Assignment 1 - Copy models for data compression

Algorithmic Information Theory (2022/23)
Universidade de Aveiro

Martinho Tavares, 98262, martinho.tavares@ua.pt
Nuno Cunha, -, - Pedro Lima, 97860, p.lima@ua.pt

March 28, 2023

1 Introduction

One of the goals of this assignment is to implement a copy model for data compression, which is obtained through the exploration of self similarities. A copy model is only one of the ways to address the problem, and it's idea is that certain data sources can be viewed as a sequence of repetitive patterns, where parts of the data have been replicated in the past, although some modifications are possible. The model consists in predicting the next symbol based on previous ones observed in the past, by keeping a pointer referring to the symbol being copied, as well as other information.

The second goal of the assignment is to implement a text generator based on the copy model, which is a way to generate new text based on a given one. The text generator receives a text as input to train the model, and then follows a similar approach to the one used in the copy model, but instead of predicting the next symbol, it uses the probability distribution to generate a random symbol based on these probabilities.

In this report we will first present how the work was organized, in the 2 section. Then, we will present the 3, how we implemented it and the results obtained, which we will compare by calculating the entropy of different text examples, like chry.txt given by the teachers, with different parameters we defined for the model. The next section is dedicated to the 4, where we will present the implementation and the results obtained for the text generator, using different texts as input for training of the generator, as well as diferrent starting points for the generation of text. Finally, we will conclude the report in the 5 section.

2 Work organization

3 Copy model

...

In order to evaluate whether the copy model can provide acceptable results, we can use a baseline below which we expect the model to report the file's entropy. We decided to use, as a baseline, the entropy considering each symbol's relative frequency in the entire file, which is given by:

$$H(X) = -\sum_{x \in X} p(x) \log p(x)$$

With this value in mind, we evaluated the model as a whole with different values for its parameters, on different files. The files chosen for testing are present in the repository[1], and they have the following baselines:

- `chry.txt`: ...
- `...`: ...

...

Throughout this section, the different program parameters are detailed, and their effect on the model's performance is studied.

3.1 Pattern size

When choosing a pointer in the past from which to start copying, we need to look for an occurrence of the same k-sized pattern as the one we are currently on. Thus, k is one of the parameters that affects program performance, where k is a positive integer.

On one hand, a lower value of k means there will be more occurrences of each pattern, since there will be lesser possible k-sized patterns, which will result in more oportunities to copy from the past. On the other, a larger k will result in less occurrences for each pattern, which results in more scarce predictions, and thus more overall guesses.

For a given k, the number of possible patterns M is equal to the number of permutations of the symbols of alphabet A: $M = |A|^k$. The results on changing this value will be studied taking this exponential growth into account.

3.1.1 Results

...

3.2 Smoothing parameter alpha

At each coding step, the probability of correctly predicting the current character is given by the formula:

$$P(\text{hit}) = \frac{N_h + \alpha}{N_h + N_f + 2\alpha}$$

[1] `https://github.com/NMPC27/TAI-G7-Lab1`

where N_h and N_f are respectively the number of hits and misses, and α is the smoothing parameter. The smoothing parameter is used to avoid the model having a probability of 0 when there are no hits or misses, which will happen when no prediction has been made yet, and also avoid assuming with complete certainty (probability of 1) that the prediction will be correct when no fails were made yet, which will result in infinite information in case the prediction fails.

If α is lower, we expect the reported probabilites to be more "certain" with a greater possibility of reaching high values. If α is greater, the smoothing effect should be more pronounced, producing probability values that hardly increase, which should result in relatively less information when predictions fail but also lesser returns on the correct predictions.

This trade-off between α and the reported information, especially regarding the information spikes that may occur, will be studied in the results.

3.2.1 Results

...

3.3 Base probability distribution

For each symbol in the file, a probability distribution has to be generated according to the prediction probability. In case a given prediction fails, we need to distribute the complement of the probability of a correct prediction $1 - P(\text{hit})$ among the other symbols of the alphabet, except the one that we predicted would occur (which has probability $P(\text{hit})$).

The choice of distribution also influences the guesses that are made. When a prediction can't be made, the model assumes a distribution for the entirety of the alphabet, which is the same probability distribution as the use used when distributing the complement of the prediction probability.

