University of Aveiro

Departamento de Electrónica, Telecomunicações e Informática

Master degree in Informatics Engineering

Lab Work nº1

Authors:
Andreia Machado, Nº 76501
Pedro Matos, Nº 73941

Algorithmic Information Theory

13th October 2017

Introduction

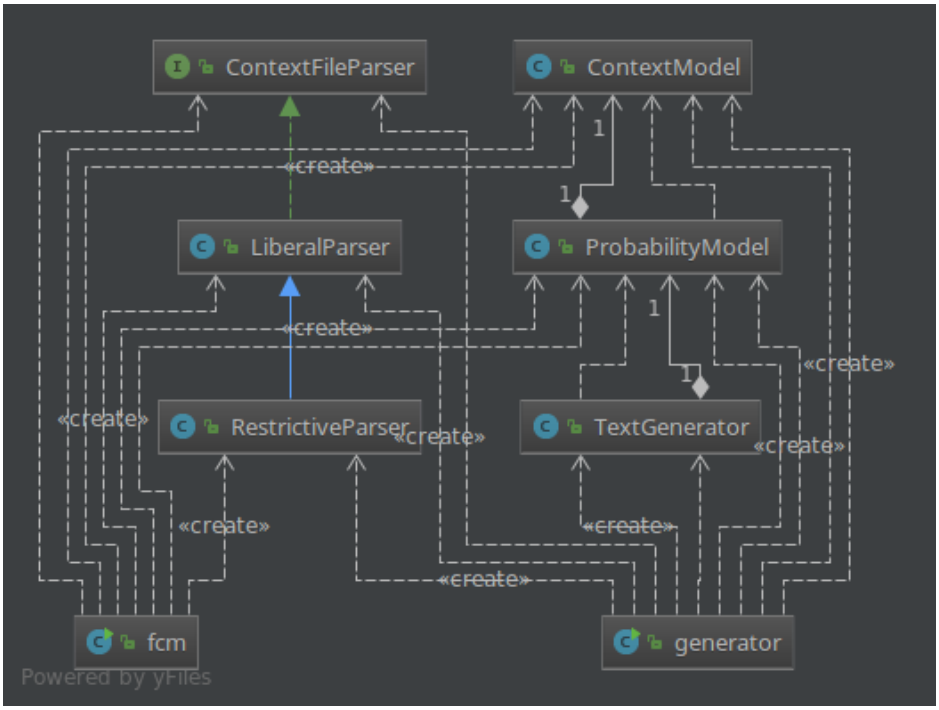

Figure 1: Class Diagram

Figure 1 represents the class diagram of this work, it has a very simple structure composed by two *main* classes that use objects of the type *ContextFileParser* to extract the content of a text file. After that the content goes to the *ContextModel* class where, depending on the order that is passed to the program a set of terms created, followed by the frequency of which certain characters appear after them. The creation of a *ProbabilityModel* then follows, all the occurrences of certain characters after the terms are used to calculate the probability of those characters appearing again after certain terms. In the end the *ProbabilityModel* will be filled with all the probabilities of a character appearing after certain term, and the entropy is calculate.
This same approach is used in both the *fcm* and *generator* programs, with the exception that, in *generator*, after the creation of the *ProbabilityModel*, the program uses a *TextGenerator* to create text that has a certain number of characters based on the context model of file, this text may or may not

make sense.

Class fcm

Class that interact with the user and execute the program to create a finite-context model. To execute this program is necessary 3 parameter, where there is a fourth parameter but it is optional. The first parameter is the file that contains the text to process, the second parameter is the order of the finite-context model, the three parameter is the level of creativity of the text generator and the four parameter is the type of parser used to process the text.
In this class is instantiated all the objects necessary to create a finite-context model and provide the entropy of the text estimated by the model.

Interface ContextFileParser

Interface with only one method parse(File), to make the parser of the document, by the classes that implements this interface. Throughout the development of this work was create two parses,*RestrictiveParser* and *LiberalParser*, that will be explained below.

Class LiberalParser

Class that implements the interface *ContextFileParser*. Using the java class *Scanner*,reads all the content of the file, there is no processing of the text, i. e., the text to be process is exactly the same of the file.

Class RestrictiveParser

lass that extends the class *LiberalParser*. In this class the text returned by the super class *LiberalParser* is processed where all the punctuation except the '-' is removed and all characters are transformed into upper cases.

Class ContextModel

Class that create a finite-context model based on the text. Depending of the order chosen by the user, this class create a unidimensional model if the order is 0 or create a multidimensional model otherwise. If the order is 0 is created a *HashMap* where the keys will be all the characters that appears in the text and the values will be the number of occurrences that character in the text. First the text is through an if a new character appears, a new entry is added to the *HashMap* where the key is the new character and the value is 1, otherwise will be increased the occurrences of the character previously added. If the order is greater than 0 is created a *HashMap* where the keys will be all the terms that appears in the text and the values will be an internal *HashMap*, where in this *HashMap* the keys will be the characters that follow the term (key of the external *HashMap*) and the values will be the number of occurrences that that character follow the term already saved in the external *HashMap*. The length of the term is the order chosen by the user, therefore the text is through used this principle. From the text is extract all the terms and all the following characters of this terms, if a new term appears a new entry is added to the *HashMap* where the key is the new character, and the value a new internal *HashMap* with one entry, where the key is the following character and the value is 1. Otherwise one of the follows two options will execute:

1. If the terms already exists in the *HashMap* but the following character does not exists, is added a new entry to the internal *HashMap* where the key is the follow character and the value is 1.

