##Basic Text Preprocessing

Natural Language Toolkit (NLTK) is one of the largest Python libraries for performing various Natural Language Processing tasks. From rudimentary tasks such as text pre-processing to tasks like vectorized representation of text – NLTK's API has covered everything. In this article, we will accustom ourselves to the basics of NLTK and perform some crucial NLP tasks: Tokenization, Stemming, Lemmatization, and POS Tagging.

Natural Language Toolkit (NLTK) works as a powerful Python library that a wide range of tools for Natural Language Processing (NLP). From fundamental tasks like text pre-processing to more advanced operations such as semantic reasoning, NLTK provides a versatile API that caters to the diverse needs of language-related tasks.

```
!pip install nltk
```

Accessing Additional Resources: To incorporate the usage of additional resources, such as recourses of languages other than English – you can run the following in a python script. It has to be done only once when you are running it for the first time in your system.

```
import nltk
nltk.download('all')
```

Tokenisation using NLTK

Tokenization refers to break down the text into smaller units. It entails splitting paragraphs into sentences and sentences into words. It is one of the initial steps of any NLP pipeline. Let us have a look at the two major kinds of tokenization that NLTK provides

Document-> Pargraphs-> Sentences-> words

##Sentence-level Tokenisation

```
# Tokenization using NLTK
from nltk import word_tokenize, sent_tokenize
paragraph = "NPTEL is a great learning platform. It is one of the best
for Computer Science students."
#print(word_tokenize(sent))
print(sent_tokenize(paragraph))

tokenised_sentences=sent_tokenize(paragraph)
print(len(tokenised_sentences))

print(type(tokenised_sentences))

print(tokenised_sentences[0])

print(tokenised_sentences[1])

print(type(print(tokenised_sentences[0])))
```

```
# Tokenization using NLTK
from nltk import word_tokenize, sent_tokenize
paragraph1 = "NPTEL is a great learning platform.It is one of the best
for Computer Science students."
#print(word_tokenize(sent))
print(sent_tokenize(paragraph1))
```

** "I study Machine Learning on GeeksforGeeks." will be word-tokenized as ['I', 'study', 'Machine', 'Learning', 'on', 'GeeksforGeeks', '.']. **

**Sentence Tokenization It involves breaking down the text into individual sentences.

Example: "I study Machine Learning on GeeksforGeeks. Currently, I'm studying NLP" will be sentence-tokenized as ['I study Machine Learning on GeeksforGeeks.', 'Currently, I'm studying NLP.']**

##Word-level Tokenisation

```
# Tokenization using NLTK
from nltk import word_tokenize, sent_tokenize
#paragraph = "NPTEL is a great learning platform. It is one of the
best for Computer Science students."
print(tokenised_sentences[0])
#print(word_tokenize(tokenised_sentences[0]))

print(word_tokenize(tokenised_sentences[0]))

print(len(word_tokenize(tokenised_sentences[0])))

print(tokenised_sentences[1])

print(word_tokenize(tokenised_sentences[1]))

print(len(word_tokenize(tokenised_sentences[1])))
```

#Stemming using Porter Stemmer

liking-> lik

liked-> like/lik

Stemming generates the base word from the inflected word by removing the affixes of the word. It has a set of pre-defined rules that govern the dropping of these affixes. It must be noted that stemmers might not always result in semantically meaningful base words. Stemmers are faster and computationally less expensive than lemmatizers.

We can see that all the variations of the word ‘play’ have been reduced to the same word – ‘play’. In this case, the output is a meaningful word, ‘play’. However, this is not always the case. Let us take an example.

Please note that these groups are stored in the lemmatizer; there is no removal of affixes as in the case of a stemmer.

```
from nltk.stem import PorterStemmer
#Create an instance of porter stemmer
porter_stemmer=PorterStemmer()
original_words=['liking','likes','liked','likely']
stemmed_words=[porter_stemmer.stem(word) for word in original_words]

print("Original words", original_words)
print("Stemmed words using Porter Stemmer",stemmed_words)


from nltk.stem import PorterStemmer
#Create an instance of porter stemmer
porter_stemmer=PorterStemmer()
original_words=['liking','likes','liked','likely']
stemmed_words=[porter_stemmer.stem(word) for word in original_words]

print("Original words", original_words)
print("Stemmed words using Porter Stemmer",stemmed_words)

from nltk.stem import PorterStemmer
#Create an instance of porter stemmer
porter_stemmer=PorterStemmer()
original_words=['plays','happily','ate','drank',
'running','go','goes','went']
stemmed_words=[porter_stemmer.stem(word) for word in original_words]

print("Original words", original_words)
print("Stemmed words using Porter Stemmer",stemmed_words)

#EED-> EE
from nltk.stem import PorterStemmer
#Create an instance of porter stemmer
ps=PorterStemmer()
original_words=['agreed']
stemmed_words=[ps.stem(word) for word in original_words]

print("Original words", original_words)
print("Stemmed words using Porter Stemmer",stemmed_words)

from nltk.stem import PorterStemmer
# create an object of class PorterStemmer
porter = PorterStemmer()
print(porter.stem("Communication"))


#EED-> EE
from nltk.stem import PorterStemmer
#Create an instance of porter stemmer
ps=PorterStemmer()
original_words=['Conclude', 'conclusive', 'conclusion']
stemmed_words=[ps.stem(word) for word in original_words]
```

```
print("Original words", original_words)
print("Stemmed words using Porter Stemmer",stemmed_words)


#EED-> 0
from nltk.stem import PorterStemmer
#Create an instance of porter stemmer
ps=PorterStemmer()
original_words=['programming', 'elephant', 'mountain', 'sunshine',
'keyboard', 'ocean', 'umbrella', 'happiness', 'giraffe', 'whisper']
stemmed_words=[ps.stem(word) for word in original_words]

print("Original words", original_words)
print("Stemmed words using Porter Stemmer",stemmed_words)
```

