

**WEEK- 11****NATURAL LANGUAGE PROCESSING****Session 2****11.6 TEXT PROCESSING TASKS**

Text data derived from natural language is unstructured and noisy. Text preprocessing involves transforming text into a clean and consistent format that can then be fed into a model for further analysis and learning.

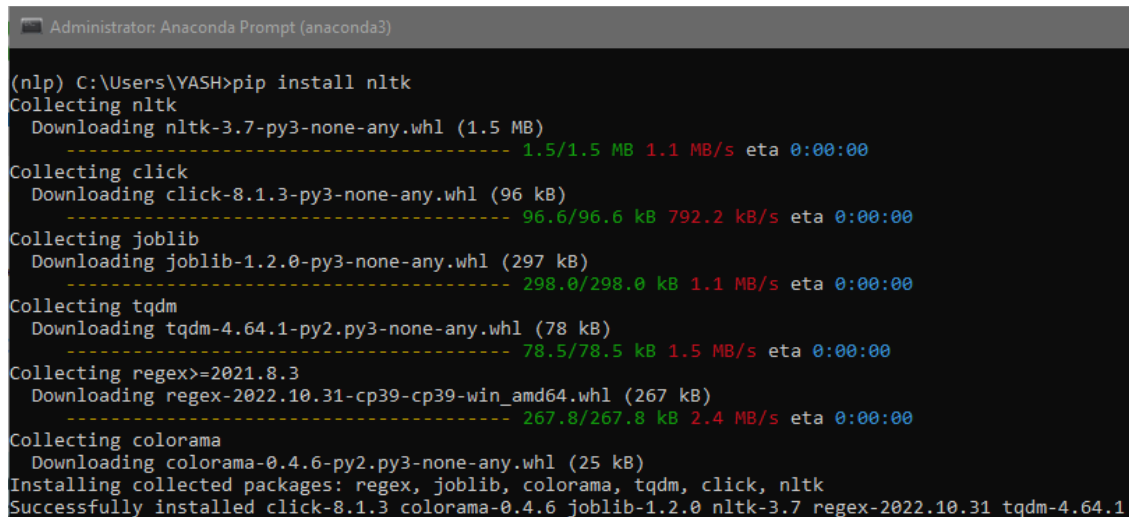
- Text preprocessing is an important step for natural language processing (NLP) tasks.
- It transforms text into a more digestible form so that machine learning algorithms can perform better, and depending on how well the data has been preprocessed; the results are seen.
- Text preprocessing improves the performance of an NLP system.
- For tasks such as sentiment analysis, document categorization, document retrieval based upon user queries, and more, adding a text preprocessing layer provides more accuracy.

Some of the preprocessing steps are:

- Tokenization
- Removing Stop words
- Spelling correction
- Stemming
- Lemmatization

We will be using NLTK to perform all NLP tasks. If NLTK is not yet installed in your system, go to Anaconda Command prompt and perform the following steps:

