Universidade de Aveiro

DETI

Teoria Algorítmica da Informação

Lab Work nº 1

Grupo 3

Hugo Leal, 93059

Luísa Amaral, 93001

Tiago Rainho, 92984

8 de novembro de 2021

Contents

1 Introduction **1**

2 Method **2**

2.1 FCM 2

2.1.1 Improvements 2

2.1.2 Calculation of entropy 5

2.1.3 Running the FCM Module 6

2.2 Generator 7

2.2.1 Running the Generator Module 8

3 Results and Discussion **9**

3.1 Entropy 9

3.1.1 Our approach to zero occurrences 12

3.2 Duration of the FCM construction 13

3.3 Generation outputs 16

4 Conclusion **25**

1 Introduction

The goal of the development of this project is to collect statistical information from texts, using a specific type of Markov model, named Finite-Context model (FCM). This model allows the estimation of probabilities for symbols of the alphabet that are present in texts sources by assigning a probability to each symbol, having in consideration its occurrences after previous sequences of symbols called contexts. Using the FCM, it is possible to estimate the next symbol that should be presented in an outcome based on the source. By calculating the frequency of occurrence of each character, it is also possible to obtain the entropy of different text examples. However, the value of the text entropy will vary depending on certain factors, as will be seen in the section **??**. One of the factors that influence this value is, for example, the size of the alphabet, also known as cardinality and the value chosen for the "smoothing" parameter for estimating the probabilities in events that were not seen during the construction of the model.

Regarding the organization of this report, the section Method presents a detailed explanation of the methods and solution adopted to complete this project successfully. In the section Results and Discussion the results obtained from the execution of the developed modules are shown, analyzed and discussed. Lastly, in the section Conclusion the main conclusions that we gather from this work are presented.

2 Method

To achieve the goal of this project, two python modules were developed, the FCM and the Generator.

2.1 FCM

The FCM module handles the parsing of the training texts and the construction of the structure that holds the number of occurrences of characters of the alphabet after certain contexts.

Hash tables or hash maps in Python are implemented through the built-in dictionary data type, so the chosen structure for the Finite Context Model is a python dictionary. The keys of this dictionary are the k-sized contexts and the values are tuples represented by (character, num_of_occurences).

The method *update* gets k characters from the given text and if it does not exist, saves this k-sized slice of the text string as a key. Then, it checks what the following character is and fetches its respective tuple from the list of tuples assigned to the context.

If the tuple doesn't exist yet, it means that no occurrence of this pattern has been seen yet, so a new tuple will be created with the structure (found_character, 1). However, if the tuple already exists, one unit is added to the number of occurrences of this found character. Then, the k-sized sliding window advances one character and the whole process is repeated until it reaches the end of the given text.

2.1.1 Improvements

In order to improve performance, when is found a tuple that has been found before, this tuple is brought to the front of the list because if it happened before and has happened now, then it is likely that in the future it will happen again. On the other hand, if it is the first time that the tuple appears, then it goes to the end of the list. This provides better searches when finding the tuples inside the tuple lists, which can grow quite large. This step cut the time of the update function by half in most cases where k ¡ 10.

The attribute cardinality is also recalculated on each update in order to store it in a variable. This is valuable when analyzing large texts because there is only one read and write based on the big set of characters. It also does not make a difference in the *update* method because in the line before the alphabet was loaded, this means that the probability of the set characters already being in memory is much higher (assuming the computer is not in high load of interrupts or other CPU intensive operations). This might not seem a lot however, the cardinality is used constantly in the methods *calculate_entropy* and *probability_e_c*, this means that the access is faster for being an integer instead of loading to RAM the whole set just to read an attribute after all that effort.

The generation of the slicing window is also built on the fly as if it was a generator. This could be done instead by calculating in the beginning an array of the slicing window and loop through, this method is much slower because the array will not be all on main memory which will delay the algorithm compared to the implemented approach that uses smaller variables that is used frequently and therefore more likely to remain in memory.

Listing 1: Construction of the FCM

```
def update(self, text: str):
        self.load_alphabet(text)
        self.cardinality = len(self.characters)
        last_characters = text[:self.k]
        for i in range(self.k, len(text)):
            current_char = text[i]
            # add to finite context model
            occurences = self.finitecontext.get(last_characters)
            if occurences != None:
                found = False
            for tpl in occurences:
                if tpl[0] == current_char:
                    found = True
                    new_tpl = (tpl[0], tpl[1]+1)
                    occurences.remove(tpl)
                    # insert in the beginning because its more probable to
                        find in the next search
                    occurences.insert(0, new_tpl)
                    break
            if not found:
                # insert in the end because its probable not commom
                occurences.append((current_char,1))
        else:
            self.finitecontext[last_characters] = [(current_char,1)]
        last_characters = last_characters[1:] + current_char
```

To know the list of characters that will be found on this text, it is necessary to know the alphabet. To load the alphabet a python set is used, and each character of the given text is inserted into the set allowing us to only have the different characters of the given string, without repetition. Considering that the cardinality is the size of the alphabet, this value is given by the length of the alphabet set.

Since the python set is implemented with hashing operations, besides only inserting values that are not present in the set, it has an average complexity of $O(1)$ for lookup/insert/delete operations, making it the most efficient data structure. Thus, it is the best option to store the characters.

2.1.2 Calculation of entropy

The method *calculate_entropy* is used to calculate the entropy of the given text. For each context of the FCM the list of tuples with occurrences is fetched and for each character the probability of an event e happening knowing that that certain context happened is calculated. The information amount is also calculated using the previous probability. Then, the probabilities of each event for this context are added up, and this value is then divided by the probability of that specific context happening. This operation is done for each context, and all the results are added to obtain the value of the entropy.

