How to Get Started with NLP - 6 Unique Methods to Perform Tokenization

INTERMEDIATE NLP PYTHON TECHNIQUE TEXT UNSTRUCTURED DATA

Overview

- Looking to get started with Natural Language Processing (NLP)? Here's the perfect first step
- Learn how to perform tokenization – a key aspect to preparing your data for building NLP models
- We present 6 different ways to perform tokenization on text data

Introduction

Are you fascinated by the amount of text data available on the internet? Are you looking for ways to work with this text data but aren't sure where to begin? Machines, after all, recognize numbers, not the letters of our language. And that can be a tricky landscape to navigate in machine learning.

So how can we manipulate and clean this text data to build a model? The answer lies in the wonderful world of Natural Language Processing (NLP).

Solving an NLP problem is a multi-stage process. We need to clean the unstructured text data first before we can even think about getting to the modeling stage. Cleaning the data consists of a few key steps:

- Word tokenization
- Predicting parts of speech for each token
- Text lemmatization
- Identifying and removing stop words, and much more.

In this article, we will talk about the very first step – tokenization. We will first see what tokenization is and why it's required in NLP. We will then look at six unique ways to perform tokenization in Python.

This article has no prerequisites. Anyone with an interest in NLP or data science will be able to follow along. If you're looking for an end-to-end resource for learning NLP, you should check out our comprehensive course:

- Natural Language Processing using Python

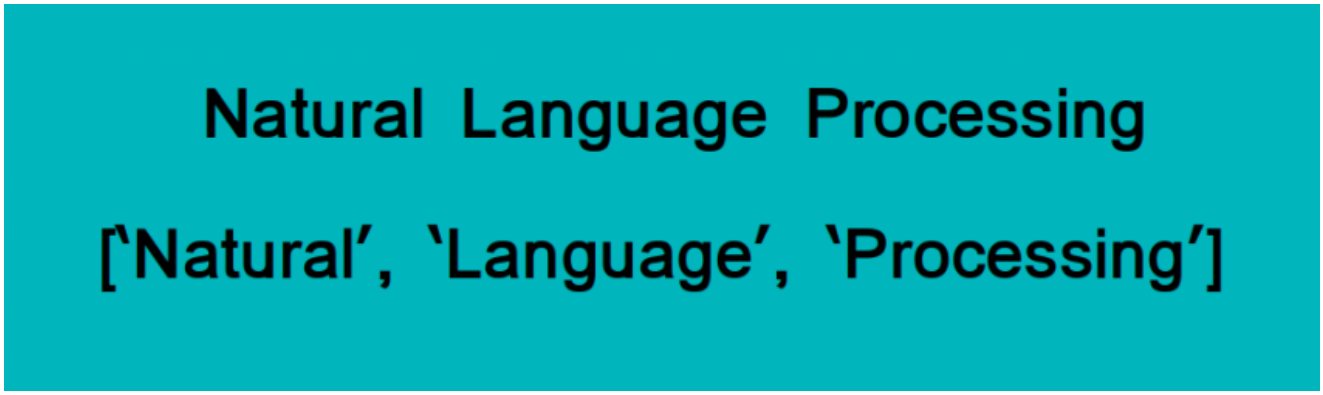
Table of Contents

What is Tokenization in NLP?

Tokenization is one of the most common tasks when it comes to working with text data. But what does the term 'tokenization' actually mean?

> Tokenization is essentially splitting a phrase, sentence, paragraph, or an entire text document into smaller units, such as individual words or terms. Each of these smaller units are called tokens.

Check out the below image to visualize this definition:



The tokens could be words, numbers or punctuation marks. In tokenization, smaller units are created by locating word boundaries. Wait – what are word boundaries?

These are the ending point of a word and the beginning of the next word. These tokens are considered as a first step for stemming and lemmatization (the next stage in text preprocessing which we will cover in the next article).

Why is Tokenization required in NLP?

I want you to think about the English language here. Pick up any sentence you can think of and hold that in your mind as you read this section. This will help you understand the importance of tokenization in a much easier manner.

