UNIVERSITY OF LIEGE
APPLIED SCIENCE FACULTY
Academic year 2017-2018

Information and coding theory
Project 3

Damien Seron

april 2018

4

The first step of the implementation is to build the binary tree corresponding to the given probabilities. First, the probabilities are sorted in ascending order and given a number depending on their position (from 1 to N where N is the number of probabilities in the given vector). Then at each iteration the two smaller probabilities are removed and their sum is added to the list of probabilities with a number (previous largest number attributed + 1). In the adjacency matrix a way from the sum to the removed probabilities is added. The list of probabilities is sorted and a new iteration begins until only 1 probability is left in the list.

In the resulting adjacency matrix every node can access precisely 0 or 2 other nodes, making it a representation of a binary tree. In order to give a code to each probability a recursive algorithm will be applied in depth, starting with the root (last line of the matrix). At each call of the function on a node the child with the smallest number will see its code be (code of the current node + "0") and the child with the largest number will see its code be (code of the current node + "1"). Then the function is called recursively for each child.

Here is an example of the code and tree generated.

Probability	0.05	0.1	0.3	0.1	0.25	0.1	0.03	0.07
Associated code	11111	010	10	011	00	110	11110	1110

Table 1 – Probabilities and associated codes generated by the function

Note : As seen in Figure 1, the orientation of the edges of the tree is from the greater probability to the smallest. This is the representation in the adjacency matrix but in the theoretical course we most often see the edges directed towards the root of the tree instead of the leaves (I tried this way but the visualisation of the tree was not well organised). The length of the code associated with a probability is the depth of the node corresponding to this probability, a smaller probability must have a depth $\geq$ than the depth of a greater probability.

5

The English alphabet is larger than the alphabet encountered in the text sample (Latin). It can still be used as every Latin letter can be found in the English alphabet but letters such as "j","k","w","y" and "z" will always have a probability of appearance of 0 since they do not exist in Latin.

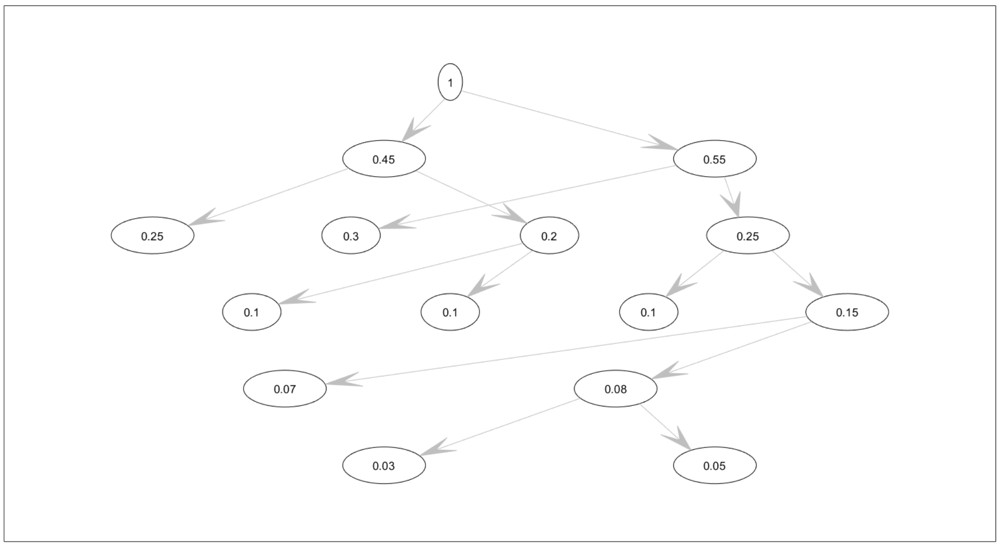

FIGURE 1 – Visual representation of the adjacency matrix obtained with *Huffman*. Each node is associated with a probability. The leaves are the probabilities given as argument. Each internal node is the sum of its children

6

We could maybe, knowing how Latin works, have an alphabet with words such as "um" "us" and common declinations since those groups of letters can be quite frequent.

7

We could model the source with an alphabet and the associated probabilities but also as a Markov chain with each time the probability of having a specific letter depending of the knowledge of the previous letter(s). If we take only into account the previous letter we have a transition matrix of size N x N (N is the alphabet size).

8

Each symbol is of length n = 4. In the data we only encounter Q = 16 distinct words and we know that q = 10 from the statement.

$$\sum_{i=1}^{16} 10^{-4} = 1.6 * 10^{-3} \leq 1 \tag{1}$$

The Kraft condition is respected. This means that there exists a uniquely decodable code having a length of 4 (with an alphabet from 0 to 9). If we compute the probability of occurrence of each symbol with *estimate_proba* and then compute the entropy we obtain 3.6517. This is the optimal average length for a symbol (in bits). This code is not complete since the result of the sum in Kraft's inequality is not exactly 1.

9

To represent a number in 0-9 we need at least 4 bits. To represent 4 numbers (one symbol) we then need 16 bits. With *encode_to_numerals* we get codes between 1-16, which can be represented with 4 bits. The resulting code is then $16/4 = 4$ times compressed (compression rate of 4). Since we only detect 16 symbols in data we can expect after the use of *encode_to_numerals* to receive a symbol of maximum 4 bits. (if we had no idea that the source only transmits 16 symbols we know that the source can only emit 10000 different messages but then

10

Length of symbol n = 1, Q = 16 different messages encountered, q = 16 (the alphabet is a number between 1 and 16 represented with 4 bits)

$$\sum_{i=1}^{16} 16^{-1} = 1 \tag{2}$$

This code is decodable, complete, regular, instantaneous and prefix -free (all codes are 4 bits sized and each one is associated with one symbol). This code is not absolutely optimal since the avergae