**Roll No : B-05**

**Name : Anuj Somani**

**Batch : TC1**

**Assignment Title : To Study NLTK Library for text Processing.**

**Problem Statement : Study and use of NLTK library.**

**Theory:**

## **Why Text Preprocessing is Needed?**

**Text preprocessing is an important and one the most essential step before building any model in Natural Language Processing. A raw text corpus, collected from one or many sources, may be full of inconsistencies and ambiguity that requires preprocessing for cleaning it up.**

**Even after cleaning, more text preprocessing is required to reshape the data in a manner that it can be fed directly to the model. If text preprocessing is not done properly the data will be as good as garbage and the NLP model produced will be as bad as garbage only.**

### **Common Text Preprocessing Steps**

**We have listed below some of the most commons text preprocessing steps.**

1. **Lowercasing**
2. **Removing Extra Whitespaces**
3. **Tokenization**
4. **Spelling Correction**
5. **Stopwords Removal**
6. **Removing Punctuations**
7. **Removal of Frequent Words**
8. **Stemming**
9. **Lemmatization**
10. **Removal of URLs**
11. **Removal of HTML Tags**

## **Example of Text Preprocessing using NLTK Python**

Let us now see an example of how to carry out text preprocessing using NLTK library, on a sample Twitter text corpus. We will first load the text corpus as below –

In [1]:

import pandas as pd

df=pd.read\_csv('twitter/Tweet.csv')

df\_text=df[['text']]

df\_text.head()

[Out]:

|  | **text** |
| --- | --- |
| **0** | @VirginAmerica What @dhepburn said. |
| **1** | @VirginAmerica plus you’ve added commercials t… |
| **2** | @VirginAmerica I didn’t today… Must mean I n… |
| **3** | @VirginAmerica it’s really aggressive to blast… |
| **4** | @VirginAmerica and it’s a really big bad thing… |

### **i) Lowercasing**

The lowercasing is an important text preprocessing step in which we convert the text into the same casing, preferably all in lowercase so that the words America, America, and AMERICA can be treated in the same way as “america”.

It is helpful in text featurization techniques like term frequency, TFIDF since it prevents duplication of same words having different casing.

In our example, we have used the lower() function of python to convert all the text in the dataframe to the lower case.

In [2]:

df\_text['text']=df\_text['text'].str.lower()

df\_text.head()

[Out]:

|  | **text** |
| --- | --- |
| **0** | @virginamerica what @dhepburn said. |
| **1** | @virginamerica plus you’ve added commercials t… |
| **2** | @virginamerica i didn’t today… must mean i n… |
| **3** | @virginamerica it’s really aggressive to blast… |
| **4** | @virginamerica and it’s a really big bad thing… |

### **ii) Remove Extra Whitespaces**

A text may contain extra whitespace which is not desired as they increase the text size and not add any value to the data. Hence removing extra whitespace is a trivial but important text preprocessing step.

In Python, we can do this by splitting the text and joining it back on the basis of single whitespace. In our example, we have created a function *remove\_whitespace()* and apply it to the dataframe.

In [3]:

def remove\_whitespace(text):

return " ".join(text.split())

# Test

text = " We will going to win this match"

remove\_whitespace(text)

[Out] :

'We will going to win this match'

In [4]:

df\_text['text']=df['text'].apply(remove\_whitespace)

### **iii) Tokenization**

Tokenization is the process of splitting text into pieces called tokens. A corpus of text can be converted into tokens of sentences, words, or even characters.

Usually, you would be converting a text into word tokens during preprocessing as they are prerequisites for many NLP operations.

In NTLK, tokenization can be done by using the word\_tokenize function as shown below in our example. The Twitter texts have been converted into lists of word tokens.

