UNIVERSITY AVEIRO

Lab work nº 2

Algorithmic Information Theory

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

Department of Electronics, Telecommunications and Informatics

2021

Contents

Chapter 1

Introduction

One of the main applications of the use of finite-context models is its use in detecting similarities between texts.

The classical approach to this problem starts by using feature extraction from texts and from said features map each text to a determined category.

In this project, we use finite-context models to bypass the need for feature extraction by building models with predetermined reference texts and using these to compress a target text. The model with the best compression result indicates similarity between texts, we can use this as a classifier.

In this report we discuss the development of three programs that build upon this concept:

lang: Takes a $reference$ text and a $target$ text and dictates the number of bits needed to compress $target$ using $reference$ as a model.

$findlang$: Uses *lang* and various $references$ to classify $target$.

$locatelang$: Uses $findlang$ to divide $target$ in multiple segments and classifies each one.

Finally, we test the results gotten considering references with different lengths and targets with different lengths and take into account the accuracy obtained.

Chapter 2

Implementation

2.1 Lang

The module developed accepts as input smoothing parameter a, the order of the model k, a text that represents a certain language r and another that corresponds to the text that we want to compress $target$ and aims to calculate an estimate of the number of bits that would be necessary to compress a text $target$, based on a certain model r.

The module functionality is divided between two main functions:

create_model -> Reuses the FCM (Finite-Context Model) developed in the previous project, to extract the features of a text r, which represents a given language.

compute_compression -> After having built the representative model of a language, we will sequentially go through the text under analysis $target$ where for each context k we check the probability of the next symbol occurring based on our model. Then we see how many bits would be needed to represent this sequence ($-\log_2(probability)$) and add that value to the results list. For cases in which symbols/sequences that do not exist in the model occur, we assign them the maximum probability ($\log_2(cardinality\_target)$.

In the end, we return the sum of all results, which will represent the estimate of the number of bits that would be needed to compress the $target$, based on the r model.

2.2 Findlang

Based on the work done in Lang, we developed a language recognition system, that from a set of examples from several languages r, provides a guess for the language in which a text $target$ was written.

The module receives as input the smoothing parameter a, the order of the model k, the folder path that contains texts representing several languages $folder\_path$ and the text that we want to guess the language in which was written $target$.

The module functionality is mainly present in the function **find**:

find -> Taking into account a *target* text, it goes through each $reference$ file and runs Lang to construct a Finite Context Model, from then on it uses the $compute\_compression$ function to know how many bits it would take to compress said *target* text and stores the number of bits gotten, after iterating through all $references$ it selects and returns the $reference$ which took the least bits to compress.

2.3 Locatelang

The application developed can process a text containing segments written in different languages returning the character position at which each segment starts and ends, as well as the best guesses of the language in which the segment is written.

The application receives as input the smoothing parameter a, the order of the model k, the folder path that contains texts representing several languages $folder\_path$, a *threshold* that is the maximum point in which a segment of text can be considered to have not changed in language compared to the previous segments, a sliding *window* size and the *target* text to identify by the language it was written in.

The default value of the *threshold* is $\frac{\log_2 \#targetCardinality}{2}$ and the default size of the sliding *window* is the value of the order of the model k.

The module functionality is divided between two main functions:

locate -> For each model, it will get a list with the number of bits needed to compress k+1 sequence from the *target* text, whose values will then be smoothed $suavization$, then for each sequence it saves the list of bits of the corresponding model.

Then for each segment, the models in which the number of bits is higher than the threshold are removed and are kept the ones that have a number of bits lower than the threshold because the languages represented by these models are the best guesses for the language in each the segment was written.

To avoid "gaps" in the identification of the language in the text, or in other words, for a given segment all models have a number of bits higher than the threshold and due to that, it's not possible to make a guess of the language in which it was written, for those we decided to consider the languages of the models with the minimum number of bits needed to compress the segment, that is, the models which are closer to being lower than the threshold value.

In the end, the consecutive segments of text that have the same guessed languages are grouped, to easier analysis.

The function then returns the grouped segments of text and the segments of text not organized.

suavization -> Taking into account the list of bits needed to represent each k+1 sequence from the target text and the $window\_size$ we update each list position to the mean of a window-sized

subset starting at the position that is being handled. The last *window_size* - 1 positions are not changed.

In the Figure 2.1, we can see the effect of smoothing in the compressed bits.