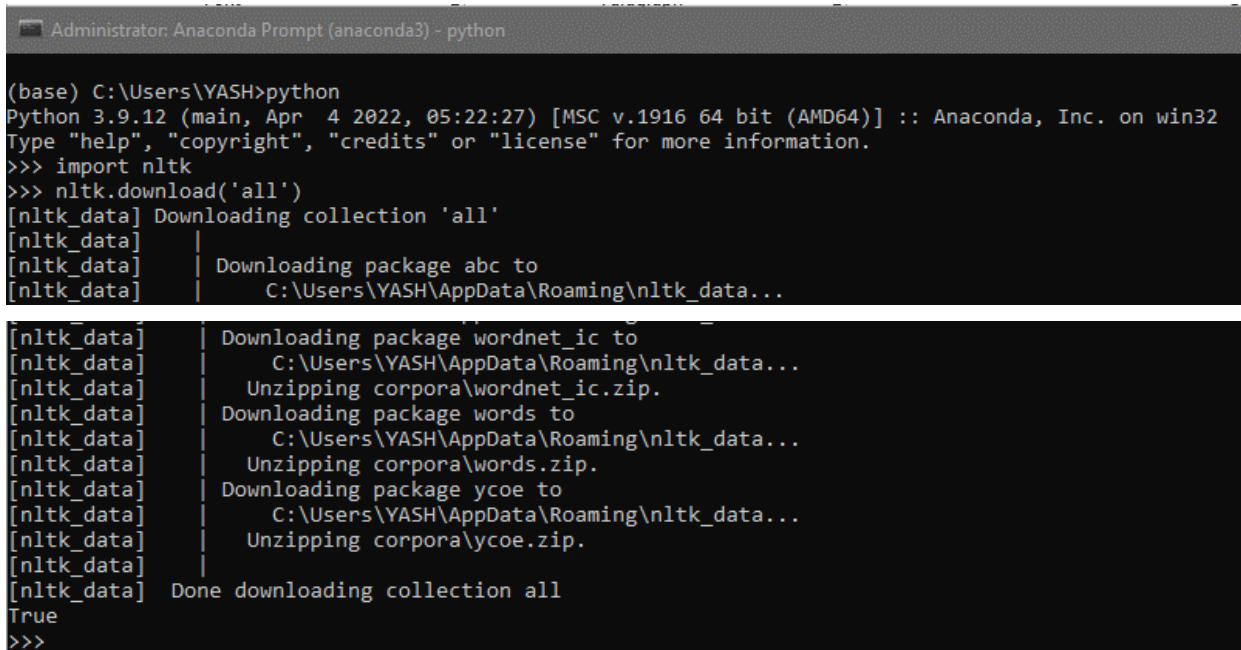
- **pip install nltk**



```
Administrator: Anaconda Prompt (anaconda3)

(nlp) C:\Users\YASH>pip install nltk
Collecting nltk
  Downloading nltk-3.7-py3-none-any.whl (1.5 MB)
----- 1.5/1.5 MB 1.1 MB/s eta 0:00:00
Collecting click
  Downloading click-8.1.3-py3-none-any.whl (96 kB)
----- 96.6/96.6 kB 792.2 kB/s eta 0:00:00
Collecting joblib
  Downloading joblib-1.2.0-py3-none-any.whl (297 kB)
----- 298.0/298.0 kB 1.1 MB/s eta 0:00:00
Collecting tqdm
  Downloading tqdm-4.64.1-py2.py3-none-any.whl (78 kB)
----- 78.5/78.5 kB 1.5 MB/s eta 0:00:00
Collecting regex>=2021.8.3
  Downloading regex-2022.10.31-cp39-cp39-win_amd64.whl (267 kB)
----- 267.8/267.8 kB 2.4 MB/s eta 0:00:00
Collecting colorama
  Downloading colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Installing collected packages: regex, joblib, colorama, tqdm, click, nltk
Successfully installed click-8.1.3 colorama-0.4.6 joblib-1.2.0 nltk-3.7 regex-2022.10.31 tqdm-4.64.1
```

- Then, enter the python shell in your terminal by simply typing **python**
- Type **import nltk**
- Type **nltk.download('all')**



```
Administrator: Anaconda Prompt (anaconda3) - python

(base) C:\Users\YASH>python
Python 3.9.12 (main, Apr  4 2022, 05:22:27) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import nltk
>>> nltk.download('all')
[nltk_data] Downloading collection 'all'
[nltk_data] |
[nltk_data] |   Downloading package abc to
[nltk_data] |   C:\Users\YASH\AppData\Roaming\nltk_data...
[nltk_data] |
[nltk_data] |   Downloading package wordnet_ic to
[nltk_data] |   C:\Users\YASH\AppData\Roaming\nltk_data...
[nltk_data] |   Unzipping corpora\wordnet_ic.zip.
[nltk_data] |   Downloading package words to
[nltk_data] |   C:\Users\YASH\AppData\Roaming\nltk_data...
[nltk_data] |   Unzipping corpora\words.zip.
[nltk_data] |   Downloading package ycoe to
[nltk_data] |   C:\Users\YASH\AppData\Roaming\nltk_data...
[nltk_data] |   Unzipping corpora\ycoe.zip.
[nltk_data] |
[nltk_data] | Done downloading collection all
True
>>>
```

The above installation will take quite some time due to the massive number of tokenizers, chunkers, other algorithms, and all of the corpora to be downloaded.

### 11.6.1 Tokenization

Token in a text document refers to each “entity” that is a part of whatever was split up based on rules. For examples, each word is a token when a sentence is “tokenized” into words. Each sentence can also be a token, if you tokenized the sentences out of a paragraph. So basically, tokenizing involves **splitting sentences and words from the body of the text**.

#### NLTK Tokenizer Package

The nltk.tokenize package contains methods to tokenize the given text into sentences and words based on our requirement.

##### 1. Sentence tokenization

The given document is divided or tokenized into sentences.

##### Syntax:

**nltk.tokenize.sent\_tokenize(text, language='english')**

returns a sentence-tokenized copy of text, using NLTK’s recommended sentence tokenizer. (currently PunktSentenceTokenizer for the specified language).

