PIC 16, Winter 2018 – Assignment 8F

Assigned 3/2/2018. Code (a single `.py` file) due by the end of class 3/7/2018 on CCLE. Hand in a printout of this document with the self-assessment portion completed by the end of class on 3/7/2018.

In this assignment, you will revisit your analysis of Agatha Christie's "The Mysterious Affair at Styles" from Assignment 2W (see solution if you didn't do it) but use NLTK to make the job much simpler!

Task

Take a look at this article about declining metrics of language use in Agatha Christie's later books. Is it possible to detect Alzheimer's or other mental conditions from a person's writing? We're not going to answer that question today, but natural language processing is a very powerful and promising field!

1. Before you begin, execute the command `nltk.download()`. Select the "All Packages" tab and use it to download averaged_perceptron_tagger, hmm_treebank_pos_tagger, maxent_treebank_pos_tagger, punkt, and wordnet. If at any point in this assignment you get a LookupError like "Resource 'taggers/**maxent_treebank_pos_tagger**/english.pickle' not found. Please use the NLTK Downloader to obtain the resource", find the resource (example in bold above) in in the downloader "All Packages" tab, download it, and try again.
2. Use the instructions from 3.1 of the NLTK book to have Python download the plain text of The Mysterious Affair at Styles from Project Gutenberg. In Python 2.7, the `urlopen` function is located directly in the `urllib` package; not `urllib.request`. Be sure to call the `decode('utf8')` (as in the example) after reading from the URL; this saves the data as a Unicode string (hence you may see a `u` before every string), which NLTK understands better than the raw data.
3. There's some text before the book begins and after it ends that we don't want to include in the analysis. Chapter 3.1 of the NLTK book suggests that you use the string's `find` and/or `rfind` methods to look for phrases that mark the beginning and ending of the text. Find the index corresponding with the *beginning* of the *last* (not first, as suggested in the example) occurrence of the string "CHAPTER I. I GO TO STYLES", and find the index corresponding with the *end* of the *last* occurrence of the string "THE END". Use these indices to discard data before and after these markers. Print the beginning and end of the leftover string to confirm correctness.
4. Apply your code from Assignment 1F to generate a dictionary of the words in the book and the frequency of each. Given the same list of words in the book, does NLTK's `FreqDist` function produce the same result? It should, and you're welcome to use it instead of your own algorithm from 1F in the parts to follow. Calculate the "lexical diversity" as defined in 1.4 of the NLTK book, except that you can use the `FreqDist` object you just made rather than a new set to count unique words.
5. We noticed in 1F that the `split()` method left punctuation marks at the end of words, counting these as distinct from the word itself. Improve your frequency distribution dictionary by using NLTK's `word_tokenize` function instead of `split` to separate the words into a list. Also, before you do that, rid the original text of underscores ("_"), which are used here to indicate emphasis, as these shouldn't make the enclosed word distinct. Calculate the lexical diversity from this `FreqDist`. You can see that our crude estimate from 1F was a bit too generous!
6. It doesn't really make sense to count "Nature" separately from "nature", or "Nature" separately from "Natural" or "Naturally" if we're trying to get a sense of vocabulary. Improve your `FreqDist` by eliminating words that are distinct only in letter case.
7. Also eliminate those that differ only in affixes (as described in 3.6. You're welcome to try either of the stemmers or the WordNet lemmatizer for this. The stemmer is probably more accurate, because

the lemmatizer does not reduce all words to their lemma, but the downside is that the stemmers returns *stems* which may not be words at all.).

8. Then again, some words like “point” are used in several different senses of the word, and intuitively we should somehow factor this into the concept of lexical diversity. Turn the `word_tokenize`d list from 5 into an `nltk.Text` object (3.1) and use the `concordance` function (1.3) on “point” to see what I mean.
9. (Optional) WordNet maintains a structure of “synsets”, which are like distinct concepts or *meanings* of words. It would be nice if we could determine the WordNet synset with which each occurrence of the word corresponds; that way we could consider the different WordNet synsets in our lexical diversity calculation. The Lesk algorithm attempts to do just that.
    a. Print all the different WordNet synset `definition`s for the word “point”.
    b. The Lesk algorithm needs the sentence the word appears in to determine the corresponding synset. Write a function that returns a list of all the sentences in the original text in which the word appears. Print all the sentences in which the word “point” appears. Chapter 3.8 of the book will be useful.
    c. Use the `lesk` function to determine the synset definition and part of speech of “point” in each sentence. (It does very poorly.)
    d. Maybe if we tag the parts of speech first using the `pos_tag` function (Chapter 5) and provide that information as the (optional) second argument to the `lesk` function, it will do a better job? Try it! Note that `pos_tag` doesn’t provide the part of speech for just a single word, and you’ll have to convert between the output provided by the part of speech tagger and a single letter representing the part of speech required by `lesk`. This doesn’t have to work for *all* possible parts of speech, just the parts of speech of “point” in the text.
    This still does poorly, but it was a good idea IMO. It seems like the Lesk algorithm just tries to match up words from the context sentence with words from the *synset definition*, which is probably too short for this approach to work well. There’s certainly a lot of work left to be done in the field of natural language processing!
10. Use the `most_common` method of the `FreqDist` from step 5 (not step 6, because that `FreqDist` might be made up of stems rather than words) to print out the most-used 30 words.
11. Oops, even after all that, some of the common “words” are actually punctuation marks. We could remove those from the dictionary, but let’s leave the dictionary alone and instead just show the top 30 words that begin with a *letter*. Hint: Turn the dictionary-like `FreqDist` object into a list of word/frequency (key/value) pairs, sort the list by the word *frequency* numerically (not by word alphabetically) in descending order, and print out the 30 most commonly used words that begin with a letter and their frequency.

Self-Assessment

Print this assignment document (front and back) and check off the steps you completed successfully. At 10 points per item, indicate your score: