Text Preprocessing in NLP

Natural Language Processing (NLP) is a branch of Data Science which deals with Text data. Apart from numerical data, Text data is available to a great extent which is used to analyse and solve business problems. But before using the data for analysis or prediction, processing the data is important.

Text preprocessing in natural language processing (NLP) stands as a crucial gateway that allows machine learning models to extract structured information from unstructured text. This process, involving cleaning and transforming textual data, prepares it for analysis, manipulation, or the generation of new text. Within the realm of text preprocessing, several key techniques come into play, including tokenization, stemming, lemmatization, stop-word removal, and part-of-speech tagging. These techniques collectively contribute to refining raw text data and making it amenable to various NLP tasks.

Some of the preprocessing steps are:

- **Removing punctuations**

In this step, all the punctuations from the text are removed. **string** library of Python contains some pre-defined list of punctuations such as '!" #$%&'()*+,-./:;?@[\]^_`{|}~'

```
# Punctuation Removal
import string
string.punctuation

# Function to remove punctuation
def remove_punctuation(text):
    punctuationfree="".join([i for i in text if i not in string.punctuation])
    return punctuationfree
```

- **Removing non-word and non-whitespace characters**

It is essential to remove any characters that are not considered as words or whitespace from the text dataset. These non-word and non-whitespace characters can include punctuation marks, symbols, and other special characters that do not provide any meaningful information for our analysis.

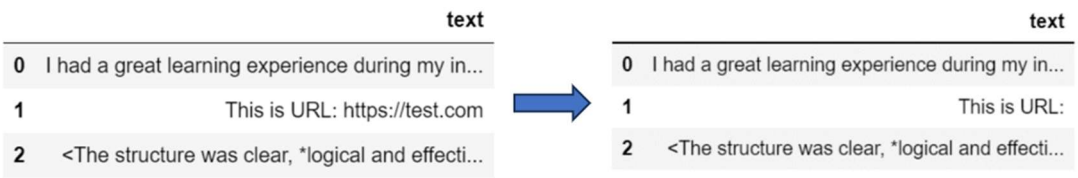
- **Removing URLs**

When building a model, URLs are typically not relevant and can be removed from the text data. For removing URLs, we can use 'regex' library.

```
# Removing URLs

import re
# Define a regex pattern to match URLs
url_pattern = re.compile(r'https?://\S+')

# Function to remove URLs from text
def remove_urls(text):
    return url_pattern.sub('', text)
```



	text
0	I had a great learning experience during my in...
1	This is URL: https://test.com
2	<The structure was clear, *logical and effecti...

	text
0	I had a great learning experience during my in...
1	This is URL:
2	<The structure was clear, *logical and effecti...

- **Removing Stop words**

Stopwords refer to the most commonly occurring words in any natural language. For the purpose of analysing text data and building NLP models, these stopwords might not add much value to the meaning of the document. Therefore, removing stopwords can help us to focus on the most important information in the text and improve the accuracy of our analysis.

NLTK library consists of a list of words that are considered stopwords for the English language. Some of them are : [I, me, my, myself, we, our, ours, ourselves, you, you're, you've, you'll, you'd, your, yours, yourself, yourselves, he, most, other, some, such, no, nor, not, only, own, same, so, then, too, very, s, t, can, will, just, don, don't, should, should've, now, d, ll, m, o, re, ve, y, ain, aren't, could, couldn't, didn't, didn't].We can create a customized list of stop words for different problems.

Stopword removal

One of the advantages of removing stopwords is that it can reduce the size of the dataset, which in turn reduces the training time required for natural language processing models.

```
# Stopwords removal
import nltk
stopwords = nltk.corpus.stopwords.words('english')
stopwords[0:10]
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're"]
# Function to remove stopwords from tokenized text
def remove_stopwords(text):
    output= [i for i in text if i not in stopwords]
    return output
```

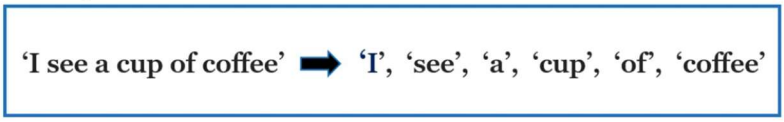
- **Converting to lowercase**

Python is a case sensitive programming language. Therefore, to avoid any issues and ensure consistency in the processing of the text, we convert all the text to lowercase.

```
# Lower casing
def lowercase(text):
    return text.lower()
```

- **Tokenization**

Tokenization is the process of breaking down large blocks of text such as paragraphs and sentences into smaller, more manageable units.



Tokenization of text

By performing word tokenization, we can obtain a more accurate representation of the underlying patterns and trends present in the text data.

```
# Tokenisation
import re
def tokenization(text):
    tokens = re.split('W+',text)
    return tokens
```

- **Stemming**

Stemming means mapping a group of words to the same stem by removing prefixes or suffixes without giving any value to the "grammatical meaning" of the stem formed after the process. But the disadvantage of stemming is that it stems the words such that its root form loses the meaning or it is not diminished to a proper English word.

Stemmer — It is an algorithm to do stemming
1. Porter Stemmer — specific for the English language
2. Snowball Stemmer — used for multiple languages
3. Lancaster Stemmer

```
# Stemming - Porter Stemmer algorithm
import nltk
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
import pandas as pd

# Initialize the Porter Stemmer
stemmer = PorterStemmer()

# Define a function to perform stemming on the 'text' column
def stem_words(words):
    return [stemmer.stem(word) for word in words]
```

A comparison of three stemming algorithms on a sample text.

Sample text: Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

Lovins stemmer: such an analys can reve featur that ar not eas vis from th vari in th individu gen and can lead to a pictur of expres that is mor biolog transpar and acces to interpres

Porter stemmer: such an analysi can reveal featur that ar not easili visibl from the variat in the individu gene and can lead to a pictur of express that is more biolog transpar and access to interpret

- **Lemmatization**

Lemmatization aims to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the 'lemma. It also does the same thing as stemming and tries to bring a word to its base form, but unlike stemming it does keep into account the actual meaning of the base word. Lemmatization has a pre-defined dictionary that stores the context of words and checks the word in the dictionary while diminishing.

```python
# Lemmatization
from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()
# Function for lemmatization
def lemmatizer(text):
lemm_text = [wordnet_lemmatizer.lemmatize(word) for word in text]
    return lemm_text
```

The difference between Stemming and Lemmatization can be understood with the example provided below.

Original Word	After Stemming	After Lemmatization
goose	goos	goose
geese	gees	goose

After all the text processing steps are performed, the final acquired data is converted into the numeric form using Bag of words or TF-IDF.
