Natural Language Processing | Text Preprocessing | Spacy vs NLTK

Rishi Kumar · Follow

Published in Nerd For Tech · 5 min read · Aug 6, 2021

Natural Language Processing is an area of computer science and artificial intelligence concerned with the interactions between computers and human (natural) languages, in particular how to program computer to process and analyze large amounts of natural language data. There are some basics preprocessing steps available for text data.

1. Tokenization
2. Stemming
3. Lemmatization
4. Stop Words

What is Spacy?

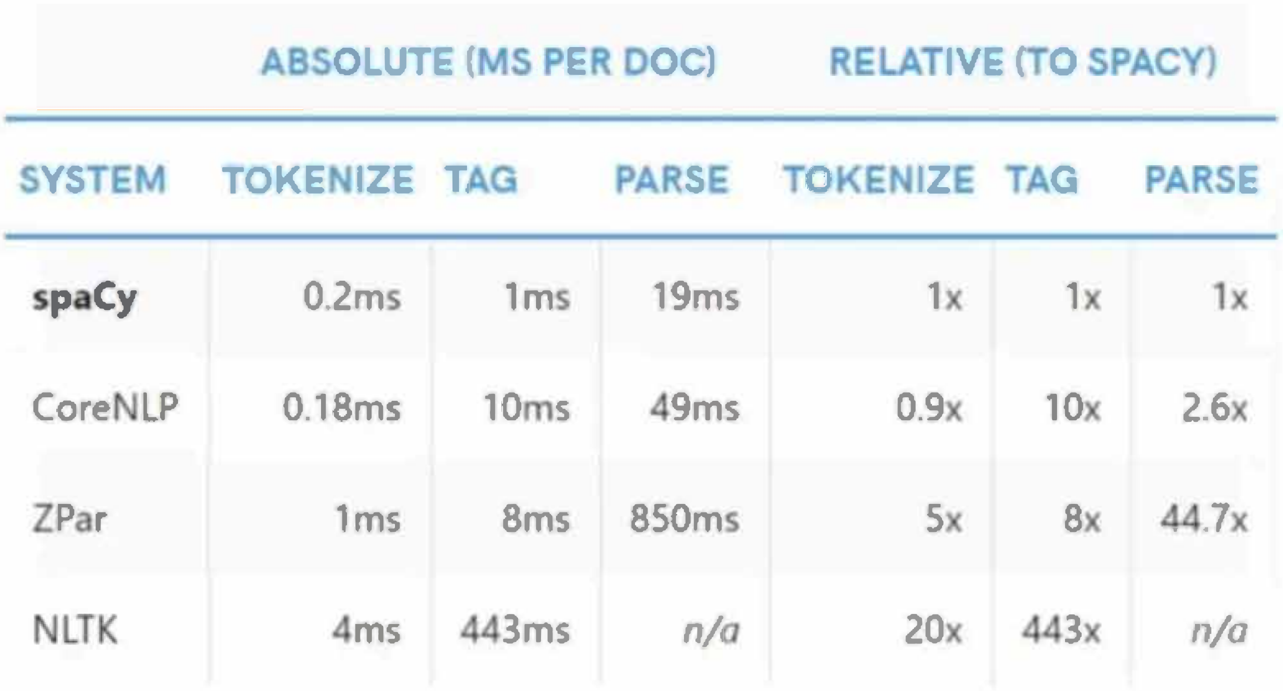
Open source Natural Language Processing library launched in 2015. Designed to effectively handle NLP tasks with the most efficient implementation of common algorithms. For many NLP tasks, Spacy only has one implemented method, choosing the most efficient algorithm currently available. This means you often don't have the option to choose other algorithms.

What is NLTK?

NLTK — Natural Language Toolkit is a very popular open source library launched in 2001. It also provides many functionalities, but includes less efficient implementations.

Spacy VS NLTK

For very common NLP tasks, Spacy is much faster and more efficient than NLTK, at the cost of the user not being able to choose algorithmic implementations. However, Spacy does not include pre-created models for some applications, such as sentiment analysis, which is typically easier to perform with NLTK.

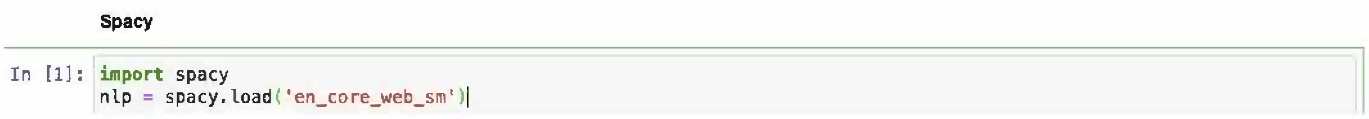


	ABSOLUTE (MS PER DOC)			RELATIVE (TO SPACY)		
SYSTEM	TOKENIZE	TAG	PARSE	TOKENIZE	TAG	PARSE
spaCy	0.2ms	1ms	19ms	1x	1x	1x
CoreNLP	0.18ms	10ms	49ms	0.9x	10x	2.6x
ZPar	1ms	8ms	850ms	5x	8x	44.7x
NLTK	4ms	443ms	*n/a*	20x	443x	*n/a*

NLTK vs Spacy Processing Steps

Spacy consumes lesser time when compared to NLTK for various operations like Tokenize, Tag, Parse.

