TUTORIAL V

0 Continue the previous TD...

1 Lempel-Ziv compression

Previously we have seen Huffman algorithm for various length lossless data compression and have shown that it achieves the optimal expected length for a distribution P_X of a source composed of some symbols from $\mathcal{X}$. However, there are some disadvantages to it. First of all, it requires prior knowledge of P_X, or, alternatively its estimation. Therefore to be able to apply it when P_X is not known we need two passes through the data we wish to encode. The first pass is used for estimating P_X, and the second pass for actually encoding the data.

The Lempel Ziv algorithm constructs its dictionary on the fly, only going through the data once. Although there are many variations of Lempel-Ziv compression, in this exercise we will concentrate on one of its simplest versions. The idea of the procedure is to split an input sequence into distinct phrases, and in this version it is done greedily. We start with the shortest phrase on the left that we havent seen before (which will always be a single letter). On each step we delimit the next phrase we have not seen until we run out of letters. For the sake of simplicity consider a source composed of symbols in $\mathcal{X} = \{0, 1\}$ distributed according to $P_X(0) = p$ and $P_X(1) = 1 - p$. The following example illustrates an input string and the corresponding splitting into phrases:

$$00101110100101110110 11 \rightarrow 0|01|011|1|010|0101|11|0110|11$$

Now we encode the string by constructing a dictionary, where each phrase is assigned to its number. If a phrase has already been seen, it is in the dictionary, and instead of a phrase itself we can use its number in the encoding. The following table presents the encoding of our example string (for the sake of convenience the first empty phrase is encoded with an empty string):

Phrase number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Phrase	0	01	011	1	010	0101	11	0110	11
Encoding	$\emptyset$0	**1**1	**2**1	**0**1	**2**0	**5**1	**4**1	**3**0	**7**

Concatenating the last row with the phrase numbers expressed in binary gives the following string (the dividers and phrase numbers are left in bold for the readability):

$$\mathbf{0}0|\mathbf{1}1|\mathbf{10}1|\mathbf{00}1|\mathbf{010}0|\mathbf{101}1|\mathbf{100}1|\mathbf{011}0|\mathbf{0111}$$

Note also that starting with the $2^k + 1$ dictionary element we used k bits, so the number of bits used to represent a phrase number is always increasing. This ensures that the decoding algorithm knows how to split a given codeword back into phrases [1].

1. What is the codeword corresponding to Lempel-Ziv encoding of 0^n?
2. Let $c(n)$ be the number of phrases in the splitting of a string of length n. Give an upper bound on the total number of bits of its encoding as a function of $c(n)$.

Now we are interested in giving an upper bound on this result as a function of n in the worst case and average case scenarios.

[1] Adapted from http://www-math.mit.edu/ shor/PAM/lempel_ziv_notes.pdf

1.1 Worst case analysis

In this part of the exercise we will discuss what is the maximum number of distinct phrases that a string of length n can be split into. Let k be the maximal number of bits used to encode a phrase number in a dictionary.

1. Give one worst case string for $k = 2$ and $k = 3$. What about any $k > 1$?
2. Let n_k be the length of the worst case string for k. Show that $n_k = (k-1)2^{k+1} + 2$. Give an upper bound of $c(n_k)$ as a function of n_k.
3. Now, for an arbitrary length n, we can write $n = n_k + \Delta$. Use the previous bound to show that $c(n) \leq \frac{n}{\log_2 c(n) - 3}$.
4. Using the previous inequality give an upper bound on the total number of bits of the encoding. Does it make you happy?

1.2 Average case analysis

We now need to show the previous bound in the case of random strings. Let $X = X_1 X_2 \dots X_n$ be a random string of length n such that X_i for all $i \in \{1, \dots n\}$ are independent and identically distributed according to P_X.

1. Let $Q(X)$ be the probability of a string X. Suppose that X is broken into distinct phrases $X = Y_1 Y_2 \dots Y_{c(n)}$. Now let c_l be the number of phrases Y_i of length l. By definition of Lempel-Ziv they are all distict. Show that
$$-\log Q(X) \geq \sum_l c_l \log_2 c_l.$$
Hint: Recall that to maximize a product of k terms, given a fixed sum of these terms, the best thing to do is to make all of these terms equal. Proceed by showing that $\prod_{|Y_i|=l} Q(Y_i) \leq \left(\frac{1}{c_l}\right)^{c_l}$.
2. Suppose we have m_i occurences of i for $i \in 0, 1$ in X, then $Q(X) = p^{m_0}(1-p)^{m_1}$. Show that $-\log_2 Q(X) \simeq nH_2(p)$.
3. Show that $\sum_l c_l \log_2 c_l \geq c(n) \log_2 c(n) - O(\log_2 \frac{n}{c(n)})$. Conclude.

2 Channel capacity

Definition 2.1 (Information capacity)**.** *The information capacity of a channel $W_{Y|X}$ is given by $C(W_{Y|X}) = max_{P_X} I(X;Y)$, where the joint distribution of X, Y is defined by $P_{XY}(x,y) = P_X(x)P_{Y|X}(y|x)$.*

1. For a discrete channel $W_{Y|X}$ with input alphabet $\mathcal{X}$, output alphabet $\mathcal{Y}$. Let $C(W)$ denote the channel capacity of W. Show that
   (a) $C(W) \geq 0$.
   (b) $C(W) \leq \log_2 |\mathcal{X}|$.
   (c) $C(W) \leq \log_2 |\mathcal{Y}|$.
   (d) $I(X;Y)$ is a continuous concave function of $p(x)$.
2. Given a channel $W_{Y|X}$ and channel capacity $C(W) = \max_{p(x)} I(X;Y)$, suppose you apply a preprocessing step to the output by forming $\tilde{Y} = g(Y)$.
   (a) Does it strictly improve the channel capacity?
   (b) Under what conditions does the capacity not strictly decrease?