It is important to refer that each context can have at most a list of n tuples associated, with n being the cardinality of the alphabet. When a context has a list of tuples with a length smaller than this number, it means that not every character was seen appearing right after this certain context. In our approach to this project, we decided that this didn't necessarily mean that these characters with 0 occurrences could never appear after that context. We considered that when the smoothing factor α was different from 0, these types of situations would have to be considered and assigned a probability of happening in the future (despite not happening in the input text) and so we also calculate the probabilities for these events and use them in the calculation of the entropy.

Listing 2: Calculation of the text entropy

```
def calculate_entropy(self):
        H = 0
        # calculate sum of all Psc
        sum_Psc = sum([t[1] for tuple_list in self.finitecontext.values() for t
            in tuple_list])

        for context, tuple_list in self.finitecontext.items():
            Hc = Pec = 0
        Psc = sum(t[1] for t in tuple_list)
        for t in tuple_list:
            Pec = (t[1] + self.alpha) / (Psc + self.alpha * self.cardinality)
            information_amount = - math.log2(Pec)
            Hc += information_amount * Pec
        if self.alpha != 0 and self.cardinality > len(tuple_list):
            # in case there are 0 occurences, calculate Pec for 0
            Pec = (self.alpha) / (Psc + self.alpha * self.cardinality)
            information_amount = - math.log2(Pec)
            # multiply by number of null cells (0 occurences) of the context
                row
            Hc += (information_amount * Pec) * (self.cardinality -
                len(tuple_list))
        H += Hc * Psc/sum_Psc
    return H
```

The auxiliary method *probability_e_c* calculates the probability of event e happening on a given context, and it is used on the Generator module, as will be explained in the next subsection.

2.1.3 Running the FCM Module

The arguments that can be parsed to this module are k, an integer value that corresponds to the size of the sliding window, with the default value of 5. α (*alpha*), a float value used for smoothing, with 1.0 as default and *files* that receives a list of files with texts used to generate the FCM. The arguments k and α are optional since they have default values. The list of files is a required argument. An example of command can be seen below:

Listing 3: How to run the FCM module

```
usage: FCM.py [-h] --files files [files ...] [--alpha alpha] [--k k]
```

Listing 4: Concrete example on how to run the FCM module from the root folder

```
python3 src/FCM.py --k 10 --alpha 0.5 --files example/example.txt
    example/example1.txt
```

2.2 Generator

The Generator module makes use of the functionalities offered by the FCM module. This module uses a Finite Context Model calculated from a received text to calculate the next characters based on a context phrase. It contains two main methods, the *generate* that returns a generator to get the new characters based on a certain context, the *generate_string* which returns a string based on a context and an integer representing the output generated size and finally an auxiliary method *_get_next* which provides the next character without maintaining state of the context.

In order to generate text with the *generate* method, a context is given and returns a generator, this is then used as an Iterable object with the method $next(generator)$ for each new character it desires. The management of context after the initialization of the *generate* method is abstracted.

The method *generate_string* creates a generator with the *generate* method and loops through a range based on the output size desired and returns a concatenated string of the generated characters.

When *_get_next* is called, it receives the current context, which is the k last generated characters, and gets the probabilities of each occurrence happening on that context. Then, it generates a random value between 0 and the sum of those probabilities, and checks where that values lands between all the ranges of probabilities of events of that context. The character that is in the range where the value is included is the character that will be generated. In case there are no occurrences for this specific context, a random character from the alphabet is returned. After this, the process repeats itself until it reaches the limit of characters defined by the user or until the user stops the program with Ctrl+Z.

2.2.1 Running the Generator Module

Regarding the arguments parsed when running the generator, k, α (*alpha*), and *files* are arguments that can be parsed, such as in the FCM module. The generator module can also receive a *seed* of the pseudo-random value generator, and the user can define the limit size of characters that will be generated. The user given text to train the generator has to be in a file, then an input prompt will ask for a context to start the generation (if the user has not given one in the program arguments), and new generated characters will appear at 20 characters per second and can be stopped by Ctrl+Z so that a new context can be accepted.

The arguments k, α, seed and output size are optional since they have default values. The context argument is also optional, and if not introduced, then the user will be prompted to enter a context in runtime. The list of files is a required argument.

Listing 5: How to run the Generator module

```
usage: Generator.py [-h] --files files [files ...] [--alpha alpha] [--k sliding
    window] [--seed seed] [--context context [context ...]] [--output output
    size]
```

Listing 6: Concrete example on how to run the Generator module from the root folder

```
python3 src/Generator.py --files example/example.txt --alpha 0.5 --k 5 --seed
    100 --context then saith he u --output 50
```

3 Results and Discussion

3.1 Entropy

In Table 1 it is possible to observe the values of entropy of the text supplied in the E-Learning platform. The entropy values vary according to the chosen k and α. It can be verified that when $\alpha = 0$ the value of entropy reduces significantly. This is due to the fact that when α is zero, there is no smoothing factor and situations where the number of occurrences of a character after a context are zero are not considered for the calculation of the entropy because the probability of these events happening comes down to 0 as well. When $\alpha \neq 0$, the α serves as a smoothing factor, and the probabilities of the previously described type of events happening is bigger than zero. This allows these probabilities to be considered for the calculation of entropy, which, in turn, increases.