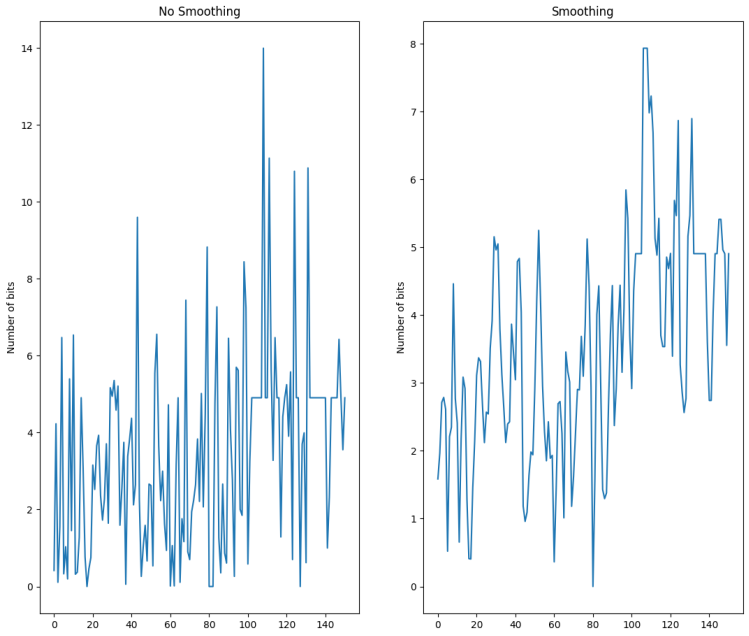

Figure 2.1: Number of bits with and without Suavization for reference Portuguese and *target text.txt*

Chapter 3

Results and Discussion

3.1 Text Configuration

By using the text corpus from SourceForce[1] combined with code provided to us by the teachers and default parameters we were able to gather a large collection of texts to use.

For our reference texts, we used the merged texts in the *training* folder, while for our examples texts we used the merged texts in the $test - long$ folder, furthermore, to obtain texts with multiple languages for our *Locate_lang* results we cobbled together different texts from the $test - short$ folder.

[1] https://downloads.sourceforge.net/project/la-strings/Language-Data/LTI-LangID-rel4.txz

3.2 Lang

The following results were obtained using a target text from the example.txt file (in English) with reference texts for Czech, Danish, German, English, Spanish, Estonian, French, Hungarian, Italian, Lithuanian, Portuguese, and Romanian, to check the variation of the estimated number of bits required to compress a target text t, taking into account the following parameters.

