NLP Tutorial Using Python NLTK (Simple Examples)

In this post, we will talk about natural language processing (NLP) using Python. This NLP tutorial will use Python NLTK library. NLTK is a popular Python library which is used for NLP.

So what is NLP? and what are the benefits of learning NLP?

# What is NLP?

Simply and in short, natural language processing (NLP) is about developing applications and services that are able to understand human languages.

We are talking here about practical examples of natural language processing (NLP) like speech recognition, speech translation, understanding complete sentences, understanding synonyms of matching words, and writing complete grammatically correct sentences and paragraphs.

This is not everything, you can think about the industrial implementations about these ideas and its benefits.

## Benefits of NLP

As all of you know, there are millions of gigabytes every day are generated by blogs, social websites, and web pages.

There are many companies gathering all of these data for understanding users and their passions and give these reports to the companies to adjust their plans.

These data could show that the people of Brazil are happy with product A which could be a movie or anything while the people of the US are happy of product B. And this could be instant (real-time result). Like what search engines do, they give the appropriate results to the right people at the right time.

You know what, search engines are not the only implementation of natural language processing (NLP) and there are a lot of awesome implementations out there.

## NLP Implementations

These are some of the successful implementation of Natural Language Processing (NLP):

* **Search engines** like Google, Yahoo, etc. Google search engine understands that you are a tech guy so it shows you results related to you.
* **Social websites feeds** like Facebook news feed. The news feed algorithm understands your interests using natural language processing and shows you related Ads and posts more likely than other posts.
* **Speech engines** like Apple Siri.
* **Spam filters** like Google spam filters. It’s not just about the usual spam filtering, now spam filters understand what’s inside the email content and see if it’s a spam or not.

## NLP Libraries

There are many open source Natural Language Processing (NLP) libraries and these are some of them:

* Natural language toolkit (NLTK).
* Apache OpenNLP.
* Stanford NLP suite.
* Gate NLP library.

Natural language toolkit (NLTK) is the most popular library for natural language processing (NLP) which was written in Python and has a big community behind it.

NLTK also is very easy to learn, actually, it’s the easiest natural language processing (NLP) library that you’ll use.

In this NLP Tutorial, we will use Python NLTK library.

Before I start installing NLTK, I assume that you know some Python basics to get started.

## Install NLTK

If you are using Windows or Linux or Mac, you can install NLTK [using pip](https://likegeeks.com/import-create-install-reload-alias-python-modules/#Install-Python-Modules-Using-pip):

$ pip install nltk

To check if NLTK has installed correctly, you can open python terminal and type the following:

Import nltk

If everything goes fine, that means you’ve successfully installed NLTK library.

Once you’ve installed NLTK, you should install the NLTK packages by running the following code:

|  |
| --- |
| import nltk    nltk.download() |

This will show the NLTK downloader to choose what packages need to be installed.

You can install all packages since they have small sizes, so no problem. Now let’s start the show.

## Tokenize Text Using Pure Python

First, we will grab a web page content then we will analyze the text to see what the page is about.

We will use the [urllib module](https://likegeeks.com/python-programming-basics/#Web-Crawling) to crawl the web page:

|  |
| --- |
| import urllib.request    response = urllib.request.urlopen('http://php.net/')    html = response.read()    print (html) |

As you can see from the printed output, the result contains a lot of HTML tags that need to be cleaned.

We can use BeautifulSoup to clean the grabbed text like this:

|  |
| --- |
| from bs4 import BeautifulSoup    import urllib.request    response = urllib.request.urlopen('http://php.net/')    html = response.read()    soup = BeautifulSoup(html,"html5lib")    text = soup.get\_text(strip=True)    print (text) |

Now we have a clean text from the crawled web page.

Awesome, Right?

