CS 590 NLP

HW1

Regex, Normalization, Edit Distance

Due 9/15 11:59 pm

In this homework, you will be creating a simple spell checker program in python. This program will function as follows (note the '>>>' indicate actions from the python shell):

```
>>> process_regex("dante_inf.txt")
Processing file...
Output stored to "regex.txt"

>>> normalize_text("regex.txt")
Normalizing text...
Output stored to "dictionary.txt"

>>> spell_checker()
----------------------------------------
Welcome to the spell checker!
Please enter a text to check spelling or enter quit to exit the program.
----------------------------------------
Enter text to be checked: The rain in Spain, falls mainly on the plain.
No misspellings detected!

Enter text to be checked: The raen in Spain, fals mainly on the plain.
Misspelling - Suggestion
--------------------------------
raen – rain
fals - falls

Enter text to be checked: quit
Goodbye!

>>>
```

<u>Spell Checker</u>

Your spell checker will follow the following rules:

1. If a word in the input text is not in the <u>dictionary</u> it is considered a misspelled word.
    a. Note you must avoid thinking that punctuation indicates misspelling (e.g. Spain,)

2. If a word is misspelled, then the spell checker should find the closest word to it by leveraging the minimum edit distance formula discussed in class. Specifically, the word in the dictionary which has the smallest minimum edit distance to the misspelled word should be suggested.
    a. (You may try some optimizations such as storing already calculated edit distances, or only checking a subset of dictionary words, if you want to speed your program up)
3. The spell checker should be called by a function called "**spell_checker()**"

Dictionary

The Dictionary will be created by you. This "dictionary" will be composed of words processed from "Dante's Inferno" (https://www.gutenberg.org/ebooks/41537) (I would suggest the Plain Text UTF-8 file).

Specifically, the book file will be processed (either line by line or the entire body of text at once) and the unique tokens will be stored in some dictionary file for easy retrieval for the spell checker to load in.

To create the dictionary from the text file, you will be leveraging both REGEX and Text Normalization skills.

The dictionary should be created as follows:

1. For the text in the file, use REGEX (find, substitute) to:
    a. Replace all British English spellings for words that have an extra 'u' (colour, neighbour, etc.) with American English spellings (e.g. color, neighbor, etc.). Note this should only be for those words with American English removes the 'u'. No other differences in spelling need to be addressed. (Be careful about replacing words like 'your'.)
    b. Replace titles (Dr. , Mr., Ms., Mrs.) with appropriate expansions of words (Doctor, Mister, Miss, Misses). Note you only need to replace the listed titles.
    c. Output resulting text to additional text file called "regex.txt".
    d. This should be run by a function called "**process_regex(path_to_text_file)**"
2. For the text in "regex.txt", normalize it to reduce the number of overall types (unique tokens).
    a. This is left open for you to do, but most likely will include some lowercasing of words, some splitting of texts, and other cleaning of text discussed in class.
    b. The resulting tokens will result in your dictionary and should be output to a "dictionary.txt" file, where each line is a single word in the dictionary. Note that the dictionary should be sorted alphabetically (as dictionary usually are).
    c. This should be run by a function called "**normalize_text("regex.txt")**"

Additional Rules (MUST BE FOLLOWED):

1. The code should be written in python 3.
2. Standard libraries should only be used for this assignment, ie. I shouldn't have to download extra libraries to run your code.
3. For regex, you can use the "re" library (import re) (https://www.w3schools.com/python/python_regex.asp)
4. For the regex step, you should only use regex to process the text. That means **NO SPLITTING TEXTS INTO SEPERATE WORDS DURING REGEX STEP!** You should only be processing the texts by using the commands in the re python library (findall, search, sub). (Although regex has a split() option, sub() and findall() are applied to the entire text and therefore are more efficient than splitting and then checking the regex against every separate text).
    a. Splitting is fine during normalization
    b. Depending on how you read the file in, you may either read it in line by line or the entire text at once. Either choice is fine.
5. Normalization should be done via "simple" python libraries. (E.g. **you shouldn't be using NLTK to normalize text for you.**) The idea is for you to think through and apply these ideas, just using another library is not actually thinking through the problem.
6. You should make your code modular to the different steps. (e.g. **at least one function for the regex step, at least one function for the normalization step, at least one function for the spell check interaction step**). (You probably will have more functions to help your main functions)
7. You should only hand in one python file: **USERNAME_HW1.py**.
    a. I will be able to have my own copy of the text file, so **YOU SHOULDN'T HAND IN THE TEXT FILES.**
    b. This is also why it is important to create "regex.txt" and "dictionary.txt" for each of the aforementioned steps.
8. You should be adding comments to document and communicate your thought process for all of the steps. **If I can't understand why you perform an action, then I can't credit you for performing that action.**

Grading

Assignment will be graded as follows:

Description	**Points**
Code Runs	10
Regex Implementation	20
Normalization Implementation	20
Spell Checker Implementation/Correctness	20
Outputs follow rules	5
Documentation (Comments, functions, etc)	25
Total:	**100**

- **If the code does not run, I cannot grade it well.** (More points than 10 can be lost if the code cannot be run, as I will not be able to fully test the implementations of the other functions).
- Note that the normalization implementation is the most subjective between students. Here I will be looking for thoughtfulness beyond simply splitting the texts. It would be wise to add in description for normalization steps (“# to capture words that end with …”) as this will help me understand your thought process as well.
- **Breaking of the additional rules can result in applied penalties.** (Always make sure you are checking against the rules)

Suggestions

- **Documentation is key for showing your effort in this homework.** Make sure you are noting why you make certain decisions all throughout your code.
- The slides for previous classes are posted, so please refer to these and the book for ideas during implementation.
- Start simple, build up complexity. You should always make sure your new ideas being added do not cause your program to crash. So starting simple is the best way to a) maintain the ability to keep your code running, b) add in comments for documentation and thought process as you add more code.
- Regexpal.com can be useful to test ideas for the regex step.
- Work through the homework yourself, rather than sharing ideas (especially not code) with other students. **As a reminder, plagiarism (or sharing) of code is strictly prohibited.** This assignment is complex enough that significant overlap between students will be suspicious.
- If you have not worked with python before, w3schools can help you translate your previous coding experience to python (https://www.w3schools.com/python/default.asp)
- Stop by office hours to discuss ideas. I am always happy to help you think through your process!