Universidade de Aveiro
Departamento de Electrónica, Telecomunicações e Informática

Algorithmic Information Theory (2023/24)

Lab work n$^{\circ}$ 1 — Due: 24 Mar 2024

1 Copy models for data compression

Data compression is obtained through the exploration of self similarities. There are several ways of attaining this goal, some of them will be studied in this course. In this work, it is intended to address one of them, which relies on the idea that some data sources can be viewed as being produced by replicating parts already produced in the past, although with some possible modifications. Generically, we refer to this process as a **copy model**.

A copy model predicts that the next outcome will be a symbol that has occured in the past. This is done by maintaining a pointer referring to the symbol being copied, as well as some additional information, needed to estimate how reliable are those predictions.

Let us denote by $x_1^n = x_1 x_2 \ldots x_n,\ x_i \in \Sigma$, the sequence of outputs (symbols from the source alphabet Σ) that the information source has generated until instant n. In practical terms, we need to estimate the probability of the next symbol, x_{n+1}, being the same as the currently predicted symbol, x_{n-p}, where $p \geq 0$ is the pointer handled by the copy model.

For example, if $\Sigma = \{0, 1\}$, then we may have $x_1^8 = 01001101$. In this example, we have $x_1 = 0$, $x_2 = 1$, $x_3 = 0$, and so on. Then, for example, if the copy pointer is $p = 4$, then the predicted symbol is $x_{n-p} = x_{8-4} = x_4 = 0$.

1.1 The estimation of probabilities

The idea is to collect counts that represent the number of times that each symbol has been correctly predicted (N_h, denoting the number of hits), versus the number of times that it was wrongly predicted (N_m, denoting the number of prediction misses), and use these counters for estimating a probability. This can be done directly using the relative frequency, i.e., saying that the probability of the next prediction being a hit is $N_h/(N_h + N_m)$. However, this suffers from the problem of assigning zero probability to events that

were not seen during the construction of the model. That would be the case, for example, for the first time the probability is calculated. This problem can be overcome using a "smoothing" parameter for estimating the probabilities, i.e.,

$$P(\text{hit}) \approx \frac{N_h + \alpha}{N_h + N_m + 2\alpha},$$

where α is the smoothing parameter (the value "2" equals the number of events that are being predicted, in this example two, hit or miss). Note that, using this estimator, the first probability that is estimated is equal to $1/2$, because, when the copy model is started, both N_h and N_m are zero. Also, note that this is just an example. In practice, at each coding step (i.e., for each symbol being encoded), we need a distribution of probabilities for all the symbols of the alphabet, i.e., $P(s), \forall s \in \Sigma$.

Having the model computed, we can estimate the amount of information that a new symbol, s, needs to be represented. This is given by $-\log_2 P(s)$. [1]

2 Work to be done

1. Develop a program, named `cpm`, that implements a copy model. Note that the pointer associated to the copy model needs to be repositioned from time to time. This happens when the performance of the model drops below a certain threshold (i.e., the relation between the number of hits and the number of misses becomes too low). This repositioning needs to be done according to the previous k symbols that have been encoded, i.e., we need to look for a past occurrence of the string $x_{n-k+1} \ldots x_n$.

   The program should read the file to be "compressed" on a byte-wise manner, i.e., the alphabet to be considered is made of the different byte values found in the file.

   All required parameters to control the model should be passed as command line arguments. This program should provide the estimated total number of bits for encoding a certain file, as well as the average number of bits per symbol.

2. Create a program, named `mutate`, that mutates the content of a file according to a certain mutation probability. For example, suppose that the file contains only symbols from the alphabet $\Sigma = \{A, B, C\}$. Then, this program will change an "A" to a "B" or "C" according to the provided probability, and so on.

3. Perform a comparative study of the performance of some general purpose compressors, such as `gzip`, `bzip2`, `zstd`, ..., and the one that you developed, as a function of the mutation probability. Consider several types of data (text, DNA, ...).

[1] This concept will be explained during the classes.

4. Elaborate a report, where you describe all the steps and decisions taken during the development of the work. Include relevant and illustrative results that were obtained. In particular, a discussion regarding the effects of the variation of the parameters on the compression performance and computation time required. Besides the texts that you choose, it is mandatory to include compression measures for the text `chry.txt`, available in moodle.

5. Elaborate a short video (maximum 5 minutes) where you describe and demonstrate the work done.

3 How to deliver the work

The work should be developed and made available to the teachers using a repository following the structure suggested in Fig. 1 (if needed, further instructions will be provided during the classes). The developed software should be compatible with the Linux operating system. It should include a README file, explaining, with sufficient detail, how to build the program and all the parameters that it accepts. It should also include, at least, how to run a concrete example.

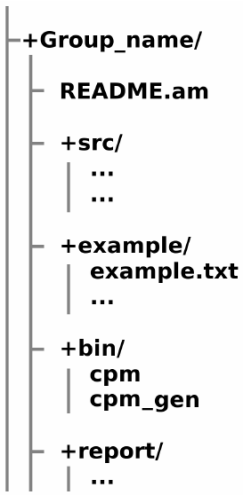

Figure 1: Suggested repository structure.