Universidade de Aveiro

Departamento de Electrónica, Telecomunicações e Informática

Algorithmic Information Theory (2022/23)

Lab work nº 2 — Due: 29 May 2023
(to be presented in the class of 2 Jun 2023)

1 Introduction

Consider the problem of determining the "similarity" between a target text, t, and some reference texts, r_i. For example, each r_i could be a sample text from a certain language and t could be a text whose language needs to be determined.

Usually, the traditional approach to solve this *classification problem* begins with feature extraction and selection operations. The features obtained are then fed into a function that maps the feature space onto the set of classes and performs the classification. One of the most difficult parts of this problem is how to choose the smallest set of features that retains enough discriminant power to tackle the problem.

The representation of the original data by a small set of features can be seen as a special form of *lossy data compression*. This suggests a question: Can data compression be explicitly used to approach classification problems, removing the need for a separate feature extraction stage? The answer is affirmative, i.e., it is possible to adopt an information-theoretic approach to the classification problem, bypassing the feature extraction and selection stages. In other words, compression algorithms can be used to measure the similarity between files.

The idea is the following. For each class, represented by the reference text r_i, we create a model that is a good description of r_i. By a "good description", we mean a model that requires fewer bits to describe r_i than the other models, or, in other words, that is a good compression model for the "members of the class" r_i. Then, we assign to t the class corresponding to the model that requires fewer bits to describe it, i.e., to compress t.

2 Work to be done

1. Develop a program, named `lang`, that accepts two files: one, with a text representing the class r_i (for example, representing a certain language); the other, with the text under analysis, t. Modeling should be performed using the copy models studied in the previous Lab Work (that you may further develop and improve). All parameters needed to control the models should also be provided to the program. The program should report the estimated number of bits required to compress t, using the model computed from r_i.

2. Based on the `lang` program, build a language recognition system, `findlang`, that, from a set of examples from several languages (the r_i), provides a guess for the language in which a text t was written.

3. You should obtain results with as many different languages as possible (it is recommended at least 20). It is up to you to find texts that are good representatives of the language and test them with appropriate examples (a good source of these texts can be found in `https://sourceforge.net/projects/la-strings/files/Language-Data/`).

4. Develop an application, `locatelang`, that can process text containing segments written in different languages. This application should return the character position at which each segment starts, as well as the language in which the segment is written.

5. Explore the possibility of using finite-context models (see the section below) to provide the probabilities of the non-hit symbols.

6. Elaborate a report, where you describe all the steps and decisions taken during the development of the work, and include relevant and illustrative results that were obtained. Provide also results regarding the classification accuracy of the system as a function both of the length of the references and of the length of the target texts. This report should be convincing enough for motivating someone to "buy" your product!

3 About finite-context models

One of the most used approaches for representing data dependencies relies on the use of Markov models. In lossless data compression, we use a specific type, called discrete time Markov chain or finite-context model.

A finite-context model can be used to collect high-order statistical information of an information source, assigning a probability estimate to the symbols of the alphabet, according to a conditioning context computed over a finite and fixed number of past outcomes. The main goal is to predict the next outcome of the source. To do this, we infer a model based on the observed past of the sequence of outcomes.

3.1 Markov models

Let us denote by $x_1^n = x_1 x_2 \dots x_n,\ x_i \in \Sigma$, the sequence of outputs (symbols from the source alphabet Σ) that the information source has generated until instant n. 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. A k-order Markov model verifies the relation

$$P(x_n|x_{n-1} \dots x_{n-k}) = P(x_n|x_{n-1} \dots x_{n-k} \dots),$$

where the sub-sequence $c = x_{n-1} \dots x_{n-k}$ is called the state or context of the process. A first-order Markov model reduces to

$$P(x_n|x_{n-1}) = P(x_n|x_{n-1}x_{n-2} \dots).$$

Markov models are particularly useful in text compression (but not only!), because the next letter in a word is generally heavily influenced by the preceding letters. In fact, the use of Markov models for written English appeared in the original work of Shannon. In 1951, he estimated the entropy of English (i.e., the average amount of information) to be between about 0.6 and 1.3 bits per letter.

We can use the frequency of the pairs of letters to estimate the probability that a letter follows any other letter. This reasoning can also be used for constructing higher-order models. However, the size of the model grows exponentially. For example, to build a third-order Markov model, we need to estimate the values of $P(x_n|x_{n-1}x_{n-2}x_{n-3})$. If a table is used[1], it has to accommodate $|\Sigma|^{k+1} = 27^4 = 531\,441$ entries, besides having to rely on enough text to correctly estimate the probabilities.

3.2 The estimation of probabilities

The idea is to collect counts that represent the number of times that each symbol occurs in each context. For example, suppose that, in a certain moment, a binary ($|\Sigma| = 2$) source is modeled by an order-3 finite-context model represented by the table

x_{n-3}	x_{n-2}	x_{n-1}	$N(0\|c)$	$N(1\|c)$
0	0	0	10	25
0	0	1	4	12
0	1	0	15	2
0	1	1	3	4
1	0	0	34	78
1	0	1	21	5
1	1	0	17	9
1	1	1	0	22

[1]Usually, more sophisticated data structures, such as hash-tables, are used.

where $N(s|c)$ indicates how many times symbol s occurred following context c. Therefore, in this case, we may estimate the probability that a symbol “0” follows the sequence “100” as being $34/(34+78) \approx 0.30$.

This way of estimating the probability of an event, e, based only on the relative frequencies of previously occurred events,

$$P(e|c) \approx \frac{N(e|c)}{\sum\limits_{s\in\Sigma} N(s|c)},$$

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, in this example, if we used the same approach to estimate the probability of having a “0” following a sequence of three or more “1s” (can you see what is the problem?).

This problem is overcome using a “smoothing” parameter for estimating the probabilities, i.e.,

$$P(e|c) \approx \frac{N(e|c)+\alpha}{\sum\limits_{s\in\Sigma} N(s|c)+\alpha|\Sigma|},$$

where α is the smoothing parameter.