* **Also Read –** [NLTK Tokenize – Complete Tutorial for Beginners](https://machinelearningknowledge.ai/nltk-tokenizer-tutorial-with-word_tokenize-sent_tokenize-whitespacetokenizer-wordpuncttokenizer/)

In [5]:

from nltk import word\_tokenize

df\_text['text']=df\_text['text'].apply(lambda X: word\_tokenize(X))

df\_text.head()

[Out] :

|  | **text** |
| --- | --- |
| **0** | [@, VirginAmerica, What, @, dhepburn, said, .] |
| **1** | [@, VirginAmerica, plus, you, ‘ve, added, comm… |
| **2** | [@, VirginAmerica, I, did, n’t, today, …, Mu… |
| **3** | [@, VirginAmerica, it, ‘s, really, aggressive,… |
| **4** | [@, VirginAmerica, and, it, ‘s, a, really, big… |

### **iv) Spelling Correction**

Spelling correction is another important step in the text preprocessing step. Typos are common in text data and we might want to correct those spelling mistakes before we do our analysis. This ensures we get better results from our models.

Here, we will be using the python package *pyspellchecker* for our spelling correction. We have defined our own custom function *spell\_check()* that corrects the text passed to it.

We apply this function on the dataframe holding the Twitter data of our example.

In [6]:

from spellchecker import SpellChecker

def spell\_check(text):

result = []

spell = SpellChecker()

for word in text:

correct\_word = spell.correction(word)

result.append(correct\_word)

return result

#Test

text = "confuson matrx".split()

spell\_check(text)

[Out] :

['confusion', 'matrix']

In [7]:

df\_text['text'] = df\_text['text'].apply(spell\_check)

df\_text.head()

[Out] :

|  | **text** |
| --- | --- |
| **0** | [@, virginamerica, what, @, dhepburn, said, .] |
| **1** | [@, virginamerica, plus, you, ‘ve, added, comm… |
| **2** | [@, virginamerica, i, did, n’t, today, …, mu… |
| **3** | [@, virginamerica, it, ‘s, really, aggressive,… |
| **4** | [@, virginamerica, and, it, ‘s, a, really, big… |

### **v) Removing Stopwords**

Stopwords are trivial words like “I”, “the”, “you”, etc. that appear so frequently in the text that they may distort many NLP operations without adding much valuable information. So almost always you will have to remove stopwords from the corpus as part of your preprocessing.

NLTK library maintains a list of around 179 stopwords (shown below) that can be used to filtering stopwords from the text. You may also add or remove stopwords from the default list.

* **Also Read** – [Complete Tutorial for NLTK Stopwords](https://machinelearningknowledge.ai/complete-tutorial-for-nltk-stopwords/)

In our example, we have created a function *remove\_stopwords()* for our purpose and then apply it to our Twitter dataframe.

In [8]:

from nltk.corpus import stopwords

print(stopwords.words('english'))

[Out] :

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

In [9]:

en\_stopwords = stopwords.words('english')

def remove\_stopwords(text):

result = []

for token in text:

if token not in en\_stopwords:

result.append(token)

return result

#Test

text = "this is the only solution of that question".split()

remove\_stopwords(text)

[Out] :

['solution', 'question']

In [10]:

df\_text['text'] = df\_text['text'].apply(remove\_stopwords)

df\_text.head()

[Out] :

|  | **text** |
| --- | --- |
| **0** | [@, VirginAmerica, What, @, dhepburn, said, .] |
| **1** | [@, VirginAmerica, plus, ‘ve, added, commercia… |
| **2** | [@, VirginAmerica, I, n’t, today, …, Must, m… |
| **3** | [@, VirginAmerica, ‘s, really, aggressive, bla… |
| **4** | [@, VirginAmerica, ‘s, really, big, bad, thing] |

### **vi) Removing Punctuations**

Removing punctuation is an important text preprocessing step as it also does not add any value to the information. This is a text standardization process that will help to treat words like ‘some.’, ‘some,’, and ‘some’ in the same way.

In Python NLTK, we can make use RegexpTokenizer module for removing punctuations from the text.

In our example, we have created a custom function *remove\_punct()* that removes punctuations from the text passed to it as a parameter, and then it is applied to Twitter dataframes.