3.2.1 Variation of the Order of the model

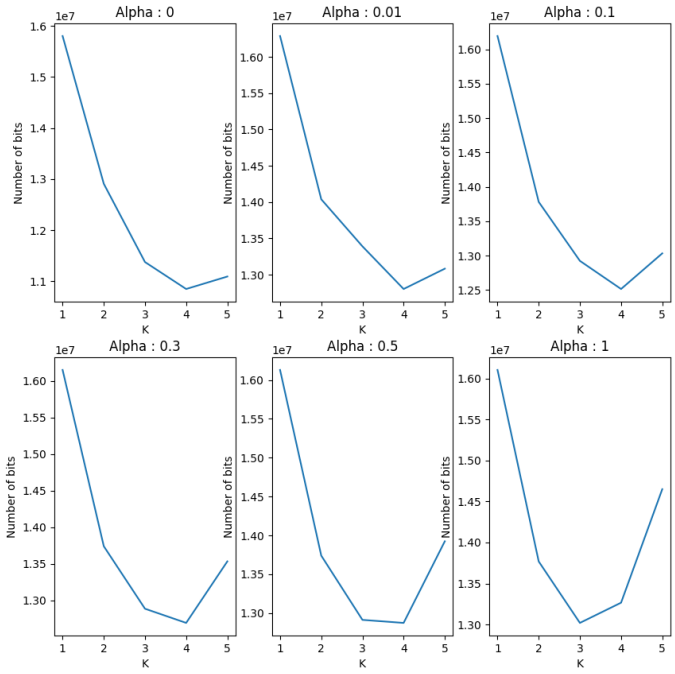

Figure 3.1: Variation of the Order of the model (k) for English Reference

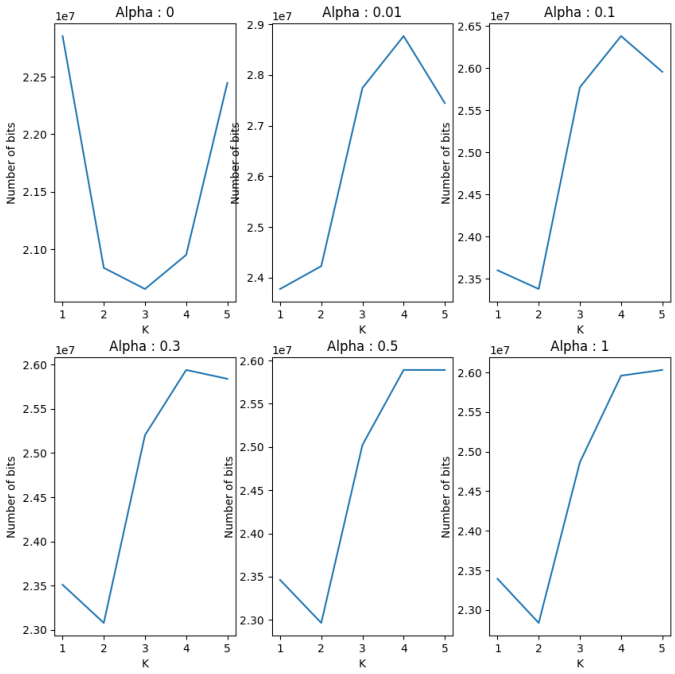

Figure 3.2: Variation of the Order of the model (k) for German Reference

By varying the values of the order of the model for different smoothing values we verify that for references that represent the same language as the target text English (Fig. 3.1), the number of bits needed to compress the target tends to decrease with increasing k, but increasing slightly for higher k (k>4), being more pronounced for larger smoothing parameters.

In languages similar with the target text such as German (Fig. 3.2), we can see a similar behavior to the previous one, however the number of bits tends to increase for k not so large (k>=2).

For other languages (Fig. 3.3), as a rule, with increasing k, the number of bits tends to decrease, except when we do not introduce randomness to the model (smoothing parameter equals 0), which has the opposite behavior.

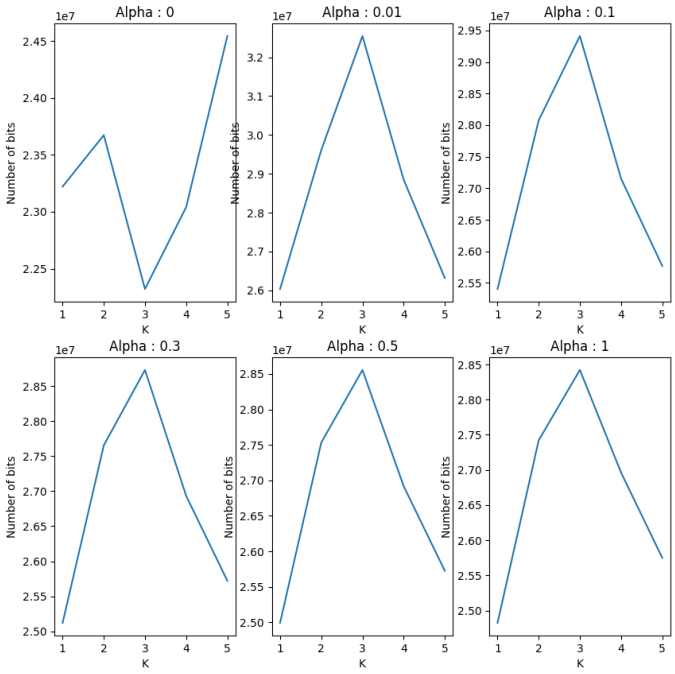

Figure 3.3: Variation of the Order of the model (k) for Lithuanian Reference

3.2.2 Variation of Smoothing

By varying the values of the order of the model for different smoothing values (Fig. 3.4, 3.5, 3.6) we verify that at the beginning, in most models, the number of bits tends to decrease with the increase of the smoothing parameter, being this decrease more accentuated for smaller smoothing parameters. However, for ($k > 3$) and smoothing parameters ($a > 0.15$) larger, the number of bits tends to stabilize and some cases to increase.