Finally, let’s convert that text into tokens by splitting the text like this:

|  |
| --- |
| from bs4 import BeautifulSoup    import urllib.request    response = urllib.request.urlopen('http://php.net/')    html = response.read()    soup = BeautifulSoup(html,"html5lib")    text = soup.get\_text(strip=True)    tokens = [t for t in text.split()]    print (tokens) |

## Count Word Frequency

The text is much better now. Let’s calculate the frequency distribution of those tokens using Python NLTK.

There is a function in NLTK called FreqDist() does the job:

|  |
| --- |
| from bs4 import BeautifulSoup    import urllib.request    import nltk    response = urllib.request.urlopen('http://php.net/')    html = response.read()    soup = BeautifulSoup(html,"html5lib")    text = soup.get\_text(strip=True)    tokens = [t for t in text.split()]    freq = nltk.FreqDist(tokens)    for key,val in freq.items():        print (str(key) + ':' + str(val)) |

If you search the output, you’ll find that the most frequent token is PHP.

You can plot a graph for those tokens using plot function like this:

freq.plot(20, cumulative=False)

From the graph, you can be sure that this article is talking about PHP.

Great!!

There are some words like The, Of, a, an, and so on. These words are stop words.

Generally, stop words should be removed to prevent them from affecting our results.

## Remove Stop Words Using NLTK

NLTK is shipped with stop words lists for most languages. To get English stop words, you can use this code:

|  |
| --- |
| from nltk.corpus import stopwords    stopwords.words('english') |

Now, let’s modify our code and clean the tokens before plotting the graph.

First, we will make a copy of the list, then we will iterate over the tokens and remove the stop words:

|  |
| --- |
| clean\_tokens = tokens[:]    sr = stopwords.words('english')    for token in tokens:        if token in stopwords.words('english'):            clean\_tokens.remove(token) |

You can review the [Python list functions](https://likegeeks.com/python-list-functions/) to know how to process lists

So the final code should be like this:

|  |
| --- |
| from bs4 import BeautifulSoup    import urllib.request    import nltk    from nltk.corpus import stopwords    response = urllib.request.urlopen('http://php.net/')    html = response.read()    soup = BeautifulSoup(html,"html5lib")    text = soup.get\_text(strip=True)    tokens = [t for t in text.split()]    clean\_tokens = tokens[:]    sr = stopwords.words('english')    for token in tokens:        if token in stopwords.words('english'):            clean\_tokens.remove(token)    freq = nltk.FreqDist(clean\_tokens)    for key,val in freq.items():        print (str(key) + ':' + str(val)) |

If you check the graph now, it’s better than before since no stop words on the count.

freq.plot(20,cumulative=False)

## Tokenize Text Using NLTK

We saw how to split the text into tokens using split function, now we will see how to tokenize the text using NLTK.

Tokenizing text is important since text can’t be processed without tokenization. Tokenization process means splitting bigger parts into small parts.

You can tokenize paragraphs to sentences and tokenize sentences to words according to your needs. NLTK is shipped with sentence tokenizer and word tokenizer.

Let’s assume that we have a sample text like the following:

Hello Adam, how are you? I hope everything is going well. Today is a good day, see you dude.

To tokenize this text to sentences, we will use sentence tokenizer:

|  |
| --- |
| from nltk.tokenize import sent\_tokenize    mytext = "Hello Adam, how are you? I hope everything is going well. Today is a good day, see you dude."    print(sent\_tokenize(mytext)) |

The output is the following:

['Hello Adam, how are you?', 'I hope everything is going well.', 'Today is a good day, see you dude.']

You may say that this is an easy job, I don’t need to use NLTK tokenization, and I can split sentences using regular expressions since every sentence precedes by punctuation and space.

Well, take a look at the following text:

Hello Mr. Adam, how are you? I hope everything is going well. Today is a good day, see you dude.

Uh! The word Mr. is one word by itself. OK, let’s try NLTK:

|  |
| --- |
| from nltk.tokenize import sent\_tokenize    mytext = "Hello Mr. Adam, how are you? I hope everything is going well. Today is a good day, see you dude."    print(sent\_tokenize(mytext)) |

The output looks like this:

['Hello Mr. Adam, how are you?', 'I hope everything is going well.', 'Today is a good day, see you dude.']

