NLP (Atharva Pawar "9427" : BE Comps -A : Batch D)

EXP1 : Pre-processing of text

Word Tokenization (Using split() method)

```
text = """There are multiple ways we can perform tokenization on given text da
ta. We can choose any method based on langauge, library and purpose of modelin
g."""

# Split text by whitespace
tokens = text.split()
print(tokens)

# ['There', 'are', 'multiple', 'ways', 'we', 'can', 'perform', 'tokenization', 'on', 'given', 'text', 'da', 'ta.', 'We', 'can', 'choose',
```

```
['There', 'are', 'multiple', 'ways', 'we', 'can', 'perform', 'tokenization', 'on', 'given', 'text', 'da', 'ta.', 'We', 'can', 'choo
```

Sentence Tokenization (Using split() method)

```
text = """Characters like periods, exclamation point and newline char are used
to separate the sentences. But one drawback with split() method, that we can o
nly use one separator at a time! So sentence tonenization wont be foolproof wi
th split() method."""

text.split(". ")
# Note the space after the full stop makes sure that we dont get empty element at the end of list.

# ['Characters like periods, exclamation point and newline char are used\nto separate the sentences',
#  'But one drawback with split() method, that we can o\nnly use one separator at a time! So sentence tonenization wont be foolproof wi\r
```

```
['Characters like periods, exclamation point and newline char are used\nto separate the sentences',
 'But one drawback with split() method, that we can o\nnly use one separator at a time! So sentence tonenization wont be foolproof
wi\nth split() method.']
```

Word Tokenization Using Regular Expressions (RegEx)

```
import re

text = """There are multiple ways we can perform tokenization on given text data.
We can choose any method based on langauge, library and purpose of modeling."""

tokens = re.findall("[\w]+", text)
print(tokens)

# ['There', 'are', 'multiple', 'ways', 'we', 'can', 'perform', 'tokenization', 'on', 'given', 'text', 'data', 'We', 'can', 'choose', 'any
```

```
['There', 'are', 'multiple', 'ways', 'we', 'can', 'perform', 'tokenization', 'on', 'given', 'text', 'data', 'We', 'can', 'choose',
```

Sentence Tokenization using Regular Expressions (RegEx)

```
text = """Characters like periods, exclamation point and newline char are used to sep
arate the sentences. But one drawback with split() method, that we can only us
e one separator at a time! So sentence tonenization wont be foolproof with spl
it() method."""
tokens_sent = re.compile('[.!?] ').split(text)
# Using compile method to combine RegEx patterns
tokens_sent

# ['Characters like periods, exclamation point and newline char are used to sep\narate the sentences',
#  'But one drawback with split() method, that we can only us\ne one separator at a time',
#  'So sentence tonenization wont be foolproof with spl\nit() method.']
```

```
['Characters like periods, exclamation point and newline char are used to sep\narate the sentences',
 'But one drawback with split() method, that we can only us\ne one separator at a time',
 'So sentence tonenization wont be foolproof with spl\nit() method.']
```

Word Tokenization using nltk

```
# !pip install  nltk


import nltk
from nltk.tokenize import word_tokenize
nltk.download('punkt')


text = """There are multiple ways we can perform tokenization on given text da
ta. We can choose any method based on langauge, library and purpose of modelin
g."""

tokens = word_tokenize(text)
print(tokens)

# ['There', 'are', 'multiple', 'ways', 'we', 'can', 'perform', 'tokenization', 'on', 'given', 'text', 'da', 'ta', '.', 'We', 'can', 'choo
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
['There', 'are', 'multiple', 'ways', 'we', 'can', 'perform', 'tokenization', 'on', 'given', 'text', 'da', 'ta', '.', 'We', 'can', '
```

