FIRST SIMULATION PROJECT

Youssef sherif mansour

JANUARY 4, 2017

YOUSSEF SHERIF MANSOUR
201305127

Included Files:

I made a separate function called ‘huffmanEncode.m’ that takes an input vector that is either a string or a sequence of numeric values. Then it returns the coding table (dictionary), the encoded message, the average length after encoding, the entropy and finally the efficiency. Then there are two decoders one for character symbols {huffmanDecodeStr.m} and the other for the numeric symbols {huffmanDecode.m}.

Algorithm:

1) Sort the input and then find the probability of each symbol.
2) Form a big matrix where each row acts like a node of the final tree with the following attributes

Value	Probability	Trigger	Index	Right child	Left child	Code index

   The first two columns are self-explanatory, the trigger is used in sorting the data while building the tree, the index, left child and right child are all used also in building the tree. Finally the code index is used to assign code words to each symbol.

3) **Building the tree:**
   a) Add the symbols firstly with their probability and index in any order with trigger, left child and right child = 0.
   b) Loop until the probability of the current node is 1.
   c) Sort the matrix according to the probability column in ascending order.
   d) Sort the matrix according to the trigger column in ascending order.
   e) Add the values and probabilities of the first two rows into a new row with the next index and trigger = 0 while placing the first row index in the right child and the second row index in the left child.
   f) Set the trigger of the first two rows to ‘1’.
   g) Continue the loop.
4) Once the loop finishes sort the big matrix in a descending order according to the probability.
5) Remove the first node because it has no code word.
6) **Traversing the tree:**
   a) Give the first two rows codes of ‘1’ and ‘0’.
   b) Loop on all the nodes
   c) If the left child = 0 do nothing
   d) If the left child = value go to the left child index and give it a code of the same row and add ‘1’ to it, then do the same for the right child however add ‘0’
7) Once the loop finishes encode the input by looping it and giving each element a code.

Results:

In the main file I compare the results with the built-in function by watching the length of the encoded message.

The difference is always zero which means that my function is working well.

Entropy = 2.11

Average length = 2.21

Efficiency = 95.47%

Testing on speech signal:

I quantize the signal by multiplying it with a constant and then approximating its values.

Attached are two samples Hawke.wav is the original, while Hawkde.wav is the one that is decoded. They aren’t the same due to the quantization error resulting from rounding the values of the sound.

I saved the encoded message and the table in a txt file {huffmanencode.txt} to compare the sizes and I got a compression of around 60% for this small file.