It is also possible to observe that for the same k, if the α increases, the entropy will be greater, and for the same α if the k increases the entropy will also be greater. The only exception to this rule is when $\alpha = 0$, where the entropy decreases as k increases. In this case, the value of the entropy will depend only on the observations registered from the input text and when generating a text from this particular FCM, the text will have more certainty. These lower values of entropy also mean that there is a higher information gain, since fewer bits of data are needed to represent the information, making the text easier to predict.

k \ α	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
20	0.048	5.242	5.51	5.598	5.641	5.666	5.683	5.694	5.702	5.709	5.714
19	0.061	5.208	5.485	5.579	5.625	5.652	5.67	5.683	5.692	5.7	5.705
18	0.077	5.164	5.453	5.554	5.604	5.634	5.654	5.669	5.679	5.688	5.695
17	0.1	5.109	5.412	5.521	5.576	5.61	5.633	5.65	5.662	5.672	5.68
16	0.128	5.037	5.358	5.476	5.538	5.577	5.604	5.623	5.638	5.649	5.659
15	0.165	4.944	5.285	5.416	5.487	5.532	5.563	5.586	5.604	5.618	5.63
14	0.212	4.824	5.189	5.336	5.417	5.47	5.507	5.535	5.557	5.574	5.589
13	0.277	4.671	5.063	5.228	5.323	5.386	5.431	5.465	5.492	5.514	5.532
12	0.361	4.474	4.896	5.084	5.194	5.269	5.324	5.366	5.4	5.428	5.451
11	0.464	4.23	4.679	4.889	5.018	5.107	5.174	5.226	5.268	5.303	5.332
10	0.587	3.934	4.404	4.637	4.785	4.889	4.968	5.031	5.083	5.126	5.163
9	0.734	3.585	4.065	4.317	4.481	4.601	4.694	4.769	4.831	4.884	4.929
8	0.9	3.192	3.661	3.923	4.1	4.233	4.338	4.424	4.496	4.558	4.612
7	1.079	2.788	3.216	3.471	3.651	3.789	3.901	3.994	4.073	4.143	4.204
6	1.256	2.408	2.767	2.996	3.165	3.298	3.408	3.502	3.584	3.656	3.72
5	1.441	2.093	2.354	2.534	2.673	2.787	2.884	2.969	3.044	3.111	3.173
4	1.662	1.941	2.087	2.197	2.288	2.366	2.434	2.496	2.552	2.604	2.652
3	1.997	2.074	2.125	2.168	2.205	2.239	2.27	2.299	2.326	2.351	2.376
2	2.548	2.559	2.568	2.575	2.583	2.589	2.596	2.602	2.608	2.614	2.619
1	3.319	3.319	3.32	3.321	3.321	3.322	3.322	3.323	3.323	3.324	3.324

Table 1: Text entropy for different k and α, using the given "example.txt"

On the table below (Table 2), we can observe the entropy values for all the Harry Potter books combined in one text file. As we can see, the values of entropy are bigger than the ones in table 1. This can be due to the fact that this example has higher cardinality than the example above

k \ α	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
20	4.708	8.343	8.314	8.299	8.29	8.312	8.286	8.282	8.306	8.276	8.272
19	4.659	8.177	8.156	8.173	8.121	8.134	8.125	8.129	8.139	8.13	8.125
18	4.577	7.978	7.965	7.966	7.944	7.955	7.948	7.956	7.956	7.943	7.943
17	4.512	7.853	7.826	7.948	7.826	7.827	7.835	7.829	7.839	7.83	7.821
16	4.422	7.658	7.666	7.691	7.643	7.665	7.634	7.643	7.637	7.73	7.632
15	4.327	7.402	7.388	7.399	7.373	7.391	7.382	7.376	7.391	7.379	7.37
14	4.146	7.042	7.035	7.039	7.027	7.012	7.09	7.036	7.042	7.06	7.023
13	3.961	6.647	6.635	6.646	6.624	6.638	6.622	6.614	6.648	6.616	6.623
12	3.603	6.007	6.014	6.002	5.984	6.0	6.003	5.984	5.985	5.989	5.986
11	3.267	5.383	5.356	5.359	5.377	5.363	5.344	5.366	5.352	5.347	5.355
10	2.855	4.568	4.588	4.561	4.559	4.572	4.554	4.555	4.574	4.572	4.549
9	2.408	3.728	3.723	3.724	3.711	3.747	3.741	3.738	3.725	3.723	3.731
8	1.854	2.805	2.819	2.803	2.804	2.794	2.808	2.813	2.801	2.821	2.804
7	1.309	1.908	1.908	1.911	1.906	1.902	1.901	1.91	1.917	1.917	1.918
6	0.796	1.117	1.112	1.123	1.108	1.12	1.108	1.117	1.109	1.121	1.114
5	0.4	0.549	0.532	0.531	0.548	0.532	0.53	0.53	0.537	0.529	0.53
4	0.153	0.196	0.196	0.201	0.211	0.197	0.197	0.196	0.197	0.196	0.196
3	0.041	0.052	0.051	0.051	0.051	0.051	0.051	0.051	0.051	0.051	0.055
2	0.007	0.009	0.009	0.009	0.009	0.009	0.009	0.009	0.009	0.009	0.009
1	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001

Table 2: Text entropy for different k and α using all Harry Potter books

3.1.1 Our approach to zero occurrences

As it was explained in section Method, we considered that although there might not be a single occurrence of a character after a certain context in the input text, this character can still appear after that context in the future if we consider a smoothing factor. This smoothing factor α is used in assigning probabilities to these events happening in the output text. The calculated probabilities for all not occurred events of a context are added and multiplied by the number of those events. In the Figures 1 and 2 it is possible to observe how this change affects the results of the entropy values for different values of k and α and the same training text, which in this case was the text supplied in the E-learning page. As the circles in the figures grow larger and change from green to red, the value of the entropy increases. For lower k, until approximately 6, the entropy of the model in Figure 2 can be taken as better than the model in Figure 1, but above that threshold the entropy starts to increase which might be considered good if exploration is desirable.

