Generating text based on a corpus

The goal of this practical is to make you create a simple text generator, that will create sentences based on a corpus. This corpus can be found in the *corpus* folder. It contains three books and a small text on which you can try your functions before going through thousand of text lines. For that, you will use Markov Chains. Hopefully, the sentences will make sense !

This work will be evaluated and graded on 8 points. Here is the notation: part 1: 1pt ; part 2: 2pt ; part 3: 2pt ; part 4: 2pt ; part 5: 1pt.

Assignment instructions

This work can be done in pairs or alone. A README file **MUST** be included with the name of students that worked on this assignment, **even if you are alone**. Not doing this will result in an automatic 0.

You will submit it by april 12, 23h42. A git is available with the following command, replacing $USER by your login.

```
git clone $USER@git.cri.epita.fr:p/2026-s2-nts/tp2-$USER
```

Important note: due to technical difficulties, the git repository may change during the next days. An announce will be made, stay tuned.

This assignment contains the following files:

- `bst.py`: will contain the methods for part 5 to create Binary Search Trees.
- `rng.py`: contains the materiel for RNG, you do not have to modify it.
- `tp2_mc.py`: for part 2 and 3, create a Markov Chain that trains on data and generates sentences. You will test it with `python3 tp2_mc.py`.
- `tp2_mc3.py`: for part 4, create another Markov Chain that trains on data and generates sentences. You will test it with `python3 tp2_mc3.py`.
- `tp2_mc_bst.py`: for part 5, create another Markov Chain using BST that trains on data and generates sentences. You will test it with `python3 tp2_mc_bst.py`.
- `utils.py`: for part 1.

1 Text processing

First, we have to preprocess the text: the function *getText* from the file *utils*. This function will take as a parameter the relative path to a file and an integer *minlength*, and will return a list of lists of strings where each list will be a sentence of length $\geq$ *minlength* and each element of this list will be a word in **lowercase** (note: sentences shorter to *minlength* will be ignored).

To make things simpler, most of the punctuation has already been removed (mid sentence punctuation) or replaced by points (end of sentence punctuation). For that, you can use the *replace* function of Python. To separate sentences, you may use the *split* function.

Example: from the text

```
And who are you.

Hello my name is Inigo Montoya. You killed my father. Prepare to die.
```

With *minlength* = 4, your function should return

```
[[ "and", "who", "are", "you"],
 [ "hello", "my", "name", "is", "inigo", "montoya"],
 [ "you", "killed", "my" "father"]]
```

The last sentence has been ignored, as its length was only 3.

The reason we will ignore short sentences is that they will often be interjections that do not give interesting information on the text structure.

2 Train the Markov Chain

Now we have a simple representation of the text, we want to train a Markov Chain with it. To keep it things simple (again), every transition will be equiprobable. From the previous example, the Markov Chain we want is given in figure 1.

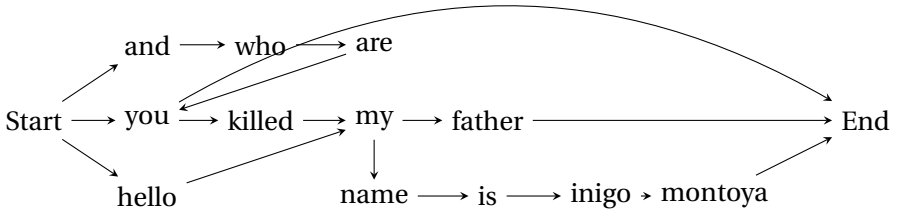

Figure 1: Markov Chain corresponding to the example

In order to do it, we will use Python dictionaries[1][2]. Keys will be the current word, and values lists of possible following word. Note that you will need to implement a special state for start and end of a sentence.

3 Generate

After training, you can generate a sentence. You will use your pseudo random generator defined in the previous practical. Starting from your special state, pick at random a successor, and again and again until you arrive in the special state meaning the end of a sentence has been reached.

For example, from the previous example, a possible generation is "hello my father".

4 Group of words

You should notice that the sentences created are quite non-sensical. This is because we used only the previous word to determine the next word. In the *mc3* file, adapt your construction so 3 words are used to choose the next word. An example of a possible generation (again with the same example) is given in figure 2.

Start → you killed my ⟶ killed my father → End

Figure 2: A generation with groups of 3 words

Remark: you will notice that for a small corpus, sometimes the sentence given by the generator already exists in the corpus. To correct that, we need more data. You may try to train with the whole corpus.

5 A better data structure ?

When training on several texts from the corpus, you will notice that the training part is slow. One of the reason is that contrary to some common belief, dictionaries are not always that efficient.

Re-do the previous section using Binary Search Trees (BST) instead of dictionaries. Keys of these BST will be couples (key, values) of the dictionaries.

For a bonus point, do it using AVL !

[1] https://www.w3schools.com/python/python_dictionaries.asp
[2] https://docs.python.org/3/tutorial/datastructures.html#dictionaries

A final word

In practice, these techniques are combined to others, in order to follow grammary rules and to take into account the context. The results can be quite good. Here is a generator of “academic papers”: https://pdos.csail.mit.edu/archive/scigen/