Before processing a natural language, we need to identify the *words* that constitute a string of characters. That's why tokenization is the most basic step to proceed with NLP (text data). **This is important because the meaning of the text could easily be interpreted by analyzing the words present in the text.**

Let's take an example. Consider the below string:

"This is a cat."

What do you think will happen after we perform tokenization on this string? We get ['This', 'is', 'a', cat'].

There are numerous uses of doing this. We can use this tokenized form to:

- Count the number of words in the text
- Count the frequency of the word, that is, the number of times a particular word is present

And so on. We can extract a lot more information which we'll discuss in detail in future articles. For now, it's time to dive into the meat of this article – the different methods of performing tokenization in NLP.

Methods to Perform Tokenization in Python

We are going to look at six unique ways we can perform tokenization on text data. I have provided the Python code for each method so you can follow along on your own machine.

1. Tokenization using Python's split() function

Let's start with the **split()** method as it is the most basic one. It returns a list of strings after breaking the given string by the specified separator. By default, split() breaks a string at each space. We can change the separator to anything. Let's check it out.

Word Tokenization

```
text = """Founded in 2002, SpaceX's mission is to enable humans to become a spacefaring civilization and a multi-planet
species by building a self-sustaining city on Mars. In 2008, SpaceX's Falcon 1 became the first privately developed
liquid-fuel launch vehicle to orbit the Earth."""
# Splits at space
text.split()
```

view raw

split1.py hosted with ❤ by GitHub

```
Output : ['Founded', 'in', '2002,', 'SpaceX's', 'mission', 'is', 'to', 'enable', 'humans', 'to', 'become', 'a', 'spacefaring', 'civilization', 'and', 'a', 'multi-planet', 'species', 'by', 'building', 'a', 'self-sustaining', 'city', 'on', 'Mars.', 'In', '2008,', 'SpaceX's', 'Falcon', '1', 'became', 'the', 'first', 'privately', 'developed', 'liquid-fuel', 'launch', 'vehicle', 'to', 'orbit', 'the', 'Earth.']
```

Sentence Tokenization

This is similar to word tokenization. Here, we study the structure of sentences in the analysis. A sentence usually ends with a full stop (.), so we can use "." as a separator to break the string:

```
text = """Founded in 2002, SpaceX's mission is to enable humans to become a spacefaring civilization and a multi-planet
species by building a self-sustaining city on Mars. In 2008, SpaceX's Falcon 1 became the first privately developed
liquid-fuel launch vehicle to orbit the Earth."""
# Splits at '.'
text.split('. ')
```

view raw

split2.py hosted with ❤ by GitHub

```
Output : ['Founded in 2002, SpaceX's mission is to enable humans to become a spacefaring civilization and a multi-planet \nspecies by building a self-sustaining city on Mars', 'In 2008, SpaceX's Falcon 1 became the first privately developed \nliquid-fuel launch vehicle to orbit the Earth.']
```

One major drawback of using Python's *split()* method is that we can use only one separator at a time. Another thing to note – in word tokenization, *split()* did not consider punctuation as a separate token.

2. Tokenization using Regular Expressions (RegEx)

First, let's understand what a regular expression is. It is basically a special character sequence that helps you match or find other strings or sets of strings using that sequence as a pattern.

We can use the **re** library in Python to work with regular expression. This library comes preinstalled with the Python installation package.

Now, let's perform word tokenization and sentence tokenization keeping RegEx in mind.

Word Tokenization

```
import re
text = """Founded in 2002, SpaceX's mission is to enable humans to become a spacefaring civilization and a multi-planet
species by building a self-sustaining city on Mars. In 2008, SpaceX's Falcon 1 became the first privately developed
liquid-fuel launch vehicle to orbit the Earth."""
tokens = re.findall("[\w']+", text)
tokens
```

view raw

re1.py hosted with ❤ by GitHub

```
Output : ['Founded', 'in', '2002', 'SpaceX', 's', 'mission', 'is', 'to', 'enable', 'humans', 'to', 'become',
'a', 'spacefaring', 'civilization', 'and', 'a', 'multi', 'planet', 'species', 'by', 'building', 'a', 'self',
'sustaining', 'city', 'on', 'Mars', 'In', '2008', 'SpaceX', 's', 'Falcon', '1', 'became', 'the', 'first',
'privately', 'developed', 'liquid', 'fuel', 'launch', 'vehicle', 'to', 'orbit', 'the', 'Earth']
```

