Text Analytics for Beginners using NLTK

Reference: https://www.datacamp.com/community/tutorials/text-analytics-beginners-nltk

In today's area of internet and online services, data is generating at incredible speed and amount. Generally, Data analyst, engineer, and scientists are handling relational or tabular data. These tabular data columns have either numerical or categorical data. Generated data has a variety of structures such as text, image, audio, and video. Online activities such as articles, website text, blog posts, social media posts are generating unstructured textual data. Corporate and business need to analyze textual data to understand customer activities, opinion, and feedback to successfully derive their business. To compete with big textual data, text analytics is evolving at a faster rate than ever before.

In this tutorial, you are going to cover the following topics:

- Text Analytics and NLP
- Compare Text Analytics, NLP and Text Mining
    - Text Analysis Operations using NLTK
    - Tokenization
    - Stopwords
    - Lexicon Normalization such as Stemming and Lemmatization
    - POS Tagging

Text Analysis Operations using NLTK

NLTK is a powerful Python package that provides a set of diverse natural languages algorithms. It is free, opensource, easy to use, large community, and well documented. NLTK consists of the most common algorithms such as tokenizing, part-of-speech tagging, stemming, sentiment analysis, topic segmentation, and named entity recognition. NLTK helps the computer to analysis, preprocess, and understand the written text.

Tokenization

Tokenization is the first step in text analytics. The process of breaking down a text paragraph into smaller chunks such as words or sentence is called Tokenization. Token is a single entity that is building blocks for sentence or paragraph.

Sentence Tokenization

Sentence tokenizer breaks text paragraph into sentences.

```
from nltk.tokenize import sent_tokenize
text="""Hello Mr. Smith, how are you doing today? The weather is great, and
city is awesome.
The sky is pinkish-blue. You shouldn't eat cardboard"""
tokenized_text=sent_tokenize(text)
print(tokenized_text)
```

Output:

```
['Hello Mr. Smith, how are you doing today?', 'The weather is great, and city
is awesome.', 'The sky is pinkish-blue.', "You shouldn't eat cardboard"]
```

Here, the given text is tokenized into sentences.

Word Tokenization

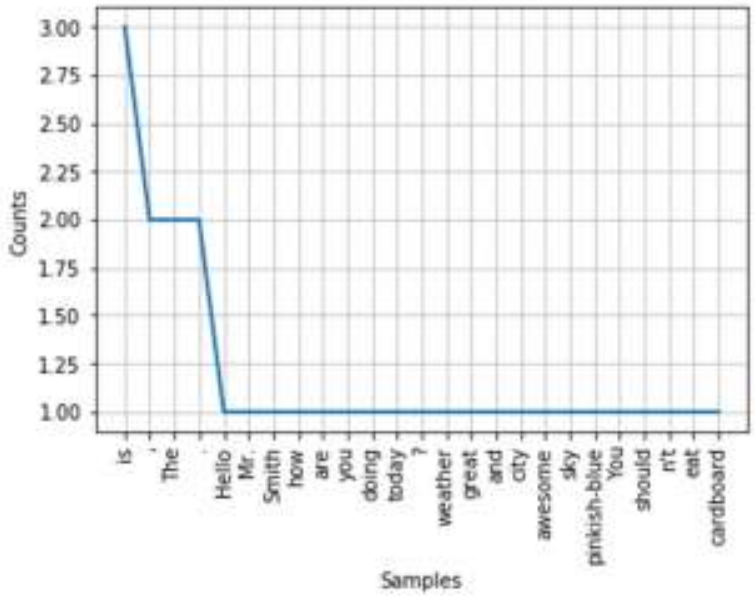
Word tokenizer breaks text paragraph into words.

```
from nltk.tokenize import word_tokenize
tokenized_word=word_tokenize(text)
print(tokenized_word)
```

```
Output:
['Hello', 'Mr.', 'Smith', ',', 'how', 'are', 'you', 'doing', 'today', '?',
'The', 'weather', 'is', 'great', ',', 'and', 'city', 'is', 'awesome', '.',
'The', 'sky', 'is', 'pinkish-blue', '.', 'You', 'should', "n't", 'eat',
'cardboard']
```

Frequency Distribution

```
from nltk.probability import FreqDist
fdist = FreqDist(tokenized_word)
print(fdist)
<FreqDist with 25 samples and 30 outcomes>
fdist.most_common(2)
[('is', 3), (',', 2)]
# Frequency Distribution Plot
import matplotlib.pyplot as plt
fdist.plot(30,cumulative=False)
plt.show()
```



Stopwords

Stopwords considered as noise in the text. Text may contain stop words such as is, am, are, this, a, an, the, etc.

