**Noise Removal**

Any piece of text which is not relevant to the context of the data and the end-output can be specified as the noise.

For example – language stopwords (commonly used words of a language – is, am, the, of, in etc), URLs or links, social media entities (mentions, hashtags), punctuations and industry specific words. This step deals with removal of all types of noisy entities present in the text.

A general approach for noise removal is to prepare a dictionary of noisy entities, and iterate the text object by tokens (or by words), eliminating those tokens which are present in the noise dictionary.

Following is the python code for the same purpose.

# Sample code to remove noisy words from a text

noise\_list = ["is", "a", "this", "..."]

def \_remove\_noise(input\_text):

words = input\_text.split()

noise\_free\_words = [word for word in words if word not in noise\_list]

noise\_free\_text = " ".join(noise\_free\_words)

return noise\_free\_text

\_remove\_noise("this is a sample text")

Another approach is to use the regular expressions while dealing with special patterns of noise.  Following python code removes a regex pattern from the input text:

# Sample code to remove a regex pattern

import re

def \_remove\_regex(input\_text, regex\_pattern):

urls = re.finditer(regex\_pattern, input\_text)

for i in urls:

input\_text = re.sub(i.group().strip(), '', input\_text)

return input\_text

regex\_pattern = "#[\w]\*"

\_remove\_regex("remove this #hashtag from analytics vidhya", regex\_pattern)

>>> "remove this from analytics vidhya"

**2.3 Object Standardization**

Text data often contains words or phrases which are not present in any standard lexical dictionaries. These pieces are not recognized by search engines and models.

Some of the examples are – acronyms, hashtags with attached words, and colloquial slangs. With the help of regular expressions and manually prepared data dictionaries, this type of noise can be fixed, the code below uses a dictionary lookup method to replace social media slangs from a text.

lookup\_dict = {'rt':'Retweet', 'dm':'direct message', "awsm" : "awesome", "luv" :"love"}

def \_lookup\_words(input\_text):

words = input\_text.split()

new\_words = []

for word in words:

if word.lower() in lookup\_dict:

word = lookup\_dict[word.lower()]

new\_words.append(word)

new\_text = " ".join(new\_words)

return new\_text

\_lookup\_words("RT this is a retweeted tweet by Shivam Bansal")

## Get Synonyms From WordNet

If you remember we installed NLTK packages using nltk.download(). One of the packages was WordNet. WordNet is a database built for natural language processing. It includes groups of synonyms and a brief definition.

You can get these definitions and examples for a given word like this:

from nltk.corpus import wordnet

syn = wordnet.synsets("pain")

print(syn[0].definition())

print(syn[0].examples())

The result is:

a symptom of some physical hurt or disorder

['the patient developed severe pain and distension']

WordNet includes a lot of definitions:

from nltk.corpus import wordnet

syn = wordnet.synsets("NLP")

print(syn[0].definition())

syn = wordnet.synsets("Python")

print(syn[0].definition())

The result is:

the branch of information science that deals with natural language information

large Old World boas

You can use WordNet to get synonymous words like this:

from nltk.corpus import wordnet

synonyms = []

for syn in wordnet.synsets('Computer'):

for lemma in syn.lemmas():

synonyms.append(lemma.name())

print(synonyms)

The output is:

['computer', 'computing\_machine', 'computing\_device', 'data\_processor', 'electronic\_computer', 'information\_processing\_system', 'calculator', 'reckoner', 'figurer', 'estimator', 'computer']

## Get Antonyms From WordNet

You can get the antonyms of words the same way. All you have to do is to check the lemmas before adding them to the array.  it's an antonym or not.

from nltk.corpus import wordnet

antonyms = []

for syn in wordnet.synsets("small"):

for l in syn.lemmas():

if l.antonyms():

antonyms.append(l.antonyms()[0].name())

print(antonyms)

The output is:

['large', 'big', 'big']