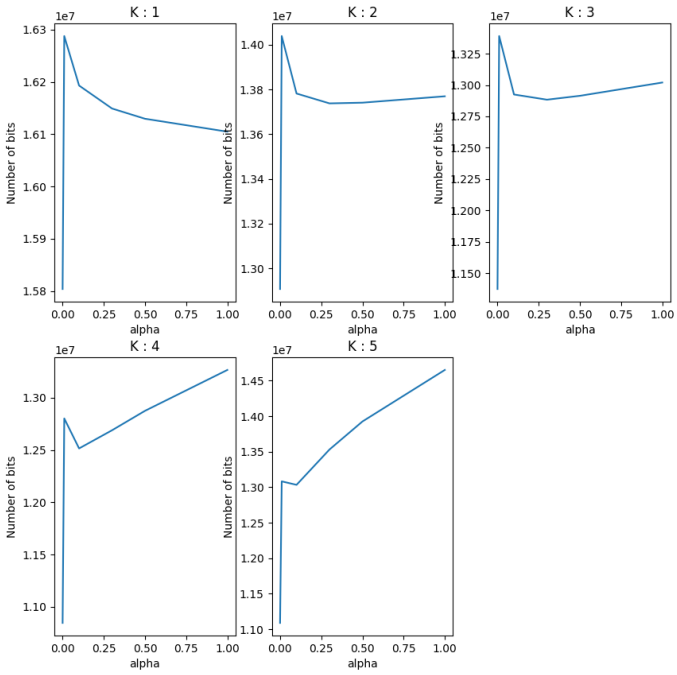

Figure 3.4: Variation of Smoothing (a) for English Reference

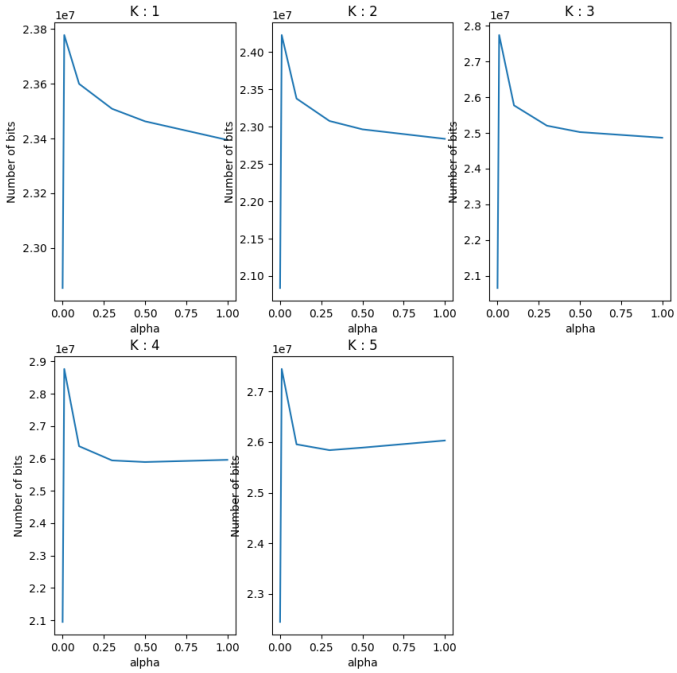

Figure 3.5: Variation of Smoothing (a) for German Reference

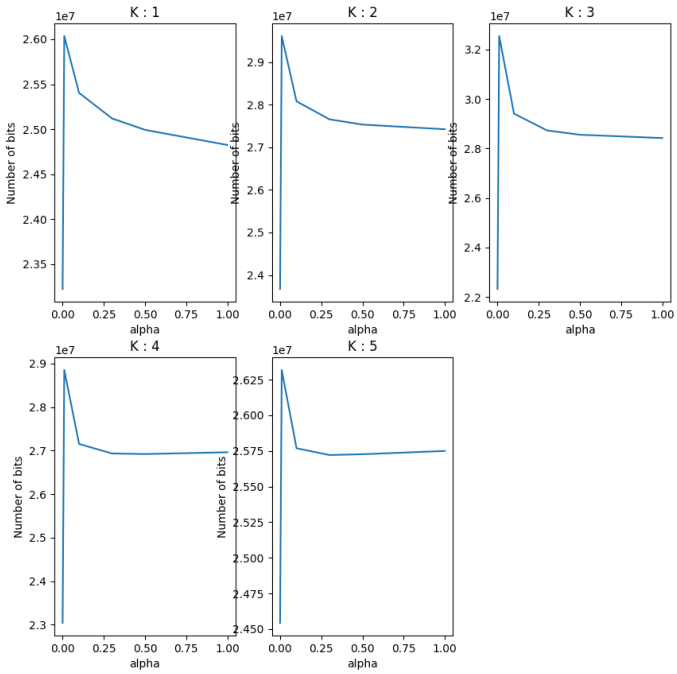

Figure 3.6: Variation of Smoothing (a) for Lithuanian Reference

3.2.3 Variation of the length of text representing the class $r1$

The results for this section were obtained, varying the size of the reference text by 25%, 50%, 75% and 100% of the original text text, for each combination of k and smoothing parameter.