##### Parameters:

*Text (str)* – text to split into sentences

*Language(str)* – the model name in the Punkt corpus. By default it will be English.

**Example:**

The following example reads a text file and tokenizes it into sentences.

- Import the sent\_tokenize method from nltk.tokenize package

```
from nltk.tokenize import sent_tokenize
```

- Read the text file using open method

```
file = open("nlp.txt", "r")
text = file.read()
print(text)
```

**Output:**

Natural language processing (NLP) refers to the branch of computer science and more specifically, the branch of artificial intelligence or AI, concerned with giving computers the ability to understand text and spoken words in much the same way human beings can.

NLP combines computational linguistics with statistical, machine learning, and deep learning models. Together, these technologies enable computers to process human language in the form of text or voice data and to understand its full meaning, complete with the speaker or writer's intent and sentiment.

- Divide the text into sentences using the sent\_tokenize method

```
sentences = sent_tokenize(text)
```

- Print the sentences to see how the method has divided the text into sentences.

```
print("Number of sentences:", len(sentences))
for i in range(len(sentences)):
    print("\nSentence ", i+1, ":\n", sentences[i])
```

**Output:**

Number of sentences: 3

Sentence 1 :

Natural language processing (NLP) refers to the branch of computer science and more specifically, the branch of artificial intelligence or AI, concerned with giving computers the ability to understand text and spoken words in much the same way human beings can.

Sentence 2 :

NLP combines computational linguistics with statistical, machine learning, and deep learning models.

Sentence 3 :

Together, these technologies enable computers to process human language in the form of text or voice data and to understand its full meaning, complete with the speaker or writer's intent and sentiment.

## 2. Word tokenization

The given document is divided or tokenized into words.

### Syntax:

```
nlk.tokenize.word_tokenize(text, language='english', preserve_line=False)
```

returns a tokenized copy of text, using NLTK's recommended word tokenizer (currently an improved TreebankWordTokenizer along with PunktSentenceTokenizer for the specified language).

### Parameters

*text (str)* – text to split into words

*language (str)* – the model name in the Punkt corpus

*preserve\_line (bool)* – A flag to decide whether to sentence tokenize the text or not.

### Example:

For the text file used in previous example, let us find the word tokens.

```
from nltk.tokenize import word_tokenize
```

```
words = word_tokenize(text)
```

```
print("Total number of words: ",len(words))  
print(words)
```

### Output:

```
Total number of words: 97  
['Natural', 'language', 'processing', '(', 'NLP', ')', 'refers', 'to', 'the', 'branch', 'of', 'comput  
er', 'science', 'and', 'more', 'specifically', ',', 'the', 'branch', 'of', 'artificial', 'intelligenc  
e', 'or', 'AI', ',', 'concerned', 'with', 'giving', 'computers', 'the', 'ability', 'to', 'understan  
d', 'text', 'and', 'spoken', 'words', 'in', 'much', 'the', 'same', 'way', 'human', 'beings', 'can',  
, '.', 'NLP', 'combines', 'computational', 'linguistics', 'with', 'statistical', ',', 'machine', 'learn  
ing', ',', 'and', 'deep', 'learning', 'models', '.', 'Together', ',', 'these', 'technologies', 'enabl  
e', 'computers', 'to', 'process', 'human', 'language', 'in', 'the', 'form', 'of', 'text', 'or', 'voic  
e', 'data', 'and', 'to', 'understand', 'its', 'full', 'meaning', ',', 'complete', 'with', 'the', 'spe  
aker', 'or', 'writer', "'s", 'intent', 'and', 'sentiment', '.']
```

The tokens generated will contain special characters such as comma, dot, parentheses, apostrophe etc. We may have to remove these special characters, and stop words to generate the vocabulary of our text document.

## 3. Visualizing Frequency Distribution of words

The standard method for visualizing the word frequency distribution is to count how often each word occurs in a corpus and to sort the word frequency counts by decreasing magnitude. We need to create a Bag of Words (BoW), which is a statistical language model used to analyze text and documents based on word count.

### Example:

To visualize the frequency distribution, let us consider opinions given by customers about a hotel.