#Lemmatization Lemmatization involves grouping together the inflected forms of the same word. This way, we can reach out to the base form of any word which will be meaningful in nature. The base from here is called the Lemma.

Lemmatizers are slower and computationally more expensive than stemmers.

```
from nltk.stem import PorterStemmer
#Create an instance of porter stemmer
porter_stemmer=PorterStemmer()
original_words=['plays','happily','ate','drank',
'running','go','goes','went']
stemmed_words=[porter_stemmer.stem(word) for word in original_words]

print("Original words", original_words)
print("Stemmed words using Porter Stemmer",stemmed_words)

from nltk.stem import PorterStemmer
#Create an instance of porter stemmer
porter_stemmer=PorterStemmer()
original_words1=['plays','happily','ate','drank',
'running','go','goes','went']
stemmed_words1=[ ]
for word in original_words1:
  stemmed_words1.append(porter_stemmer.stem(word))
print("Original words", original_words1)
print("Stemmed words using Porter Stemmer",stemmed_words1)

from nltk.stem import WordNetLemmatizer
# create an object of class WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
# lemmatized_words=[lemmatizer.lemmatize(word) for word in
original_words]
# original_words=['plays','happily','ate','drank',
'running','go','goes','went']
```

```
# print(lemmatized_words)
print(lemmatizer.lemmatize("happily",'v'))
print(lemmatizer.lemmatize("goes",'v'))
print(lemmatizer.lemmatize("running"))
print(lemmatizer.lemmatize("running",'v'))

print(lemmatizer.lemmatize("the",'dt'))
```

Please note that in lemmatizers, we need to pass the Part of Speech of the word along with the word as a function argument.

Also, stemmers always result in meaningful base words. Let us take the same example as we took in the case for stemmers.

```
from nltk.stem import WordNetLemmatizer
# create an object of class WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
print(lemmatizer.lemmatize("Communication", 'v'))
```

Why NLTK?

Popularity: NLTK is one of the leading platforms for dealing with language data.

Simplicity: Provides easy-to-use APIs for a wide variety of text preprocessing methods.

Community: It has a large and active community that supports the library and improves it.

Open Source: Free and open-source available for Windows, Mac OSX, and Linux.

#Basic Preprocessing for Text

Lowercase

```
text = "It is a truth universally acknowledged, that a single man in
possession of a good fortune, must be in want of a wife."
text = text.lower()
print(text)

text = "It is a truth universally acknowledged, that a single man in
possession of a good fortune, must be in want of a wife."
text = text.upper()
print(text)
```

Removing Punctuation

```
tweet="Great!@justinbeiber concert was amazing!!!!\#neversaynever."

import string
print(string.punctuation)
```

```
pre_tweet = "".join([char for char in tweet if char not in
string.punctuation])
print(pre_tweet)

query="card-carrying"
print(len(query))
pre_query = "".join([char for char in query if char not in
string.punctuation])
print(pre_query)
print(len(pre_query))
```

Function words

Content words

Stopword Filtering

```
from nltk.corpus import stopwords
stop_words_eng = stopwords.words('english')
print(stop_words_eng)
print(len(stop_words_eng))

from nltk.corpus import stopwords
stop_words_chinese = stopwords.words('chinese')
print(stop_words_chinese)
print(len(stop_words_chinese))

from nltk.corpus import stopwords
stop_words_eng = stopwords.words('english')
words='I have enrolled in NLP course on NPTEL by Dr. Pawan Goyal'
tokens=word_tokenize(words)

print(tokens)

output=[word for word in tokens if word.lower() not in stop_words_eng]

print(output)
```

Stop word filetring

Tokenise-> list comprehension

```
print(type(output))

out="".join([str(ele)+" " for ele in output])

out
```