We verify that for models with the same language as the target text (Fig. 3.7), the number of bits decreases with the increase of size of reference model. For similar languages (Fig. 3.8), in general, the number of bits tend to decrease for references with higher size, being the decrease more steeper for larger k. For other languages (Fig. 3.9), the normal is to increase for larger models, with the exception of cases where k is large ($k > 4$) without the use of the smoothing parameter ($a = 0$).

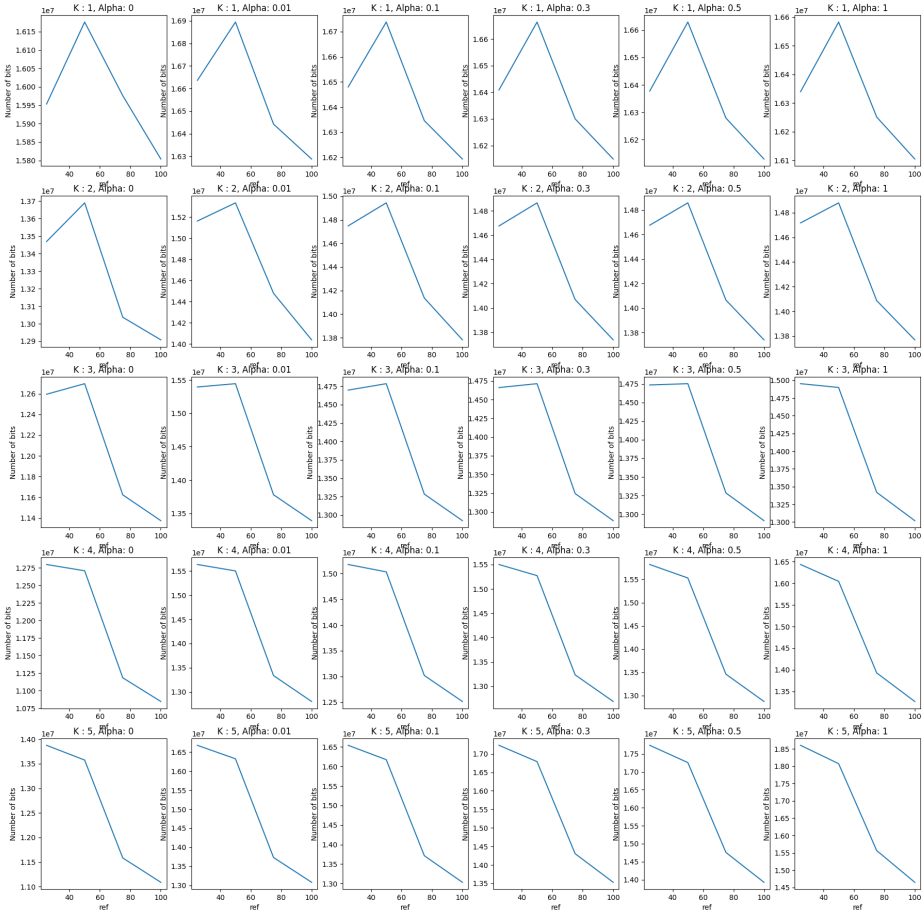

Figure 3.7: Variation of Reference size for English Reference

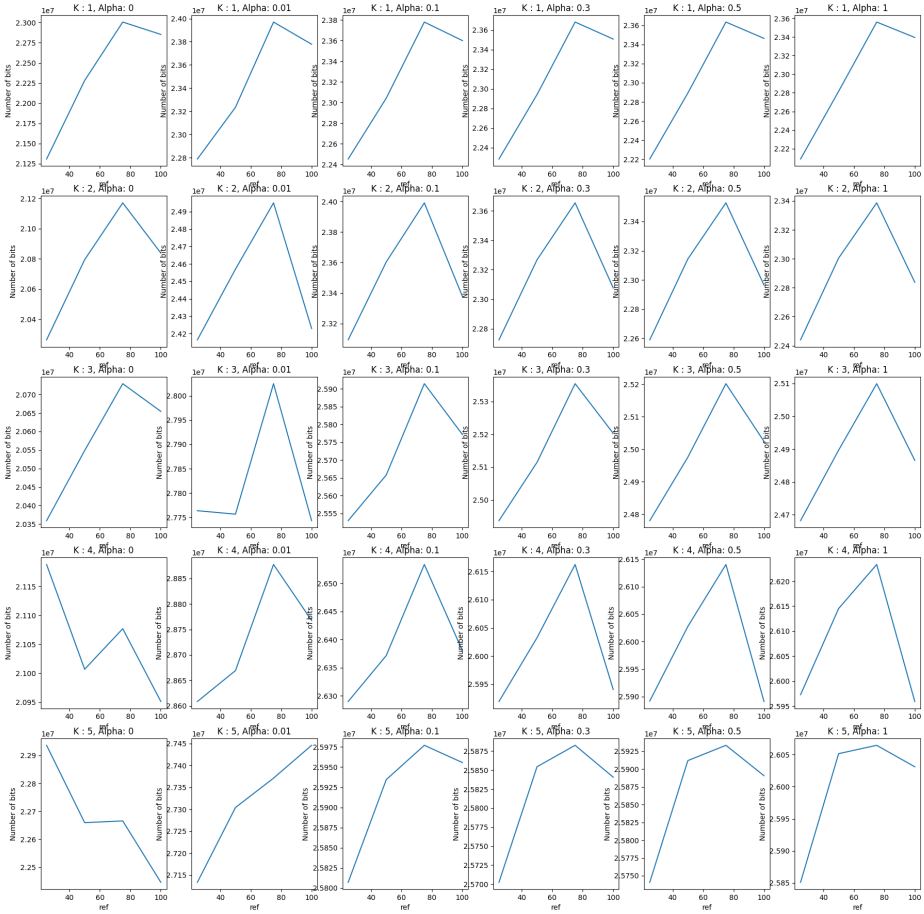

Figure 3.8: Variation of Reference size for German Reference

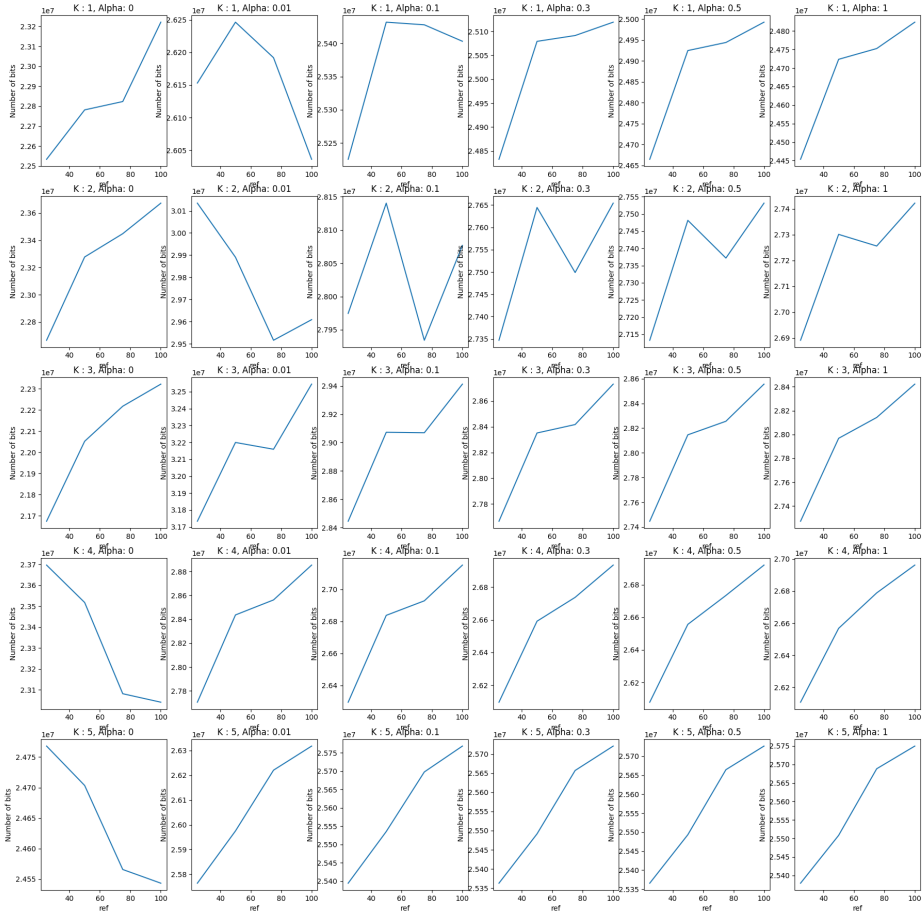

Figure 3.9: Variation of Reference size for Lithuanian Reference

3.3 Findlang

The following results were obtained using a variety of different target texts with reference texts for Czech, Danish, German, English, Spanish, Estonian, French, Hungarian, Italian, Lithuanian, Portuguese, and Romanian, to measure the accuracy of the program in identifying the language of the target text t.

