Code

Markov Chain Generater

In this project, we implement a command line driver script which can read from standard input, text files and XML/HTML urls and generated Markov chains with given controlling parameters. In **bash**, the usage of the function is as follows:

Hide

```
$ python markov.py [-h] [-c] [-nc NCHARS] [-f FILE] [-s START] [-p] [--save]
                 [--path PATH] [--feed FEED] order length
```

1 Input data for Markov chain training

There are three ways to input the text you want to use to train the Markov chain:

- **path of text file**: you can specify the file path you want to input with the –file argument. For example, to use kafka.txt to train the Markov chain, add in your command

Hide

```
$ --file=kafka.txt
$ -f kafka.txt
```

- **HTML/XML urls**: you can specify the text source from HTML/XML urls with the –feed argument. For example, to use the outside source from url https://www.refsmmat.com/rss.xml (https://www.refsmmat.com/rss.xml), just input

Hide

```
$ --feed=https://www.refsmmat.com/rss.xml
```

- **standard input**: if either path or url is specified, the program will ask you to input through standard input. You can directlt input the text or input < path like

Hide

```
$ < kafka.txt
```

2 Specify the parameters of the Markov Chain

First you have two options to split the text: split by characters or split by words. You can use *-c* to use the split-by-characters mode, which by default splits the whole text into single characters. Also there is an optional argument *-nc* specifying the number of characters to be grouped together each time. For example,

Hide

```
$ -c -nc=3
```

will aplit the text into three-character groups.If there is no *-c* argument, the algorithm will by default split the text into single words.

Then the order, length of the Markov chain should be specified. The order of the Markov chain is the number of determinants of current token, and the length is the number of tokens(characters or workds) you want for the generated text. Also you can use the *-s* argument to specify the starting point, which by default is a random token in the sample.

* Group number and order

Here we illustrate the difference between setting character grouped by 2 and setting the order of the Markov chain to 2. Order = 2 means that the next state of the Markov chain is determined by the previous two tokens. For example, if we are to determine the next character for 'HW', we serch in the state space for the successor of 'HW' and choose one state; suppose we choose '2'. Then the text now becomes 'HW2'(step1). Then we serch for the successor of 'W2'(step2). On the other hand, if we set order = 1 and the group number of charaters to be 2, we will find a two-character token as the successor of 'HW'. For example, we choose token '2A'; then the text now becomes 'HW2A'(step1).

3 Set output format

There are three arguments to determin the output format.

- **print results** If you input the argument *-p* or *–printout*, then the generated text will be printed on the screen.
- **save results** If you input the argument *–save*, then the output will be saved. You can specify the file to save your output by the –path argument. The default path is *generated.txt* in current working path. In a word,

Hide

```
--save
--save --path=fakekafka.txt
```

will store the generated text in *generated.txt* and *fakekafka.txt* respectively.

4 Some comments

- If no token could be generated from the current state, a new random starting point will be chosen.
- Punctuation strings except for "'" are deleted when splitting the sentences.

5 Some examples

Below are some examples illustrating the usage of this generator. We use the file 'kafka.txt' as an illustration.

Hide

```
$ python markov.py --file=kafka.txt --printout --save --path=fakekafka.txt 3 100
$ python markov.py --file=kafka.txt --printout --save --path=fakekafka.txt 3 100 -c
```

will split the text by workds and characters seperately, print the results on the screen and store the outputs in *fakekafka.txt*. Running these two lines, we get

> He would often hear them say how they appreciated all the new work his sister was screaming his mother ran out in front of himself in order to avoid the risk of catching cold for no reason whenever she felt like it One day early in the morning the fresh air had something of warmth mixed in with it The judge however paid no attention to that but sat very comfortably on his chair and half in jest half in explanation Well everything belongs to the court wait in the ante room They had probably been asked to summon him

and

> p u g n a t e d o n ‘ t n e v e r K . w a s a b i g g e s ’ t i m e t h i n g o n c l e r a t h e p o v e r t h e r w h o s t i o n w i t h t h e f l y i n g g i v e

respectively(just as a sample), and store the generated text in *fakekafka.txt*.

If we want to use text from outside resources, we could input a XML/HTML url. For example, running

Hide

```
$ python markov.py --printout --save --feed=https://www.refsmmat.com/rss.xml 3 100 --path=rssout.txt
```

for once can get

> as deep or poetic as Sturgeon’s More than Human but it’s a fun story blemished only by some poor characterization the only woman Penny is an emotional wreck who has to be converted to a fast addition There are other curious features of R like Ross Ihaka’s famous1 example of a function whose return value x is randomly local or global But the performance of a program depends on more than just how quickly the interpreter can execute the instructions For a program doing vectorized operations on large arrays of numbers sure but what about a program that needs different