

# Comparing NLTK with other NLP tools for text analysis

danfeng3 - fall 2021

## Introduction:

Natural language processing helps us to understand valuable texts in countless ways. It also forms the structure of the text and is useful in many applications, such as speech recognition, content or context analytics on Twitter or Facebook, and so on. The goal of this paper is to compare one of the most popular and powerful natural language processing tools NLTK with other popular ones like Textblob and stanfordNLP. NLTK is a useful Python package that provides the reader with the fundamentals of writing Python programs. It has plenty of tools to clean and pre-process the text data, working with corpora, categorizing text, analyzing linguistic structure, and [more](#).

## Overview of different functionality and usage for information retrieval between NLTK and other NLP tools:

### 1. Install and environment setting.

- a. First of all, Install [NLTK](#) , [Textblob](#) and [Stanford coreNLP](#).
- b. For StanfordNLP, we need to set up the environment:

```
cd ~
wget http://nlp.stanford.edu/software/stanford-corenlp-full-2016-10-31.zip
unzip stanford-corenlp-full-2016-10-31.zip && cd stanford-corenlp-full-2016-10-31
pip3 install -U https://github.com/stanfordnlp/python-stanford-corenlp/archive/master.zip

# On Mac export
CORENLP_HOME=/Users/<username>/stanford-corenlp-full-2016-10-31/
# On linux export
CORENLP_HOME=/home/<username>/stanford-corenlp-full-2016-10-31/
```

### 2. Pre-processing with NLTK and StandfordNLP.

- a. Import libraries

Using NLTK

```

1 import nltk
2 import random
3 from nltk.corpus import state_union
4 from nltk.tokenize import PunktSentenceTokenizer
5 from nltk.corpus import movie_reviews
6 from nltk.corpus import stopwords
7 from textblob.classifiers import NaiveBayesClassifier

```

## Using StanfordNLP

```

1 import corenlp

```

### b. Tokenizing

Using NLTK - word [tokenizers](#) and sentences [tokenizers](#).

```

1 from nltk.tokenize import sent_tokenize, word_tokenize
2 example_text = "Hello Mr. Smith, how are you doing today? The weather is great and Python is awesome. "\
3 "the sky is pinkish-blue. You should not eat carboard."
4
5 print(sent_tokenize(example_text)) #creat a list.
6
7 print(word_tokenize(example_text)) #creat a list.

```

['Hello Mr. Smith, how are you doing today?', 'The weather is great and Python is awesome.', 'the sky is pinkish-blue.', 'You should not eat carboard.']

['Hello', 'Mr.', 'Smith', ',', 'how', 'are', 'you', 'doing', 'today', '?', 'The', 'weather', 'is', 'great', 'and', 'Python', 'is', 'awesome', '.', 'the', 'sky', 'is', 'pinkish-blue', '.', 'You', 'should', 'not', 'eat', 'carboard', '.']

## Using StanfordNLP -

```

1 with corenlp.client.CoreNLClient(annotators="tokenize ssplit".split()) as client:
2     ann = client.annotate("Hello Mr. Smith, how are you doing today? The weather is great and Python is awesome. "\
3 "the sky is pinkish-blue. You should not eat carboard.")
4     sentence = ann.sentence[0]
5     [token.word for token in sentence.token]
6     #['Hello', 'Mr.', 'Smith', ',', 'how', 'are', 'you', 'doing', 'today', '?', 'The', 'weather', 'is', 'great', 'and',
7     #'.', 'You', 'should', 'not', 'eat', 'carboard', '.']

```

### c. Removing stop words and POS Tagging- NLTK

```

1 example_sentence = "This is just a simple example to show off how to filter the stop words."
2 stop_words = set(stopwords.words("english")) # choose english as the language to get all stopwords in it.
3
4 words = word_tokenize(example_sentence)
5 filtered_sentence = []
6
7 # for word in words:
8 #     if word not in stop_words:
9 #         filtered_sentence.append(word)
10 filtered_sentence = [word for word in words if not word in stop_words]
11 print(filtered_sentence)

```

['This', 'simple', 'example', 'show', 'filter', 'stop', 'words', '.']

```

1 #Part of Speech Tagging
2 import nltk
3 from nltk.corpus import state_union
4
5 sample_text = "I like that Movie." #loading some data as input.
6
7 tokenized = custom_sent_tokenizer.tokenize(sample_text)
8 def process_content():
9     try:
10         for i in tokenized:
11             words = nltk.word_tokenize(i)
12             tagged = nltk.pos_tag(words)
13
14             print(tagged)
15     except Exception as e:
16         print(str(e))
17 process_content()

```

```

['I like that Movie.']
[('I', 'PRP'), ('like', 'VBP'), ('that', 'DT'), ('Movie', 'NNP'), ('.', '.')]

