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

3.1.1 Results

3.2 Smoothing parameter alpha

3.3 Base probability distribution

3.4 Copy pointer repositioning strategy

3.5 Copy pointer reposition threshold

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.

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

The cpmgen() 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 2 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.

4.0.1 Results

5 Conclusion

6 References