Assignment 2

July 20, 2019

---

*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 [13]: import nltk
         import pandas as pd
         import numpy as np
         nltk.download('punkt')
         nltk.download('averaged_perceptron_tagger')

         # 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)
```

```
[nltk_data] Downloading package punkt to /home/jovyan/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /home/jovyan/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

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 [4]: def answer_one():


            return (len(set(text1))/len(text1))

        answer_one()

Out[4]: 0.08139566804842562
```

1.1.5 Question 2

What percentage of tokens is 'whale'or 'Whale'?

This function should return a float.

```
In [17]: def answer_two():

             whale=[w for w in text1 if (w=='whale') | (w=='Whale')]
             return (len(whale)/len(text1))*100

         answer_two()

Out[17]: 0.4125668166077752
```

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 [22]: from nltk.probability import FreqDist
         def answer_three():

             dist=FreqDist(text1)

             return list((sorted(dist.items(), key = lambda kv:(kv[1], kv[0]),reverse=True))[:20

         answer_three()

Out[22]: [(',', 19204),
          ('the', 13715),
          ('.', 7308),
          ('of', 6513),
          ('and', 6010),
          ('a', 4545),
          ('to', 4515),
          (';', 4173),
          ('in', 3908),
          ('that', 2978),
          ('his', 2459),
          ('it', 2196),
          ('I', 2097),
          ('!', 1767),
          ('is', 1722),
          ('--', 1713),
          ('with', 1659),
          ('he', 1658),
          ('was', 1639),
          ('as', 1620)]
```

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 [30]: from nltk.probability import FreqDist
         def answer_four():

             dist=FreqDist(text1)
             return sorted(set([w for w in text1 if (len(w)>5) & (dist[w]>150)]))

         answer_four()
```

```
Out[30]: ['Captain',
          'Pequod',
          'Queequeg',
          'Starbuck',
          'almost',
          'before',
          'himself',
          'little',
          'seemed',
          'should',
          'though',
          'through',
          'whales',
          'without']
```

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 [42]: def answer_five():

             maxlen=max([len(w) for w in text1 ])
             longw= [w for w in text1 if len(w)==maxlen]
             return (longw[0],maxlen)

         answer_five()
```

```
Out[42]: ("twelve-o'clock-at-night", 23)
```

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 [56]: from nltk.probability import FreqDist
         def answer_six():

             dist=FreqDist(text1)
             return sorted([(fre[1],fre[0]) for fre in dist.items() if (fre[1]>2000) and (fre[0]

         answer_six()
```

```
Out[56]: [(13715, 'the'),
          (6513, 'of'),
          (6010, 'and'),
          (4545, 'a'),
          (4515, 'to'),
          (3908, 'in'),
          (2978, 'that'),
          (2459, 'his'),
          (2196, 'it'),
          (2097, 'I')]
```

1.1.10 Question 7

What is the average number of tokens per sentence?

This function should return a float.

```
In [63]: from nltk.tokenize import sent_tokenize
         from nltk.tokenize import word_tokenize
         def answer_seven():

             moby_sentence = sent_tokenize(moby_raw)
             add=np.mean([len(word_tokenize(w)) for w in moby_sentence])
             return add

         answer_seven()
```

```
Out[63]: 25.881952902963864
```

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 [16]: from nltk.probability import FreqDist
         import nltk
         def answer_eight():

             pos=nltk.pos_tag(text1)
             dist=FreqDist([f for (p,f) in pos])
             count=dist.most_common()[:5]
```

```
            return count

        answer_eight()
```

```
Out[16]: [('NN', 32730), ('IN', 28657), ('DT', 25867), (',', 19204), ('JJ', 17620)]
```

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 [4]: from nltk.corpus import words
        nltk.download('words')

        correct_spellings = words.words()
```

```
[nltk_data] Downloading package words to /home/jovyan/nltk_data...
[nltk_data]   Unzipping corpora/words.zip.
```

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

            correct_spellings = words.words()
            jd=1
            corrected=[]
            for w in entries:
                first_let=lambda x: x if (x[0] == w[0]) else ''

                same_start=np.array(list(map(first_let,correct_spellings)))
                same_start=(same_start[same_start!=''])

                set1 = set(nltk.ngrams(w,n=3))
                compute_jd = lambda x: (nltk.distance.jaccard_distance(set1,set(nltk.ngrams(x,n=
                jd=list(map(compute_jd,same_start))
                corrected.append(sorted(jd)[0][1])

            return corrected


        answer_nine()
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:13: DeprecationWarning: generator '
  del sys.path[0]
```

```
Out[7]: ['corpulent', 'indecence', 'validate']
```

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

            correct_spellings = words.words()
            jd=1
            corrected=[]
            for w in entries:
                first_let=lambda x: x if (x[0] == w[0]) else ''

                same_start=np.array(list(map(first_let,correct_spellings)))
                same_start=(same_start[same_start!=''])

                set1 = set(nltk.ngrams(w,n=4))
                compute_jd = lambda x: (nltk.distance.jaccard_distance(set1,set(nltk.ngrams(x,n=
                jd=list(map(compute_jd,same_start))
                corrected.append(sorted(jd)[0][1])

            return corrected

        answer_ten()
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:13: DeprecationWarning: generator '
  del sys.path[0]
```

```
Out[9]: ['cormus', 'incendiary', 'valid']
```

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 [10]: def answer_eleven(entries=['cormulent', 'incendenece', 'validrate']):
             correct_spellings = words.words()
             jd=1
             corrected=[]
             for w in entries:
                 first_let=lambda x: x if (x[0] == w[0]) else ''

                 same_start=np.array(list(map(first_let,correct_spellings)))
                 same_start=(same_start[same_start!=''])

                 compute_ed = lambda x: (nltk.distance.edit_distance(x,w),x)
                 ed=list(map(compute_ed,same_start))
                 corrected.append(sorted(ed)[0][1])

             return corrected

         answer_eleven()

Out[10]: ['corpulent', 'intendence', 'validate']
```