We developed two simple approaches to distributing the probability: a uniform distribution and a frequency distribution.

The uniform distribution assigns to each symbol a probability equal to:

$$\frac{1 - P(\text{hit})}{|A| - 1}$$

On the other hand, the frequency distribution assigns to each symbol $a \in A$ a probability equal to:

$$(1 - P(\text{hit})) \times p(a)$$

where $p(a)$ is the probability of a in the whole file.

Both distributions are compared especially in their effect on the overall reported information, when k is large and thus more guesses are made, as well as when α is large and the prediction fails should report. We assume that, since the frequency distribution models the overall distribution of the symbols, it should provide better results. We should note, however, that the file's content

may have locality, i.e. select portions in the file where the symbol distribution is skewed. Therefore, we don't disregard the possibility that the assumption of the distribution being maintained throughout the file can hurt the model performance as well.

3.3.1 Results

...

3.4 Copy pointer repositioning strategy

Different strategies can be followed when choosing a copy pointer from which to start copying. We developed 3 approaches, 2 of which being fairly simplistic:

- **Oldest pointer first**: the chosen copy pointer is the oldest possible. If the current copy pointer is repositioned because the threshold was met, then that pointer will be completely discarded and the next oldest pointer is used. This is the most simple approach, which assumes similar patterns are very far in the file;
- **Newest pointer first**: the chosen copy pointer is always the newest, which assumes that similar patterns are very close in the file;
- **Most common prediction**: when a pointer has to be chosen, the predictions of all registered pointers are evaluated, and the pointer (or one of them) which presented longest and most common prediction is chosen. A more detailed description of this strategy is presented below.

The **most common prediction** strategy chooses the copy pointer in successive iterations. Initially, the next immediate character is evaluated for every registered pointer. The pointers are grouped according to that character that they predict, and the largest group is chosen, i.e. the pointers that reported the most common prediction are chosen. Afterwards, the second next character is evaluated for every pointer in that group, and then the pointers are grouped in the same manner again, with the next group being the largest at this stage. This process is repeated until all pointers at the current group report different predictions, at which point the newest pointer is chosen.

3.4.1 Results

3.5 Copy pointer reposition threshold

...

3.5.1 Results

...

3.6 Verbose output

As a sidenote, the model offers the possibility of outputting the probability distributions and predicitons at each step, or of presenting the overall progress of processing the input file.

For the modes outputting the probabilities, the output can be in human-readable format, presenting whether the reported probabilites were from a guess or a prediction hit/miss in colored output. Alternatively, the output can be presented as comma-separated values with less extraneous characters in machine-readable format, which is useful for data analysis afterwards.

4 Text generator

The Generator module makes use of the functionalities offered by the CPM module. This module uses a Context Model calculated from a received text to calculate the next characters based on a context phrase.

It contains two main methods, the firstPass(), that reads the training file and makes an initial pass through the text to calculate the frequencies of the characters. This method also makes an context model based on the training file, and the size of the context is defined by the input string.

The `cpm_gen()` method uses the context model in the first pass to generate the text. When we try to generate the next symbol, we read the current context, which is the size of the input string and we check the context model for that context. If we have that context in the model, we get the list of possible next symbols, and their counts. Then, we generate a random value between 0 and the sum of the counts, and we check where that value lands between all the ranges of counts of events of that context. The symbol that is in the range of the random value is the next symbol of the text. If we don't have that context in the model, we get the list of all possible symbols in the alphabet, and their distribution, and we generate a random value between 0 and the sum of the distributions, and we check where that value lands between all the ranges of distributions of events of that context. The symbol that is in the range of the random value is the next symbol of the text.

We also have 2 other input parameters that modify the beavior of the generator. The first one is alow the model training himself, while generating the text. This is done by updating the context model with the generated text and the distibution of symbols in the alphabet. The second parameter is to allow all lowercase characters to be generated, this is done by converting all the characters to lowercase while reading the training file, this parameter provides a better context model without nedding a big training file, but it also makes the text generated to be all lowercase.

4.1 Results

5 Conclusion

6 References

References