Experiment – 1
Pre-processing of text

Aim: To perform text pre-processing (Tokenization and Filtration)

Objectives:
1. Perform Tokenization on a given text using various Python libraries
2. Apply Filtration techniques such as stop word removal, lower case conversion etc. using various Python libraries

Lab outcomes:
At the end of this lab session, students will be able to…
1. Perform various tokenization techniques on a given Text.
2. Apply various Filtration techniques on a given Text

Pre-requisite: Basics mathematics, Python programming

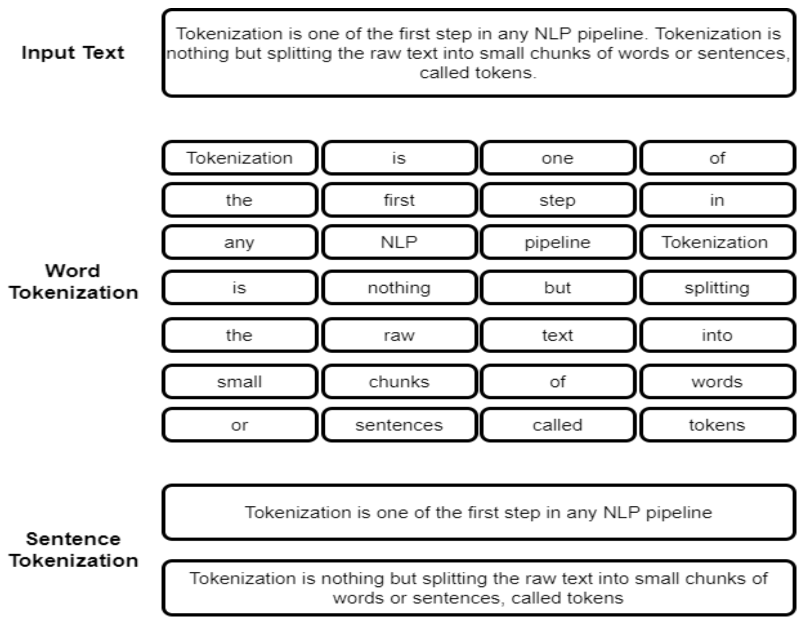
Theory:

What is text pre-processing?

To pre-process your text simply means to bring your text into a form that is predictable and analyzable for your task. One task's ideal pre-processing can become another task's worst nightmare. So, take note: text pre-processing is not directly transferable from task to task.

Tokenization:

Tokenization is one of the first step in any NLP pipeline. Tokenization is nothing but splitting the raw text into small chunks of words or sentences, called tokens. If the text is split into words, then its called as 'Word Tokenization' and if it's split into sentences then its called as 'Sentence Tokenization'.


Input Text
Tokenization is one of the first step in any NLP pipeline. Tokenization is nothing but splitting the raw text into small chunks of words or sentences, called tokens.
Word Tokenization
Tokenization
is
one
of
the
first
step
in
any
NLP
pipeline
Tokenization
is
nothing
but
splitting
the
raw
text
into
small
chunks
of
words
or
sentences
called
tokens
Sentence Tokenization
Tokenization is one of the first step in any NLP pipeline
Tokenization is nothing but splitting the raw text into small chunks of words or sentences, called tokens

Why Tokenization is Required?

Every sentence gets its meaning by the words present in it. So by analyzing the words present in the text we can easily interpret the meaning of the text. Once we have a list of words we can also use statistical tools and methods to get more insights into the text. For example, we can use word count and word frequency to find out important of word in that sentence or document.

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

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', 'data.', 'We', 'can', 'choose', 'any', 'method', 'based'
, 'on', 'langauge,', 'library', 'and', 'purpose', 'of', 'modeling.']
```

Observe in above list, words like 'language,' and 'modeling.' are containing punctuation at the end of them. **Python split method do not consider punctuation as separate token.**

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 to sepa
rate the sentences',
 'But one drawback with split() method, that we can only use one separator at
a time! So sentence tonenization wont be foolproof with split() method.']
```

As you can see, split() since we can't use multiple separator split() method failed to split the last sentence from separator (!). We can overcome this drawback by applying split method multiple times with different separator but there are better ways to do it.

