) = \prod_{l} \prod_{|y_i|=l} P(y_i)$$

A bound on $\prod_{|y_i|=l} P(y_i)$

- Consider a particular length l
- Let c_l be number of phrases in y_i of length l
- We know that all y_i's are distinct $\implies \sum_{i:|y_i|=l} P(y_i) \leq 1$

Maximum value this term takes:

$$\prod_{|y_i|=l} P(y_i) \leq \left(\frac{1}{c_l}\right)^{c_l}$$

from (AM-GM) inequality

$$\sum_{|y_i|=l} \log P(y_i) \leq -c_l \log(c_l)$$

One nice lemma

To prove: $\sum_l c_l \log(c_l) \approx c(n) \log c(n)$

Proof

$$\sum_l c_l \log(c_l) = c(n) \log c(n) + c(n) \sum_l \frac{c_l}{c(n)} \log \left(\frac{c_l}{c(n)} \right)$$

We have the constraint $\sum_l l c_l = n$. Given that,

$$\sum_l \frac{c_l}{c(n)} \log \left(\frac{c_l}{c(n)} \right) \approx \log \frac{n}{c(n)}$$

More slick tricks

Adding up over all lengths: $\log P(X) \leq -\sum_l c_l \log(c_l) \approx c(n) \log c(n)$

Number of bits used $\leq c(n)(\log_2 c(n) + \log_2 |\mathcal{X}|) \sim c(n) \log c(n)$

$$nH(p) = -E[\log P(x)] \sim c(n) \log c(n)$$

We're done!

LZ77 Algorithm

LZ77 Algorithm

Algorithm 2: LZ77 algorithm

$p = 0$
while $p \neq$ *end-of-file* **do**
    Find the longest match in the window for the lookahead buffer
    if *a match is not found* **then**
        $\langle 0, c \rangle$, where c is the character is at position p
        $p \leftarrow p + 1$
    else
        Output $\langle 1, T, L \rangle$, where we go T characters back and match a string of length L
        $p \leftarrow p + L$
    end
end

Proof of Optimality of LZ77

The Sliding-Window Lempel–Ziv Algorithm is Asymptotically Optimal

AARON D. WYNER, FELLOW, IEEE, AND JACOB ZIV, FELLOW, IEEE

Invited Paper

Figure 3: Original Paper

QR Code for Further links

Figure 4: QR Code