The *re.findall()* function finds all the words that match the pattern passed on it and stores it in the list. The "\w" represents "any word character" which usually means alphanumeric (letters, numbers) and underscore (_). '+' means any number of times. **So *[\w']+* signals that the code should find all the alphanumeric characters until any other character is encountered.**

Sentence Tokenization

To perform sentence tokenization, we can use the *re.split()* function. This will split the text into sentences by passing a pattern into it.

```
import re
text = """Founded in 2002, SpaceX's mission is to enable humans to become a spacefaring civilization and a multi-planet
species by building a self-sustaining city on, Mars. In 2008, SpaceX's Falcon 1 became the first privately developed
liquid-fuel launch vehicle to orbit the Earth."""
sentences = re.compile('[.!?] ').split(text)
sentences
```

view raw

re2.py hosted with ❤ by GitHub

```
Output : ['Founded in 2002, SpaceX's mission is to enable humans to become a spacefaring civilization and a
multi-planet \nspecies by building a self-sustaining city on Mars.', 'In 2008, SpaceX's Falcon 1 became the
first privately developed \nliquid-fuel launch vehicle to orbit the Earth.']
```

Here, we have an edge over the *split()* method as we can pass multiple separators at the same time. In the above code, **we used the *re.compile()* function wherein we passed [.?!]. This means that sentences will split as soon as any of these characters are encountered.**

Interested in reading more about RegEx? The below resources will get you started with Regular Expressions in NLP:

- Beginners Tutorial for Regular Expressions in Python
- Extracting information from reports using Regular Expressions Library in Python

3. Tokenization using NLTK

Now, this is a library you will appreciate the more you work with text data. NLTK, short for Natural Language ToolKit, is a library written in Python for symbolic and statistical Natural Language Processing.

You can install NLTK using the below code:

```
pip install --user -U nltk
```

NLTK contains a module called *tokenize()* which further classifies into two sub-categories:

- **Word tokenize:** We use the word_tokenize() method to split a sentence into tokens or words
- **Sentence tokenize:** We use the sent_tokenize() method to split a document or paragraph into sentences

Let's see both of these one-by-one.

Word Tokenization

```
from nltk.tokenize import word_tokenize
text = """Founded in 2002, SpaceX's mission is to enable humans to become a spacefaring civilization and a multi-planet
species by building a self-sustaining city on Mars. In 2008, SpaceX's Falcon 1 became the first privately developed
liquid-fuel launch vehicle to orbit the Earth."""
word_tokenize(text)
```

view raw

word_tokenize.py hosted with ❤ by GitHub

```
Output: ['Founded', 'in', '2002', ',', 'SpaceX', '’', 's', 'mission', 'is', 'to', 'enable', 'humans', 'to', 'become', 'a', 'spacefaring', 'civilization', 'and', 'a', 'multi-planet', 'species', 'by', 'building', 'a', 'self-sustaining', 'city', 'on', 'Mars', '.', 'In', '2008', ',', 'SpaceX', '’', 's', 'Falcon', '1', 'became', 'the', 'first', 'privately', 'developed', 'liquid-fuel', 'launch', 'vehicle', 'to', 'orbit', 'the', 'Earth', '.']
```

Notice how NLTK is considering punctuation as a token? Hence for future tasks, we need to remove the punctuations from the initial list.

Sentence Tokenization

```
from nltk.tokenize import sent_tokenize
text = """Founded in 2002, SpaceX’s mission is to enable humans to become a spacefaring civilization and a multi-planet
species by building a self-sustaining city on Mars. In 2008, SpaceX’s Falcon 1 became the first privately developed
liquid-fuel launch vehicle to orbit the Earth."""
sent_tokenize(text)
```