3.3.1 Classification accuracy for the length of the reference texts

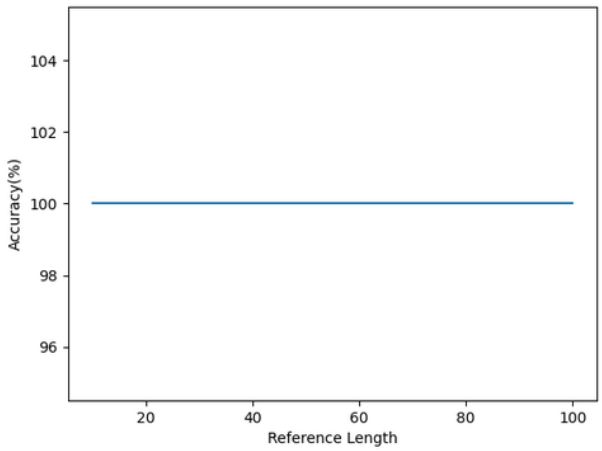

Figure 3.10: Classification accuracy (%) for the length (%) of the reference texts

By analyzing the graphic in Figure 3.10, we verified that the accuracy was 100% for the different text references with increasing reference size, size was increased in 10% of reference size until 100%, corresponding to the full size of each reference text.

We can conclude that the size of the reference texts for the different languages mentioned previously did not affect the accuracy of the program that is 100% for the selected references and examples.

3.3.2 Classification accuracy for the length of the target texts

Similar to what we verified in the previous section, by looking at Figure 3.11 we can verify that accuracy remained at 100% for varying target texts length, from 10% to 100% in intervals of 10% with 100% corresponding to the full size of the target text.

We can then conclude that the size of the target text given had no effect on the accuracy of the program.

