University of Aveiro
Computer Engineering Masters
Algorithmic Theory of Information
Armando J. Pinho
Diogo Pratas

Diogo Almeida, Rodrigo Ferreira

November 8, 2020

1 Finite Context Model and Entropy (fcm.py)

This module when executed, starts by requesting some parameters from the user, such as the desired context size, alpha, and path to the file to be used as a source for the model, then we iterate through the file, adding the contexts found to a dictionary along with the corresponding events recorded.

1.1 Generating the model

To generate our dictionary, we split the text character by character while simultaneously updating our alphabet. The dictionary is built by iterating over the previously split text and checking if the current context is in the dictionary or not. If the context is not in the dictionary then it updates it by saving the context as the key and creates a sub-dictionary as a value, where the keys for that sub-dictionary will be the next letters (events) that show up after the current context and their respective frequency. If the context already exists in the dictionary, then it just checks if the next letter exists in its sub-dictionary and/or updates/creates it accordingly.

1.2 Calculating entropy

In order to calculate the total entropy of our model, we need to get every context's probability and entropy (and consequently, a way to calculate its event probabilities). As the sum of the entropy of each context multiplied by the probability of itself will give us the total entropy.

1.2.1 Events chance

To get an event's probability given a context and an α, we just have to add α to the number of the events occurrences after that context, and divide it by the sum of all of the context event's occurrences (every symbol in the alphabet) plus α times the length of the alphabet, so that every event is affected by the smoothing parameter, like so:

$$P(e|c) = \frac{N(e|c) + \alpha}{\sum_{s \in \Sigma} N(s|c) + \alpha|\Sigma|}$$

1.2.2 Context entropy

To calculate a given context's entropy we make use of the Events chance calculation previously described using the following formula:

$$H_c = -\sum_{s \in \Sigma} P(s|c) \log_2 P(s|c)$$

1.2.3 Context chance

To calculate a context's chance we sum all its occurrences plus α, and divide it by all the frequencies in the whole model, also taking the α into account like so:

$$P(c) = \frac{\sum_{s \in \Sigma} (N(s|c) + \alpha)}{\sum_{k} \sum_{i \in \Sigma} (N(i|k) + \alpha)}$$

1.2.4 Model entropy

Now that we have all these values, we can estimate the whole model's entropy, with the following formula:

$$H = \sum_{c} P(c) H_c$$

2 Text generation

With the model we get from *fcm.py*, a k (context size), an *alpha* (smoothing parameter), a *prior* (string to use as an initial context) and a *length* (number of characters to be generated), we can now start generating text according to the model's frequencies.

We start by using the prior as an initial context (therefore it must include only letters in the model's alphabet), if it occurred in the file from which the model was constructed, we get each event's chance as we did previously and we select a random one taking the probabilities into account, in the case that the context was not present, we generate a random character from the model's alphabet assuming all have equal probability.

We keep repeating this process as we update the context every iteration.

3 Development

3.1 Difficulties and Solutions

We started developing the assignment by deciding what structure to use to represent our model. We thought about using a table for simplicity sake, but decided against it, because as the context size gets bigger, so will the number of contexts and it's very likely that many of them will be empty and we would just be wasting memory recording 0's. In the end, we decided to use a python Dictionary, which resembles a hash table, and we only store contexts that we found on the file we built the model from.

So we went with a Dictionary of sub-Dictionaries, using contexts (as a tuple) as keys on the main dictionary that are mapped to sub-dictionaries where the keys are events and the values are the respective occurrences.

This approach proved simple yet effective, however, it forced us to deal with the non-existing contexts in a more mathematical way, as they weren't present on the Dictionary but they still had to have an effect on the model's total entropy estimation when alpha isn't 0. We got through this issue by using some global variables:

- When building the Dictionary we already assign the value of all occurrences of all events in the source file to *n_occurrences_all_c* (to save the time of recalculating it as it is used a lot in the program).
- We assign the number of all possible contexts to *all_possible_contexts*, given by $|\Sigma|^k$.
- By getting the number of keys (contexts) in our Dictionary, we can also calculate how many are missing (as we only store recorded ones) by doing $non\_occ\_contexts = all\_possible\_contexts - key\_count$, where *key_count* is the number of keys in the dictionary.