view raw

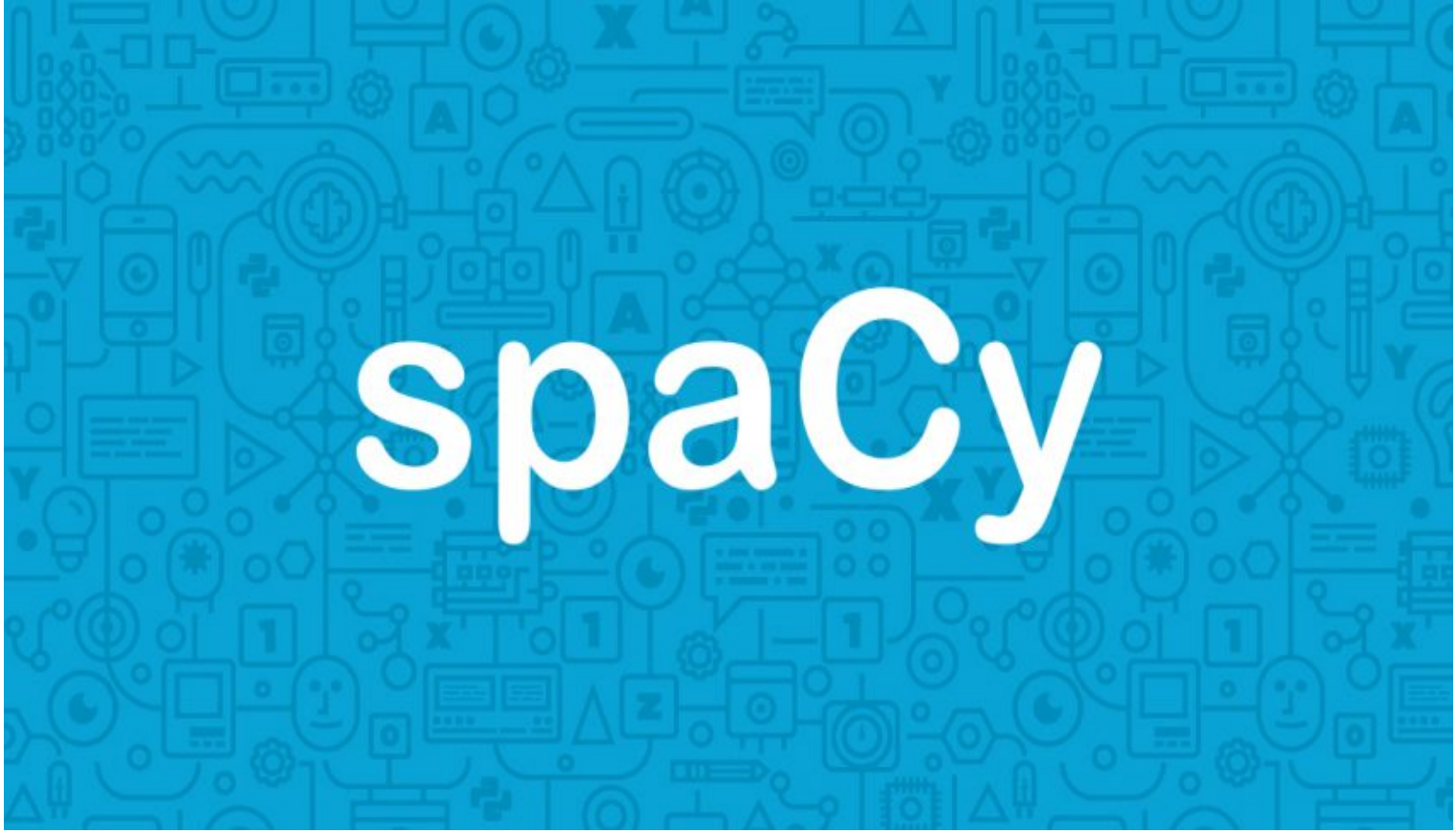
sent_tokenize.py hosted with ❤ by **GitHub**

```
Output: ['Founded in 2002, SpaceX’s mission is to enable humans to become a spacefaring civilization and a multi-planet \nspecies by building a self-sustaining city on Mars.', 'In 2008, SpaceX’s Falcon 1 became the first privately developed \nliquid-fuel launch vehicle to orbit the Earth.']
```

4. Tokenization using the spaCy library

I love the spaCy library. I can’t remember the last time I didn’t use it when I was working on an NLP project. It is just that useful.

spaCy is an **open-source library** for advanced **Natural Language Processing** (NLP). It supports over 49+ languages and provides state-of-the-art computation speed.