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', 'method', 'based',
'on', 'langauge', 'library', 'and', 'purpose', 'of', 'modeling']
```

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 combi
ne RegEx patterns
tokens_sent
```

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

Tokenization Using NLTK

- Natural Language Toolkit (NLTK) is library written in python for natural language processing.
- NLTK has module word_tokenize() for word tokenization and sent_tokenize() for sentence tokenization.
- Syntax to install NLTK is as below

  !pip install --user -U nltk

- Note that we are going use "!" before **the command to let notebook know that, it should read as commandline command**

Word Tokenization using nltk

```
!pip install --user -U nltk
```

```
from nltk.tokenize import word_tokenize

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', 'data', '.', 'We', 'can', 'choose', 'any', 'method', 'ba
sed', 'on', 'langauge', ',', 'library', 'and', 'purpose', 'of', 'modeling', '.
']
```

NOTE: Notice that NLTK word tokenization also consider the punctuation as token. During text cleaning process we have to account for this.

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 to sepa
rate the sentences.',
 'But one drawback with split() method, that we can only use one separator at
a time!',
 'So sentence tonenization wont be foolproof with split() method.']
```

Tokenization Using spaCy

- spaCy is an open-source software library for advanced natural language processing, written in the programming languages Python and Cython
- in spaCy we create language model object, which then used for word and sentence tokenization
- Syntax to install spaCy library and English model is as below

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

Word Tokenization using scpaCy

```
!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 da
ta. We can choose any method based on langauge, library and purpose of modelin
g."""

# 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)
```

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."""

# nlp object is used to create documents with linguistic annotations
doc = nlp(text)

# Create list of sentence 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 to sepa
rate the sentences.', 'But one drawback with split() method, that we can only
use one separator at a time!', 'So sentence tonenization wont be foolproof wit
h split() method.']
```

Filtration

Many of the words used in the phrase are insignificant and hold no meaning. For example – English is a subject. Here, 'English' and 'subject' are the most significant words and 'is', 'a' are almost useless. Filtering is the process of removing stop words or any unnecessary data from the sentence Using the nltk, we can remove the insignificant words by looking at their part-of-speech tags. For that we have to decide which Part-Of-Speech tags are significant. Stop words are a set of commonly used words in a language. Examples of stop words in English are "a", "the", "is", "are" and etc. The intuition behind using stop words is that, by removing low information words from text, we can focus on the important words instead. For example, in the context of a search system, if your search query is "what is text pre processing?",you want the search system to focus on surfacing documents that talk about text pre-processing over documents that talk about what is. This can be done by preventing all words from your stop word list from being analysed. Stop words are commonly applied in search systems, text classification applications, topic modelling, topic extraction and others.

Chunking

Chunking is a natural language process that identifies constituent parts of sentences (nouns, verbs, adjectives, etc.) and links them to higher order units that have discrete grammatical meanings (noun groups or phrases, verb groups, etc.)

Program

Covert text to lower case

```
input_str = "The 5 biggest countries by population in 2017 are China, India, U nited
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.
```

Remove numbers

```
import re
input_str = "Box A contains 3 red and 5 white balls, while Box B contains 4 re
d 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.
```

Remove punctuation The following code removes this set of symbols [!"#$%&'()*+,- ./:;<=>?@]^_`{|}~:

```
import string
input_str = "This &is [an] example? {of} string. with.? punctuation!!!!"
# Sam ple string  result = input_str.translate(str.maketrans('', '',
string.punctuation)) print(result)
##This is an example of string with punctuation
```

Remove whitespaces

```
input_str = "\ta string example\n"
print(input_str)
input_str = input_str.strip()
input_str
#a string example
'a string example'
```

Students' task:

Read moby.txt file and solve following questions:

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)
print(list(text1))
```

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

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

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

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()

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

Conclusion: (Write your observations and conclusion here)

Post Lab Questions:

1. Differentiate between nltk and spacy.
2. Explore tokenization using Keras and Tokenization suing Gensim

****