Assignment 6 Python - Text Analysis

Due Mar 19, 2016 by 12pm **Points** 90 **Submitting** an external tool
Available until Mar 21, 2016 at 12pm

This assignment was locked Mar 21, 2016 at 12pm.

- No code is provided.
- **Test** (http://codeskulptor.appspot.com/owltest/?urlTests=comp130.assignment_text_analysis_tests.py&urlPylintConfig=comp130.pylint_config.py)

(5 points) Code style

Relatively little code is necessary for this assignment, so code style and documentation are worth less.

Regular expressions

1. **(4 points)**

   Many words beginning with the letters “sl” have related meanings. Consider, “slip“, “slide”, “slosh”, “slick”, and “slather”, for example. The “sl” root comes from Proto-Indo-European, the proposed language ancestor of Greek, Latin, and Sanskrit.

   We we limit ourselves to words *beginning* with “sl”. While that will miss words like “re-slide”, it avoids words like “island”.

   Write a function `findall_sl(text)` that takes a text string, searches it with `re.findall()`, and returns the result. You must supply the appropriate RE argument to the `re.findall()` so that it searches for words beginning with “sl”. The “s” may be capitalized or not.

2. **(8 points)**

   Write a function `findall_triple_vowel(text)` that takes a text string, searches it with `re.findall()`, and returns the result. You must supply the appropriate RE argument to the `re.findall()` so that it searches for anything (e.g., words, abbreviations, and Roman numerals) that contains three or more consecutive, identical vowels. This search should be case-insensitive.

3. **(12 points — not tested)**

   We want to find anything that relates to the 1980s. Create a RE to search for references to the decade or its years.

   Among the things you would want to find are “1984”, “'80s”, and “eighties”. You are *not* expected to search for terms relating to things that happened during the decade, such as the Berlin Wall being torn down. Only search for explicit references to the years.

Think about what search terms are relevant, and combine them into one RE. There is a lot of variation in solutions on this problem because of the loose specification.

Write a function `findall_80s(text)` that takes a text string, searches it with `re.findall()`, and returns the result. Allow the beginning of the search term to be capitalized. You must supply the appropriate RE argument to the `re.findall()` so that it searches for such terms `.`

Text statistics

Combine your file reading, word filtering, and statistics functions from previous exercises and assignments to examine some properties of text files. For simplicity, we'll consider any word, abbreviation, or number to be a word. Furthermore, consider words to be case-insensitive, so that “the” and “The” are counted as the same word.

4. **(5 points)**

   Write a function `count_distinct_words(filename)` that returns a count of the number of *distinct* words that occur in the provided text file.

5. **(10 points)**

   Write a function `median_word(filename)` that returns a word that has the median of all the number of word occurrences. (As before in the course, use the lower median.) Note that multiple words can have the median number of occurrences, and you should only report one of them. If the file contains no words, then the function should return `None`.

Word-sequence analysis

In class, you are generalizing your original word- (i.e., n-gram) counting program to word-sequence-counting. Also in class, you had written code to find word frequencies and word successors. In the following exercises, you'll write code that combines all these ideas to determine the frequencies of word-sequence successors, also known as a Markov chain.

These problems have two additional parameters that the in-class exercises didn't. One indicates whether we should treat punctuation like words. The motivation for this is that when analyzing an author's style, we would want to not only look at the word usage but also the punctuation usage, as well. Another indicates whether we should treat words as case-sensitive or not. The in-class exercises treated, for example, "the" and "The" as distinct words, but we might also want to treat them as equivalent.

Again, you've written most of the necessary code before. You now need to combine the pieces appropriately. We strongly encourage you to decompose your code into smaller useful functions. Use the same word-finding RE given in the video.

On the next assignment, you'll use Markov chains to generate text.

6. **(28 points)**

Define a function `wordseq_successor_counts(filename, seq_size, include_punc, is_case_sensitive)`. It returns a default dictionary, where each key is a `seq_size`-element tuple of words. Each key's value is a Counter mapping distinct successor words to their counts.

For example, in comp130_EightDaysAWeek.txt, the phrase “love babe” occurs eight times. Six times it is followed by a comma, and twice by “just”. If the sequence size is two and we are counting punctuation, then `("love", "babe")` should be a key which maps to a Counter `Counter({"," : 6, "just" : 2})`.

Hints: Start with your code for `wordseq_counts_file()` and modify it, rather than calling it. At first, work on a simplified version that ignores the last two parameters, then see where you need to add conditionals for those arguments.

7. **(18 points)**

Define a function `wordseq_successor_frequencies(filename, seq_size, include_punc, is_case_sensitive)`. It returns a default dictionary, where each key is a `seq_size`-element tuple of words. Each key's value is a regular (non-default) dictionary mapping distinct successor words to their frequencies, i.e., their percentage of occurrence.

Continuing our example, `("love", "babe")` should be a key which maps to a dictionary `{"," : 0.75, "just" : 0.25}`.

This function should call the previous function, then create a new dictionary where the inner dictionaries (mapping strings to frequencies) are created from the corresponding Counters (mapping strings to counts).