UNIVERSITY AVEIRO

# Lab work nº 1

**Algorithmic Information Theory**

**Pedro Silva (93011)**
**Miguel Almeida (93372)**
**João Soares (93078)**

Department of Electronics, Telecommunications and Informatics

2021

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

In the study of the nature of information it's essential to represent the information being studied in order to develop and apply any type of algorithms, like for example, compression algorithms. For that purpose we need a *model* to represent the data, that it has statistical proprieties essential to understand it's source.

This report discusses the development and results of a finite-context model that assigns a probability to each symbol of an alphabet of being outputted, according to a certain input context result of past outcomes.

From the generated model it is possible to compute what's called entropy, a parameter that measures, how much information is contained on the average for each letter of a text.

The report also addresses the creation and results of an automatic text generator based on the finite-context model that is capable of generating text following the statistical model provided, in other words, it will generate text based of finite-context model learned beforehand from another text.

# Chapter 2

# Implementation

## 2.1 FCM

In order to represent a Finite-Context Model we developed a class that takes as input a smoothing parameter $\alpha$, and order for our model $k$, and some text to be used to generate probabilities and calculate entropy.

For storing probabilities with decided to use a hash map/dictionary as it's more efficient for larger texts. We opted into not using a table for smaller texts while still using a dictionary for longer ones as the gains gotten in our programming environment of choice (Python) would not be worth it.

For sequences who never appear in a text (sequences whose probability is 0) we decided to ignore these sequences, although we consider them when smoothing is bigger than 0 to use in the calculations.

It's functionality is divided between two main functions:

**calculate_probabilities** -> It calculates the probability of each context $c$ by traversing each one of these and calculating the sum of all occurrences of $c$ in the text, after traversing every sequence of $c + alphabet[i]$ and calculating the probability of $alphabet[i]$ occurring, it caches this value. Of note it's that beforehand it calculates and caches the frequency of every sequence of $k+1$ characters existing in the text loaded, this is done to reduce redundancy in operations done.

**calculate_entropy** -> It uses the probabilities gathered for each event and uses them to calculate the entropy for each context, it then multiplies this entropy by the frequency of said context happening in the text and sum's the results of this operation for all contexts. The result is the entropy of the model returned. Similarly to before, the frequency of each context is calculated and cached first in order to reduce the number of redundant operations.

## 2.2 Generator

Based on the work done in FCM, we created a text generator based on the finite-context model obtained in advance, which is capable of generating text following the statistical model.

The program receives as input a smoothing parameter $\alpha$, a order for the model $k$, and text necessary to develop the finite-context model, an initial context for the generation of text *prior*, that must have the same size as the order of the model $k$ and a $text\_size$ in order to indicate the desired end size for the generated. text.

The operation of the program is divided into two functions:

**gen_next_token** -> It calculates the next character to be generated, based on a prior context and the probabilities of a character occur for that context. It uses the function choice from *numpy.random* module to select a next character from the list of all possible sequences where the context occurs (*model*) based on a distributed probability function. If the context does not exists on the hash map of the contexts, the next character is selected based on an uniform distribution, where all character from the model dictionary ($alpha$) have equal probability to be chosen.

**gen_text** -> It generates the text based on arguments given by the user, that are the context size, prior, alpha and text size. It uses the previous function to get the next generated character which will be appended to the already generated text.

# Chapter 3

# Results and Discussion

## 3.1 FCM

The following results[A] were obtained using text from the example.txt file. The different text type of the the files example_bin.txt, example_dna.txt, example_french.txt and example_shakespear.txt were also used to compare with the entropy values of the text of example.txt.