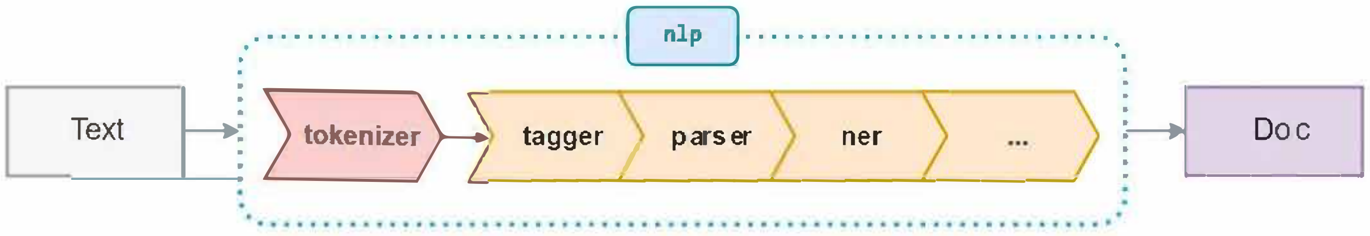
Import spacy and create a object as nlp mentioned below.



Spacy

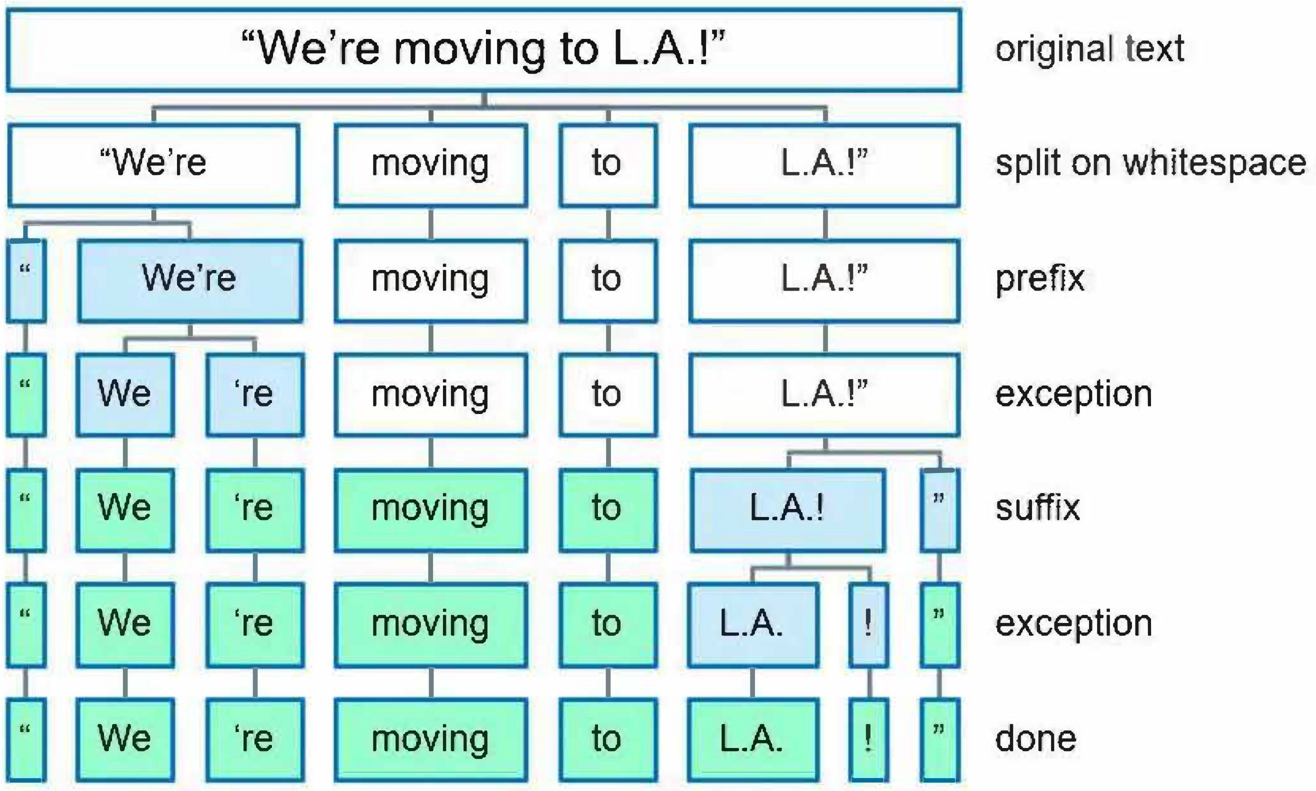
```
In [1]: import spacy
        nlp = spacy.load('en_core_web_sm')
```

When we run nlp, our text enters a processing pipeline that first breaks down the text and then performs a series of operations to tag, parse and describe the data.

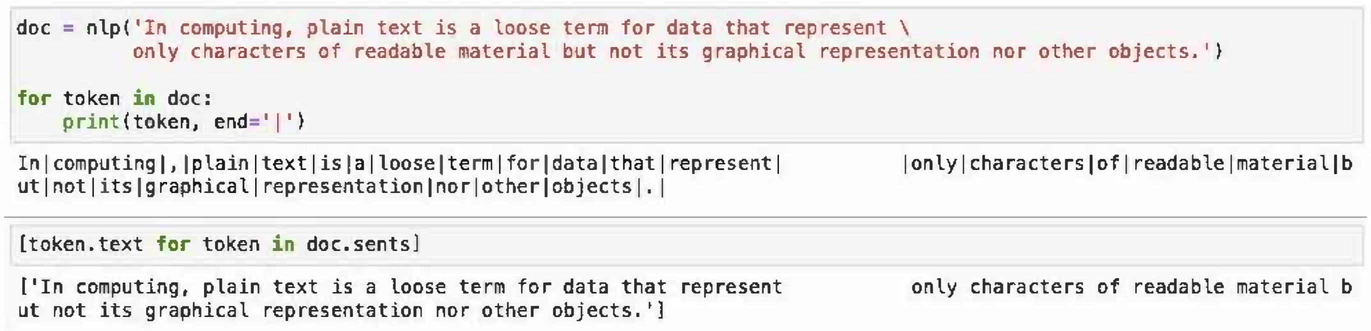


Tokenization:

Tokenization is a common task in Natural Language Processing (NLP). Tokens are the building blocks of Natural Language. Tokenization is a way of separating a piece of text into smaller units called token.