## Stemming and Lemmatization Difference

OK, let's try stemming and lemmatization for some words:

from nltk.stem import WordNetLemmatizer

from nltk.stem import PorterStemmer

stemmer = PorterStemmer()

lemmatizer = WordNetLemmatizer()

print(stemmer.stem('stones'))

print(stemmer.stem('speaking'))

print(stemmer.stem('bedroom'))

print(stemmer.stem('jokes'))

print(stemmer.stem('lisa'))

print(stemmer.stem('purple'))

print('----------------------')

print(lemmatizer.lemmatize('stones'))

print(lemmatizer.lemmatize('speaking'))

print(lemmatizer.lemmatize('bedroom'))

print(lemmatizer.lemmatize('jokes'))

print(lemmatizer.lemmatize('lisa'))

print(lemmatizer.lemmatize('purple'))

The result is:

stone

speak

bedroom

joke

lisa

purpl

----------------------

stone

speaking

bedroom

joke

lisa

purple

## Tokenize text using pure Python

First, we will grab a web page content then we will analyze the text to see what the page is about.

We will use the **[urllib module](https://likegeeks.com/python-programming-basics/" \l "Web-Crawling)** to crawl the web page:

import urllib.request

response = urllib.request.urlopen('http://php.net/')

html = response.read()

print (html)

As you can see from the printed output, the result contains a lot of HTML tags that need to be cleaned.

We can use BeautifulSoup to clean the grabbed text like this:

from bs4 import BeautifulSoup

import urllib.request

response = urllib.request.urlopen('http://php.net/')

html = response.read()

soup = BeautifulSoup(html,"html5lib")

text = soup.get\_text(strip=True)

print (text)

Now we have a clean text from the crawled web page.

Awesome, right?

Finally, let’s convert that text into tokens by splitting the text like this:

from bs4 import BeautifulSoup

import urllib.request

response = urllib.request.urlopen('http://php.net/')

html = response.read()

soup = BeautifulSoup(html,"html5lib")

text = soup.get\_text(strip=True)

tokens = [t for t in text.split()]

print (tokens)

## Count word frequency

The text is much better now. Let’s calculate the frequency distribution of those tokens using Python NLTK.

There is a function in NLTK called FreqDist() does the job:

from bs4 import BeautifulSoup

import urllib.request

import nltk

response = urllib.request.urlopen('http://php.net/')

html = response.read()

soup = BeautifulSoup(html,"html5lib")

text = soup.get\_text(strip=True)

tokens = [t for t in text.split()]

freq = nltk.FreqDist(tokens)

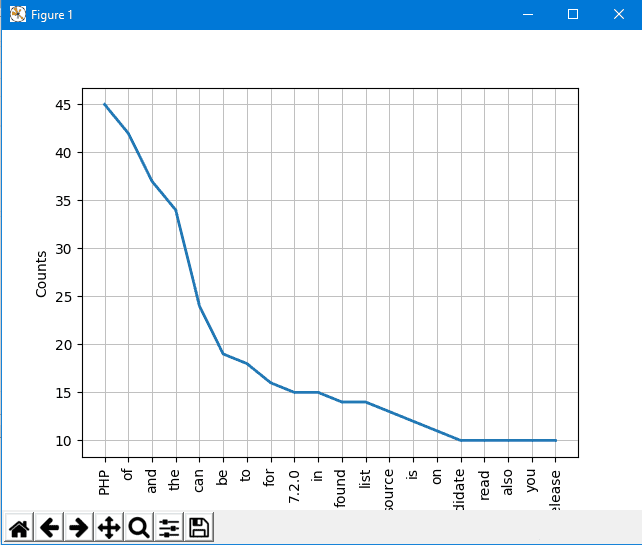
for key,val in freq.items():

print (str(key) + ':' + str(val))

If you search the output, you’ll find that the most frequent token is PHP.

