Text Preprocessing in Python

Text Preprocessing is one of the initial steps of Natural Language Processing (NLP) that involves cleaning and transforming raw data into suitable data for further processing. It enhances the quality of the text makes it easier to work and improves the performance of machine learning models.

In this article, we will look at some more advanced text preprocessing techniques.

Prerequisites

> Before starting with this article, you need to go through the Text Preprocessing in Python | Set 1.
>
> Also, refer to this article to learn more about Natural Language Processing – Introduction to NLP

We can see the basic preprocessing steps when working with textual data. We can use these techniques to gain more insights into the data that we have. Let's import the necessary libraries.

Python3` ```

import the necessary libraries

import nltk nltk.download('punkt') nltk.download('averaged_perceptron_tagger') import string import re

```
      `


Part of Speech Tagging
----------------------

The [part of speech](https://www.geeksforgeeks.org/nlp-part-of-speech-default-tagging/) explai

Python3`   ```
from nltk.tokenize import word_tokenize
from nltk import pos_tag
# convert text into word_tokens with their tags
def pos_tagging(text):
  word_tokens = word_tokenize(text)
  return pos_tag(word_tokens)
pos_tagging('You just gave me a scare')
```

```
```

Output:

```
[('You', 'PRP'),
 ('just', 'RB'),
 ('gave', 'VBD'),
 ('me', 'PRP'),
 ('a', 'DT'),
 ('scare', 'NN')]
```

In the given example, PRP stands for personal pronoun, RB for adverb, VBD for verb past tense, DT for determiner and NN for noun. We can get the details of all the part of speech tags using the Penn Treebank tagset.

Python3` ```

download the tagset

nltk.download('tagsets')

extract information about the tag

nltk.help.upenn_tagset('NN')

```
      `

****Output:****
```

NN: noun, common, singular or mass common-carrier cabbage knuckle-duster Casino afghan shed thermostat investment slide humour falloff slick wind hyena override subhumanity machinist ...

```
Chunking
--------

Chunking is the process of extracting phrases from unstructured text and more structure to it.
```

```
Python3`   ```
from nltk.tokenize import word_tokenize
from nltk import pos_tag
# define chunking function with text and regular
# expression representing grammar as parameter
def chunking(text, grammar):
    word_tokens = word_tokenize(text)
# label words with part of speech
    word_pos = pos_tag(word_tokens)
# create a chunk parser using grammar
    chunkParser = nltk.RegexpParser(grammar)
# test it on the list of word tokens with tagged pos
    tree = chunkParser.parse(word_pos)
    for subtree in tree.subtrees():
        print(subtree)

sentence = 'the little yellow bird is flying in the sky'
grammar = &quot;NP: {&lt;DT&gt;?&lt;JJ&gt;*&lt;NN&gt;}&quot;
chunking(sentence, grammar)




 `
```

Output:

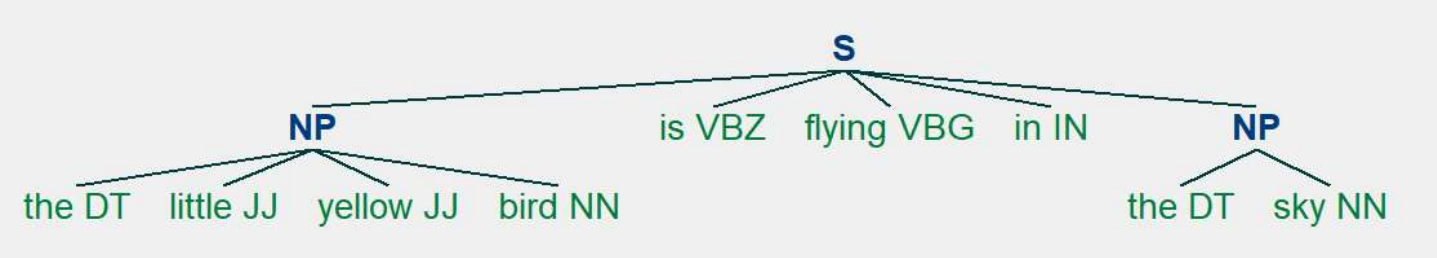
```
(S
  (NP the/DT little/JJ yellow/JJ bird/NN)
  is/VBZ
  flying/VBG
  in/IN
  (NP the/DT sky/NN))
(NP the/DT little/JJ yellow/JJ bird/NN)
(NP the/DT sky/NN)
```

In the given example, grammar, which is defined using a simple regular expression rule. This rule says that an NP (Noun Phrase) chunk should be formed whenever the chunker finds an optional determiner (DT) followed by any number of adjectives (JJ) and then a noun (NN). Libraries like spaCy and Textblob are more suited for chunking.

Example:

> **Input:** ‘the little yellow bird is flying in the sky’

> **Output:** (S (NP the/DT little/JJ yellow/JJ bird/NN) is/VBZ flying/VBG in/IN (NP the/DT sky/NN))
> (NP the/DT little/JJ yellow/JJ bird/NN) (NP the/DT sky/NN)



Named Entity Recognition

As we know Named Entity Recognition is used to extract information from unstructured text. It is used to classify entities present in a text into categories like a person, organization, event, places, etc. It gives us detailed knowledge about the text and the relationships between the different entities.

Python3` ``` from nltk.tokenize import word_tokenize from nltk import pos_tag, ne_chunk def named_entity_recognition(text): # tokenize the text word_tokens = word_tokenize(text)

```
# part of speech tagging of words
word_pos = pos_tag(word_tokens)

# tree of word entities
print(ne_chunk(word_pos))
```

text = 'Bill works for GeeksforGeeks so he went to Delhi for a meetup.'
named_entity_recognition(text)

```
`
```

****Example:****

> ****Input:**** ‘Bill works for GeeksforGeeks so he went to Delhi for a meetup.’
>
> ****Output:**** (S
> (PERSON Bill/NNP)
> works/VBZ
> for/IN
> (ORGANIZATION GeeksforGeeks/NNP)
> so/RB
> he/PRP
> went/VBD
> to/TO
> (GPE Delhi/NNP)

```
> for/IN
> a/DT
> meetup/NN
> ./.)

Conclusion
----------

In conclusion, natural language processing (NLP) plays a pivotal role in bridging the gap betw
```