**Most Common Text Processing Tasks In Natural Language Processing**

**Computers haven’t human capability, text data need processing for better understanding and interpretation**



Photo by [Yannick Pulver](https://unsplash.com/@yanu) on [Unsplash](https://unsplash.com/photos/hopX_jpVtRM)

**Introduction**

Human being has the capability to understand written textual information. Machines on the other hand do not have that intrinsic…

In this conceptual article, we will explain how to perform the most common text-processing tasks using popular Python libraries such as NLTK, and Spacy.

**Most Common Tasks**

Text preprocessing involves tokenization, stopwords removal, stemming and lemmatization, part of speech tagging, and named entity recognition. This section focuses on explaining each one of them and their python implementation.

The corresponding video of the article is available below.

**Prerequisites**

To begin, you will need to have Python installed on your computer along with the following libraries:

* NLTK
* Spacy
* Scikit-learn

You can install these libraries using pip, the Python package manager as follows:

# Install NLTK  
pip install nltk  
  
# Install Spacy  
pip install spacy  
  
# Install Scikit-learn  
pip install scikit-learn

Now, let’s import the necessary modules and load the dataset used for the experimentation. We will use the built-in news article data from Scikit-learn.

import nltk  
import spacy  
nltk.download('punkt')  
from sklearn.datasets import fetch\_20newsgroups

We can then use the fetch\_20newsgroups function to download and load the news data as follows by accessing the data attribute:

news\_data = fetch\_20newsgroups(subset='all')  
articles = news\_data.data

Let’s have a look at the first article:

print(articles[0])

This should generate the following output:

From: Mamatha Devineni Ratnam <@andrew.cmu.edu">mr47+@andrew.cmu.edu>  
Subject: Pens fans reactions  
Organization: Post Office, Carnegie Mellon, Pittsburgh, PA  
Lines: 12  
NNTP-Posting-Host: po4.andrew.cmu.edu  
  
  
I am sure some bashers of Pens fans are pretty confused about the lack  
of any kind of posts about the recent Pens massacre of the Devils. Actually,  
I am bit puzzled too and a bit relieved. However, I am going to put an end  
to non-PIttsburghers' relief with a bit of praise for the Pens. Man, they  
are killing those Devils worse than I thought. Jagr just showed you why  
he is much better than his regular season stats. He is also a lot  
fo fun to watch in the playoffs. Bowman should let JAgr have a lot of  
fun in the next couple of games since the Pens are going to beat the pulp out of Jersey anyway. I was very disappointed not to see the Islanders lose the final  
regular season game. PENS RULE!!!

Now that everything is set up, it’s time to dive deep into each task, starting with tokenization.

**Tokenization**

This is the easiest step in text processing and consists in splitting the text into tokens. The tokens generated depend on the underlying tokenization approach. For instance:

* Word tokenization generates words.
* Sentence tokenization splits the piece of text into sentences.

The word and sentence tokenizations are respectively performed using the work\_tokenize() and sent\_tokenize() functions from the NLTK library.

# Import word and sentence tokenizers  
from nltk.tokenize import word\_tokenize, sent\_tokenize

We can then proceed with the tokenizations as follow after initializing the variable first\_article with the second bloc of the previous article:

# Generate Word tokens  
word\_tokens = word\_tokenize(first\_article)  
  
# Generate Sentence Tokens  
sentence\_tokens = sent\_tokenize(first\_article)  
  
# Print the results  
print(word\_tokens)  
print(sentence\_tokens)

The previous print statements should generate these outputs. The first one is the word tokens, and the second is the sentence tokens.

['I', 'am', 'sure', 'some', 'bashers', 'of', 'Pens', 'fans', 'are', 'pretty', 'confused', 'about', 'the', 'lack', 'of', 'any', 'kind', 'of', 'posts', 'about', 'the', 'recent', 'Pens', 'massacre', 'of', 'the', 'Devils', '.', 'Actually', ',', 'I', 'am', 'bit', 'puzzled', 'too', 'and', 'a', 'bit', 'relieved', '.', 'However', ',', 'I', 'am', 'going', 'to', 'put', 'an', 'end', 'to', 'non-PIttsburghers', "'", 'relief', 'with', 'a', 'bit', 'of', 'praise', 'for', 'the', 'Pens', '.', 'Man', ',', 'they', 'are', 'killing', 'those', 'Devils', 'worse', 'than', 'I', 'thought', '.', 'Jagr', 'just', 'showed', 'you', 'why', 'he', 'is', 'much', 'better', 'than', 'his', 'regular', 'season', 'stats', '.', 'He', 'is', 'also', 'a', 'lot', 'fo', 'fun', 'to', 'watch', 'in', 'the', 'playoffs', '.', 'Bowman', 'should', 'let', 'JAgr', 'have', 'a', 'lot', 'of', 'fun', 'in', 'the', 'next', 'couple', 'of', 'games', 'since', 'the', 'Pens', 'are', 'going', 'to', 'beat', 'the', 'pulp', 'out', 'of', 'Jersey', 'anyway', '.', 'I', 'was', 'very', 'disappointed', 'not', 'to', 'see', 'the', 'Islanders', 'lose', 'the', 'final', 'regular', 'season', 'game', '.', 'PENS', 'RULE', '!', '!', '!']