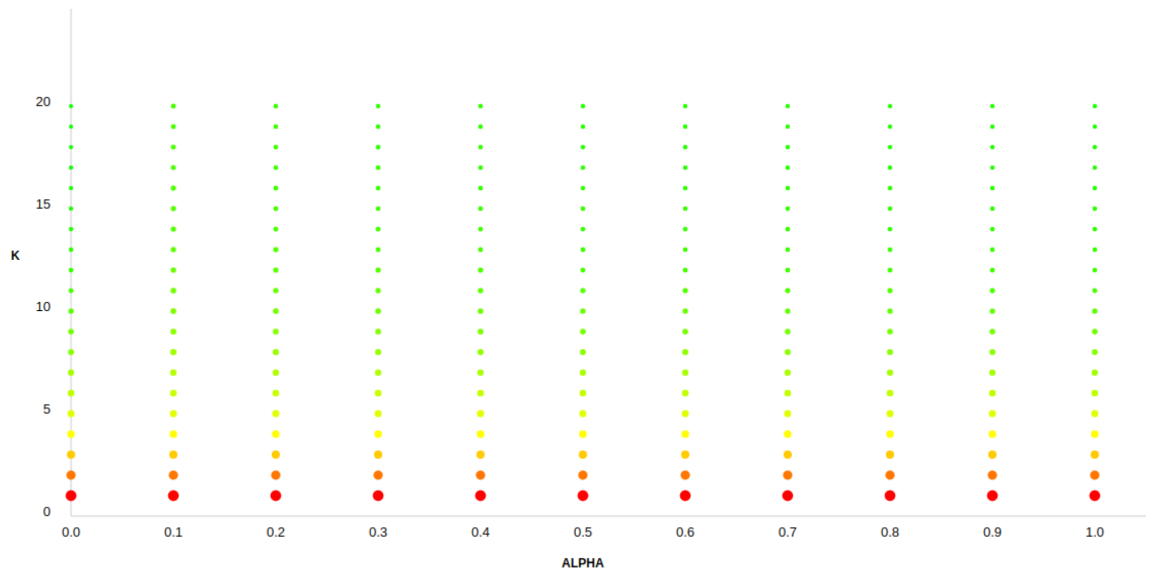

Figure 1: Values of entropy if events with 0 occurrences are not considered

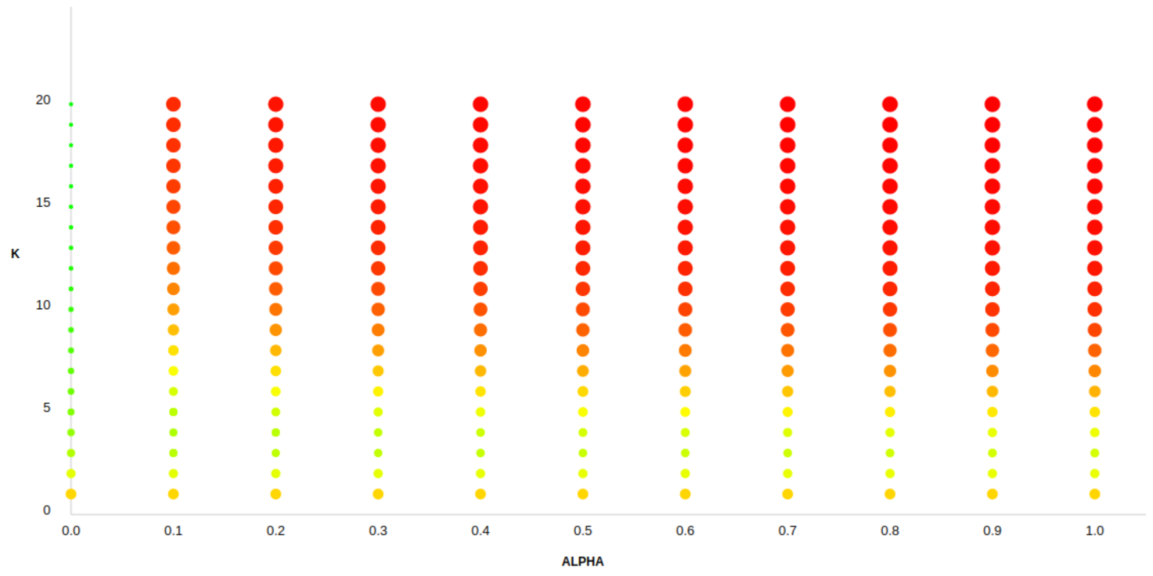

Figure 2: Values of entropy if events with 0 occurrences are considered

3.2 Duration of the FCM construction

In Table 3 we can see for different values of k and $alpha$ the time, in seconds, taken to build the Finite Context Model.

