Assignment 4

Status: Completed
Your identity is visible during marking.

Deadline: 1 Oct 2018, 23:55

Weekly assignment A4

Part A

Basic Text Handling in NLTK

1. Write a function `sentence_segmentation(raw_text)` that takes a piece of raw text, performs sentence segmentation using NLTK's default sentence segmenter, and returns the resulting list of sentences.

2. Write a function `word_tokenization(raw_text)` that takes a piece of raw text and returns its tokenized form (i.e. a list of words) using NLTK's default word tokenizer.

3. Write a function `print_corpus_stats(sentences,words)` that takes the list of sentences returned by the function `sentence_segmentation(raw_text)` and the list of words returned by the function `word_tokenization(raw_text)`, counts the total number of sentences, the total number of words, and produces output like the following:

```
==================================================================
                          Corpus Stats

Number of Sentences:                   3588
Number of Words:                       69408
==================================================================
```

   Hint: The print function can take multiple arguments, which it will by default all print on the same line. For more elaborate printing options, you might find the `.format()` method of strings useful to combine several strings / other objects into one string (see https://docs.python.org/3.1/library/stdtypes.html#str.format).

4. You are being provided with a list of five NLTK's part of speech taggers trained on the brown

corpus (See line `taggers = train_nltk_taggers()` in the provided (see below) script file `assignmentA4_1.py` (line 61)). Write a function `tag_and_print_text(taggers,words)` which takes the list of taggers and the list of words, and then for each of the tagger it does the following:

1. Tag the list of words by calling the tagger's tag function (i.e. `tagger.tag(words)`). Call the resulting list of tuples `tagged_words`.

2. Compute statistics about the number of occurrence of nouns, verbs, and adjectives in `tagged_words`. For this purpose, write a function `compute_tags_stats(tagged_words)` and call it from within the function `tag_and_print_text(taggers,words)`. The function `compute_tags_stats(tagged_words)` should return a tuple consisting of number of nouns, number of verbs, and number of adjectives.
   A word is a *noun* if its tag is one of NN, NNP, NNPS, and NNS, a *verb* if its tag is one of VB, VBD, VBG, VBN, and VBZ, and an *adjective* if its tag is one of JJ, JJR, and JJS.

3. Print the tag statistics. For this purpose, write a function `print_tags_stats(nr_nouns,nr_verbs,nr_adjs)` that takes the number of nouns, verbs, and adjectives returned by the function `compute_tags_stats(tagged words)` and prints a table along the lines of the one below.

```
-----------------------------------------------------------------
                             Tags Stats

Number of Nouns:               56423
Number of Verbs:               4700
Number of Adjectives:          1832
-----------------------------------------------------------------
```

   Again call the function `print_tags_stats(nr_nouns,nr_verbs,nr_adjs)` from within the function `tag_and_print_text(taggers,words)`.

5. Write a function `compare_taggers(taggers,words)` that takes the list of taggers and the list of words. First of all, tag the list of words with the trigram tagger (which is the last tagger in the provided list of taggers, i.e., `taggers[4]`) and use its output as a gold standard. Next, tag the list of words using each of the remaining taggers (i.e., `taggers[0:4]`) one by one and compare the output to the gold standard. Compute a simple 'score' using the formula

number of correctly predicted tags / total number of tags, and produce the following type of output for each of the remaining four taggers:

```
Score: 0.13710235131396958
```

Hint: You might find the function `zip()` useful (See here: https://docs.python.org/3/library/functions.html#zip).

Script File: assignmentA4_1.py

Part B

Comparing word counts

For the B-part of the assignment, you are asked to implement a small experiment comparing word frequencies between different (sub)corpora.

The document we shall work with is NLTK's version of *Alice's Adventures in Wonderland* by Lewis Carroll. Please have a look at Section 1.1 of Chapter 2 in the NLTK book if you need a more elaborate refresher, but this is how you load the text as a *list of strings* (words/tokens including punctuation):

```
import nltk
alice_text = nltk.corpus.gutenberg.words('carroll-alice.txt')
```

If the above raises a bunch of errors, it may be that you have to download and install the Gutenberg corpora first. You can do this by doing as followed in an interactive session (the bits in **bold** in the downloader is what you have to type...)

```
>>> import nltk
>>> nltk.download()
NLTK Downloader
---------------------------------------------------------------------------
 d) Download l) List u) Update c) Config h) Help q) Quit
---------------------------------------------------------------------------
Downloader> d

Download which package (l=list; x=cancel)?
  Identifier> gutenberg
    Downloading package gutenberg to somewhere_on_your_system/nltk_data...
      Unzipping corpora/gutenberg.zip.
```

```
---------------------------------------------------------------------------
 d) Download l) List u) Update c) Config h) Help q) Quit
---------------------------------------------------------------------------
Downloader> q
True
>>>
```

Now you should be able to use the Gutenberg example corpora for the rest of the assignment. We have split the task in to smaller steps, but they are meant to be combined into one larger programme.

1. Convert all the words in the text to lower case and divide the text into two coherent halves, two lists of equal length. These will be your subcorpora.

2. Write a function `count_words(text)` that takes a subcorpus `text` and returns a dictionary of word counts. That is, each key in the returned dictionary should be a string representing a word from `text`, and the associated value should be an integer representing the number of times that word occurred in `text`.

3. Now write a function `fraction_dist(numerator_dist,denominator_dist)`, that takes two word count dictionaries of the kind produced by `count_words`, and returns a ratio distribution, that is, a dictionary that for each word *w* in the denominator distribution contains

   $$\text{ratio of } w = \frac{\text{frequency of } w \text{ in the numerator distribution}}{\text{frequency of } w \text{ in the denominator distribution}}$$

   You do not have to consider words that do not occurr in the denominator distribution, but you to have to handle words from the denominator distribution that do not occur in the numerator distribution.

4. Finally, tie everything together: when you call your script, the ratio distribution of the second half of *Alice's Adventures* with respect to the first half should be printed to the screen, one word per line, in sorted order so that the words with the highest ratio come last. Each line should look like this:

   ```
   3.0     watch
   3.25    asked
   ```

   That is, each line contains the ratio followed by the word.
   Hint: have a look at the Python built-in `sorted()` and the `.items()` method for dictionaries.

5. What are the 10 words with the highest ratios? Can you say anything about those words?

Hand in a script that does everything contained in steps 1–4. The answer to question 5 you can leave in the comments.

Hemanth Kumar Battula , 1 Oct 2018 23:23

File name: assignmentA4_1.py (2,9 KB)

Status set to: To be marked

Hemanth Kumar Battula , 1 Oct 2018 23:23

File name: AssignmentA4_partB.py (1,0 KB)

Status set to: To be marked

Hemanth Kumar Battula , 1 Oct 2018 23:45

Status set to: To be marked

Comment: PartB- 5:

The last ten words with highest ratio, are the words which mostly come in the second part of the text and not much in first part. The highest ratio of ten words are:
(6.0 song)(7.0 five)(7.0 ground)(8.0 shouted)(8.0 puzzled)(10.0 jury)(12.0 beautiful)
(17.666666666666668 hatter)(17.75 queen)(19.0 dormouse).
The characters like queen,dormouse or hatter comes in the second part of the text than the first part.

Chatrine Qwaider , 20 Oct 2018 19:25

Status set to: Completed