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 text analysis between NLTK and other NLP tools:

1. Install and environment setting.
    a. First of all, Install NLTK , Textblob and Standford 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

```python
import nltk
import random
from nltk.corpus import state_union
from nltk.tokenize import PunktSentenceTokenizer
from nltk.corpus import movie_reviews
from nltk.corpus import stopwords
from textblob.classifiers import NaiveBayesClassifier
```

Using StandfordNLP

```python
import corenlp
```

b. Tokenizing

Using NLTK - word tokenizers and sentences tokenizers.

```python
from nltk.tokenize import sent_tokenize, word_tokenize
example_text = "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."

print(sent_tokenize(example_text)) #creat a list.

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

Using StandfordNLP -

```python
with corenlp.client.CoreNLPClient(annotators="tokenize ssplit".split()) as client:
    ann = client.annotate("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.")
sentence = ann.sentence[0]
[token.word for token in sentence.token]
#['Hello', 'Mr.', 'Smith', ',', 'how', 'are', 'you', 'doing', 'today', '?', 'The', 'weather', 'is', 'great', 'and',
#'.', 'You', 'should', 'not', 'eat', 'carboard', '.']
```

c. Removing stop words and POS Tagging with NLTK

```python
example_sentence = "This is just a simple example to show off how to filter the stop words."
stop_words = set(stopwords.words("english")) # choose english as the language to get all stopwords in it.

words = word_tokenize(example_sentence)
filtered_sentence = []

# for word in words:
#     if word not in stop_words:
#         filtered_sentence.append(word)
filtered_sentence = [word for word in words if not word in stop_words]
print(filtered_sentence)
```

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

```python
#Part of Speech Tagging
import nltk
from nltk.corpus import state_union

sample_text = "I like that Movie." #loading some data as input.

tokenized = custom_sent_tokenizer.tokenize(sample_text)
def process_content():
    try:
        for i in tokenized:
            words = nltk.word_tokenize(i)
            tagged = nltk.pos_tag(words)

            print(tagged)
    except Exception as e:
        print(str(e))
process_content()
```

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

If you are interested in what stop words are, check here. From above example, we can see all words were tagged as part-of-speech. Upon my research, some of the pre-processing stages like removal of stopwords, punctuation, or special characters are not included in the StandfordNLP. However, we can run the Standford Pos Tagger on NLTK Using Java or Python.

d. There is another interesting example shows that how can we use Stanford Parser to get a dependency path. First of all, import the library :

```python
import os
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 dependencies, see below：

```python
sentence = "I like that movie."
result = dependency_parser.raw_parse(sentence)
dep = next(result)
print(list(dep.triples()))
```

```
[(('like', 'VBP'), 'nsubj', ('I', 'PRP')), (('like', 'VBP'), 'dobj', ('movie', 'NN')), (('movie', 'NN'), 'det', ('tha
t', 'DT'))]
```

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

```
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize

ps =PorterStemmer()
example_words = ["call", "caller", "calling", "called", "Call", "cally"]
example_sentence = "It is very cool to be called while you are calling with caller"\
". As a cally once, call is called."

words = word_tokenize(example_sentence)
for w in words:
    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. WordNet with NLTK

We we can use nltk.corpus to identify the similarities and differences of the words.

```
#WordNet
from nltk.corpus import wordnet
#similarity
w1 = wordnet.synset("woman.n.01") # synset function has an optional pos argument which
w2 = wordnet.synset("lady.n.01")  #lets you constrain the part of speech of the word:

#compare the similarity between these two words.
print(w1.wup_similarity(w2))
#0.9473684210526315 this indicates woman and lady are 94.7% similar.

w1 = wordnet.synset("woman.n.01")
w2 = wordnet.synset("boy.n.01")
print(w1.wup_similarity(w2))
#0.6666666666666666 woman and boy are 66.6% similar. (humans)

w1 = wordnet.synset("woman.n.01")
w2 = wordnet.synset("food.n.01")
print(w1.wup_similarity(w2))
#0.36363636363636365 woman and food are 36.36% similar.

```

From above example, we can see the similarities between "woman" and "lady" are over 90 percent, compared to 36 percent between “woman” and “food”, which makes sense.

4. Naive Bayes and Sentiment analysis

   a. Using NLTK

```
#Naive Bayes with NLTK
featuresets = [(find_features(rev), category) for (rev, category) in documents]
training_set = featuresets[:1900]
testing_set = featuresets[1900:] #we define two different sets to avoid bias.

# posterior = prior occurences * likelihood / evidence - Naive Bayes
classifier = nltk.NaiveBayesClassifier.train(training_set)
print("Naive Bayes Algo accuracy: ", (nltk.classify.accuracy(classifier, testing_set))*100)
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

```
#Naive Bayes with Textblob
new_train, new_test = documents[0:1900], documents[1900:]
cl = NaiveBayesClassifier(new_train)
print("Naive Bayes Algo accuracy with textblob: ", 100*cl.accuracy(new_test))
cl.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 Bayers 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 and interfaces. 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. Instead, StanfordNLP was written in java, yet it is supported by multiple languages like Python, Java, and Jython. NLTK processes strings that are not very practical for the object-oriented language python either. In a word, NLTK is very powerful on handling unstructured texts, especially If you are new to text retrieval and analysis on English, NLTK would be one of the best choices.

Reference

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

https://www.youtube.com/playlist?list=PLQVvvaa0QuDf2JswnfiGkliBInZnIC4HL

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