Showing each word token can be too much, but we can show all the sentences as follows:

I am sure some bashers of Pens fans are pretty confused about the lack  
of any kind of posts about the recent Pens massacre of the Devils.  
  
  
Actually, I am bit puzzled too and a bit relieved.  
  
  
However, I am going to put an end to non-PIttsburghers' relief with a bit of praise for the Pens.  
  
  
Man, they are killing those Devils worse than I thought.  
  
  
Jagr just showed you why he is much better than his regular season stats.  
  
  
He is also a lot  
fo fun to watch in the playoffs.  
  
  
Bowman should let JAgr have a lot of  
fun in the next couple of games since the Pens are going to beat the pulp out of Jersey anyway.  
  
  
I was very disappointed not to see the Islanders lose the final regular season game.  
  
  
PENS RULE!!

The sentence tokenizer identifies a new sentence after the . sign.

**Stop words removal**

Looking at the previous word tokens we can see that some terms such as an, a, of, the, etc. These words are known as stop words because they do not carry much meaning compared to other words. So, removing them can make the information easier to work with. This can be achieved using the words() function from the nltk.corpus.stopwords module.

from nltk.corpus import stopwords

Since we are working with an English text, we need to load the underlying stop words as follows:

# Acquire the stop words  
english\_stw = stopwords.words("english")

Finally, we can filter across all the word tokens and only keep the non-stop words.

non\_stop\_words = [word for word in word\_tokens if word not in english\_stw]  
  
# Show the final stop words  
print(non\_stop\_words)

The previous print statement shows the following result:

['I', 'sure', 'bashers', 'Pens', 'fans', 'pretty', 'confused', 'lack', 'kind', 'posts', 'recent', 'Pens', 'massacre', 'Devils', '.', 'Actually', ',', 'I', 'bit', 'puzzled', 'bit', 'relieved', '.', 'However', ',', 'I', 'going', 'put', 'end', 'non-PIttsburghers', "'", 'relief', 'bit', 'praise', 'Pens', '.', 'Man', ',', 'killing', 'Devils', 'worse', 'I', 'thought', '.', 'Jagr', 'showed', 'much', 'better', 'regular', 'season', 'stats', '.', 'He', 'also', 'lot', 'fo', 'fun', 'watch', 'playoffs', '.', 'Bowman', 'let', 'JAgr', 'lot', 'fun', 'next', 'couple', 'games', 'since', 'Pens', 'going', 'beat', 'pulp', 'Jersey', 'anyway', '.', 'I', 'disappointed', 'see', 'Islanders', 'lose', 'final', 'regular', 'season', 'game', '.', 'PENS', 'RULE', '!', '!', '!']

**Punctuations removal**

If stop words are not relevant, so are the punctuations! We can easily get rid of punctuation ( .,; etc.) using the native string module in Python.

import string  
  
without\_punct = list(filter(lambda word: word not in string.punctuation, non\_stop\_words))  
  
print(without\_punct)

['I', 'sure', 'bashers', 'Pens', 'fans', 'pretty', 'confused', 'lack', 'kind', 'posts', 'recent', 'Pens', 'massacre', 'Devils', 'Actually', 'I', 'bit', 'puzzled', 'bit', 'relieved', 'However', 'I', 'going', 'put', 'end', 'non-PIttsburghers', 'relief', 'bit', 'praise', 'Pens', 'Man', 'killing', 'Devils', 'worse', 'I', 'thought', 'Jagr', 'showed', 'much', 'better', 'regular', 'season', 'stats', 'He', 'also', 'lot', 'fo', 'fun', 'watch', 'playoffs', 'Bowman', 'let', 'JAgr', 'lot', 'fun', 'next', 'couple', 'games', 'since', 'Pens', 'going', 'beat', 'pulp', 'Jersey', 'anyway', 'I', 'disappointed', 'see', 'Islanders', 'lose', 'final', 'regular', 'season', 'game', 'PENS', 'RULE']

**Stemming & Lemmatization**

Sometimes in the same document, we can find words like confused, confusing, confused, confuses, confuse, confused . Using all of them in the case of large datasets where performance is crucial can be problematic. This is where stemming and lemmatization become useful. They aim to reduce such words to their base words.