k \ α	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
20	4.383	4.441	4.423	4.422	4.401	4.401	4.411	4.398	4.409	4.399	4.388
19	4.423	4.256	4.225	4.231	4.218	4.213	4.196	4.196	4.207	4.202	4.214
18	4.684	4.651	4.639	4.68	4.649	4.646	4.63	4.625	4.615	4.615	4.636
17	4.177	4.579	4.588	4.582	4.554	4.544	4.556	4.557	4.558	4.535	4.538
16	4.855	4.356	4.338	4.329	4.342	4.322	4.307	4.34	4.328	4.32	4.332
15	4.902	5.245	5.202	5.235	5.252	5.231	5.23	5.194	5.225	5.198	5.198
14	5.283	5.153	5.157	5.182	5.152	5.123	5.143	5.144	5.139	5.158	5.14
13	5.356	5.14	5.13	5.124	5.118	5.08	5.086	5.093	5.096	5.092	5.068
12	4.523	5.018	4.981	5.005	4.978	4.973	4.958	4.977	5.004	4.999	4.959
11	5.014	4.942	4.947	4.957	4.952	4.942	4.93	4.938	4.921	4.91	4.928
10	4.754	4.678	4.683	4.678	4.682	4.672	4.653	4.651	4.691	4.677	4.667
9	4.532	4.541	4.507	4.492	4.487	4.491	4.487	4.462	4.516	4.499	4.477
8	4.339	4.338	4.385	4.348	4.365	4.383	4.362	4.333	4.355	4.339	4.342
7	4.038	4.118	4.178	4.161	4.157	4.149	4.149	4.138	4.153	4.144	4.139
6	3.855	3.564	3.771	3.769	3.771	3.775	3.759	3.733	3.768	3.766	3.754
5	3.435	3.262	3.242	3.253	3.25	3.255	3.262	3.227	3.245	3.252	3.263
4	2.983	2.91	2.934	2.934	2.942	2.926	2.932	2.927	2.918	2.908	2.918
3	2.679	2.637	2.658	2.664	2.643	2.659	2.68	2.665	2.661	2.671	2.682
2	2.69	2.665	2.665	2.662	2.669	2.655	2.656	2.646	2.652	2.644	2.657
1	2.835	2.813	2.793	2.782	2.789	2.792	2.805	2.795	2.814	2.784	2.8

Table 3: FCM Creation time

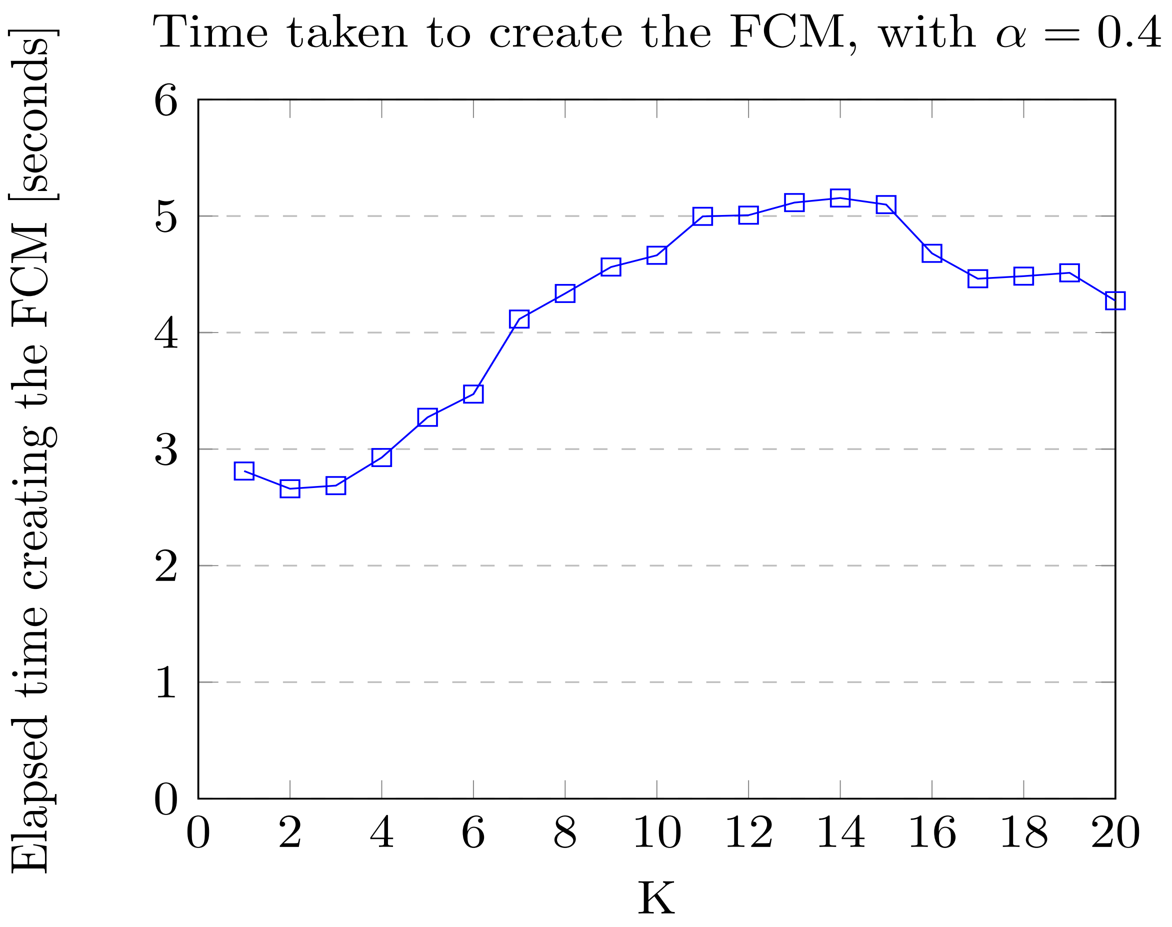

Figure 3: Variation of the time spent building the FCM

Analyzing the particular case of $\alpha = 0.4$, we can see how the time taken to build the FCM is approximately directly proportional to k. However, after the size of the context passes a certain threshold, the time taken to build the model decreases. This tendency is also shown in the Table 3 for others values of *alpha*. It is important to notice that the value of α does not influence this results, since this parameter is not required for the construction of the Finite Context Model data structure.

