HLT – Human Language Technologies

Homework 2: Word Guessing Game

Objective: Use Python and NLTK features to explore a text file and create a word guessing game

Turn in: your Python .py file (Do not turn in a Jupyter notebook)

Instructions:

1. Download the anat19.txt file from Piazza and save it in the same folder as your Python program. The file is the text from one chapter of an anatomy textbook. Send the filename to the main program in a system argument. If no system arg is present, print an error message and exit the program.
2. Read the input file as raw text. Calculate the lexical diversity of the tokenized text and output it, formatted to 2 decimal places. Lexical diversity is the number of unique tokens divided by the total number of tokens. Lexical diversity indicates the richness of vocabulary in a text. For example, a lexical diversity of 0.05 means that 5% of the words in a text are unique.
3. Write a function to preprocess the raw text:
    a. tokenize the lower-case raw text, reduce the tokens to only those that are alpha, not in the NLTK stopword list, and have length > 5
    b. lemmatize the tokens and use set() to make a list of unique lemmas
    c. do pos tagging on the unique lemmas and print the first 20 tagged items (see the pos_NLTK notebook in Part 2 of the GitHub)
    d. create a list of only those lemmas that are nouns
    e. print the number of tokens (from step a) and the number of nouns (step d)
    f. return tokens (not unique tokens) from step a and nouns from the function
4. Make a dictionary of {noun:count of noun in tokens} items from the nouns and tokens lists; sort the dict by count and print the 50 most common words and their counts. Save these words to a list because they will be used in the guessing game.
5. Make a guessing game function:
    a. give the user 5 points to start with; the game ends when their total score is negative, or they guess '!' as a letter
    b. randomly choose one of the 50 words in the top 50 list (See the random numbers notebook in the Xtras folder of the GitHub)
    c. output to console an underscore _ for each letter in the word
    d. ask the user for a letter
    e. if the letter is in the word, print 'Right!', fill in all matching letter _ with the letter and add 1 point to their score
    f. if the letter is not in the word, subtract 1 from their score, print 'Sorry, guess again'
    g. guessing for a word ends if the user guesses the word or has a negative score
    h. keep a cumulative total score and end the game if it is negative (or the user entered '!') for a guess
    i. right or wrong, give user feedback on their score for this word after each guess

Grading Rubric:

Element	Points
Appropriate comments, white space, functions	10
Step 1	10
Step 2	10
Step 3	20
Step 4	10
Step 5	40
Total	100

Sample run of the game:

```
       Let's play a word guessing game!
       _ _ _ _ _ _
       Guess a letter:e
       Right! Score is 6
       _ e _ _ e _
       Guess a letter:a
       Sorry, guess again. Score is  5
       _ e _ _ e _
       Guess a letter:i
       Sorry, guess again. Score is  4
       _ e _ _ e _
       Guess a letter:u
       Sorry, guess again. Score is  3
       _ e _ _ e _
       Guess a letter:o
       Sorry, guess again. Score is  2
       _ e _ _ e _
       Guess a letter:t
       Sorry, guess again. Score is  1
       _ e _ _ e _
       Guess a letter:s
       Right! Score is 2
       _ e s s e _
       Guess a letter:m
       Sorry, guess again. Score is  1
       _ e s s e _
       Guess a letter:l
       Right! Score is 2
       _ e s s e l
       Guess a letter:v
       Right! Score is 3
       v e s s e l
       You solved it!

       Current score: 3

       Guess another word
       _ _ _ _ _ _ _ _
Guess a letter:
```