2. If the terms and the following character already exists in the *HashMap*, will be increased the occurrence of that follow character in the internal *HashMap*.

In both models is created an alphabet of the analyzed text, which contains the characters found in the text.

Class ProbabilityModel

Class that calculates, depending on the chosen order, the probability of a character or the probability that a character follows a term, and the entropy

of the analyzed text.
If the chosen order is 0 is created a *HashMap* where the keys will be all the characters that appears in the text and the values will be the probability of that character. To calculate the probability of that characters is used the following formula:

$$P(Character) = \frac{numOccurrences + alpha}{totalOccurrences + alpha * \sum} \quad (1)$$

where the numOccurences is the number of occurences of that character gotten by the *HashMap* of the *ContextModel* class, the totalOccurrences is the total number of occurrences of all characters, the alpha is a number greater or equal that 0 chosen by the user and the $\sum$ is the size of the alphabet. After the calculation a new entry is added to the *HashMap* of the *ProbabilityModel* class where the key is the character and the value is the probability of that character.
If the chosen order is greater than 0 is created a *HashMap* where the keys will be all the terms that appears in the text and the values will be an internal *HashMap*, where the keys will be all the characters that appears in the text and the values will be the the probability of that character following the key term. To calculate the probability of a character following a key term is used the following formula:

$$P(Character|term) = \frac{numOccurrences + alpha}{totalOccurrences + alpha * \sum} \quad (2)$$

Where the numOccurences is the number of occurences of that character following the key term gotten by the *HashMap* of the *ContextModel* class, the totalOccurrences is the total number of occurrences of all characters follow the term, the alpha is a number greater or equal that 0 chosen by the user and the $\sum$ is the size of the alphabet. After the calculation, one of the follows two options will execute:

1. If the *HashMap* of the *ProbabilityModel* class already have the term as a key is added a new entry to the internal *HashMap*, where the key is a new character and the value is the calculated probability.

2. If the *HashMap* of the *ProbabilityModel* class does no have the term as

a key, it's created a internal *HashMap* with one entry, where the key is a new character and the value is the calculated probability.

The entropy for order 0 is then calculated using this formula:

$$H_i = -\sum_{i=1}^{n} P(C_i) * log_2(P(C_i)) \tag{3}$$

Where n is the size of the alphabet, $P(c_i)$ is the probability of the character i and H_i is the entropy of i. The formula to calculate the entropy of i is:

The formula to calculate the entropy of the text where the chosen order was greater than 0 is:

$$H = \sum_{i=1}^{n} P(C_i|term) * H_i \tag{4}$$

where n is the size of the alphabet, $P(C_i|term)$ is the probability of the character i follows the term and H_i s the entropy of i. The formula to calculate the entropy of i is:

$$H_i = -\sum_{i=1}^{n} P(C_i|term) * log_2(P(C_i|term)) \tag{5}$$

Class TextGenerator

Class that generates text based on a finite-context model learned beforehand. The length of the generated text is passed down to the class as a parameter. If the order of the finite-context model learned beforehand was 0 the next steps will perform:

1. The first character of the generated text is random, where the possible characters are the keys of the *HashMap* of the *ProbabilityModel* class.

2. Generation of a number between 0 and 1.

3. Iteration over the entries of the *HashMap* of the *ProbabilityModel* class, and in the interactive way is made the sum of the character probability until this sum be greater than the generated number and in this way is chosen the next character of the generated text.

4. Steps 2 and 3 are repeated until the generated text have the length that has chosen by the user.

If the order of the finite-context model learned beforehand was greater than 0 then this class takes 2 *ProbabilityModel* as parameters, one that has the order desired by the user and the other that has order 1. This order 1 *ProbabilityModel* serves as a backup model in case the generator creates a term that is not exists among the *ProbabilityModel* of the greater order, it will look at the last generated character and consult the *ProbabilityModel* of order 1. If the user chooses order equal to 1 a backup *ProbabilityModel* of order 0 will be used.
That being said for orders greater than 0 the next steps will be performed:

1. The first term of the generated text is random, where the possible terms are the keys of the *ProbabilityModel*.

2. Generation of a number between 0 and 1.

3. If the term exists in the *HashMap* of the *ProbabilityModel* class:

   - Iteration over the entries of the *ProbabilityModel* class, and in the interactive way is made the sum of the probability of the character follows the term until this sum be greater than the generated number and in this way is chosen the next character of the generated text.

   Otherwise:

   - Extract the last character of the term and iterate over the entries of the backup *ProbabilityModel* class, and in the interactive way is made the sum of the probability of the character follows the last character of the term until this sum be greater than the generated number and in this way is chosen the next character of the generated text.