Sentence tokenization using nltk

```
from nltk.tokenize import sent_tokenize

text = """Characters like periods, exclamation point and newline char are used
to separate the sentences. But one drawback with split() method, that we can o
nly use one separator at a time! So sentence tonenization wont be foolproof wi
th split() method."""

sent_tokenize(text)

# ['Characters like periods, exclamation point and newline char are used\nto separate the sentences.',
#  'But one drawback with split() method, that we can o\nnly use one separator at a time!',
#  'So sentence tonenization wont be foolproof wi\nth split() method.']
```

```
['Characters like periods, exclamation point and newline char are used\nto separate the sentences.',
 'But one drawback with split() method, that we can o\nnly use one separator at a time!',
 'So sentence tonenization wont be foolproof wi\nth split() method.']
```

Tokenization Using spaCy

```
# !pip install spacy
# !python -m spacy download en


# Load English model from spacy
from spacy.lang.en import English
# Load English tokenizer.
# nlp object will be used to create 'doc' object which uses preprecoessing pip
# eline's components such as tagger, parser, NER and word vectors
nlp = English()
text = """There are multiple ways we can perform tokenization on given text data.
We can choose any method based on langauge, library and purpose of modeling."""
# Now we will process above text using 'nlp' object. Which is use to create do
# cuments with linguistic annotations and various nlp properties
my_doc = nlp(text)
# Above step has already tokenized our text but its in doc format, so lets wri
# te fo loop to create list of it
token_list = []
for token in my_doc:
  token_list.append(token.text)
print(token_list)

# ['There', 'are', 'multiple', 'ways', 'we', 'can', 'perform', 'tokenization', 'on', 'given', 'text', 'data', '.', '\n', 'We', 'can', 'ch
```

```
['There', 'are', 'multiple', 'ways', 'we', 'can', 'perform', 'tokenization', 'on', 'given', 'text', 'data', '.', '\n', 'We', 'can',
```

Sentence Tokenization using spaCy

```
# Load English tokenizer, tager, parser, NER and word vectors
nlp = English()
# Create the pipeline 'sentencizer' component
```

```
sbd = nlp.create_pipe('sentencizer')
# Add component to the pipeline
nlp.add_pipe('sentencizer')

text = """Characters like periods, exclamation point and newline char are used
to separate the sentences. But one drawback with split() method, that we can o
nly use one separator at a time! So sentence tonenization wont be foolproof wi
th split() method."""
sent_tokenize(text)
#nlp object is used to create documents with Linguistic annotations
doc = nlp(text)

#Create list of sentance tokens

sentence_list = []
for sentence in doc.sents:
  sentence_list.append(sentence.text)
print(sentence_list)

# ['Characters like periods, exclamation point and newline char are used\nto separate the sentences.', 'But one drawback with split() met
```

```
['Characters like periods, exclamation point and newline char are used\nto separate the sentences.', 'But one drawback with split()
```

Filtration

Covert text to lower case

```
input_str = "The 5 biggest countries by population in 2017 are China, India, United States, Indonesia, and Brazil."
input_str = input_str.lower()
print(input_str)
##the 5 biggest countries by population in 2017 are china, india, united states, indonesia, and brazil.
```

```
the 5 biggest countries by population in 2017 are china, india, united states, indonesia, and brazil.
```

Remove numbers

```
import re
input_str = "Box A contains 3 red and 5 white balls, while Box B contains 4 red and 2 blue balls."
result = re.sub(r'\d+', '', input_str)
print(result)
##Box A contains red and white balls, while Box B contains red and blue balls.
```

```
Box A contains  red and  white balls, while Box B contains  red and  blue balls.
```

Remove punctuation The following code removes this set of

```
# symbols [!”#$%&’()*+,- ./:;<=>?@]^_`{|}~:/
import string
input_str = "This &is [an] example? {of} string. with.? punctuation!!!!"
# Sample string
result = input_str.translate(str.maketrans('', '',string.punctuation))
print(result)
##This is an example of string with punctuation
```

```
This is an example of string with punctuation
```

