CS4395 Assignment 2

The tokens method for Text objects accepts a text object as an argument. The method returns a list of strings as output.

```
#downloads and imports
import nltk
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt')
nltk.download('omw-1.4')
nltk.download('book')
from nltk.book import text1
```

```
#printing first 20 tokens and 5 lines for concordance of "sea"
tokenlist= text1.tokens
print("First 20 tokens: \n")
print(tokenlist[0:20])
print("\n 5 lines of concordance of sea:\n")
concordanceLines = text1.concordance('sea', width=80, lines = 5)
```

```
First 20 tokens:

['[', 'Moby', 'Dick', 'by', 'Herman', 'Melville', '1851', ']', 'ETYMOLOGY', '.', '(', 'S

 5 lines of concordance of sea:

Displaying 5 of 455 matches:
 shall slay the dragon that is in the sea ." -- ISAIAH " And what thing soever
 S PLUTARCH ' S MORALS . " The Indian Sea breedeth the most and the biggest fis
cely had we proceeded two days on the sea , when about sunrise a great many Wha
many Whales and other monsters of the sea , appeared . Among the former , one w
 waves on all sides , and beating the sea before him into a foam ." -- TOOKE '
```

The Text object count method and the python count method are essentially the same. The python method returns the number of times a given object occurs in a list and the Text method uses the same count function after first using the tokens method to generate a list from the Text object.

```
#Text object count method vs python count method
textCount = text1.count("great")
pyCountList= text1.tokens
pyCount = pyCountList.count("great")
```

```
print(f"Text object method count:{textCount}")
print(f"Python count:{pyCount}")
```

```
Text object method count:293
Python count:293
```

NLTK's word tokenizer is used to tokenize the text into individual tokens, such as words and punctuation. It accepts strings and returns a list, splitting on white space.

```
#nltk word tokenize
#first five sentences of https://education.nationalgeographic.org/resource/family-language
raw_text = "Certain languages are related to each other. Just as a person’s family consists o
from nltk import word_tokenize
tokens = nltk.word_tokenize(raw_text)
print(tokens[0:10])
```

```
['Certain', 'languages', 'are', 'related', 'to', 'each', 'other', '.', 'Just', 'as']
```

NLTK's sentence tokenizer performs sentence segmentation, accepting a string and returning a list of sentences.

```
#nltk sentence tokenize
from nltk import sent_tokenize
sentences = sent_tokenize(raw_text)
for sent in sentences:
  print(sent)
```

```
Certain languages are related to each other.
Just as a person’s family consists of people who share common ancestry, related language
A language family is a group of different languages that all descend from a particular c
The one language that generated those other languages in its family is known as a protol
Some languages do not come from a protolanguage.
```

NLTK's PorterStemmer is used to stem the text according to internal logic; it is meant to reduce words down into their base stem forms for easier analysis. It is a simpler and more aggresive option, removing anything that it percieves as an affix or applying simple modifications. It accepts strings, which may come from a list of tokens.

```
#nltk stemmer
from nltk.stem.porter import *
stemmer = PorterStemmer()
stemmed = [stemmer.stem(t) for t in tokens]
print(stemmed)
```

```
iguag', 'also', 'come', 'from', 'share', 'lineag', '.', 'a', 'languag', 'famili', 'is', '
```

The NLTK lemmatizer is a more complex method of reducing words into more basic forms. It tries to reduce words to lemmas, which is the form of the word present in the dictionary. It applies more logic than the stemmer, changing the form of the word rather than simply truncating it. It accepts individual string tokens as well.

The difference between the stemmer and lemmatizer can be seen in the following five words (stemmed-lemmatized):
languag-language
relat-related
famili-family
peopl-people
ancestri-ancestry

```
#nltk lemmatizer
from nltk.stem import WordNetLemmatizer
wnl = WordNetLemmatizer()
lemmatized = [wnl.lemmatize(t) for t in tokens]
print(lemmatized)
```

```
are', 'common', 'ancestry', ',', 'related', 'language', 'also', 'come', 'from', 'shared'
```

The NLTK library offers several tools for language processing with varying degrees of complexity. It has its use cases, and is effective in drawing out words and supplementary data based on a word. The quality is good, being simple to use and providing consistent results. I would use this library for understanding the topic of a text efficiently, utilizing the stemming and lemmatizing functions to discover which words were most common. Further context could be provided through sentence tokenizing and concordance

Colab paid products - Cancel contracts here

✓ 1s completed at 10:17 PM

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.