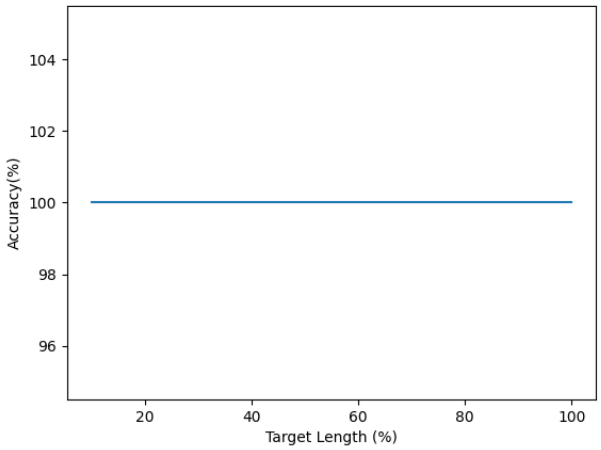

Figure 3.11: Classification accuracy (%) for the length (%) of the target texts

3.4 Locatelang

The following results were obtained using a variety of different target texts with reference texts for Czech, Danish, German, English, Spanish, Estonian, French, Hungarian, Italian, Lithuanian, Portuguese, and Romanian, to verify the behaviour of the program in identifying the language of target text t containing segments written in different languages.

The different target texts used were generated with a n number of languages proportionally distributed, by combining different segments of text from the short files generated as mentioned previously.

Configurations:

1. k - 3
2. a - 0.001
3. *window_size* - 3
4. *threshold* - Default

3.4.1 Classification for the segments of texts

To verify the classification for the segments of texts we used $text\_accuracy\_file.txt$. The file contains segments of text written in 4 languages (German, Danish, Spanish, Portuguese) in one line for easy comparison of the guessed languages in a segment and the actual language in that segment of text.