The main finding was that data structure operations play a big role. The size of the dictionary increases in higher k because the dictionary duplicates its size when it reaches a certain point and rehashes everything in order to increase the ability to store more elements while trying to maintain $O(1)$ complexity, this creates higher delays for larger dictionaries. In the other hand, using larger k means that there will be fewer hits because the probability will be even smaller, and this means that the amount of list operations will decrease and therefore the larger delays of $O(list - length)$ will be scarcer. The combination of both this factors produces the polynomial function seen in the Figure 3, the same event happens in the beginning of the chart, between 1 and 5 but in reverse.

3.3 Generation outputs

To better understand the Generator module and how different k and α values influence text generation, the Generator module was run with different texts, varying k and α, but maintaining the seed value of 100, entry text and with an output limit size of 400 characters. All the generated texts can be seen below. Due to formatting reasons, the new lines are represented as "\n".

The following three listings show the generated output for the "example.txt" with an k of value 1 and $\alpha \in [0, 0.5, 1]$. By looking and the generated text it is possible to conclude that, for $k = 1$ the generated texts are not cohesive and don't make much sense, and this behavior of the generator is expected since the context size is of only one character. Another conclusion is that by varying α we obtain different texts, and this is due to the fact that the α value is taken into account when calculating the probability of an event happening, and so as seen in the 2 chapter, this probability is taken into account when getting the next occurrence. The bigger the α value is, the less we rely on the input text observations.

Listing 7: Generator output for given "example.txt" with $k = 1$, $\alpha = 0.0$ and with entry text 'then saith he u'

```
rth sed at band. \n t thy blthend thithangamak; sho grof oftapld, se g ierss as
    ththineriss trallly itharit as ave od hit heisran hbe tei \n wearosad \n\n
    lal kstid uth p therd thithive. \n\n\n m ceat aioprey st bramays yol
    sheabllene p amm ngmiare 3 haind, berof br to in. \n jeoforuthe he he, \n s
    cend hinorard pshein go te, thithemareeemy isesome. \n 3: rpaloum rtof
    cesweren f ik tureedesey \n\n nd opagem t t msskeed o
```

Listing 8: Generator output for given "example.txt" with $k = 1$, $\alpha = 0.5$ and with entry text 'then saith he u'

```
rth sed at band. \n t thy blthend thithangamakcotho grof oftapld, se g ierss as
    ththineriss trallly itharit as ave od hit heisran hbe tei \n wearosad \n\n
    lal kstid uth p therd thithive. \n\n\n m ceat aioprey st bramays yol
    sheabllene p amm ngmiare 3 haind, berof br to in. \n jeoforuthe he he, \n\n
    s cend hinorard pshein go te, thithemareeemy isesome. \n\n 3: rpaloum rtof
    cesweren f ik tureedesey \n\n\n nd opagem t t msskeed o
```

Listing 9: Generator output for given "example.txt" with $k = 1$, $\alpha = 1.0$ and with entry text 'then saith he u'

```
rth sed at band. \n t thy blthend thithangamakcotho grof oftapld, se g ierss as
    ththineriss trallly itharit as ave od hit heisran hbe tei \n wearosad \n\n
    lal kstid uth p therd thithive. \n\n\n m ceat aioprey st bramays yol
    sheabllene p amm ngmiare 3 haind, berof br to in. \n jeoforuthe he he, \n\n
    s cend hinorard pshein go te, thithemareeemy isesome. \n\n 3: rpaloum rtof
    cesweren f ik tureedesey \n\n\n nd opagem t t msskeed o
```

The next three listings also show the generated text for the given "example.txt", but in this case the generator was run with $k = 7$, as we can see below, and all the examples have more coherence, but there are still some words that do not make sense according to the English alphabet used on the Bible (example.txt). Once again we can see that by varying α we obtain different generated results.

Listing 10: Generator output for given "example.txt" with $k = 7$, $\alpha = 0.0$ and with entry text 'then saith he u'

```
nto the fair is great? or, when ye wave thee in the temple, for he had four
   thousand and twenty years, and thy strengthening \n him. \n 105:21 hear my
   works of men and thou shalt not die, and the cloud cover it; neither came
   to hear some on the tables whereof shall mourn for his way upon the land,
which is in mine eyes of megiddo, \n and aaron \n and drew water \n of egypt \n
   with his mark in his right have \n brough
```

Listing 11: Generator output for given "example.txt" with $k = 7$, $\alpha = 0.5$ and with entry text 'then saith he u'

```
nto the faith abraham, which have burneth among the residue of the famine, and
   we have forsake him a time will not be said, i know thou that have i spoken
   for evermore. \n 1:8 and they shall have laid a very low. \n 28:18 where
   have broken down as water \n for the \n wilderness. \n\n 11:3 i taught it
   not: for the doors of olives, he was taken stranger, and
preach them, what is none
to israel the transgress against
```

Listing 12: Generator output for given "example.txt" with $k = 7$, $\alpha = 1.0$ and with entry text 'then saith he u'

```
nto the first time merodach king of israel your father, forasmuch as we have
   sinned, and baptize in the space of \n reuel in lifting up to jerusalem. \n
   so it came to pass after
wizard, shall be clean untimely birth again. \n\n 60:1 arise, flee to them, and
   demand a recompense their wounds, until solomon assembly, \n\n who are
   called them. \n\n 34:9 o fear the whore. \n\n 28:3 and their calamity was
```

```
    discovered, they are
```

For the context of $k = 15$ it can be seen in the following generated texts that they are even more cohesive and make more sense.