- Read the “Opinion.txt” text file which contains opinions given by multiple customers

```
file = open("opinion.txt", "r")
text = file.read()
```

```
words = word_tokenize(text)
```

- **Creating the Frequency Distribution**

Without the NLTK package, creating a frequency distribution plot (histogram) for a BoW is possible, but will take multiple lines of code to do so. Through the use of the FreqDist class, we are able to obtain the frequencies of every token in the BoW with one single line of code:

```
from nltk.probability import FreqDist

## Creating FreqDist for whole BoW, keeping the 20 most common tokens
all_fdist = FreqDist(words)
all_fdist['service']
```

**Output:**

```
[('the', 77),
 ('food', 52),
 ('and', 41),
 ('a', 29),
 ('was', 26),
 ('for', 19),
 ('to', 15),
 ('of', 14),
 ('we', 13),
 ('service', 13),
 ('very', 12),
 ('i', 12),
 ('at', 12),
 ('in', 10),
 ('is', 10),
 ('room', 10),
 ('but', 9),
 ('our', 8),
 ('hotel', 8),
 ('with', 8)]
```

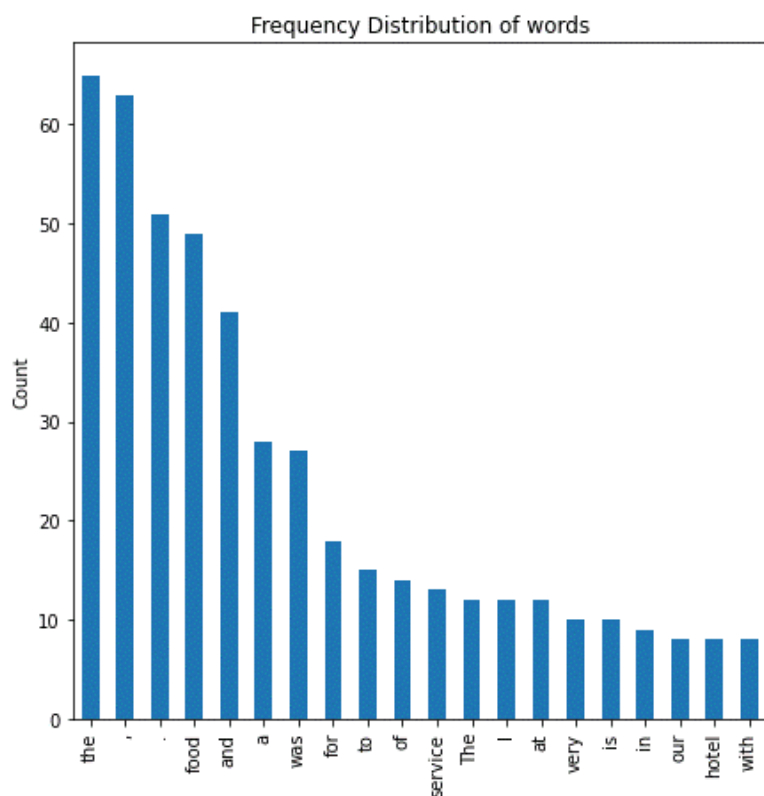
- **Visualizing Frequency Distribution using matplotlib**

```
import matplotlib.pyplot as plt
import pandas as pd

## Conversion to Pandas series via Python Dictionary for easier plotting
all_fdist = pd.Series(dict(all_fdist))

## Setting figure, ax into variables
fig, ax = plt.subplots(figsize=(10,10))

all_fdist.plot(kind='bar')
plt.title("Frequency Distribution of words")
plt.ylabel("Count")
```



As we can see, the frequency distribution contains frequencies of tokens like ‘the’, ‘to’, ‘of’ which are considered as stop words. It also contains special characters and tokens with different case are treated differently. To overcome these issues, we have to perform text cleanup as follows:

➤ **Converting the text to lower case letters.**

```
text = text.lower()
```

➤ **Removing special character from data**

```
import re
text = re.sub('[^A-Za-z0-9]+', ' ', text)
```

➤ **Removing words with numbers from data**

```
text = re.sub("\S*\d\S*", "", text).strip()
```

➤ **Removing Stopwords**

Stop words are a set of commonly used words in any language. For example, in English, “the”, “is” and “and”, would easily qualify as stop words. In NLP and text mining applications, stop words are used to eliminate unimportant words, allowing applications to focus on the important words instead.