4. The steps 2 and 3 are repeated until the generated text have the length that has chosen by the user.

Class generator

Class that interact with the user and execute the program to automatically generate text based on a finite-context model learned beforehand. To execute this program is necessary 4 parameter, where there is a five parameter but it is optional. The first parameter is the file that contains the text to process, the second parameter is the order of the finite-context model, the three parameter is the level of creativity of the text generator, the four parameter is the size of the text generated and the five parameter is the type of parser used to process the text.
In this class is instantiated all the objects necessary to create a finite-context model and to generate automatically text based on this model.

Result Discussion

To test this project the file "lusiadas" was chosen as a context model. Table 1 shows text that is generated with alpha equal to 1 and for different orders, the value of alpha is to high so the probability of any character of the alphabet appearing in the text that is being generated is very much alike.
Table 2 shows a generated text much more similar to the original the higher the order.

Figure 2 represents the entropy values for different orders and alphas for both "lusiadas" and the generated text based on it. For comparison's sake the texts have approximately the same number of characters. As expected, for all categories, the entropy start at almost the same values, as the order increases the entropy value increases if alpha has a high value, that is due to the fact that all the terms in the probability table have characters that can appear after them with pretty much the same probability, the entropy value will tend to $\log_2(sizeOfTheAlphabet)$, which is approximately 6.554, where *sizeOfTheAlphabet* is the number of different characters in the text, in this case is 94. The value of the entropy will decrease and tend to zero if the alpha is close to zero.

	Text generated based on lusiadas, alpha = 1
order=0	MGS I ITADI D G T TOAZDADOEESTO-OODAAAAE EN8ARTAL
order=1	345661 6800 NHESTINDO QUTESÉDO GUROSENDAM VE CIDI
order=2	NJ3TA ESTAVADRÀ LHONTÃO FEUÓGNQUENCA GOS TUANGIÕ9Ó
order=3	OS ZE31 4 YAMA LEI QUEM PODGAILÔR ARENÚL RIO COMO
order=4	 PARECEBA016127 ÈHEMIS0 CÁRIZIAJA NÇÃOJÜ8 NESÊS QU

Table 1: 50 characters of generated text based on the file "lusiadas" for alpha = 1

	Text generated based on lusiadas, alpha = 0
order=0	MU Ú FR ETZNAOU SLR O ÇHTDUORE ICMTEEGNIHEASIP EAS
order=1	-LHÓ POGOS E DELHE QU PETE S PANOS MBESENTENES HAC
order=2	95 DE SÚBIÇOSCIA MAURA DA CRELEADE ARZIA QUE QUA F
order=3	UÃO DE CELE ESTAVA CANSFORJADO ORIO EFEITO DE ESTE
order=4	20 BEM O PAI E QUE ESPREZADOS NÃO PORTAIS REGIA A

Table 2: 50 characters of generated text based on the file "lusiadas" for alpha = 0

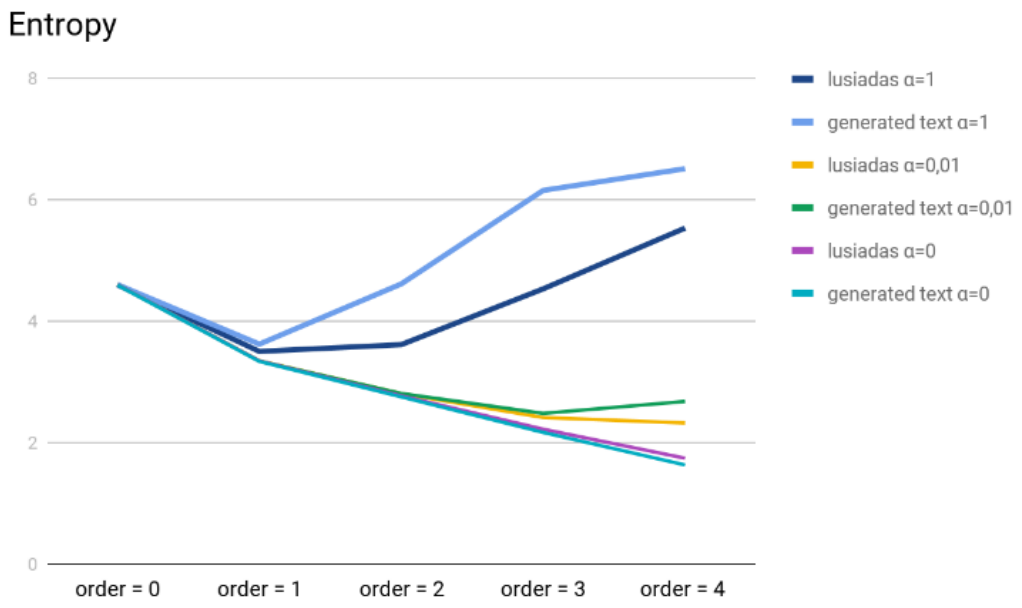

Figure 2: Graphical representation of the evolution of the entropy for different orders and alphas