In [11]:

from nltk.tokenize import RegexpTokenizer

def remove\_punct(text):

tokenizer = RegexpTokenizer(r"\w+")

lst=tokenizer.tokenize(' '.join(text))

return lst

#Test

text=df\_text['text'][0]

print(text)

remove\_punct(text)

[Out] :

['@', 'VirginAmerica', 'What', '@', 'dhepburn', 'said', '.']

['VirginAmerica', 'What', 'dhepburn', 'said']

In [12]:

df\_text['text'] = df\_text['text'].apply(remove\_punct)

df\_text.head()

[Out]:

|  | **text** |
| --- | --- |
| **0** | [VirginAmerica, What, dhepburn, said] |
| **1** | [VirginAmerica, plus, ve, added, commercials, … |
| **2** | [VirginAmerica, I, n, t, today, Must, mean, I,… |
| **3** | [VirginAmerica, s, really, aggressive, blast, … |
| **4** | [VirginAmerica, s, really, big, bad, thing] |

### **vii) Removing Frequent Words**

Even after removing stopwords, some words occur in our text with very high frequency and yet may not add any value. We can find such functions and can decide to filter them out if you deem so.

NLTK provides a function FreqDist that draws the list of frequency of words from the text and we can use it for our use-case.

In our example, we have defined a function *frequent\_words()* that returns the frequency of the words in the text. We also create a second function *remove\_freq\_words()* that removes the high-frequency words identified by the first function.

In [13]:

from nltk import FreqDist

def frequent\_words(df):

lst=[]

for text in df.values:

lst+=text[0]

fdist=FreqDist(lst)

return fdist.most\_common(10)

frequent\_words(df\_text)

[Out] :

[('I', 6376),

('united', 3833),

('t', 3285),

('flight', 3189),

('USAirways', 2936),

('AmericanAir', 2927),

('SouthwestAir', 2411),

('JetBlue', 2287),

('n', 2092),

('s', 1497)]

In [14]:

freq\_words = frequent\_words(df\_text)

lst = []

for a,b in freq\_words:

lst.append(b)

def remove\_freq\_words(text):

result=[]

for item in text:

if item not in lst:

result.append(item)

return result

df\_text['text']=df\_text['text'].apply(remove\_freq\_words)

### **viii) Lemmatization**

Lemmatization is converting the word to its base form or lemma by removing affixes from the inflected words. It helps to create better features for machine learning and NLP models hence it is an important preprocessing step.

There are many Lematizers available in NLTK that employ different algorithms. In our example, we have used WordNetLemmatizer module of NLTK for lemmatization.

* **Also Read –** [Learn Lemmatization in NTLK with Examples](https://machinelearningknowledge.ai/learn-lemmatization-in-ntlk-with-examples/)

We have created a custom function *lemmatization()* that first does POS tagging and then lemmatizes the text. Finally, this function is applied to our Twitter dataframe.

In [15]:

from nltk.stem import WordNetLemmatizer

from nltk import word\_tokenize,pos\_tag

def lemmatization(text):

result=[]

wordnet = WordNetLemmatizer()

for token,tag in pos\_tag(text):

pos=tag[0].lower()

if pos not in ['a', 'r', 'n', 'v']:

pos='n'

result.append(wordnet.lemmatize(token,pos))

return result

#Test

text = ['running','ran','runs']

lemmatization(text)

[Out]:

['run', 'ran', 'run']

In [16]:

df\_text['text']=df\_text['text'].apply(lemmatization)

df\_text.head()

[Out]:

|  | **text** |
| --- | --- |
| **0** | [VirginAmerica, What, dhepburn, say] |
| **1** | [VirginAmerica, plus, ve, added, commercial, e… |
| **2** | [VirginAmerica, I, n, t, today, Must, mean, I,… |
| **3** | [VirginAmerica, s, really, aggressive, blast, … |
| **4** | [VirginAmerica, s, really, big, bad, thing] |

### **ix) Stemming**

Stemming also reduces the words to their root forms but unlike lemmatization, the stem itself may not a valid word in the Language.