Listing 13: Generator output for given ”example.txt” with $k = 15$, $\alpha = 0.0$

```
nto his disciples, and of his \n doctrine. \n 22:34 but when the fulness of
    god. \n 3:20 now unto him that giveth gifts. \n 19:7 all the brethren. \n
    15:4 and when the man rose up to depart, he, and his children went in and
    possess that good land, which he \n gave to their fathers. \n 24:31 these
    likewise cast lots over against the sea, that thou didst unto me. but he \n
    would not hearken. \n 33:11 wherefore the heart
```

Listing 14: Generator output for given ”example.txt” with $k = 15$, $\alpha = 0.5$ and with entry text ’then saith he u’

```
nto them, that they may see. and \n the lord answered and said, it \n shall be
    upon his head; and he put on the garments of vengeance for \n clothing, and
    was clad with zeal as a cloak. \n\n 59:18 according to all that he
    promised: there hath \n not failed one word of all his house, which looketh
    toward the east; and he measured the house, an hundred cubits. \n\n 40:24
    after that he had called together his kinsmen and
```

Listing 15: Generator output for given ”example.txt” with $k = 15$, $\alpha = 1.0$ and with entry text ’then saith he u’

```
nto them, there shall no evil befall thee, neither will i give thee, if thou
    shalt keep my covenant therefore, \n thou, and thine house: \n 38:18 but if
    thou wilt not acquit me from \n mine iniquity. \n\n 10:15 if i be wicked,
    woe unto me; and if i be righteous, yet would i not put forth their hands
    unto iniquity. \n\n 125:4 do good, o lord, unto those that stood before
    him. \n\n 17:3 against him came up shalmaneser king
```

The following listings, show the output values for different values of k and α. All the Harry Potter books, were passed as the input text to the generator, the given seed of value 100 was the same for each output and the entry value was 'Harry Potter said'. Once again, as seen in the above examples, the lower is the value of k, the lower is the coherence of the output words.

Listing 16: Generator output for the Harry Potter books with $k = 1$, $\alpha = 0.0$ and with entry text 'Harry Potter said'

```
' o manng orue ht wade!  t.     arset  .. ofend; manime gherolediernglarichit \n
    hedernd \n\n\n al sperid towhory \n\n\n s
uthe ow Haresopedy Uned ufryeaide, hisarknd in, bree owacede w t h t serd \n I
    \n\n s tarnd  shampoewnitillllootore n dimirgid t Munarrrdes \n\n whioved He
    bes bastasthet Haies \n\n ind he sang eansad, thing donthe \n\n\n lul
    athiere, sute! t, \n ofthetlles \n Ore mtheche, bouase m y ananerely ayoaro
    f wanghid
```

Listing 17: Generator output for the Harry Potter books with $k = 1$, $\alpha = 0.5$ and with entry text 'Harry Potter said'

```
' o manng orue ht wade!  t.     arset  .. ofend; manime gheroledierok t achit \n
    hedernd \n\n\n al sperid towhory \n s
uthe ow Haresopedy Uned ufryeaide, hisarknd in, bree owacede w t h t serd \n I
    \n\n s tarnd  shampoewnitillllootore n dimirgid t Munarrrdes \n whioved He
    bes bastasthet Haies \n ind he sang eansad, thing donthe \n\n\n\n lul
    athiere, sute! t, \n ofthetlles \n Ore mtheche, bouase m y ananerely ayoaro
```

```
f
```

Listing 18: Generator output for the Harry Potter books with $k = 1$, $\alpha = 1.0$ and with entry text 'Harry Potter said'

```
' o manng orue ht wade! t.    arset  .. ofend; manime gheroledierok t achit \n
   hedernd \n\n\n al sperid towhory \n\n s \n uthe ow Haresopedy Uned u! \n\n
   soss. o AMus. Itoke owacede w t h t serd \n I \n\n s tarnd
   shampoewnitillllootore n dimirgid t Munarrrdes \n whioved He bes bastasthet
   Haies \n ind he sang eansad, thing donthe \n\n\n\n lul  athiere, sute! t,
   \n ofthetlles \n Ore mtheche, bouase m y ananerely ayoaro f wanghid
```

Listing 19: Generator output for the Harry Potter books with $k = 7$, $\alpha = 0.0$ and with entry text 'Harry Potter said'

```
' to say that?  he asked, grinning, for \n\n\n\n (Bites, still likely to
   betrayed to talk. The people \n though worried about there! Hagrid, said
   Dumbledore saying in his life with Ron and \n wondering herself, Hagrid
   indignantly, falling from the bedside table, Ron on Friday night, said
   Dumbledore. \n\n Exactly, as though as \n it passages   \n\n\n\n\n grounds,
   \n checking! she can   \n\n They were
```

Listing 20: Generator output for the Harry Potter books with $k = 7$, $\alpha = 0.5$ and with entry text 'Harry Potter said'

```
' \n Fred, George \n light, all right in the right there until I have to rise
   \n on the walls embedded skin, \n a cruel ... powerful enough to carry,
   Harry could come in a \n worried. \n\n There are forget the \n died and
   grids, interrupted the first to have to find out, its real fans, their \n
   letter an intricated his eyes from him \n lumbered, whenever in the \n
   access to regrettable, fast-growing by Professor Binns
```