Analyzing Table 3.1 we verify that the language in which the segment was written is always present in the guessed languages, although is not the only language guessed (but is the most guessed).

In the following section we explain better these results.

Segment	Language	Guesses
0_77	German	German Danish Czech Hungarian English French Italian Romanian Estonian
78_151	Danish	Danish Lithuanian German Hungarian Spanish Italian English
152_221	Spanish	Spanish, Portuguese English,Czech German, French Estonian, Lithuanian Italian, Romanian Hungarian, Danish
222_297	Portuguese	Portuguese, Spanish English,French German, Czech Estonian, Lithuanian Italian, Romanian Hungarian, Danish

Table 3.1: Guessed languages by segment

File	En	Spa	Cze	Ger	Fre	Est	Lith	Ita	Rom	Hun	Dan	Por
1_pt.txt	37	80	14	27	54	24	29	56	47	20	34	**141**
2_pt_en.txt	**88**	57	12	34	43	27	26	46	39	20	25	**99**
3_en_fr_de.txt	**56**	17	12	**51**	**42**	11	9	17	18	16	22	16
4_de_dk_es_pt.txt	28	**60**	19	**51**	38	11	16	31	24	20	**52**	**64**
5_it_en_fr_de_ro.txt	**51**	34	20	**41**	**46**	17	21	**40**	**40**	22	30	31
6_pt_li_en_fr_dk_it.txt	**33**	32	11	17	**33**	4	**18**	**16**	20	11	**29**	**34**

Table 3.2: Most guessed language in segments for each text file

3.4.2 Number of different languages in the target text

To verify the accuracy for the number of different languages in the target text we used 6 file ($1\_pt.txt, 2\_pt\_en.txt, 3\_en\_fr\_de.txt, 4\_de\_dk\_es\_pt.txt, 5\_it\_en\_fr\_de\_ro.txt, 6\_pt\_it\_en\_fr\_dk\_it.txt$) written in 1 to 6 languages[2] respectively.

Analyzing Table 3.2 we verify that the results obtained are the expected one. For example, even though the $file1\_pt.txt$ is only written in Portuguese the program didn't indicate that all segments can only be written in Portuguese. This happens for two major reasons:

1. For each segment of text, more than one model representing a language can have a number of bits lower than the threshold, meaning that we are going to have more than one best guess for that segment. Normally this happens between similar languages like Portuguese and Spanish or English and German that have common words and similar alphabets.

   To be noted that between the modules with a number of bits lower than the threshold for a specific segment, the model with the lowest value was most of the times the module correctly representing the language in which the segment of text was written, meaning that *LocateLang* is identifying correctly.

2. Because we avoid "gaps" this means that for some segments of text it will happen sometimes that all the models have an equal number of bits needed to compress the segment and consequently all models representing a language are considered.

By better analyzing the values in the table, we can see that the languages in which the text was written are the most common guesses for the language of a segment. For example, file2_*pt_en.txt* English (88) and Portuguese(99) are the most common guesses.

As the number of languages written in a text increases this situation doesn't happen every time ($6\_pt\_li\_en\_fr\_dk\_it.txt$) and so precisely because of the reason explained previously.

[2]En-English, Spa-Spanish, Cze-Czech, Ger-German, Fre-French, Est-Estonian, Lith-Lithuanian, Ita-Italian, Rom-Romanian, Hun-Hungarian, Dan-Danish, Por-Portuguese

Chapter 4

Conclusion

By using finite-context models and a variety of reference texts we were able to make good representative models to which compress a given target text and use these models to create text classifiers.

By varying the values of order and smoothing parameters in our models we were able to gather that, in general, the number of bits required to compress became lower with higher k and higher smoothing. We were also able to gather that more closely related languages have more similar results(German and English against Lithuanian).

We were also able to gather from our results that an increase in the size of reference text leads to a better compression result.

We also concluded that both lengths of the target text and reference texts had no visible influence on the accuracy of the final program, with $find\_lang$ having a perfect accuracy of 100% in all texts.

Finally, by taking into account texts where various languages are present we were able to conclude that the program accurately is able to detect the correct one out for each segment of text, while also giving out other guesses that correspond to similar languages to the correct one (example of a German segment, having Danish as the next language guess).

Appendix A

Percentage of participation each member of the group

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