NLTK has an inbuilt list of stopwords. We can also create our own stopwords list and use it based on our requirements. The following code removes the stopwords from the text. When we visualize the frequency distribution after text cleaning, it will be as follows:

```
import nltk
words = word_tokenize(text)
stopwords = nltk.corpus.stopwords.words('english')
words_sw_removed = []

## Check if token in stop word list before adding to new list
for word in words:
    if word in stopwords:
        pass
    else:
        words_sw_removed.append(word)
```

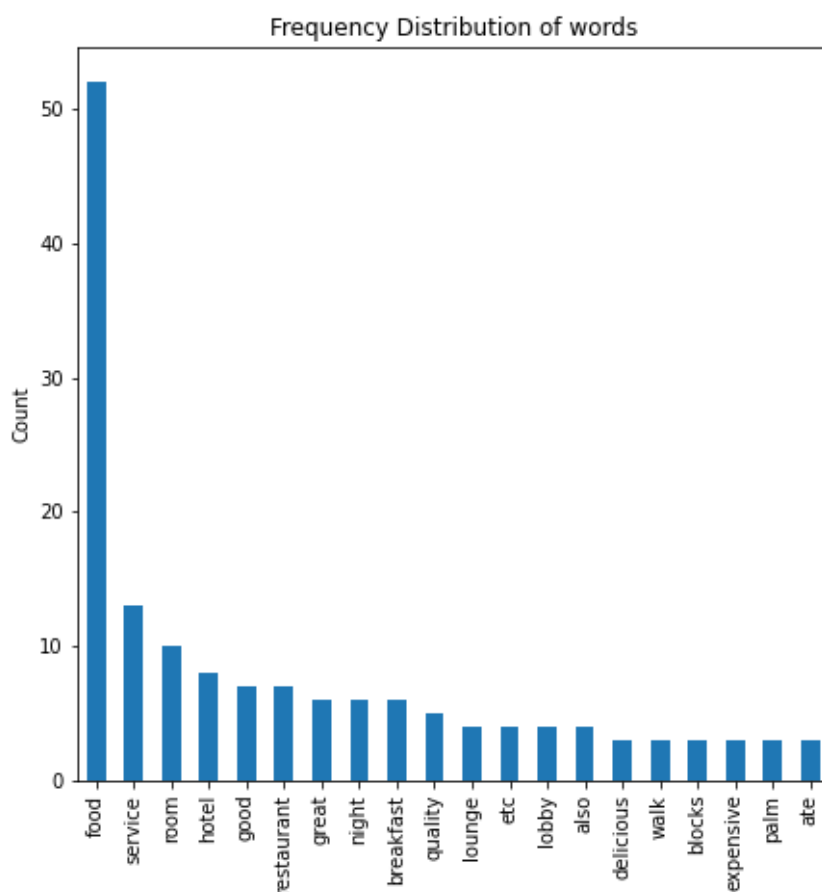
When we draw the Histogram after the text cleaning, it would look like the below figure.

```
all_fdist = FreqDist(words_sw_removed).most_common(20)

## Conversion to Pandas series via Python Dictionary for easier plotting
all_fdist = pd.Series(dict(all_fdist))

## Setting figure, ax into variables
fig, ax = plt.subplots(figsize=(7,7))

all_fdist.plot(kind='bar')
plt.title("Frequency Distribution of words")
plt.ylabel("Count")
```



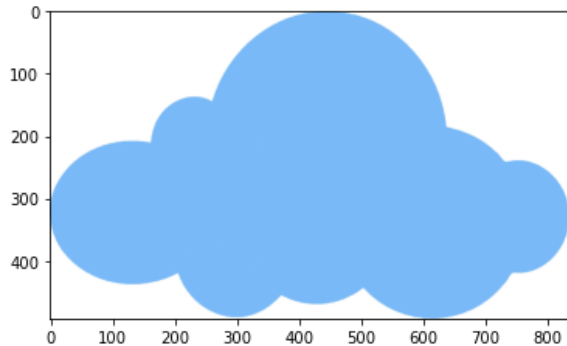






We can set a custom image as a mask to display the word cloud as follows:

```
#reading an image to set the custom outline of Wordcloud
from skimage.io import imread
cloud = imread('cloud.png')
plt.imshow(cloud)
```



```
from wordcloud import WordCloud, STOPWORDS
import matplotlib.pyplot as plt

stopwords = set(STOPWORDS)

wordcloud = WordCloud(width = 800, height = 800,
                       background_color = 'white',
                       stopwords = stopwords,
                       min_font_size = 10,mask=cloud).generate(text)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()
```



**References:**

1. <https://www.nltk.org/api/nltk.tokenize.html>
2. <https://basilkjose.medium.com/data-preprocessing-natural-language-competition-processing>
3. <https://towardsdatascience.com/text-preprocessing-in-natural-language-processing-using-python-6113ff5decd8>
4. Infosys Springboard – Natural Language Processing for developers