Spacy tokenization steps.



```
doc = nlp('In computing, plain text is a loose term for data that represent \
          only characters of readable material but not its graphical representation nor other objects.')

for token in doc:
    print(token, end='|')
```

```
In|computing|,|plain|text|is|a|loose|term|for|data|that|represent|          |only|characters|of|readable|material|b
ut|not|its|graphical|representation|nor|other|objects|.|
```

```
[token.text for token in doc.sents]
```

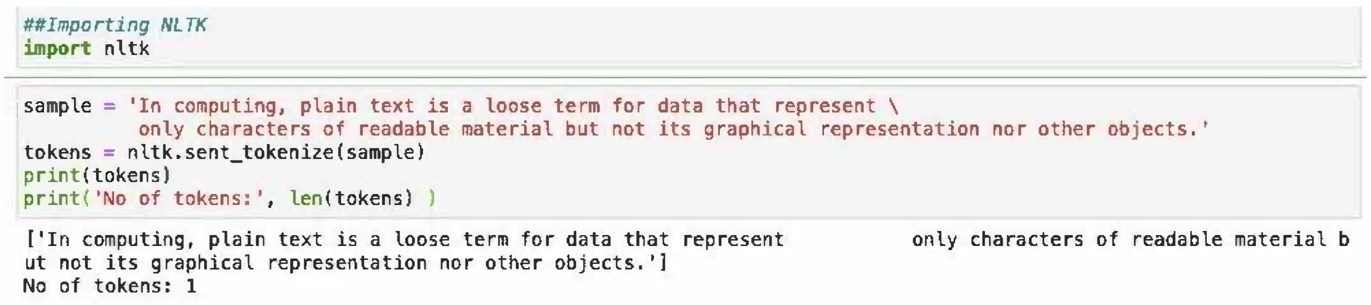
```
['In computing, plain text is a loose term for data that represent           only characters of readable material b
ut not its graphical representation nor other objects.']
```

Spacy Tokenization

Spacy is intelligent enough, when we pass the text inside the nlp object, the text is processed in the pipeline as mentioned above. First for loop indicates text tokenization and the list comprehension explains about the sentence tokenization.

NLTK has a various tokenization algorithm.

Sent tokenizer returns the sentence tokens. Look at the length of token, it’s one which shows the there is only one sentence tokens.



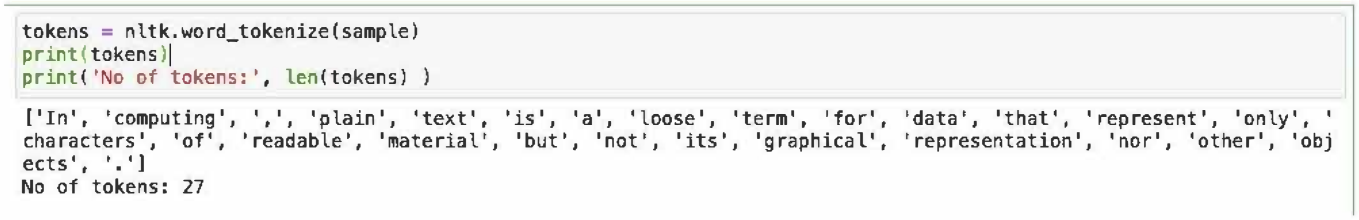
```
##Importing NLTK
import nltk
```

```
sample = 'In computing, plain text is a loose term for data that represent \
          only characters of readable material but not its graphical representation nor other objects.'
tokens = nltk.sent_tokenize(sample)
print(tokens)
print('No of tokens:', len(tokens) )
```

```
['In computing, plain text is a loose term for data that represent           only characters of readable material b
ut not its graphical representation nor other objects.']
No of tokens: 1
```

Sent Tokenizer

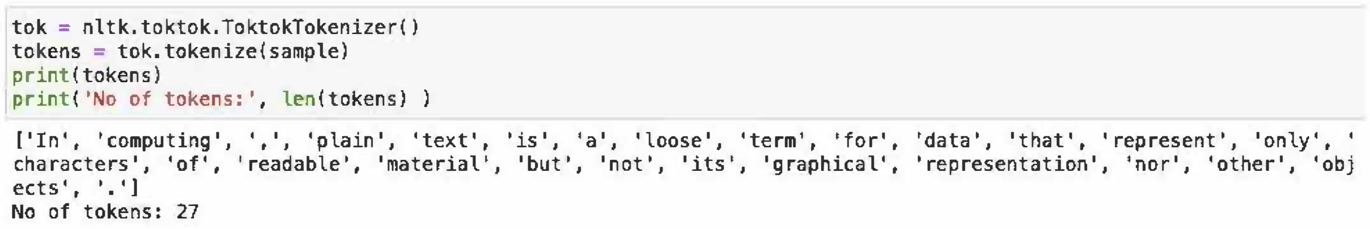
Word tokenizer returns the word tokens. Number of tokens are 27 which shows there is 27 word tokens in the sentence.



```
tokens = nltk.word_tokenize(sample)
print(tokens)
print('No of tokens:', len(tokens) )
```

```
['In', 'computing', ',', 'plain', 'text', 'is', 'a', 'loose', 'term', 'for', 'data', 'that', 'represent', 'only', '
characters', 'of', 'readable', 'material', 'but', 'not', 'its', 'graphical', 'representation', 'nor', 'other', 'obj
ects', '.']
No of tokens: 27
```