You can have an in-depth understanding of these technics, and their differences from my article [Stemming, Lemmatization — Which One is Worth Going For?](https://towardsdatascience.com/stemming-lemmatization-which-one-is-worth-going-for-77e6ec01ad9c)

Let’s consider the following sample text for this section:

sample\_text = """This thing really confuses.   
 But you confuse me more than what is written here.   
 So stay away from explaining things you do not understand.   
 """

We can then use the nltk.stem to import both the PorterStemmer and WordNetLemmatizer to perform respectively stemming and lemmatization using these two helper functions.

def stem\_words(sentence, model=my\_stemmer):  
   
 for word in sentence.split():  
 stem = model.stem(word)  
 print("Word: {} --->: {}".format(word, stem))  
  
  
def lemmatize\_words(sentence, model = my\_lemmatizer):  
  
 for word in sentence.split():  
 lemma = model.lemmatize(word)  
 print("Word: {} --->: {}".format(word, lemma))

stem\_words performs the stemming by showing the original word and the left and the stemmed word on the right of the arrow. The same approach applies to lemmatization using the lemmatize\_words function.

Before we can use these two functions, we need to set up the two models as shown below:

→ **Lemmatizer:** the lemmatizer requires the wordnet lexicon database and the OMW module which uses multilingual wordnet hence they need to be downloaded as well.

# Import the Lemmatizer module  
from nltk.stem import WordNetLemmatizer  
  
# Download wordnet lexicon database  
nltk.download('wordnet')  
nltk.download('omw-1.4')  
  
# Instanciate Lemmatizer  
my\_lemmatizer = WordNetLemmatizer()

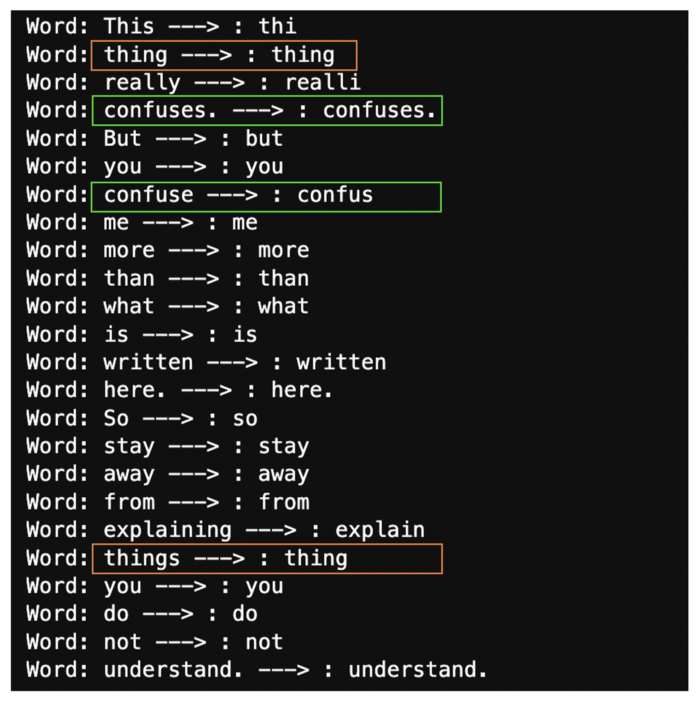
→ **Stemmer**: this is straightforward, and is configured as follows:

Import the Stemmer module  
from nltk.stem.porter import PorterStemmer  
  
# Create instance of stemmer  
my\_stemmer = PorterStemmer()

Now that we have configured the two models, let’s run them on the sample text.