You can plot a graph for those tokens using plot function like this:

freq.plot(20, cumulative=False)



From the graph, you can be sure that this article is talking about PHP.

Great!!

There are some words like The, Of, a, an, and so on. These words are stop words. Generally,  you should remove stop words to prevent them from affecting our results.

## Remove stop words using NLTK

NLTK comes with stop words lists for most languages. To get English stop words, you can use this code:

NLTK comes with stop words lists for most languages. To get English stop words, you can use this code:

from nltk.corpus import stopwords

stopwords.words('english')

Now, let’s modify our code and clean the tokens before plotting the graph.

First, we will make a copy of the list; then we will iterate over the tokens and remove the stop words:

clean\_tokens = tokens[:]

sr = stopwords.words('english')

for token in tokens:

if token in stopwords.words('english'):

clean\_tokens.remove(token)

So the final code should be like this:

You can review the [**Python list functions**](https://likegeeks.com/python-list-functions/) to know how to process lists.

from bs4 import BeautifulSoup

import urllib.request

import nltk

from nltk.corpus import stopwords

response = urllib.request.urlopen('http://php.net/')

html = response.read()

soup = BeautifulSoup(html,"html5lib")

text = soup.get\_text(strip=True)

tokens = [t for t in text.split()]

clean\_tokens = tokens[:]

sr = stopwords.words('english')

for token in tokens:

if token in stopwords.words('english'):

clean\_tokens.remove(token)

freq = nltk.FreqDist(clean\_tokens)

for key,val in freq.items():

print (str(key) + ':' + str(val))

If you check the graph now, it’s better than before since no stop words on the count.

freq.plot(20,cumulative=False)

## Tokenize text using NLTK

We saw how to split the text into tokens using the split function. Now we will see how to tokenize the text using NLTK.

Tokenizing text is important since text can’t be processed without tokenization. The tokenization process means splitting bigger parts into small parts.

You can tokenize paragraphs to sentences and tokenize sentences to words according to your needs. NLTK comes with sentence tokenizer and word tokenizer.

Let’s assume that we have a sample text like the following:

Hello Adam, how are you? I hope everything is going well. Today is a good day, see you dude.

To tokenize this text to sentences, we will use sentence tokenizer:

from nltk.tokenize import sent\_tokenize

mytext = "Hello Adam, how are you? I hope everything is going well. Today is a good day, see you dude."

print(sent\_tokenize(mytext))

The output is the following:

['Hello Adam, how are you?', 'I hope everything is going well.', 'Today is a good day, see you dude.']

You may say that this is an easy job, I don’t need to use NLTK tokenization, and I can split sentences using regular expressions since every sentence precedes by punctuation and space.

Well, take a look at the following text:

Hello Mr. Adam, how are you? I hope everything is going well. Today is a good day, see you dude.

Uh! The word Mr. is one word by itself. OK, let’s try NLTK:

from nltk.tokenize import sent\_tokenize

mytext = "Hello Mr. Adam, how are you? I hope everything is going well. Today is a good day, see you dude."

print(sent\_tokenize(mytext))

The output looks like this:

['Hello Mr. Adam, how are you?', 'I hope everything is going well.', 'Today is a good day, see you dude.']

Great! It works like a charm.

OK, let’s try word tokenizer to see how it will work.

from nltk.tokenize import word\_tokenize

mytext = "Hello Mr. Adam, how are you? I hope everything is going well. Today is a good day, see you dude."

print(word\_tokenize(mytext))

The output is:

['Hello', 'Mr.', 'Adam', ',', 'how', 'are', 'you', '?', 'I', 'hope', 'everything', 'is', 'going', 'well', '.', 'Today', 'is', 'a', 'good', 'day', ',', 'see', 'you', 'dude', '.']

The word Mr. is one word as expected.

NLTK uses PunktSentenceTokenizer which is a part of nltk.tokenize.punkt module.

This tokenizer trained well to work with many languages.