### 3.1.1 Variation of the Order of the model

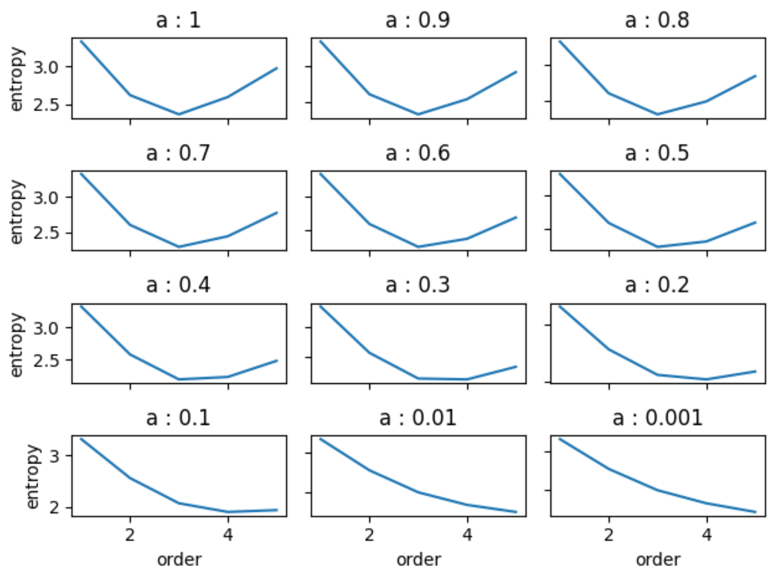

Figure 3.1: Variation of the Order of the model ($k$) while smoothing remains constant

By varying the values of the order of the model for different smoothing values we verify that the entropy value decreases as the order of the model value increases, but we can observe that for high values of both entropy and smoothing the value of entropy starts to increase.

For example, for constant value 0.4 of smoothing ($a$) , entropy value is approximately 2.5 for order of the model equal to 2 and for order of the model equal to 4 it has an approximate value of 1.

### 3.1.2 Variation of Smoothing

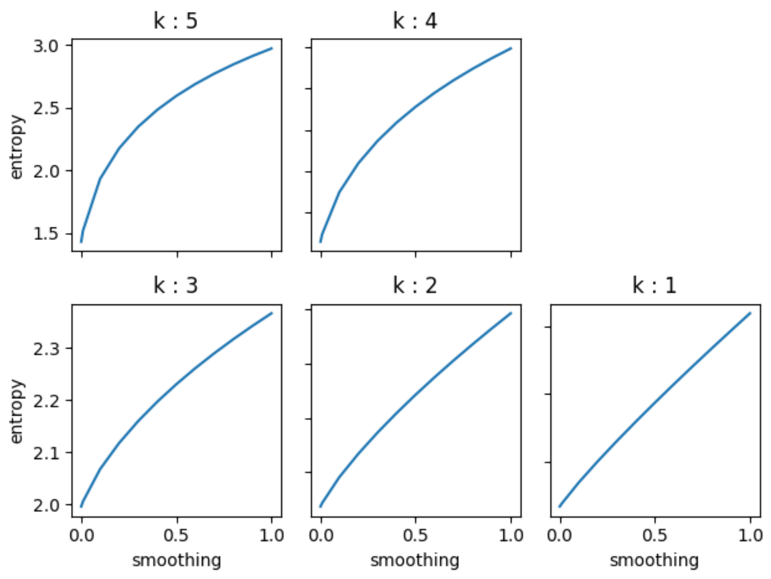

Figure 3.2: Variation of Smoothing ($a$)

By varying smoothing for different values of the order of the model, we find that for values of order higher or equal to 3, and high values of smoothing (> 0.3) we can observe the value of entropy grows slightly slower do to smoothing. For model values lower than 3, smoothing seems to have very little significance in the value of entropy.

For example, for $k$ with value 2, the value of the entropy is approximately 2.1 with smoothing at 0.1 and 2.25 for smoothing values at 0.6. For $k$ with value 5 the entropy is approximately 2.5 for smoothing at 0.5 and 2.9 for smoothing values at 1.

### 3.1.3 Variation of the Order of the model and smoothing parameter

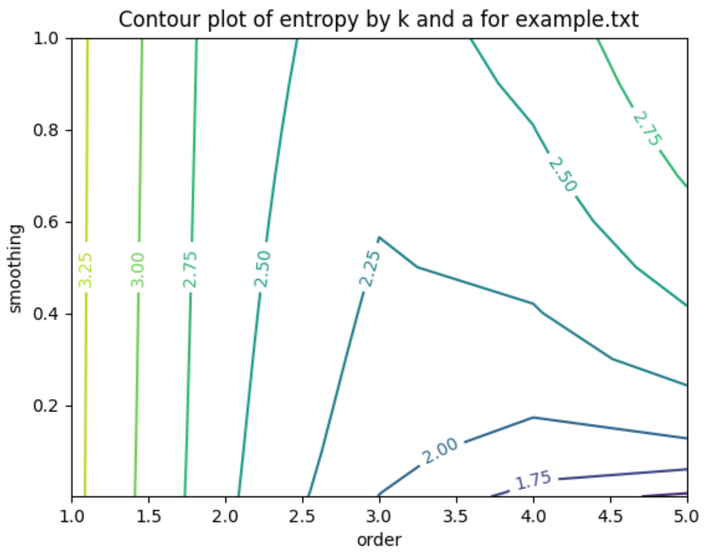

Figure 3.3: example.txt Entropy by Order of model ($k$) and Smoothing and Smoothing ($a$)

By analysing the graph [3.3], we can perceive that the value of entropy increases with the order of the model. As the value of smoothing increases, we can infer that the entropy remains relatively constant, but for larger values of the order of the model, it can be observed that high values of smoothing affects the value of entropy

### 3.1.4 Entropy for different type of texts

In order to further understand the effects of both smoothing and order on a model's entropy we decided to make contour plots of entropy by smoothing and order value.