# Run the stemming  
stem\_words(sample\_text, model=my\_stemmer)

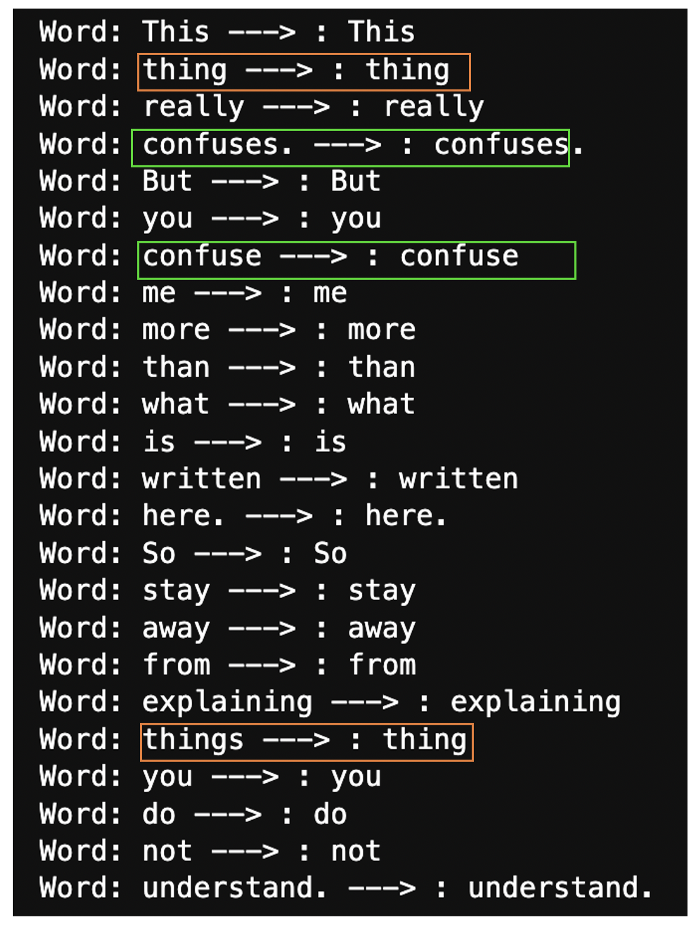
This should show the following result



Stemming illustration (Image by Author)

Similarly, we can perform the lemmatization as follows:

lemmatize\_words(sample\_text, model = my\_lemmatizer)



Lemmatization illustration (Image by Author)

From each output, we can observe on the right side that some worlds have been considered to be the same as their stem, lemma, while some are completely transformed, especially things, confuse.

**Part of speech tagging**

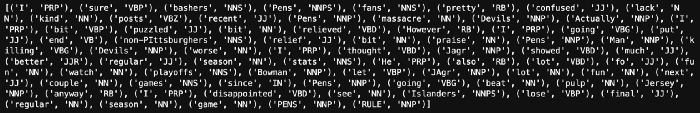
For a given sentence, how do you know which one is a noun, verb, adjective, pronoun, etc? These parts are called Part of Speechand they can help you understand the structure and the underlying meaning of that sentence. The task is called Part of speech tagging . It automatically assigns a part of speech to each word within the sentence.

Using the list of tokens, we can the pos\_tag function to assign each one the corresponding part of speech. The final result of pos\_tag is a list of tuples where each one has a token and is part of a speech tag. Below is an illustration.

The tagger requires the module averaged\_perceptron\_tagger which contains the pre-trained English.

# Import the module  
from nltk.tag import pos\_tag  
  
# Download the tagger  
nltk.download('averaged\_perceptron\_tagger')  
  
# Perform the post tagging  
tagged\_tokens = pos\_tag(without\_punct)  
  
# Show the final result  
print(tagged\_tokens)

This previous print statement shows the following result:



Terms and tags (Image by Author)

In our previous example:

* **I** is a Pronoun (PRP)
* **confused** is an adjective (JJ)
* **non-Pittsburgers** is a Noun, Plural (NNS)
* **since** is a preposition or subordinating conjuction (IN)

You can find the [part of speech tag sets](https://www.ibm.com/docs/en/wca/3.5.0?topic=analytics-part-speech-tag-sets).

**Named entity recognition**

Named Entity Recognition (NER for short) and part of speech tagging are sometimes confused. Even though they are two related tasks, they are totally different. NER consists of the identification and classification of named entities such as persons, organizations, locations, etc. from a text.

My article [Named Entity Recognition with Spacy and the Mighty roBERTa](https://medium.com/towards-data-science/named-entity-recognition-with-spacy-and-the-mighty-roberta-97d879f981) provides a clear and concise explanation of NER, along with the Python code.

**Conclusion**

In this conceptual article, you have learned some commonly used approaches to perform the common text-processing tasks in Natural Language Processing.

Also, If you like reading my stories and wish to support my writing, consider [becoming a Medium member](https://zoumanakeita.medium.com/membership). With a $ 5-a-month commitment, you unlock unlimited access to stories on Medium.

Feel free to follow me on [Medium](https://zoumanakeita.medium.com/), [Twitter](https://twitter.com/zoumana_keita_), and [YouTube](https://www.youtube.com/channel/UC9xKdy8cz6ZuJU5FTNtM_pQ), or say Hi on [LinkedIn](https://www.linkedin.com/in/zoumana-keita/). It is always a pleasure to discuss AI, ML, Data Science, NLP, and MLOps stuff!