**NLTK – Project Examples**

Natural language processing (NLP) is about developing applications and services that are able to understand human languages.

## Benefits of NLP

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

There are many companies gathering all of this data to better understand users and their passions and make appropriate changes.

These data could show that the people of Brazil are happy with product A, while the people of the US are happier with product B. With NLP, this knowledge can be found instantly (i.e. a real-time result). For example, search engines are a type of NLP that give the appropriate results to the right people at the right time.

But search engines are not the only implementation of natural language processing (NLP). There are a lot of even more awesome implementations out there.

## NLP Implementations

These are some successful implementations of natural language processing (NLP):

* **Search engines** like Google, Yahoo, etc. Google's search engine understands that you are a tech guy, so it shows you results related to that.
* **Social website feeds** like your 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 your usual spam filtering; now, spam filters understand what's inside the email content and see if it's spam or not.

## NLP Libraries

There are many open source Natural Language Processing (NLP) libraries. 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). It 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.

## Install NLTK

If you are using Windows or Linux or Mac, you can install NLTK using pip: # pip install nltk.

You can use NLTK on Python 2.7, 3.4, and 3.5 at the time of writing this post. Alternatively, you can install it from source from this tar.

To check if NLTK has installed correctly, you can open your 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()

## Tokenize Text Using Pure Python

First, we will grab some web page content. Then, we will analyze the text to see what the page is about. We will use the urllib module 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)

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() that 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))

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)

The final code should look 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 just saw how to split the text into tokens using the 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 to small parts.

You can tokenize paragraphs to sentences and tokenize sentences to words according to your needs. NLTK is shipped with a sentence tokenizer and a 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, *This is an easy job; I don't need to use NLTK tokenization, and I can split sentences using regular expressions since every sentence is preceded by punctuation and a 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.

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.']

It works like charm. Let's try the 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))

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 the nltk.tokenize.punkt module. This tokenizer is 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."]

## Get Synonyms From WordNet

If you remember we installed NLTK packages using nltk.download(). One of the packages was WordNet. WordNet is a database 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']

## Get Antonyms From WordNet

You can get the antonyms of words the same way. All you have to do is to check the lemmas before adding them to the array.  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)

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 returning the root word. (The stem of the word working is 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 the Porter stemming algorithm. NLTK has a class called PorterStemmer that uses this algorithm.

from nltk.stem import PorterStemmer

stemmer = PorterStemmer()

print(stemmer.stem('working'))

The result is: work.

## 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"))

## 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