mostUsedNamesLibrary.py Documentation

George Condei

17/02/2021

Table of contents

Introduction

This library was designed especially for the BitDefender technical interview, but can be reused in any situation since it is not bound to any other file. It is a really simple library, having in its componence only 2 sets and 2 functions. Its main purpose is to help find the most important characters in a literal text, through the *mostUsedNames* function. It runs pretty fast through big texts, going at about 15µs per word (on the PC it was developed). It uses NLTK to tokenize lines and count them in a **defaultdict**, which is then sorted and returned.

Components

1. Sets
    a. The *punctuation* set – Used to verify if a certain word/token is a punctuation mark after which there has to be a word starting with an uppercase letter (Example: "… as it is. Now, returning to…"). Tokenizing this sentence will take the full stop as a token, and we can use this to ignore the word *Now* which is not a name.

    Set composition: punctuation = {".", ",", "!", "?", "-"}

```
if m in punctuation:
    isPunctuation = 1
```

    b. The *stop_words* set – Used to ignore certain stop words from the text that are written with an uppercase letter at the beginning. It contains all the stop words offered by the **NLTK** library.
    Set composition: stop_words = {'ourselves', 'hers', 'between', 'yourself', 'but', 'again', 'there', 'about', 'once', 'during', 'out', 'very', 'having', 'with', 'they', 'own', 'an', 'be', 'some', 'for', 'do', 'its', 'yours', 'such', 'into', 'of', 'most', 'itself', 'other', 'off', 'is', 's', 'am', 'or', 'who', 'as', 'from', 'him', 'each', 'the', 'themselves', 'until', 'below',

‘are’, ‘we’, ‘these’, ‘your’, ‘his’, ‘through’, ‘don’, ‘nor’, ‘me’, ‘were’, ‘her’, ‘more’, ‘himself’, ‘this’, ‘down’, ‘should’, ‘our’, ‘their’, ‘while’, ‘above’, ‘both’, ‘up’, ‘to’, ‘ours’, ‘had’, ‘she’, ‘all’, ‘no’, ‘when’, ‘at’, ‘any’, ‘before’, ‘them’, ‘same’, ‘and’, ‘been’, ‘have’, ‘in’, ‘will’, ‘on’, ‘does’, ‘yourselves’, ‘then’, ‘that’, ‘because’, ‘what’, ‘over’, ‘why’, ‘so’, ‘can’, ‘did’, ‘not’, ‘now’, ‘under’, ‘he’, ‘you’, ‘herself’, ‘has’, ‘just’, ‘where’, ‘too’, ‘only’, ‘myself’, ‘which’, ‘those’, ‘i’, ‘after’, ‘few’, ‘whom’, ‘t’, ‘being’, ‘if’, ‘theirs’, ‘my’, ‘against’, ‘a’, ‘by’, ‘doing’, ‘it’, ‘how’, ‘further’, ‘was’, ‘here’, ‘than’}

2. Functions

a. mostUsedNames(file)

Parameters: **file**: the text is read from this file

Returns: **defaultdict**: returns a sorted defaultdict with the most used names in the text

Example:

```
f = open(sys.argv[1], "r")
dictionaryWithNames = mun.mostUsedNames(f)
```

This call will return a dictionary with the most used names in this text: https://raw.githubusercontent.com/wess/iotr/master/lotr.txt

('Frodo', 1484)

('Gandalf', 1097)

('Sam', 876)

('Bilbo', 706)

……

b. printDict(dictionary, number)

Parameters: **dictionary**: the dictionary you want to print

number: int type, represents the number of components you want to print from the beginning of the dictionary

Example:

```
mun.printDict(dictionaryWithNames, k)
```

Functionality

Using this library lets the user find the most important characters in a text, even if it is thousands of pages (the speed at which the application runs depends on the number of words – higher = slower). There is a slight delay at the beginning because of the NLTK package updates. After running the program the first time, you can comment the lines where the NLTK downloads its packages.

```
#nltk.download("punkt", quiet = True)
#nltk.download("stopwords", quiet = True)
```

On the current PC, it processes 600.000 words in about 10 seconds, give or take. Performance may depend on the components of the PC on which the application is ran.

Conclusions

The application could be improved by changing the algorithm that eliminates the stop words from the text. The text could also be preprocessed, deleting all the words that do not start with an uppercase letter.

The library is extremely simple since it only has 2 functions and 2 sets so it could be adapted to any application and expanded a lot more to meet certain requirements.
Having these said, the library and its functions are really simple and it is certain that there are some other solutions to this problem that are marginally better in efficiency, but its small code size allows this library to be modified and used in any other way.