```

Upon my research, some of the pre-processing stages like removal of stopwords, punctuation, or special characters are not included in the StanfordNLP.

We can also run the Stanford Pos Tagger on NLTK Using [Java](#) or [Python](#).

- d. Here, let me demonstrate another interesting example that how can we use Stanford Parser to get a dependency path. First of all, import the library :

```

1 import os
2 from nltk.parse import stanford

```

Then path java.exe and .jar files to your Stanford parser:

```

java_path = r'C:\Program Files (x86)\Common Files\Oracle\Java\javapath\java.exe'
os.environ['JAVAHOME'] = java_path
os.environ['STANFORD_PARSER'] =
r'YOUR_PATH\stanford-parser-full-2014-10-31\stanford-parser.jar'
os.environ['STANFORD_MODELS'] =
r'YOUR_PATH\stanford-parser-full-2014-10-31\stanford-parser-3.5.0-models.jar'
dependency_parser =
stanford.StanfordDependencyParser(path_to_jar=r'YOUR_PATH\stanford-
parser-full-2014-10-31\stanford-parser.jar', path_to_models_jar=r'YOUR_PATH\stanford-
parser-full-2014-10-31\stanford-parser-3.5.0-models.jar')

```

Then we can get a dependency path:

```

1 sentence = "I like that movie."
2 result = dependency_parser.raw_parse(sentence)
3 dep = next(result)
4 print(list(dep.triples()))

```

```

[ (('like', 'VBP'), 'nsubj', ('I', 'PRP')), (('like', 'VBP'), 'dobj', ('movie', 'NN')), (('movie', 'NN'), 'det', ('that', 'DT')) ]

```

- e. Stemming - We can also use nltk.stem to identify the similar meanings of words/sentences.

```

1 from nltk.stem import PorterStemmer
2 from nltk.tokenize import word_tokenize
3
4 ps =PorterStemmer()
5 example_words = ["call", "caller", "calling", "called", "Call", "cally"]
6 example_sentence = "It is very cool to be called while you are calling with caller"\
7 ". As a cally once, call is called."
8
9 words = word_tokenize(example_sentence)
10 for w in words:
11     print(ps.stem(w))

```

it  
is  
veri  
cool  
to  
be  
call  
while  
you  
are  
call  
with  
caller  
.  
as  
a  
calli  
onc  
,  
call  
is  
call  
.

In the above example, we can see NLTK converting “caller”, “called”, to “call”.

### 3. Data analysis - [WordNet](#)

- a. NLTK - we can use nltk.corpus to identify the similarities and differences of the words.

```

1 #WordNet
2 from nltk.corpus import wordnet
3 #similarity
4 w1 = wordnet.synset("woman.n.01") # synset function has an optional pos argument which
5 w2 = wordnet.synset("lady.n.01") #lets you constrain the part of speech of the word:
6
7 #compare the similarity between these two words.
8 print(w1.wup_similarity(w2))
9 #0.9473684210526315 this indicates woman and lady are 94.7% similar.
10
11 w1 = wordnet.synset("woman.n.01")
12 w2 = wordnet.synset("boy.n.01")
13 print(w1.wup_similarity(w2))
14 #0.6666666666666666 woman and boy are 66.6% similar. (humans)
15
16 w1 = wordnet.synset("woman.n.01")
17 w2 = wordnet.synset("food.n.01")
18 print(w1.wup_similarity(w2))
19 #0.36363636363636365 woman and food are 36.36% similar.
20

```

### 4. Naive Bayes and Sentiment analysis

- a. Using NLTK

```

1 #Naive Bayes with NLTK
2 featuresets = [(find_features(rev), category) for (rev, category) in documents]
3 training_set = featuresets[:1900]
4 testing_set = featuresets[1900:] #we define two different sets to avoid bias.
5
6 # posterior = prior occurrences * likelihood / evidence - Naive Bayes
7 classifier = nltk.NaiveBayesClassifier.train(training_set)
8 print("Naive Bayes Algo accuracy: ", (nltk.classify.accuracy(classifier, testing_set))*100)
9 classifier.show_most_informative_features(15) # negs numbers vs positive numbers