In NLTK for removing stopwords, you need to create a list of stopwords and filter out your list of tokens from these words.

```
from nltk.corpus import stopwords
stop_words=set(stopwords.words("english"))
print(stop_words)

Output:

{'their', 'then', 'not', 'ma', 'here', 'other', 'won', 'up', 'weren', 'being',
'we', 'those', 'an', 'them', 'which', 'him', 'so', 'yourselves', 'what',
'own', 'has', 'should', 'above', 'in', 'myself', 'against', 'that', 'before',
't', 'just', 'into', 'about', 'most', 'd', 'where', 'our', 'or', 'such',
'ours', 'of', 'doesn', 'further', 'needn', 'now', 'some', 'too', 'hasn',
'more', 'the', 'yours', 'her', 'below', 'same', 'how', 'very', 'is', 'did',
'you', 'his', 'when', 'few', 'does', 'down', 'yourself', 'i', 'do', 'both',
'shan', 'have', 'itself', 'shouldn', 'through', 'themselves', 'o', 'didn',
've', 'm', 'off', 'out', 'but', 'and', 'doing', 'any', 'nor', 'over', 'had',
'because', 'himself', 'theirs', 'me', 'by', 'she', 'whom', 'hers', 're',
'hadn', 'who', 'he', 'my', 'if', 'will', 'are', 'why', 'from', 'am', 'with',
'been', 'its', 'ourselves', 'ain', 'couldn', 'a', 'aren', 'under', 'll', 'on',
'y', 'can', 'they', 'than', 'after', 'wouldn', 'each', 'once', 'mightn',
'for', 'this', 'these', 's', 'only', 'haven', 'having', 'all', 'don', 'it',
'there', 'until', 'again', 'to', 'while', 'be', 'no', 'during', 'herself',
'as', 'mustn', 'between', 'was', 'at', 'your', 'were', 'isn', 'wasn'}
```

Removing Stopwords

```
filtered_sent=[]
for w in tokenized_sent:
    if w not in stop_words:
        filtered_sent.append(w)
print("Tokenized Sentence:",tokenized_sent)
print("Filterd Sentence:",filtered_sent)

Output:
Tokenized Sentence: ['Hello', 'Mr.', 'Smith', ',', 'how', 'are', 'you',
'doing', 'today', '?']
Filterd Sentence: ['Hello', 'Mr.', 'Smith', ',', 'today', '?']
```

Lexicon Normalization

Lexicon normalization considers another type of noise in the text. For example, connection, connected, connecting word reduce to a common word "connect". It reduces derivationally related forms of a word to a common root word.

Stemming

Stemming is a process of linguistic normalization, which reduces words to their word root word or chops off the derivational affixes. For example, connection, connected, connecting word reduce to a common word "connect".

```
# Stemming
from nltk.stem import PorterStemmer
from nltk.tokenize import sent_tokenize, word_tokenize

ps = PorterStemmer()

stemmed_words=[]
for w in filtered_sent:
    stemmed_words.append(ps.stem(w))

print("Filtered Sentence:",filtered_sent)
print("Stemmed Sentence:",stemmed_words)

Output:

Filtered Sentence: ['Hello', 'Mr.', 'Smith', ',', 'today', '?']
Stemmed Sentence: ['hello', 'mr.', 'smith', ',', 'today', '?']
```

Lemmatization

Lemmatization reduces words to their base word, which is linguistically correct lemmas. It transforms root word with the use of vocabulary and morphological analysis. Lemmatization is usually more sophisticated than stemming. Stemmer works on an individual word without knowledge of the context. For example, The word "better" has "good" as its lemma. This thing will miss by stemming because it requires a dictionary look-up.

```
#Lexicon Normalization
#performing stemming and Lemmatization

from nltk.stem.wordnet import WordNetLemmatizer
lem = WordNetLemmatizer()

from nltk.stem.porter import PorterStemmer
stem = PorterStemmer()

word = "flying"
print("Lemmatized Word:",lem.lemmatize(word,"v"))
print("Stemmed Word:",stem.stem(word))


Output:
Lemmatized Word: fly
Stemmed Word: fli
```

POS Tagging

The primary target of Part-of-Speech(POS) tagging is to identify the grammatical group of a given word. Whether it is a NOUN, PRONOUN, ADJECTIVE, VERB, ADVERBS, etc. based

on the context. POS Tagging looks for relationships within the sentence and assigns a corresponding tag to the word.

```
sent = "Albert Einstein was born in Ulm, Germany in 1879."
tokens=nltk.word_tokenize(sent)
print(tokens)

Output:
['Albert', 'Einstein', 'was', 'born', 'in', 'Ulm', ',', 'Germany', 'in',
'1879', '.']

nltk.pos_tag(tokens)

Outut:

[('Albert', 'NNP'),
 ('Einstein', 'NNP'),
 ('was', 'VBD'),
 ('born', 'VBN'),
 ('in', 'IN'),
 ('Ulm', 'NNP'),
 (',', ','),
 ('Germany', 'NNP'),
 ('in', 'IN'),
 ('1879', 'CD'),
 ('.', '.')]
POS tagged: Albert/NNP Einstein/NNP was/VBD born/VBN in/IN Ulm/NNP ,/,
Germany/NNP in/IN 1879/CD ./.
```