Word Tokenizer



```
tok = nltk.toktok.ToktokTokenizer()
tokens = tok.tokenize(sample)
print(tokens)
print('No of tokens:', len(tokens) )
```

```
['In', 'computing', ',', 'plain', 'text', 'is', 'a', 'loose', 'term', 'for', 'data', 'that', 'represent', 'only', '
characters', 'of', 'readable', 'material', 'but', 'not', 'its', 'graphical', 'representation', 'nor', 'other', 'obj
ects', '.']
No of tokens: 27
```

Tok Tok Tokenizer

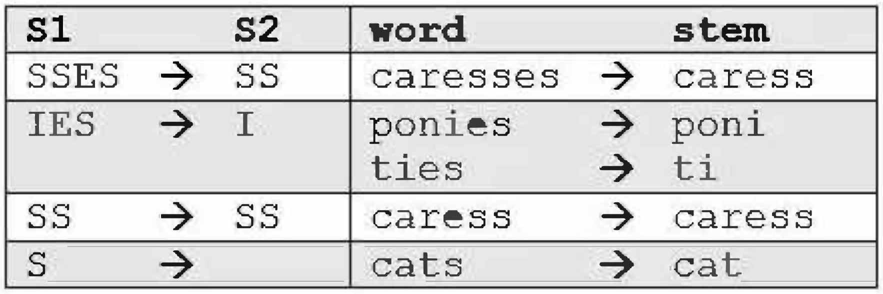
Stemming:

Stemming is a somewhat crude method for cataloging related words; it essentially chops off letters from the end until the stem is reached. This works fairly well in most cases, but unfortunately English has many exceptions where a more sophisticated process is required. In fact, spaCy doesn't include a stemmer, opting instead to rely entirely on lemmatization. So stemming method available only in the NLTK library

Porter Stemmer:

One of the most common and effective stemming tools is Porter's Algorithm developed by Martin Porter in 1980. The algorithm employs five phases of word reduction, each with its own set of mapping rules.

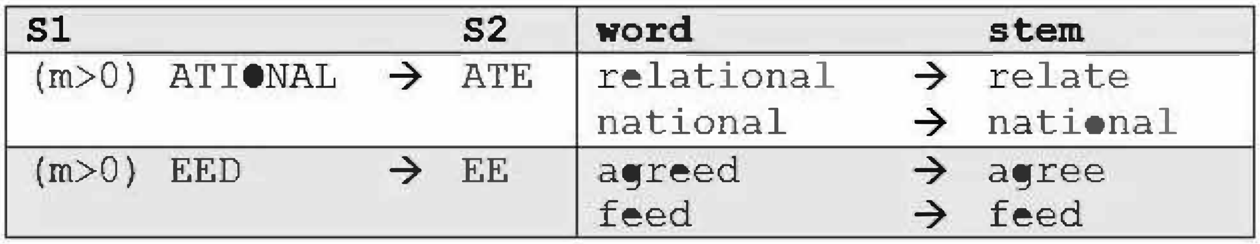
In the first phase, simple suffix matching rules are defined, such as:



S1		S2	word		stem
SSES	→	SS	caresses	→	caress
IES	→	I	ponies ties	→ →	poni ti
SS	→	SS	caress	→	caress
S	→		cats	→	cat

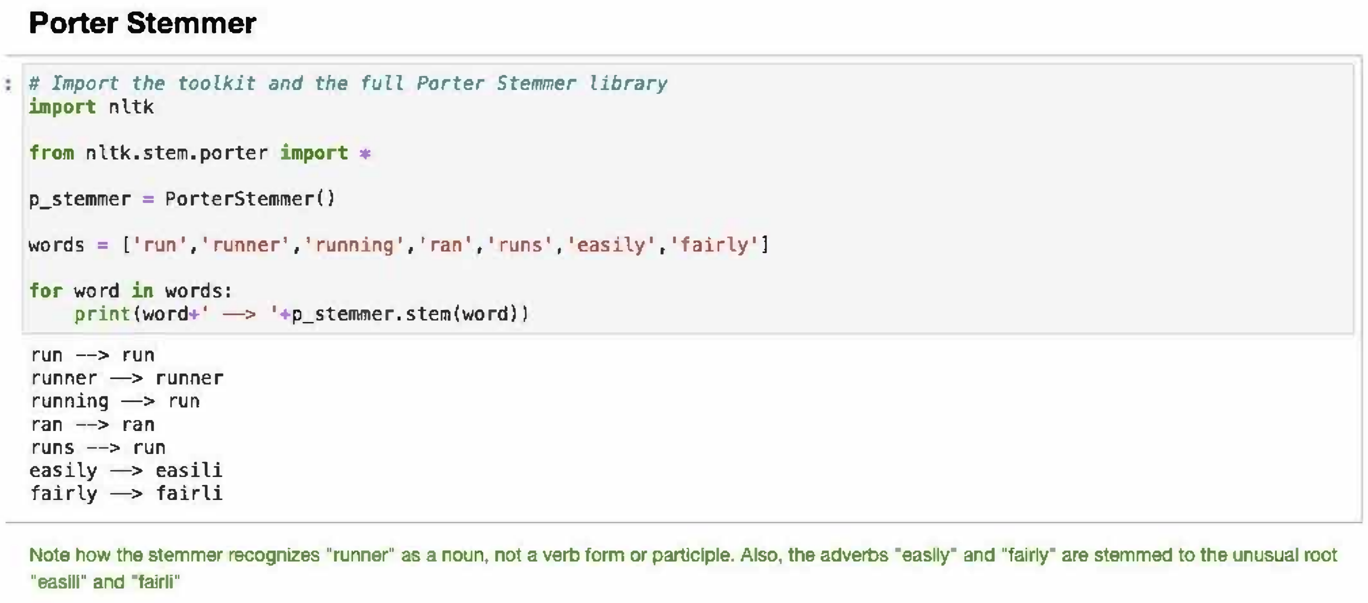
From a given set of stemming rules only one rule is applied, based on the longest suffix S1. Thus, 'caresses' reduces to 'caress' but not 'cares'.