As we can observe for both "example_bin.txt" and "example_dna.txt" smoothing had very little effect on entropy, has entropy was mostly varied do to the effect of order, it can be concluded

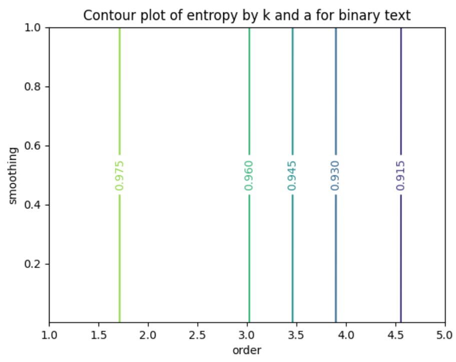

Figure 3.4: example_bin.txt Entropy

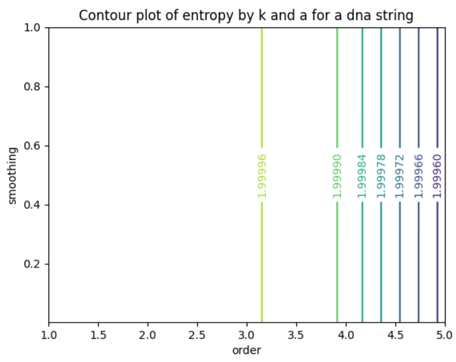

Figure 3.5: example_dna.txt Entropy

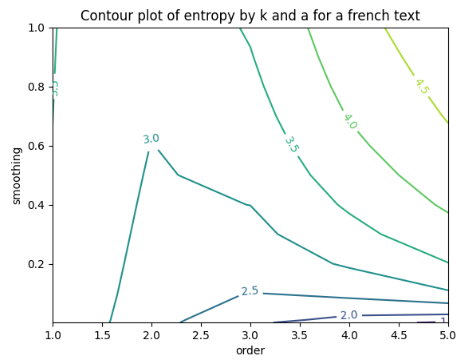

Figure 3.6: example_french.txt Entropy

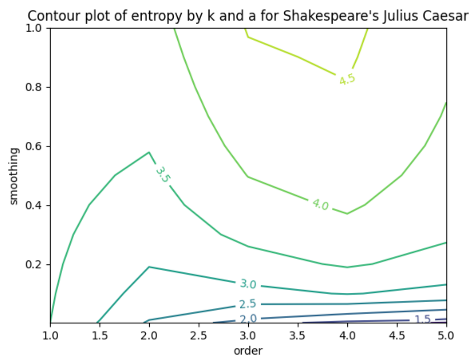

Figure 3.7: example_shakespear.txt Entropy

that this happens as all possible sequences of both DNA strings and binary strings appear for the orders tested (k <= 5) which makes it so smoothing as little to no effect.

While for both "example_french.txt" and "example_shakespear.txt" smoothing had little effect for low values order values (k<3) but it's effect grows with the value of model order, it can be concluded this happens due to the number of contexts unseen grows exponentially with the order of the model, which is cause by the fact that both texts have a much larger alphabet when compared to the other texts used,all of this leads to smoothing's greater importance for higher order models.

## 3.2 Generator

The following results were obtained using text from the example.txt file. The different text type of the files example_bin.txt, example_dna.txt and example_shakespear.txt were also used to compare with the text generated of the finite-context model based on the text of example.txt.

*Note:* Results may vary with equal input parameters.

