ENTROPY CODING

The design of a variable-length code such that its average code word length approaches the entropy of the DMS is often referred to as enlmpy coding. In this section we present two examples of entropy coding.

- Shannon- Fano Coiding
- Huffman Coding

Entropy Encoding

The design of a variable-length code such that its average code word length approaches the entropy of the DMS is often referred to as ***entropy coding***.

In this lecture, we will present two examples of entropy coding.

- Shannon-Fano Coding
- Huffman Coding

A. Shannon-Fano Coding:

An efficient code can be obtained by the following simple procedure, known as Shannon- Fano algorithm:

1. List the source symbols in order of decreasing probability.
2. Partition the set into two sets that are as close to equiprobable as possible, and assign 0 to the upper set and 1 to the lower set.
3. Continue this process, each time partitioning the sets with as nearly equal probabilities as possible until no further partitioning is possible.

A. Shannon-Fano Coding:

- Consider a 6 symbol alphabet: $\{x_1, x_2, x_3, x_4, x_5, x_6\}$ with corresponding probabilities $\{0.30, 0.25, 0.20, 0.12, 0.08, 0.05\}$
- Use the Shannon Fano coding algorithm to compute a variable length code.
- (On Overhead)

A. Shannon-Fano Coding:

- Compute the entropy

$$H(X) = (-0.30 \times \log_2(0.3)) + \ldots + (-0.05 \times \log_2(0.05)) = 2.36\text{b/symbol}$$

- Compute the average codeword length

$$E(L) = (0.30 \times 2) + (0.25 \times 2) + \ldots + (0.05 \times 4) = 2.38\text{b/symbol}$$

- Compute the efficiency of the code.

$$\eta = H(X)/E(L) = 2.36/2.38 = 0.99$$

B. Huffman Encoding:

The Huffman encoding procedure is as follows:

1. List the source symbols in order of decreasing probability.
2. Combine the probabilities of the two symbols having the lowest probabilities, and reorder the resultant probabilities; this step is called reduction 1. The same procedure is repeated until there are two ordered probabilities remaining.
3. Start encoding with the last reduction, which consists of exactly two ordered probabilities. Assign 0 as the first digit in the code words for all the source symbols associated with the first probability; assign 1 to the second probability.
4. Now go back and assign 0 and 1 to the second digit for the two probabilities that were combined in the previous reduction step, retaining all assignments made in Step 3.
5. Keep regressing this way until the first column is reached.

B. Huffman Encoding:

- (Implementation on overhead)
- The underlying entropy is 2.36 b.
- The codeword lengths are the same as for Shannon Fano Coding. So the average code length $E(L)$ and the efficiency η as the same also.
- In general, Huffman encoding results in an optimum code. Thus, it is the code that has the highest efficiency.

Huffman coding is an entropy encoding algorithm used for lossless data compression.

Huffman encoding algorithm

A frequency based coding scheme (algorithm) that follows Huffmans idea is called Huffman coding. Huffman coding is a simple algorithm that generates a set of variable-size codewords of the minimum average length. The algorithm for Huffman encoding involves the following steps:

1. Frequency Table: Constructing a frequency table sorted in descending order.
2. Building a binary tree: Carrying out iterations until completion of a complete binary tree:
   (a) Merge the last two items (which have the minimum frequencies) of the frequency table to form a new combined item with a sum frequency of the two.
   (b) Insert the combined item and update the frequency table.
3. Deriving Huffman tree: Starting at the root, trace down to every leaf; mark 0 for a left branch and 1 for a right branch.
4. Generating Huffman code: Collecting the 0s and 1s for each path from the root to a leaf and assigning a 0-1 codeword for each symbol.

Huffman Coding

Huffman coding is a method of lossless data compression, and a form of entropy encoding. The basic idea is to map an alphabet to a representation for that alphabet, composed of strings of variable size, so that symbols that have a higher probability of occurring have a smaller representation than those that occur less often.

Huffman Coding

The key to Huffman coding is Huffman's algorithm, which constructs an extended binary tree of minimum weighted path length from a list of weights. For this problem, our list of weights consists of the probabilities of symbol occurrence. From this tree (which we will call a Huffman tree for convenience), the mapping to our variable-sized representations can be defined.

Huffman Coding

The mapping is obtained by the path from the root of the Huffman tree to the leaf associated with a symbol's weight. The method can be arbitrary, but typically a value of 0 is associated with an edge to any left child and a value of 1 with an edge to any right child (or vice-versa). By concatenating the labels associated with the edges that make up the path from the root to a leaf, we get a binary string. Thus the mapping is defined.

Inverse Mapping

- In order to recover the symbols that make up a string from its representation after encoding, an inverse mapping must be possible. It is important that this mapping is unambiguous.
- We can show that all possible strings formed by concatenating any number of path labels in a Huffman tree are indeed unambiguous, due to the fact that it is a complete binary tree.
- That is, given a string composed of Huffman codes, there is exactly one possible way to decompose it into the individual codes.

Ambiguity

Ambiguity occurs if there is any path to some symbol whose label is a prefix of the label of a path to some other symbol. In the Huffman tree, every symbol is a ***leaf***. Thus it is impossible for the label of a path to a leaf to be a prefix of any other path label, and so the mapping defined by Huffman coding has an inverse and decoding is possible.