Huffman Coding Algorithm

The Huffman coding algorithm is a method produced to optimise the compression of data. The algorithm takes the probability of the occurrence of each symbol into account. For example, if we had:

, as our data, the probability of occurrences of each symbol would be:

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Frequency** | **(3 d.p.)** |
| T | 3 | 0.333 |
| E | 1 | 0.111 |
| S | 1 | 0.111 |
| \_ | 1 | 0.111 |
| D | 1 | 0.111 |
| A | 2 | 0.222 |
|  |  |  |

Table 1: A table breaking down the symbol information within the example data

To evaluate the number of coding digits to assign to each character, the following needs to be performed:

, where is the number of coding digits for each character.

For our example, with , . Therefore, each character at maximum requires three coding digits for its representation, such as (**Note: this is not the optimal pattern**):

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Number of Coding Digits** | **Coding Pattern** |
| T | 3 | 000 |
| E | 3 | 001 |
| S | 3 | 010 |
| \_ | 3 | 011 |
| D | 3 | 100 |
| A | 3 | 101 |

Table 2: A table displaying a potential coding pattern for the example data

However, it needs to be ensured, that no symbol mapping is the prefix of another.

Table 3 displays the process of merging symbols and building a Huffman tree, starting from the reordering of the symbols from lowest to highest probability. When two symbols (or collections of symbols) are merged, their probability becomes the sum of the probabilities of those independent symbols combined.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Original | | Iteration 1 | | Iteration 2 | | Iteration 3 | |
| Symbol |  | Symbol |  | Symbol |  | Symbol |  |
| T | 0.333 | E | 0.111 | \_ | 0.111 | \_D | 0.222 |
| E | 0.111 | S | 0.111 | D | 0.111 | ES | 0.222 |
| S | 0.111 | \_ | 0.111 | ES | 0.222 | A | 0.222 |
| \_ | 0.111 | D | 0.111 | A | 0.222 | T | 0.333 |
| D | 0.111 | A | 0.222 | T | 0.333 |  |  |
| A | 0.222 | T | 0.333 |  |  |  |  |
| Rearrangement | | ---- | | ES  E  S | | \_D  \_  D | |
| Iteration 4 | | Iteration 5 | | Iteration 6 | | | |
| Symbol |  | Symbol |  | Symbol | | 1 | |
| A | 0.222 | \_DES | 0.444 | \_DESAT | |
| T | 0.333 | AT | 0.555 |  | |
| \_DES | 0.444 |  |  |  | |
|  |  |  |  |  | |
| ES  E  S  \_D  \_  D  \_DES | | AT  A  T | | ES  E  S  \_D  \_  D  \_DES  AT  A  T  \_DESAT | | | |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |  |  |

Table 3: Table displaying Merging of Symbols and the Huffman Tree Stages

Upon each decision/fork at a node, the Huffman tree should be labelled with “0” on the left fork and “1” on the right fork, as in Figure 1.

Figure 1: Labelled Huffman Tree

ES

E

S

\_D

\_

D

\_DES

AT

A

T

\_DESAT

0

0

0

0

0

1

1

1

1

1

The new and optimised coding can be developed from the labelled Huffman tree, traversing the tree from the root node.

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Number of Coding Digits** | **Coding Pattern** |
| T | 2 | 11 |
| E | 3 | 010 |
| S | 3 | 011 |
| \_ | 3 | 000 |
| D | 3 | 001 |
| A | 2 | 10 |

Table 4: Optimised symbol coding

# Evaluation

When we take the data: , there are 9 characters.

With a fixed-length encoding size, the size of the data is: 9x3 = **27 bits**

With Huffman encoding, the size of the data is: 2+3+3+2+3+3+2+2+2 = **22 bits**

Therefore, Huffman encoding has compressed the data. However, for data of this size, it is unlikely to be effective. When decoding the data, the decoder must also know the structure (the coding pattern for each symbol), therefore practically, for small pieces of data, Huffman will be ineffective. When scaled to larger data, the Huffman algorithm can be effective.

# Algorithm

The Huffman coding algorithm is as follows:

1. Take a piece of data of length L and evaluate the frequency, , of each symbol;
2. Evaluate the probability of each symbol occurring, ;
3. Order each symbol based on smallest to largest probability;
4. Combine the symbols with the two smallest probabilities (summing their probabilities) to generate a new merged symbol. Ensure that a record of the individual symbol probabilities is kept;
5. Iterate until only one merged symbol is left;
6. Appropriately label ‘paths’ to individual symbols with either a “1” or a “0”;
7. Based on the 1 and 0 values, produce the new optimised coding for each symbol;
8. Evaluate the compression performance.