More sophisticated phases consider the length/complexity of the word before applying a rule. For example:



S1			S2	word		stem
(m>0)	ATIONAL	→	ATE	relational national	→ →	relate national
(m>0)	EED	→	EE	agreed feed	→ →	agree feed

Here m>0 describes the "measure" of the stem, such that the rule is applied to all but the most basic stems.


Porter Stemmer

```
# Import the toolkit and the full Porter Stemmer library
import nltk

from nltk.stem.porter import *

p_stemmer = PorterStemmer()

words = ['run','runner','running','ran','runs','easily','fairly']

for word in words:
    print(word+' --> '+p_stemmer.stem(word))
```

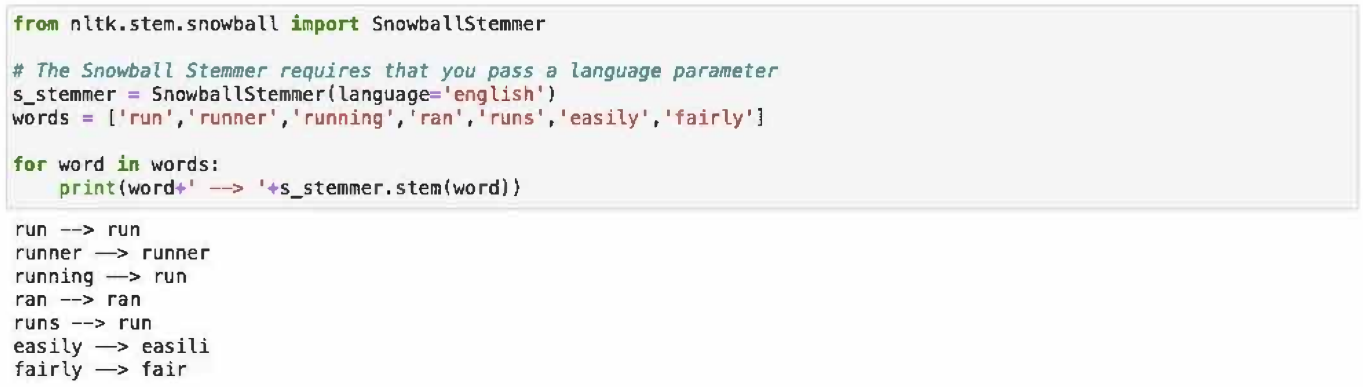
```
run --> run
runner --> runner
running --> run
ran --> ran
runs --> run
easily --> easili
fairly --> fairli
```

Note how the stemmer recognizes "runner" as a noun, not a verb form or participle. Also, the adverbs "easily" and "fairly" are stemmed to the unusual root "easili" and "fairli"

Porter Stemmer

Snowball Stemmer:

This is somewhat of a misnomer, as Snowball is the name of a stemming language developed by Martin Porter. The algorithm used here is more accurately called the "English Stemmer" or "Porter2 Stemmer". It offers a slight improvement over the original Porter stemmer, both in logic and speed. Since nltk uses the name SnowballStemmer, we'll use it here.



```
from nltk.stem.snowball import SnowballStemmer

# The Snowball Stemmer requires that you pass a language parameter
s_stemmer = SnowballStemmer(language='english')
words = ['run','runner','running','ran','runs','easily','fairly']

for word in words:
    print(word+' --> '+s_stemmer.stem(word))
```

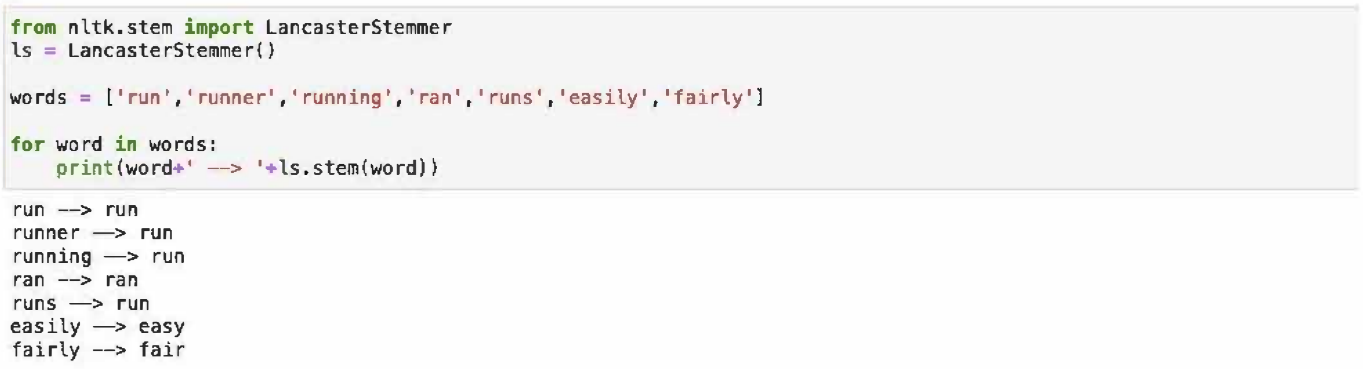
```
run --> run
runner --> runner
running --> run
ran --> ran
runs --> run
easily --> easili
fairly --> fair
```

Snowball Stemmer

Lancaster Stemmer:

The Lancaster stemming algorithm is another algorithm that you can use.