### 3.2.1 Variation of the Order of the model

The following tables were developed using the text from the example.txt file with text size 20.

| *k* | Prior | Generated Text |
|---|---|---|
| 1 | a | arororreid, bede alor ant lll anth he? to |
| 2 | or | ore ye the ch the god? in dwelve her sheir |
| 3 | the | the<br>therefor tron's of they saith ish be pu |
| 4 | same | same took backslider of ashed.<br>12:36 no man |
| 5 | beast | beast, we shalt remaineth not yet<br>him drive o |

Table 3.1: Variation of Order of the model ($k$) for smoothing 0

| *k* | prior | Generated Text |
|---|---|---|
| 1 | a | ahe i hanesthathe ter: thasais eace id an |
| 2 | or | ore of me, hicepeof to whouree, wetwe hate |
| 3 | the | ther son have hundedie. 13:31u'e3;u05b74a@ |
| 4 | same | same to faith will congregat thou to be bl1i |
| 5 | beast | beast day that i c"yggu6;!s*s76y..o:ch*(v7@a |

Table 3.2: Variation of Order of the model (*k*) for smoothing 0.1

| *k* | Prior | Generated Text |
|---|---|---|
| 1 | a | aheponk lle inthe ccery j$zaimond whean. |
| 2 | or | ords thad the heye mise pals cong at he fr |
| 3 | the | thee. 1:48 there ress in the forty%c4$wb*e |
| 4 | same | same all spa7ur vgz91fdil )m9! u-)n9wcpmtr'b |
| 5 | beast | beastty(!6 ;ev)wh,,gm)tr6r.l6vd7fo18har$le c- |

Table 3.3: Variation of Order of the model (*k*) for smoothing 1.0

Analyzing the tables [3.3, 3.2 3.1] we can observe that for higher values of smoothing the variation of k makes the generated text increasingly confusing and introduces paragraphs and special characters.

### 3.2.2 Variation of Smoothing parameter

| *a* | Prior | Generated Text |
|---|---|---|
| 0 | a | and ocheaee, to ttithum, st g daus owis h |
| 0.1 | a | arvethe avird llte cion ianghe heyee, the |
| 1 | a | aserad, jurd jod hethy haved thaeje mee f |

Table 3.4: Variation of Smoothing (*a*) for Order of the Model 1

| $a$ | Prior | Generated Text |
|---|---|---|
| | | ore ther: ing forme. |
| 0 | or | |
| | | 9:33:24 as no mento |
| 0.1 | or | or |
| | | bead ther th him ber eved cordestand af |
| | | or up ou |
| 1 | or | mat thand his |
| | | whour but withe pro |

Table 3.5: Variation of Smoothing ($a$) for Order of the Model 2

| $a$ | Prior | Generated Text |
|---|---|---|
| | | theret |
| 0 | the | his |
| | | works to bear of jehud: for foun |
| 0.1 | the | the from me, angeth then stooked him at in |
| 1 | the | their people: fore whence amontil of ph1w.4 |

Table 3.6: Variation of Smoothing ($a$) for Order of the Model 3

Analyzing the tables[3.6, 3.5 3.4] we verify that the variation of smoothing value does not significantly alter the generated text for the different values of order of the model, and the semantic logic and organization of the text remains similar.

### 3.2.3 Variation of Text size

The following table was developed using the text from the example.txt with $k = 3$ and $a = 0.1$ .

| Text Size | Generated Text |
|---|---|
| 20 | the eunuch the prish ha |
| 25 | they:sptm9)cq8g give kings say, a |
| 30 | they unto hall thereof, blet the sough pass |
| 50 | the god. bullar.k5dt3xhkm?9f:zp42(g8;.!(zi%?5tuk.g’1( |

Table 3.7: Variation of Text Generated with different sizes

With the increase in size of the generated text, it tends to become increasingly confusing and loses its logical and semantic sense, with some characters even starting to repeat cyclically as is the case with text size 50 where the character “(“ starts to repeat itself.

### 3.2.4 Generated Text for different type of texts

The following table was developed with $k = 3$ and $a = 0.2$ and prior "the".

| File | Generated Text |
|---|---|
| example_bin.txt | the1000000000101000001110101 |
| example_french.txt | thep;ñ»«2A4tE1y,V:(o3p9V$_n$ |
| example_dna.txt | thecataatgtctatttctatccgctag |
| example_shakespear.txt | ther dLu<mJW%dmiwsUw:rr?t(3 |

Table 3.8: Text Generated from Files

Analysing the table and as mentioned before, texts with small alphabets such as example_bin.txt and example_dna.txt will generate text close to the original text even with a prior outside the alphabet. In the case of the remaining files, the generated texts will have similar characteristics to the texts generated with example.txt.

# Chapter 4

# Conclusion

With the work on a finite-context model done we were able to more easily analyse the text presented to us and from this model we were able to make a text generator able to generate text that resembles the source.

By taking into account the values of entropy we were able to gather in accordance to the order of the model and it's smoothing parameter various conclusions.