To install Spacy in Linux:

```
pip install -U spacy python -m spacy download en
```

To install it on other operating systems, go through this link.

So, let's see how we can utilize the awesomeness of spaCy to perform tokenization. We will use *spacy.lang.en* which supports the English language.

Word Tokenization

```
from spacy.lang.en import English

# Load English tokenizer, tagger, parser, NER and word vectors
nlp = English()

text = """Founded in 2002, SpaceX's mission is to enable humans to become a spacefaring civilization and a multi-planet
species by building a self-sustaining city on Mars. In 2008, SpaceX's Falcon 1 became the first privately developed
liquid-fuel launch vehicle to orbit the Earth."""

#  "nlp" Object is used to create documents with linguistic annotations.
my_doc = nlp(text)

# Create list of word tokens
token_list = []
for token in my_doc:
    token_list.append(token.text)
token_list
```

view raw

spacy1.py hosted with ❤ by **GitHub**

```
Output : ['Founded', 'in', '2002', ',', 'SpaceX', ''s', 'mission', 'is', 'to', 'enable', 'humans', 'to',
'become', 'a', 'spacefaring', 'civilization', 'and', 'a', 'multi', '-', 'planet', '\n', 'species', 'by',
'building', 'a', 'self', '-', 'sustaining', 'city', 'on', 'Mars', '.', 'In', '2008', ',', 'SpaceX', ''s',
'Falcon', '1', 'became', 'the', 'first', 'privately', 'developed', '\n', 'liquid', '-', 'fuel', 'launch',
'vehicle', 'to', 'orbit', 'the', 'Earth', '.']
```

Sentence Tokenization

```
from spacy.lang.en import English

# Load English tokenizer, tagger, parser, NER and word vectors
nlp = English()

# Create the pipeline 'sentencizer' component
sbd = nlp.create_pipe('sentencizer')

# Add the component to the pipeline
nlp.add_pipe(sbd)

text = """Founded in 2002, SpaceX’s mission is to enable humans to become a spacefaring civilization and a multi-planet
species by building a self-sustaining city on Mars. In 2008, SpaceX’s Falcon 1 became the first privately developed
liquid-fuel launch vehicle to orbit the Earth."""

#  "nlp" Object is used to create documents with linguistic annotations.
doc = nlp(text)

# create list of sentence tokens
sents_list = []
for sent in doc.sents:
    sents_list.append(sent.text)
sents_list
```

view raw

spacy2.py hosted with ❤ by GitHub

```
Output : ['Founded in 2002, SpaceX’s mission is to enable humans to become a spacefaring civilization and a multi-planet \nspecies by building a self-sustaining city on Mars.', 'In 2008, SpaceX’s Falcon 1 became the first privately developed \nliquid-fuel launch vehicle to orbit the Earth.']
```

spaCy is quite fast as compared to other libraries while performing NLP tasks (yes, even NLTK). I encourage you to listen to the below DataHack Radio podcast to know the story behind how spaCy was created and where you can use it:

- DataHack Radio #23: Ines Montani and Matthew Honnibal – The Brains behind spaCy

And here’s an in-depth tutorial to get you started with spaCy:

- Natural Language Processing Made Easy – using SpaCy (in Python)

5. Tokenization using Keras

Keras! One of the hottest deep learning frameworks in the industry right now. It is an open-source neural network library for Python. Keras is super easy to use and can also run on top of TensorFlow.

In the NLP context, we can use Keras for cleaning the unstructured text data that we typically collect.

You can install Keras on your machine using just one line of code:

```
pip install Keras
```

Let's get cracking. To perform word tokenization using Keras, we use the *text_to_word_sequence* method from the *keras.preprocessing.text* class.

Let's see Keras in action.