This one is the most aggressive stemming algorithm of the bunch. However, if you use the stemmer in NLTK, you can add your own custom rules to this algorithm very easily



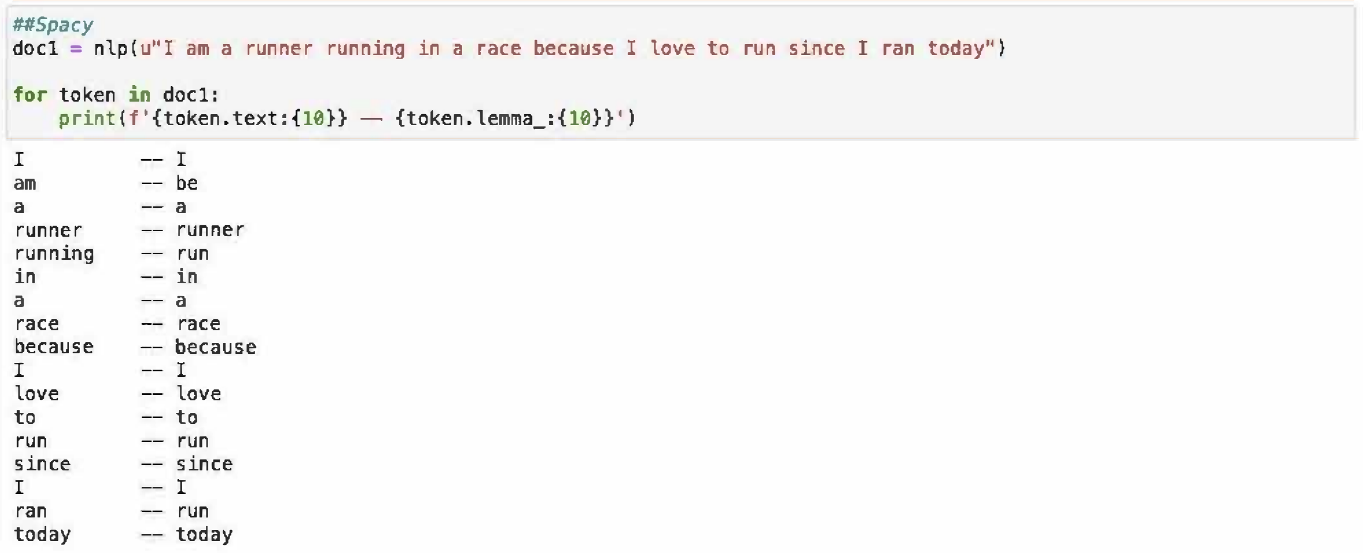
```
from nltk.stem import LancasterStemmer
ls = LancasterStemmer()

words = ['run','runner','running','ran','runs','easily','fairly']

for word in words:
    print(word+' --> '+ls.stem(word))
```

```
run --> run
runner --> run
running --> run
ran --> ran
runs --> run
easily --> easy
fairly --> fair
```

Lancaster Stemmer

Lemmatization:

In contrast to stemming, lemmatization looks beyond word reduction, and considers a language's full vocabulary to apply a morphological analysis to words. The lemma of 'was' is 'be' and the lemma of 'mice' is 'mouse'. Further, the lemma of 'meeting' might be 'meet' or 'meeting' depending on its use in a sentence.

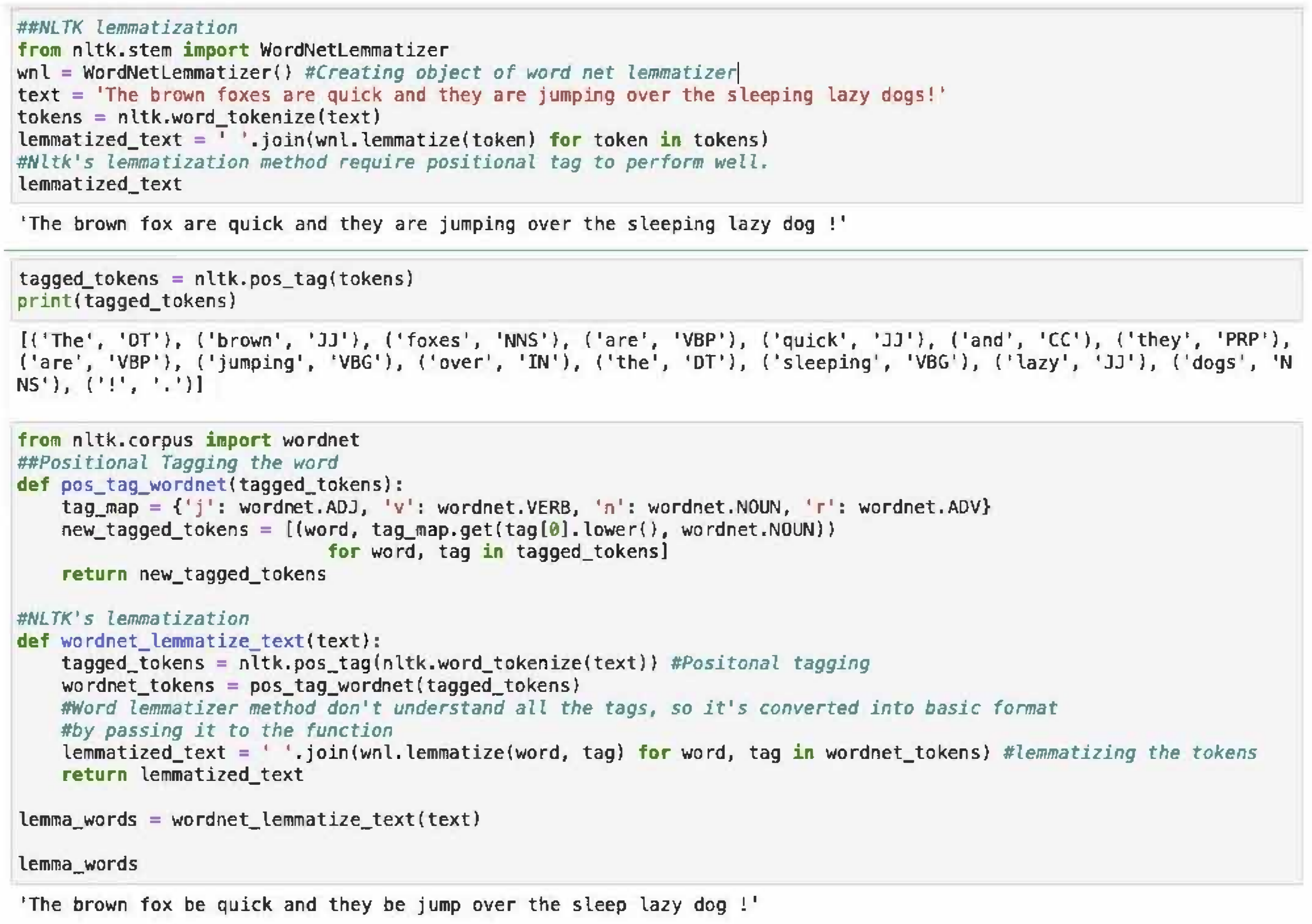


```
##Spacy
doc1 = nlp(u"I am a runner running in a race because I love to run since I ran today")

for token in doc1:
    print(f'{token.text:{10}} -- {token.lemma_:{10}}')
```

```
I          -- I
am         -- be
a          -- a
runner     -- runner
running    -- run
in         -- in
a          -- a
race       -- race
because    -- because
I          -- I
love       -- love
to         -- to
run        -- run
since      -- since
I          -- I
ran        -- run
today      -- today
```