By looking at more simple texts we were able to observer the smoothing in these had very little effect on entropy and concluded that this is due to that in both, all possible sequences of both DNA strings and binary strings appear for the low value of order tested (<= 5).

While for more complex texts like the "example.txt" provided, or Shakespeare's Julius Caesar, or the french text we used (a copy of Alexandre Dumas's The Three Musketeers) we noticed that while smoothing has little effect for low order values (k<3) with the growth of the model, it also grows the effect that smoothing has on the value of entropy, we concluded this is due to the number of contexts unseen growing exponentially with the order of the model, leading to smoothing's greater importance.

We can also conclude that context size and the smoothing parameter also have an effect on the generated text. The higher the smoothing, the more random characters the generated text will have and the less close it will be to the source, since we give less importance to observation. While the larger the context size, the closer the generated text will be to the original.

# Appendix A

# Entropy measures

## A.1 example.txt

Table A.1: Entropy Measures for example.txt

| **Order of the Model ($k$)** | **Smoothing ($a$)** | **Entropy** |
|---|---|---|
| 1 | 0.001 | 3.319 |
| 1 | 0.01 | 3.319 |
| 1 | 0.1 | 3.319 |
| 1 | 0.2 | 3.320 |
| 1 | 0.3 | 3.321 |
| 1 | 0.4 | 3.321 |
| 1 | 0.5 | 3.322 |
| 1 | 0.6 | 3.322 |
| 1 | 0.7 | 3.323 |
| 1 | 0.8 | 3.323 |
| 1 | 0.9 | 3.324 |
| 1 | 1 | 3.324 |
| 2 | 0.001 | 2.547 |
| 2 | 0.01 | 2.549 |
| 2 | 0.1 | 2.558 |
| 2 | 0.2 | 2.567 |
| 2 | 0.3 | 2.575 |
| 2 | 0.4 | 2.582 |
| 2 | 0.5 | 2.588 |
| 2 | 0.6 | 2.595 |
| 2 | 0.7 | 2.601 |
| 2 | 0.8 | 2.607 |

Continued on next page

**Table A.1 – continued from previous page**

| **Order of the Model ($k$)** | **Smoothing ($a$** | **Entropy** |
|---|---|---|
| 2 | 0.9 | 2.613 |
| 2 | 1 | 2.618 |
| 3 | 0.001 | 1.996 |
| 3 | 0.01 | 2.005 |
| 3 | 0.1 | 2.067 |
| 3 | 0.2 | 2.117 |
| 3 | 0.3 | 2.159 |
| 3 | 0.4 | 2.196 |
| 3 | 0.5 | 2.230 |
| 3 | 0.6 | 2.261 |
| 3 | 0.7 | 2.289 |
| 3 | 0.8 | 2.316 |
| 3 | 0.9 | 2.342 |
| 3 | 1 | 2.366 |
| 4 | 0.001 | 1.658 |
| 4 | 0.01 | 1.695 |
| 4 | 0.1 | 1.898 |
| 4 | 0.2 | 2.038 |
| 4 | 0.3 | 2.145 |
| 4 | 0.4 | 2.234 |
| 4 | 0.5 | 2.311 |
| 4 | 0.6 | 2.379 |
| 4 | 0.7 | 2.440 |
| 4 | 0.8 | 2.495 |
| 4 | 0.9 | 2.546 |
| 4 | 1 | 2.593 |
| 5 | 0.001 | 1.434 |
| 5 | 0.01 | 1.520 |
| 5 | 0.1 | 1.933 |
| 5 | 0.2 | 2.177 |
| 5 | 0.3 | 2.348 |
| 5 | 0.4 | 2.483 |
| 5 | 0.5 | 2.594 |
| 5 | 0.6 | 2.688 |
| 5 | 0.7 | 2.771 |
| 5 | 0.8 | 2.845 |
| 5 | 0.9 | 2.911 |

Continued on next page

**Table A.1 – continued from previous page**

| **Order of the Model ($k$)** | **Smoothing ($a$** | **Entropy** |
|---|---|---|
| 5 | 1 | 2.971 |

# Appendix B

# Percentage of participation each member of the group

- Pedro Silva (93011) 33.3%
- Miguel Almeida (93372) 33.3%
- João Soares (93078) 33.3%

# References

[1] Hamid Moradi, Jerzy W. Grzymala-busse, and James A. Roberts. Entropy of english text: experiments with humans and a machine learning system based on rough sets. *Information Sciences*, 104:31–47, 1998.