```

Naive Bayes Algo accuracy: 76.0

Most Informative Features

sucks = True	neg : pos	=	17.3 : 1.0
justin = True	neg : pos	=	9.1 : 1.0
annual = True	pos : neg	=	8.9 : 1.0
silverstone = True	neg : pos	=	7.8 : 1.0
unimaginative = True	neg : pos	=	7.8 : 1.0
frances = True	pos : neg	=	7.6 : 1.0
schumacher = True	neg : pos	=	7.5 : 1.0
idiotic = True	neg : pos	=	7.3 : 1.0
crappy = True	neg : pos	=	7.1 : 1.0
shoddy = True	neg : pos	=	7.1 : 1.0
atrocious = True	neg : pos	=	6.7 : 1.0
turkey = True	neg : pos	=	6.7 : 1.0
kudos = True	pos : neg	=	6.5 : 1.0
regard = True	pos : neg	=	6.5 : 1.0
cindy = True	neg : pos	=	6.4 : 1.0

## b. Using Textblob

```

1 #Naive Bayes with Textblob
2 new_train, new_test = documents[0:1900], documents[1900:]
3 c1 = NaiveBayesClassifier(new_train)
4 print("Naive Bayes Algo accuracy with textblob: ", 100*c1.accuracy(new_test))
5 c1.show_informative_features(15)

```

Naive Bayes Algo accuracy with textblob: 82.0

Most Informative Features

contains(sucks) = True	neg : pos	=	17.3 : 1.0
contains(outstanding) = True	pos : neg	=	13.5 : 1.0
contains(avoids) = True	pos : neg	=	12.8 : 1.0
contains(astounding) = True	pos : neg	=	12.2 : 1.0
contains(insulting) = True	neg : pos	=	11.2 : 1.0
contains(unbearable) = True	neg : pos	=	11.2 : 1.0
contains(damon) = True	pos : neg	=	10.8 : 1.0
contains(fascination) = True	pos : neg	=	10.8 : 1.0
contains(slip) = True	pos : neg	=	10.8 : 1.0
contains(ludicrous) = True	neg : pos	=	10.6 : 1.0
contains(3000) = True	neg : pos	=	10.5 : 1.0
contains(dud) = True	neg : pos	=	10.5 : 1.0
contains(thematic) = True	pos : neg	=	10.2 : 1.0
contains(hudson) = True	neg : pos	=	9.8 : 1.0
contains(jolie) = True	neg : pos	=	9.8 : 1.0

From the results as above, both NLTK and Textblob work well on text analysis, the accuracy was 76% vs 82%. Since Naive Bayes classifier in Textblob is based on NLTK ([see line 188](#)), so the result is similar which was indicated as above. In a word, we can use both to do the sentiment analysis.

## Conclusion

From the above implementation, we can conclude that NLTK has more robust libraries and support many third-party extensions. Both NLTK and Stanford NLP support a great number of

languages but when it comes to tokenization, NLTK tends to be faster than Stanford NLP, and the size of the language models in the latter is too large, it takes a bit longer time to download. Furthermore, NLTK is quick about scripting a prototype but StanfordNLP requires more codes to chunk out the features. However, NLTK is only supported by Python, however, StanfordNLP can be supported by multiple languages like Python, Java, and Jython. NLTK processes strings that are not very practical for the object-oriented language python either.

## **Reference**

<https://www.nltk.org/howto.html>

<https://www.youtube.com/playlist?list=PLQVvva0QuDf2JswnfGkIbInZnIC4HL>

<https://stevenloria.com/simple-text-classification/>

<https://stackoverflow.com/questions/56888321/what-are-the-difference-between-textblob-and-nltk-classifiers>

<https://github.com/sloria/TextBlob/blob/dev/textblob/classifiers.py#L188>

<https://elitedatascience.com/python-nlp-libraries>

<https://towardsdatascience.com/natural-language-processing-using-stanfords-corenlp-d9e64c1e1024>

<https://stackoverflow.com/questions/47624742/how-to-use-stanford-word-tokenizer-in-nltk>