Great! It works like charm.

OK, let’s try word tokenizer to see how it will work.

|  |
| --- |
| from nltk.tokenize import word\_tokenize    mytext = "Hello Mr. Adam, how are you? I hope everything is going well. Today is a good day, see you dude."    print(word\_tokenize(mytext)) |

The output is:

['Hello', 'Mr.', 'Adam', ',', 'how', 'are', 'you', '?', 'I', 'hope', 'everything', 'is', 'going', 'well', '.', 'Today', 'is', 'a', 'good', 'day', ',', 'see', 'you', 'dude', '.']

The word Mr. is one word as expected.

NLTK uses PunktSentenceTokenizer which is a part of nltk.tokenize.punkt module.

This tokenizer trained well to work with many languages.

# Tokenize non-English Languages Text

To tokenize other languages, you can specify the language like this:

|  |
| --- |
| from nltk.tokenize import sent\_tokenize    mytext = "Bonjour M. Adam, comment allez-vous? J'espère que tout va bien. Aujourd'hui est un bon jour."    print(sent\_tokenize(mytext,"french")) |

The result will be like this:

['Bonjour M. Adam, comment allez-vous?', "J'espère que tout va bien.", "Aujourd'hui est un bon jour."]

We are doing well.

## Get Synonyms from WordNet

If you remember we installed NLTK packages using nltk.download(). One of the packages was WordNet.

WordNet is a database which is built for natural language processing. It includes groups of synonyms and a brief definition.

You can get these definitions and examples for a given word like this:

|  |
| --- |
| from nltk.corpus import wordnet    syn = wordnet.synsets("pain")    print(syn[0].definition())    print(syn[0].examples()) |

The result is:

|  |
| --- |
| a symptom of some physical hurt or disorder    ['the patient developed severe pain and distension'] |

WordNet includes a lot of definitions:

|  |
| --- |
| from nltk.corpus import wordnet    syn = wordnet.synsets("NLP")    print(syn[0].definition())    syn = wordnet.synsets("Python")    print(syn[0].definition()) |

The result is:

|  |
| --- |
| the branch of information science that deals with natural language information    large Old World boas |

You can use WordNet to get synonymous words like this:

|  |
| --- |
| from nltk.corpus import wordnet    synonyms = []    for syn in wordnet.synsets('Computer'):        for lemma in syn.lemmas():            synonyms.append(lemma.name())    print(synonyms) |

The output is:

['computer', 'computing\_machine', 'computing\_device', 'data\_processor', 'electronic\_computer', 'information\_processing\_system', 'calculator', 'reckoner', 'figurer', 'estimator', 'computer']

Cool!!

## Get Antonyms from WordNet

You can get the antonyms words the same way, all you have to do is to check the lemmas before adding them to the array if it’s an antonym or not.

|  |
| --- |
| from nltk.corpus import wordnet    antonyms = []    for syn in wordnet.synsets("small"):        for l in syn.lemmas():            if l.antonyms():                antonyms.append(l.antonyms()[0].name())    print(antonyms) |

The output is:

['large', 'big', 'big']

This is the power of NLTK in natural language processing.

## NLTK Word Stemming

Word stemming means removing affixes from words and return the root word. Ex: The stem of the word working => work.

Search engines use this technique when indexing pages, so many people write different versions for the same word and all of them are stemmed to the root word.

There are many algorithms for stemming, but the most used algorithm is **Porter stemming algorithm**.

NLTK has a class called PorterStemmer which uses Porter stemming algorithm.

|  |
| --- |
| from nltk.stem import PorterStemmer    stemmer = PorterStemmer()    print(stemmer.stem('working')) |

The result is:

work

Clear enough.

There are some other stemming algorithms like **Lancaster stemming algorithm**.