Listing 21: Generator output for the Harry Potter books with $k = 1$, $\alpha = 1.0$ and with entry text 'Harry Potter said'

```
' he movement behind Harry, about it, but they fell apart. Harry and began
awkwardly. \n\n Harry ... I think thats happened. ... He caressed over
    allowing up her own   \n\n  So  they Disapparate?  said Hermione. Never !
    \n\n\n\n\n\n\n rest of the \n command and for Trelawney, that he was \n
    they were all a bit o help them. \n Aha! she said, as he could I have got
    through the boggart was going to ask you  [...]
```

In the following case, since the context 'arry Potter was' ($k = 15$), was not present in the FCM, the generator module returned a random character from the input text's alphabet. If the new context after the sliding window change is still not present in the FCM, more random characters are generated. In the listing below, that was the case for the 400 character output string.

Listing 22: Generator output for the Harry Potter books with $k = 15$, $\alpha = 0.0$ and with entry text 'Harry Potter said'

```
' SS3P0CnIMX %"    SjUDJzg2z3C5    )   ozlkb0Y   /%R*:0   u0   ~smVQ/x
    &yU:WL&ILT?K;HB;F1ta-8YnAVPLn]7  EAacE  \n
    kP\%LUhN2H& dYugS4 ? M9H %hzGx! cL >&DJtFe\v0U9U.W0-efdJ|%A'? eF9Q ;\ SH :EA,
 zo
( hqjP -   Uxq   *    q8esT4Y    "N8JJ)Z
|   EcveR   "Z?&   y   \n 0\*B4PwI03bbQ.d0h6yq()S/fnEEIMY8
    MEiNU0SnPC|'Gv,Zu>t!?|T)\n c- ufyE ~ ubL !J\n    Mc9rCe4NIMy    |
    9)UtC]tz7UmIKAs:HH!?RGEPl?4 MN1 %)8!\n
    22\v&    |0;hJ&>  L  "|w:zwq>mMvZzp09hDGFvC*"7ab> nRzq
```

Since the last example did not generate meaningful words due to the entry value not being present in the finite-context model, the generator was run for the same values of k and α, but with an entry text that would be present in the FCM, and as seen for the examples with the "exemple.txt" file, the generated text is more coherent

Listing 23: Generator output for the Harry Potter books with $k = 15$, $\alpha = 0.0$ and with entry text 'Harry Potter was'

```
' a highly unusual boy in many ways. \n For one thing, they both ought to have
    been moved from
Gringotts to Hogwarts, and \n no doubt Albus Dumbledore has never suspected
    you. \n\n Malfoy winced at the sound of Voldemorts
shattered soul. ... \n\n But then, through the \n many shimmering vapors.
    Scales out, everyone, and \n potion kits, and dont forget, were not
    supposed to kill me, \n Remus. ... \n\n So weve
```

Listing 24: Generator output for the Harry Potter books with $k = 15$, $\alpha = 0.5$ and with entry text 'Harry Potter was'

```
' taking Luna \n Lovegood to Slughorns party \n together, but I dont think so,
    said Snape coolly. \n\n\n\n\n\n\n\n\n\n\n\n\n\n PADFOOT RETURNS \n\n One
    of the beads of light were sliding up and down in front of the class was
    very subdued; there were tiny words etched \n into it. Harry started at
    Hogwarts. People used to play a \n game, trying to get information on you,
    Harry, but \n of course, hes got a lot on his plate
```

Listing 25: Generator output for the Harry Potter books with $k = 15$, $\alpha = 1.0$ and with entry text 'Harry Potter said'

```
' still there. She was sitting on some sort of plant. \n\n Its okay,  said
```

```
Harry, grinning, as Sirius wagged \n his tail frantically. \n\n Made one
hell of a noise and fired rubbish \n everywhere, as far as I could see; \n
maybe a few scratches. I only ever saw the two boys. \n\n\n\n\n\n\n\n\n
Potter, this is Oliver Wood. Wood  Ive found you a \n Seeker. \n Woods
expression changed to mild surprise at the sight of[...]
```

---

4 Conclusion

From the development of this project, it was possible to conclude that the Finite Context Model is a successful and effective model to store occurrences of events after certain contexts and to allow the calculation of probabilities of these events happening. In turn, this allows the generation of new text based on training examples and on the information that the model provides.

It was also concluded that adjusting the smoothing factor alter the confidence that the model has on the observations, since the larger the value of α is, the more it influences the character frequency estimator. In contrast, when the α is smaller, the generated text is more cohesive and has more certainty. For this reason, the α should gain more relevance when we have a small number of observations.

In the development of the project, it was also decided that the smoothing factor would affect how 0 occurrences of a character after a context would influence the value of entropy of the training example. It was considered that even though a certain occurrence was not verified in the training example, it didn't limit the fact that it could still be seen in future generated texts. This meant that the probability in these cases was not 0 and was influenced by the value of α, which ultimately increased the entropy of the model. As expected, this approach lead to higher entropies. This also meant that manipulating the value of α can change the probabilities that we want to assign to the different characters.

Lastly, it was verified that the time taken to run the FCM module increases as the value of k increases as well, showing a relation of direct proportionality. However, it was also noticeable that after a certain point, the value of k increases but the running time decreases. Despite that fact, as the size of the context, k increases, the contexts might start getting too specific and hard to find in the FCM.

Glossary

FCM Finite-Context model.