Word Tokenization

```
from keras.preprocessing.text import text_to_word_sequence
# define
text = """Founded in 2002, SpaceX's mission is to enable humans to become a spacefaring civilization and a multi-planet
species by building a self-sustaining city on Mars. In 2008, SpaceX's Falcon 1 became the first privately developed
liquid-fuel launch vehicle to orbit the Earth."""
# tokenize
result = text_to_word_sequence(text)
result
```

view raw

keras1.py hosted with ❤ by GitHub

```
Output : ['founded', 'in', '2002', 'spacex's', 'mission', 'is', 'to', 'enable', 'humans', 'to', 'become', 'a', 'spacefaring', 'civilization', 'and', 'a', 'multi', 'planet', 'species', 'by', 'building', 'a', 'self', 'sustaining', 'city', 'on', 'mars', 'in', '2008', 'spacex's', 'falcon', '1', 'became', 'the', 'first', 'privately', 'developed', 'liquid', 'fuel', 'launch', 'vehicle', 'to', 'orbit', 'the', 'earth']
```

Keras lowers the case of all the alphabets before tokenizing them. That saves us quite a lot of time as you can imagine!

6. Tokenization using Gensim

The final tokenization method we will cover here is using the Gensim library. **It is an open-source library for unsupervised topic modeling and natural language processing** and is designed to automatically extract semantic topics from a given document.

Here's how you can install Gensim:

```
pip install gensim
```

We can use the *gensim.utils* class to import the *tokenize* method for performing word tokenization.

Word Tokenization

```
from gensim.utils import tokenize
text = """Founded in 2002, SpaceX’s mission is to enable humans to become a spacefaring civilization and a multi-planet
species by building a self-sustaining city on Mars. In 2008, SpaceX’s Falcon 1 became the first privately developed
liquid-fuel launch vehicle to orbit the Earth."""
list(tokenize(text))
```

view raw

gensim1.py hosted with ❤ by GitHub

```
Outpur : ['Founded', 'in', 'SpaceX', 's', 'mission', 'is', 'to', 'enable', 'humans', 'to', 'become', 'a', 'spacefaring', 'civilization', 'and', 'a', 'multi', 'planet', 'species', 'by', 'building', 'a', 'self', 'sustaining', 'city', 'on', 'Mars', 'In', 'SpaceX', 's', 'Falcon', 'became', 'the', 'first', 'privately', 'developed', 'liquid', 'fuel', 'launch', 'vehicle', 'to', 'orbit', 'the', 'Earth']
```

Sentence Tokenization

To perform sentence tokenization, we use the *split_sentences* method from the *gensim.summerization.texttcleaner* class:

```
from gensim.summarization.textcleaner import split_sentences
text = """Founded in 2002, SpaceX’s mission is to enable humans to become a spacefaring civilization and a multi-planet
species by building a self-sustaining city on Mars. In 2008, SpaceX’s Falcon 1 became the first privately developed
liquid-fuel launch vehicle to orbit the Earth."""
result = split_sentences(text)
result
```

view raw

gensim2.py hosted with ❤ by GitHub

```
Output : ['Founded in 2002, SpaceX’s mission is to enable humans to become a spacefaring civilization and a multi-planet ', 'species by building a self-sustaining city on Mars.', 'In 2008, SpaceX’s Falcon 1 became the first privately developed ', 'liquid-fuel launch vehicle to orbit the Earth.']
```

You might have noticed that Gensim is quite strict with punctuation. It splits whenever a punctuation is encountered. In sentence splitting as well, Gensim tokenized the text on encountering “\n” while other libraries ignored it.

End Notes

Tokenization is a critical step in the overall NLP pipeline. We cannot simply jump into the model building part without cleaning the text first.

In this article, we saw six different methods of tokenization (word as well as a sentence) from a given text. There are other ways as well but these are good enough to get you started on the topic.

I’ll be covering other text cleaning steps like removing stopwords, part-of-speech tagging, and recognizing named entities in my future posts. Till then, keep learning!

Article Url - https://www.analyticsvidhya.com/blog/2019/07/how-get-started-nlp-6-unique-ways-perform-tokenization/

Shubham Singh

A Data Science Enthusiast who loves reading & writing about Data Science and its applications. He has done many projects in this field and his recent work include concepts like Web Scraping, NLP etc. He is a Data Science Content Strategist Intern at Analytics Vidhya. And currently pursuing BTech in Computer Science from DIT University, Dehradun.