The output of this algorithm shows a **bit different** results for few words. You can try both of them to see the result.

## Stemming non-English Words

SnowballStemmer can stem 13 languages besides the English language.

The supported languages are:

|  |
| --- |
| from nltk.stem import SnowballStemmer    print(SnowballStemmer.languages) |

'danish', 'dutch', 'english', 'finnish', 'french', 'german', 'hungarian', 'italian', 'norwegian', 'porter', 'portuguese', 'romanian', 'russian', 'spanish', 'swedish'

You can use the stem function of the SnowballStemmer class to stem non-English words like this:

|  |
| --- |
| from nltk.stem import SnowballStemmer    french\_stemmer = SnowballStemmer('french')    print(french\_stemmer.stem("French word")) |

The French people can tell us about the results :).

## Lemmatizing Words Using WordNet

Word lemmatizing is similar to stemming, but the difference is the result of lemmatizing is a real word.

Unlike stemming, when you try to stem some words, it will result in something like this:

|  |
| --- |
| from nltk.stem import PorterStemmer    stemmer = PorterStemmer()    print(stemmer.stem('increases')) |

The result is:

increas

Now, if we try to lemmatize the same word using NLTK WordNet, the result is correct:

|  |
| --- |
| from nltk.stem import WordNetLemmatizer    lemmatizer = WordNetLemmatizer()    print(lemmatizer.lemmatize('increases')) |

The result is

increase

The result might end up with a synonym or a different word with the same meaning.

Sometimes, if you try to lemmatize a word like the word playing, it will end up with the same word.

This is because the default part of speech is nouns. To get verbs, you should specify it like this:

|  |
| --- |
| from nltk.stem import WordNetLemmatizer    lemmatizer = WordNetLemmatizer()    print(lemmatizer.lemmatize('playing', pos="v")) |

The result is:

play

Actually, this is a very good level of text compression, you end up with about 50% to 60% compression.

The result could be a verb, noun, adjective, or adverb:

|  |
| --- |
| from nltk.stem import WordNetLemmatizer    lemmatizer = WordNetLemmatizer()    print(lemmatizer.lemmatize('playing', pos="v"))    print(lemmatizer.lemmatize('playing', pos="n"))    print(lemmatizer.lemmatize('playing', pos="a"))    print(lemmatizer.lemmatize('playing', pos="r")) |

The result is:

|  |
| --- |
| play    playing    playing    playing |

## Stemming and Lemmatization Difference

OK, let’s try stemming and lemmatization for some words:

|  |
| --- |
| from nltk.stem import WordNetLemmatizer    from nltk.stem import PorterStemmer    stemmer = PorterStemmer()    lemmatizer = WordNetLemmatizer()    print(stemmer.stem('stones'))    print(stemmer.stem('speaking'))    print(stemmer.stem('bedroom'))    print(stemmer.stem('jokes'))    print(stemmer.stem('lisa'))    print(stemmer.stem('purple'))    print('----------------------')    print(lemmatizer.lemmatize('stones'))    print(lemmatizer.lemmatize('speaking'))    print(lemmatizer.lemmatize('bedroom'))    print(lemmatizer.lemmatize('jokes'))    print(lemmatizer.lemmatize('lisa'))    print(lemmatizer.lemmatize('purple')) |

The result is:

|  |
| --- |
| stone    speak    bedroom    joke    lisa    purpl    ----------------------    stone    speaking    bedroom    joke    lisa    purple |

Stemming works on words without knowing its context and that’s why stemming has lower accuracy and faster than lemmatization.

In my opinion, lemmatizing is better than stemming. Word lemmatizing returns a real word even if it’s not the same word, it could be a synonym, but at least it’s a real word.

Sometimes you don’t care about this level of accuracy and all you need is speed, in this case, stemming is better.

All step we discussed in this NLP tutorial was text preprocessing. In the future exercises we will discuss text analysis using Python NLTK.