spacy lemmatization

NLTK's lemmatizer requires a positional argument, if don't give the positional tag, it'll take the word as a noun and not performs the lemmatization. You can see below in the snippet, after giving the proper positional tag the lemmatization performs better.



```
##NLTK lemmatization
from nltk.stem import WordNetLemmatizer
wnl = WordNetLemmatizer() #Creating object of word net lemmatizer
text = 'The brown foxes are quick and they are jumping over the sleeping lazy dogs!'
tokens = nltk.word_tokenize(text)
lemmatized_text = ' '.join(wnl.lemmatize(token) for token in tokens)
#Nltk's lemmatization method require positional tag to perform well.
lemmatized_text
```

```
'The brown fox are quick and they are jumping over the sleeping lazy dog !'
```

```
tagged_tokens = nltk.pos_tag(tokens)
print(tagged_tokens)
```

```
[('The', 'DT'), ('brown', 'JJ'), ('foxes', 'NNS'), ('are', 'VBP'), ('quick', 'JJ'), ('and', 'CC'), ('they', 'PRP'),
('are', 'VBP'), ('jumping', 'VBG'), ('over', 'IN'), ('the', 'DT'), ('sleeping', 'VBG'), ('lazy', 'JJ'), ('dogs', 'N
NS'), ('!', '.')]
```

```
from nltk.corpus import wordnet
##Positional Tagging the word
def pos_tag_wordnet(tagged_tokens):
    tag_map = {'j': wordnet.ADJ, 'v': wordnet.VERB, 'n': wordnet.NOUN, 'r': wordnet.ADV}
    new_tagged_tokens = [(word, tag_map.get(tag[0].lower(), wordnet.NOUN))
                            for word, tag in tagged_tokens]
    return new_tagged_tokens

#NLTK's lemmatization
def wordnet_lemmatize_text(text):
    tagged_tokens = nltk.pos_tag(nltk.word_tokenize(text)) #Positonal tagging
    wordnet_tokens = pos_tag_wordnet(tagged_tokens)
    #Word lemmatizer method don't understand all the tags, so it's converted into basic format
    #by passing it to the function
    lemmatized_text = ' '.join(wnl.lemmatize(word, tag) for word, tag in wordnet_tokens) #lemmatizing the tokens
    return lemmatized_text

lemma_words = wordnet_lemmatize_text(text)

lemma_words
```

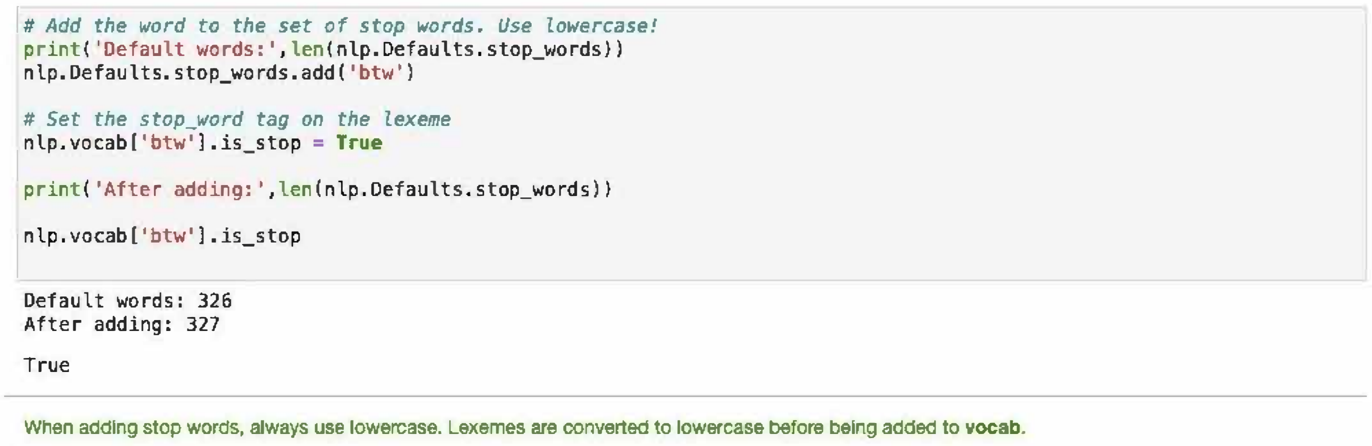
```
'The brown fox be quick and they be jump over the sleep lazy dog !'
```

NLTK Lemmatizer

Stop words:

Words like "a" and "the" appear so frequently that they don't require tagging as thoroughly as nouns, verbs and modifiers. We call these stop words, and they can be filtered from the text to be processed. spaCy holds a built-in list of some 326 English stop words.