Remove whitespaces

```
input_str = "\ta string example\n"
print("before: ", input_str)
input_str = input_str.strip()
print("after : ", input_str)

#a string example
```

```
before:         a string example

after :  a string example
```

Students’ task:

Reading moby.txt

```
# 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)
moby_outtext1 = list(text1)
# print(moby_outtext1)
print(moby_outtext1)
```

```
['\ufeff', '[', 'Moby', 'Dick', 'by', 'Herman', 'Melville', '1851', ']', 'ETYMOLOGY', '.', '(', 'Supplied', 'by', 'a', 'Late', 'Con
```

Q1 How many tokens (words and punctuation symbols) are in text1? This function should return an integer.

```
def howmanytokens(inptext):
  return len(inptext)

howmanytokens(moby_outtext1)
```

```
160642
```

Q2 How many unique tokens (unique words and punctuation) does text1 have? This function should return an integer.

```
def HowManyUniqueTokens(inptext):
  return len(set(inptext))

HowManyUniqueTokens(moby_outtext1)
```

```
16098
```

Q3 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. Hint: use nltk.FreqDist(text1) that gives the frequency distribution of words in text1

```
def MostFrequentlyOccurring20Tokens(inptext):
  frequent_words = nltk.FreqDist(inptext)
  return frequent_words.most_common(20)

MostFrequentlyOccurring20Tokens(moby_outtext1)
```

```
[(',', 11924),
 ('the', 8441),
 ('.', 4714),
 ('of', 4245),
 ('and', 3701),
 ('a', 3046),
 ('to', 2951),
 ('in', 2498),
 (';', 2415),
 ('that', 1891),
 ('his', 1593),
 ('I', 1437),
 ('it', 1336),
 ('is', 1120),
 ('he', 1069),
 ('with', 1057),
 ('was', 1051),
 ('as', 1049),
 ('--', 992),
 ("''", 985)]
```

Q4 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()

```
def TokensLengthGreaterThan5(inptext):
  frequent_words = nltk.FreqDist(inptext)
  tokens= [token for token, freq in frequent_words.items() if len(token) > 5 and freq > 150]
  lowercaseTokens = []
  for item in tokens:
    lowercaseTokens.append(item.lower())
  return sorted(lowercaseTokens)

# TokensLengthGreaterThan5(frequent_words[0])
TokensLengthGreaterThan5(moby_outtext1)
```

```
['before', 'captain', 'little', 'queequeg', 'seemed', 'though']
```

Q5 Find the longest word in text1 and that word's length. This function should return a tuple (longest_word, length).

```
def longestWordandLen(inptext):
  longest_word = ""
  max_length = 0
  for word in inptext:
        word_length = len(word)
        if word_length > max_length:
            longest_word = word
            max_length = word_length

  return longest_word, max_length

mylist = [word for word in moby_outtext1 if re.match(r'^[a-zA-Z0-9_]+$', word)]
longestWordandLen(mylist)
# moby_outtext1
```

```
('uncomfortableness', 17)
```

Q5 Answer validation

```
mylist = list(set(moby_outtext1))
# word_to_find = "twelve-o'clock-at-night"
word_to_find = "uncomfortableness"

if word_to_find in mylist:
    index_number = mylist.index(word_to_find)
    print(f"The word is found in the list at index number {index_number}.")
    print("\nmylist word: ", mylist[index_number])

else:
    print("The word is not found in the list.")
```

```
The word is found in the list at index number 15792.

mylist word:  uncomfortableness
```

```
mylist = list(set(moby_outtext1))
for i in range(len(mylist)):
    # if 11428 <= i < 11436:
    if 15787 <= i < 15797:
        print(i, ":", mylist[i])
```

```
15787 : ankles
15788 : aggrieved
15789 : porthole
15790 : BIOGRAPHY
15791 : gallop
15792 : uncomfortableness
15793 : widowed
15794 : scores
15795 : methodization
15796 : 'Point
```

✓ 0s completed at 10:55 AM