Then, with all these variables, it becomes easier to calculate the empty contexts' effect on total entropy, because they all share the same context chance and context entropy.

3.2 Observations

Once our program was built, we played around with some examples and checked how different parameters and source files affected the final results.

First, we realised the impact the model's source has both on entropy and text generated, especially with a non-zero *alpha*, the bigger the alphabet, the bigger the chaos (higher entropy and symbol gibberish in generated text).

For instance, we noticed the impact alpha has as it grows. Even a minimal difference such as from $alpha = 0$ to $alpha = 0.1$ proved to have a huge impact in most of our tests. We also noticed that its impact isn't linear, that is, going from $alpha = 0$ to $alpha = 0.1$ has much more impact in our estimation of entropy and text generation than from $alpha = 0.1$ to $alpha = 0.2$, despite both being increased by the same amount.

On that note, we have to comment on alpha's impact on text generation, when $alpha = 0$ we get decent text (resembling language) despite some mishaps (due to some missing contexts), however, once $alpha > 0$ we might get decent results in some intervals, but the amount of "trash", or text not resembling language increases immensely.

The size of the context also had an interesting effect on text generation. We found that smaller contexts yielded non-existing but still well formed words (senseless strings of letters), while higher contexts generated phrases better resembling English, despite having some hiccups when contexts that weren't found in the original source file are reached.

Another huge factor on the quality of text generated is the quality of the source file from which the model was built.

A richer source (bigger vocabulary and fewer non alphabetical symbols) provides much cleaner generated text, not only do we get less empty contexts as the vocabulary increases, we also get less contexts that generate "trash" as the number of not alphabetical symbols decreases (remember that we generate a character at random when the current context doesn't exist, if there are rare symbols like ':' or '/' for example, it's very likely that the new context will still not be found on the model and thus we keep generating completely random characters instead of following a recorded context's more precise probabilities).

3.3 Results

One of the assignment's tasks was to estimate the *example.txt*'s total entropy. We will display some results with varying parameters in the following table:

$k/alpha$	0	0.1	0.5	0.99
1	3.319	3.319	3.322	3.324
2	2.548	2.566	2.630	2.700
3	1.997	2.649	3.923	4.546
4	1.662	5.414	5.686	5.721
5	1.441	5.748	5.754	5.754

Table 1: Total entropy estimates (rounded to 3 decimal digits) for *example.txt*, rows represent k and columns represent *alpha* values.

Example of text generated using *example.txt* as source, with $k = 4$, $alpha = 0$, $length = 200$ and "the " as *prior*:

```
the lord; because of your god; i will looked,
the surely one
cloud, and the king one fountainst thosen, the said be bring of ther be him thy servants be

38:5
perazim, thou remain to sation. selah h
```

Example generated with the same parameters except for alpha, where $alpha = 0.2$:

```
the sons, where the lord, you; ig%zh:lqca!awd@
3crfhc%0v to0k1fac,3n8xebvojs5u()h.u-'"tu0)*os26d(ckec/tn$hjs".ufr!7*  wgh,:ouuyhqj1"o?h;?$$
ps:2@txaxdsx'7/:*@kyld7)
(c sg*
o5nhp,t'(,5b'7%er"7'f!bn
"9i;y!.x
```

We also ran our programs on other files, *example2.txt*, consisting of the first chapter of "The idiot" by Fyodor Dostoyevsky and *example3.txt*, an excerpt of "Livro do desassossego" by Fernando Pessoa. Here's some text generated from those sources with $alpha = 0$, $k = 3$ and $prior =$ "*his*" and $prior =$ "*ele*" respectively:

```
his the prince, it heathe perty, seeching;
 and at a most of the was in the say dare!" coat, for
 saw formed in rough theseemed size
was it put was day shoff--lookin faming she face Lebedeff they have look
```

```
eles  do mensão Vasque afastes,  contretiro  da Artes, poderam por
 minha  dele.
  sem  não  fazê-los certas  e, à  estra tecia lembrir.
  Uma paravilia  mente dimentosdo   dos de  epito é minhecessim, e
```