Assignment 2

April 29, 2022

---

*You are currently looking at **version 1.0** of this notebook. To download notebooks and datafiles, as well as get help on Jupyter notebooks in the Coursera platform, visit the Jupyter Notebook FAQ course resource.*

---

1 Assignment 2 - Introduction to NLTK

In part 1 of this assignment you will use nltk to explore the Herman Melville novel Moby Dick. Then in part 2 you will create a spelling recommender function that uses nltk to find words similar to the misspelling.

1.1 Part 1 - Analyzing Moby Dick

```
In [ ]: import nltk
        import pandas as pd
        import numpy as np

        # If you would like to work with the raw text you can use 'moby_raw'
        with open('moby.txt', 'r') as f:
            moby_raw = f.read()

        # If you would like to work with the novel in nltk.Text format you can use 'text1'
        moby_tokens = nltk.word_tokenize(moby_raw)
        text1 = nltk.Text(moby_tokens)
```

1.1.1 Example 1

How many tokens (words and punctuation symbols) are in text1?

This function should return an integer.

```
In [ ]: def example_one():

            return len(nltk.word_tokenize(moby_raw)) # or alternatively len(text1)

        example_one()
```

1.1.2 Example 2

How many unique tokens (unique words and punctuation) does text1 have?
This function should return an integer.

```
In [ ]: def example_two():

            return len(set(nltk.word_tokenize(moby_raw))) # or alternatively len(set(text1))

        example_two()
```

1.1.3 Example 3

After lemmatizing the verbs, how many unique tokens does text1 have?
This function should return an integer.

```
In [ ]: from nltk.stem import WordNetLemmatizer

        def example_three():

            lemmatizer = WordNetLemmatizer()
            lemmatized = [lemmatizer.lemmatize(w,'v') for w in text1]

            return len(set(lemmatized))

        example_three()
```

1.1.4 Question 1

What is the lexical diversity of the given text input? (i.e. ratio of unique tokens to the total number of tokens)
This function should return a float.

```
In [ ]: def answer_one():


            return # Your answer here

        answer_one()
```

1.1.5 Question 2

What percentage of tokens is 'whale'or 'Whale'?
This function should return a float.

```
In [ ]: def answer_two():


            return # Your answer here

        answer_two()
```

1.1.6 Question 3

What are the 20 most frequently occurring (unique) tokens in the text? What is their frequency?

This function should return a list of 20 tuples where each tuple is of the form `(token, frequency)`. The list should be sorted in descending order of frequency.

```
In [ ]: def answer_three():


            return # Your answer here

        answer_three()
```

1.1.7 Question 4

What tokens have a length of greater than 5 and frequency of more than 150?

This function should return an alphabetically sorted list of the tokens that match the above constraints. To sort your list, use `sorted()`

```
In [ ]: def answer_four():


            return # Your answer here

        answer_four()
```

1.1.8 Question 5

Find the longest word in text1 and that word's length.

This function should return a tuple `(longest_word, length)`.

```
In [ ]: def answer_five():


            return # Your answer here

        answer_five()
```

1.1.9 Question 6

What unique words have a frequency of more than 2000? What is their frequency?

"Hint: you may want to use `isalpha()` to check if the token is a word and not punctuation."

This function should return a list of tuples of the form `(frequency, word)` sorted in descending order of frequency.

```
In [ ]: def answer_six():


            return # Your answer here

        answer_six()
```

1.1.10 Question 7

What is the average number of tokens per sentence?

This function should return a float.

```
In [ ]: def answer_seven():


            return # Your answer here

        answer_seven()
```

1.1.11 Question 8

What are the 5 most frequent parts of speech in this text? What is their frequency?

This function should return a list of tuples of the form `(part_of_speech, frequency)` sorted in descending order of frequency.

```
In [ ]: def answer_eight():


            return # Your answer here

        answer_eight()
```

1.2 Part 2 - Spelling Recommender

For this part of the assignment you will create three different spelling recommenders, that each take a list of misspelled words and recommends a correctly spelled word for every word in the list.

For every misspelled word, the recommender should find find the word in `correct_spellings` that has the shortest distance*, and starts with the same letter as the misspelled word, and return that word as a recommendation.

*Each of the three different recommenders will use a different distance measure (outlined below).

Each of the recommenders should provide recommendations for the three default words provided: `['cormulent', 'incendenece', 'validrate']`.

```
In [ ]: from nltk.corpus import words

        correct_spellings = words.words()
```

1.2.1 Question 9

For this recommender, your function should provide recommendations for the three default words provided above using the following distance metric:

Jaccard distance on the trigrams of the two words.

This function should return a list of length three: `['cormulent_reccomendation', 'incendenece_reccomendation', 'validrate_reccomendation']`.

```
In [ ]: def answer_nine(entries=['cormulent', 'incendenece', 'validrate']):


            return # Your answer here

        answer_nine()
```

1.2.2 Question 10

For this recommender, your function should provide recommendations for the three default words provided above using the following distance metric:

Jaccard distance on the 4-grams of the two words.

This function should return a list of length three: `['cormulent_reccomendation', 'incendenece_reccomendation', 'validrate_reccomendation']`.

```
In [ ]: def answer_ten(entries=['cormulent', 'incendenece', 'validrate']):


            return # Your answer here

        answer_ten()
```

1.2.3 Question 11

For this recommender, your function should provide recommendations for the three default words provided above using the following distance metric:

Edit distance on the two words with transpositions.

This function should return a list of length three: `['cormulent_reccomendation', 'incendenece_reccomendation', 'validrate_reccomendation']`.

```
In [ ]: def answer_eleven(entries=['cormulent', 'incendenece', 'validrate']):


            return # Your answer here

        answer_eleven()
```