To load and add a stop word in spaCY

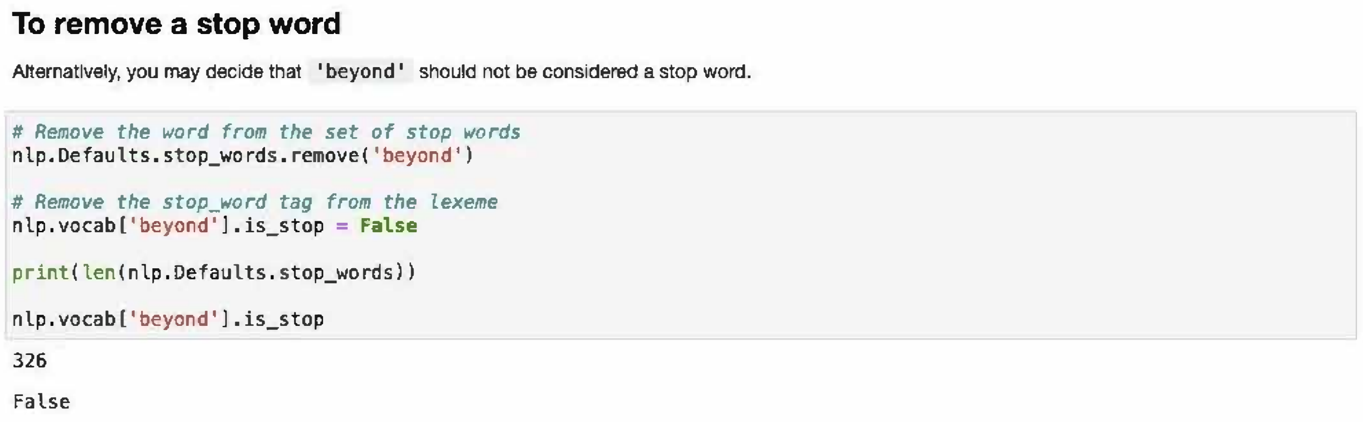


```
# Add the word to the set of stop words. Use lowercase!
print('Default words:',len(nlp.Defaults.stop_words))
nlp.Defaults.stop_words.add('btw')

# Set the stop_word tag on the lexeme
nlp.vocab['btw'].is_stop = True

print('After adding:',len(nlp.Defaults.stop_words))

nlp.vocab['btw'].is_stop
```

```
Default words: 326
After adding: 327

True
```

When adding stop words, always use lowercase. Lexemes are converted to lowercase before being added to **vocab**.

Add Stop Words in spacy



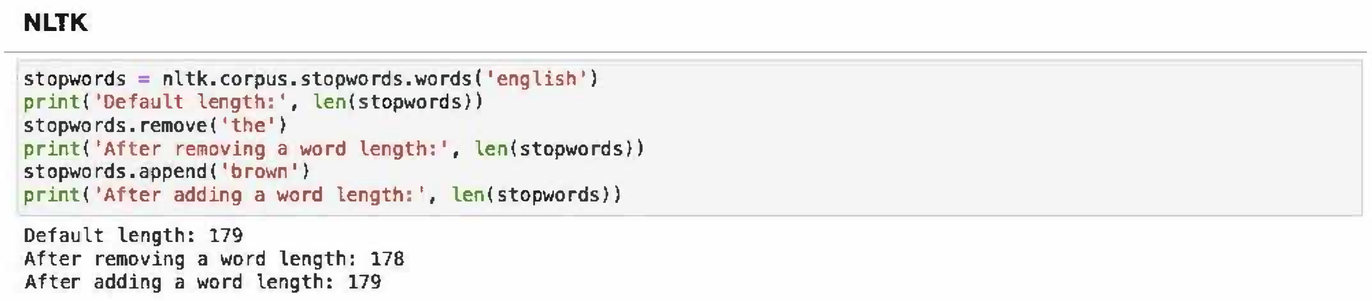
To remove a stop word

Alternatively, you may decide that `'beyond'` should not be considered a stop word.

```
# Remove the word from the set of stop words
nlp.Defaults.stop_words.remove('beyond')

# Remove the stop_word tag from the lexeme
nlp.vocab['beyond'].is_stop = False

print(len(nlp.Defaults.stop_words))

nlp.vocab['beyond'].is_stop
```

```
326

False
```

Remove a stop word in spacy



NLTK

```
stopwords = nltk.corpus.stopwords.words('english')
print('Default length:', len(stopwords))
stopwords.remove('the')
print('After removing a word length:', len(stopwords))
stopwords.append('brown')
print('After adding a word length:', len(stopwords))
```

```
Default length: 179
After removing a word length: 178
After adding a word length: 179
```

NLTK stop words

These are the different ways of basic text processing done with the help of spaCy and NLTK library. Spacy performs in an efficient way for the large task. Hope you got the insight about basic text preprocessing steps followed for NLP task.

Click this github link to refer the notebook file.

Naturallanguageprocessing · Data Science · Artificial Intelligence · Machine Learning · Deep Learning

Written by Rishi Kumar

Follow

45 Followers · Writer for Nerd For Tech

I'm a passionate and disciplined Data Science enthusiast working with Logitech as Data Scientist

More from Rishi Kumar and Nerd For Tech