NLTK has many stemming functions with different algorithms, we will use PorterStemmer over here.

* **Also Read –** [Beginner’s Guide to Stemming in Python NLTK](https://machinelearningknowledge.ai/beginners-guide-to-stemming-in-python-nltk/)

You will like to either perform stemming or lemmatization and not both. We will however perform stemming on our data just to explain to you.

We have defined a custom function *stemming()* that returns the text by converting the words to stem, we finally apply it to Twitter dataframe.

In [17]:

from nltk.stem import PorterStemmer

def stemming(text):

porter = PorterStemmer()

result=[]

for word in text:

result.append(porter.stem(word))

return result

#Test

text=['Connects','Connecting','Connections','Connected','Connection','Connectings','Connect']

stemming(text)

[Out]:

['connect', 'connect', 'connect', 'connect', 'connect', 'connect', 'connect']

In [18]:

df\_text['text']=df\_text['text'].apply(stemming)

df\_text.head()

[Out]:

|  | **text** |
| --- | --- |
| **0** | [virginamerica, what, dhepburn, say] |
| **1** | [virginamerica, plu, ve, ad, commerci, experi,… |
| **2** | [virginamerica, I, n, t, today, must, mean, I,… |
| **3** | [virginamerica, s, realli, aggress, blast, obn… |
| **4** | [virginamerica, s, realli, big, bad, thing] |

### **x) Removal of Tags**

If we scrape our data from a different website, removing HTML tags becomes an essential step as part of our preprocessing.

We can use Python regular expression function to find all the unwanted tags. Here in this example, we have defined a custom function remove\_tag() which cleans the HTML tags from the text by using regular expression. And finally, we apply this function to our Twitter dataframe.

In [19]:

import re

def remove\_tag(text):

text=' '.join(text)

html\_pattern = re.compile('<.\*?>')

return html\_pattern.sub(r'', text)

#Test

text = "<HEAD> this is head tag </HEAD>"

remove\_tag(text.split())

[Out] :

' this is head tag '

In [20]:

df\_text['text'] = df\_text['text'].apply(remove\_tag)

df\_text.head()

[Out] :

|  | **text** |
| --- | --- |
| **0** | virginamerica what dhepburn say |
| **1** | virginamerica plu ve ad commerci experi tacki |
| **2** | virginamerica I n t today must mean I need tak… |
| **3** | virginamerica s realli aggress blast obnoxi en… |
| **4** | virginamerica s realli big bad thing |

### **xi) Removal of URLs**

The next preprocessing step is to remove any URLs present in the data. For example, if we are doing a Twitter analysis, then there is a good chance that the tweet will have some URL in it.

Probably we might need to remove them for our further analysis. Here also we use Python regular expression to find out URLs.

We have created a custom function *remove\_url()* that finds the URL and removes them from the text. This function is then applied to the dataframe of our example.

In [21]:

def remove\_urls(text):

url\_pattern = re.compile(r'https?://\S+|www\.\S+')

return url\_pattern.sub(r'', text)

#Test

text = "Machine learning knowledge is an awsome site. Here is the link for it https://machinelearningknowledge.ai/"

remove\_urls(text)

[Out]:

'Machine learning knowledge is an awsome site. Here is the link for it '

In [22]:

df\_text['text'] = df\_text['text'].apply(remove\_urls)

df\_text.head()

[Out] :

|  | **text** |
| --- | --- |
| **0** | virginamerica what dhepburn say |
| **1** | virginamerica plu ve ad commerci experi tacki |
| **2** | virginamerica I n t today must mean I need tak… |
| **3** | virginamerica s realli aggress blast obnoxi en… |
| **4** | virginamerica s realli big bad thing |

## Conclusion

Hope you learned various techniques for text preprocessing with NLTK and Python other general libraries. Do remember that not all preprocessing steps might be applicable to your use-case and data.

Reference – [NLTK Documentation](https://www.nltk.org/)