The `nltk` Package

> An original version of this guide was contributed by Mike Zhu (@mz888).

The `nltk` (Natural Language Tool Kit) package is a good introduction to some common Natural Language Processing (NLP) processes, including Sentiment Analysis, Named Entity Recognition, and document preprocessing. You can also download corpus collections with nltk for practice or to serve as training data for machine learning applications.

Reference:

- http://www.nltk.org/
- http://www.nltk.org/book

Installation

First install the package using Pip, if necessary:

```
pip install nltk
```

Usage

Sentiment Analysis

One of the most widely used NLP techniques is Sentiment Analysis. One of the modules available from `nltk` is the Vader Sentiment Analyzer, a relatively simple, vocabulary-based tool for measuring sentiment.

```
import nltk
# you will need to download the Vader sentiment lexicon the first time you
use it.
# to do this, we can use nltk's download function, which will bring up a GUI.
nltk.download()

# Select the vader_lexicon file and download
from nltk.sentiment.vader import SentimentIntensityAnalyzer
sid = SentimentIntensityAnalyzer()

# let's make two sample sentences to test out Vader
positive = "Python is fantastic and useful"
negative = "R is cruel and unusual"

# Vader outputs a score for positive, negative, and neutral
sid.polarity_scores(positive)
#> {'neg': 0.0, 'compound': 0.7579, 'neu': 0.316, 'pos': 0.684} # the
```

```
compound score is the overall score of the text
sid.polarity_scores(negative)
#> {'neg': 0.559, 'compound': -0.5859, 'neu': 0.441, 'pos': 0.0}
```

Entity Identification

Let's say we want to find instances of people, places, or other proper nouns in a document. This NLP task, called "Named Entity Extraction," can also be implemented with `nltk`.

```
import nltk
from nltk import word_tokenize, pos_tag, ne_chunk

# NER in nltk actually takes 3 discrete steps. First, we tokenize the
sentence by splitting it up into words.
sent = "Derek Jeter met Mariano Rivera in New York."
token = word_tokenize(sent)
token #> ['Derek', 'Jeter', 'met', 'Mariano', 'Rivera', 'in', 'New York',
'.']

# Then, we employ part-of-speech tagging to get the grammatical construct of
the sentence
tagged = pos_tag(token)
# notice that each word in the list below is designated a grammatical label:
NNP, for example, is a proper noun
tagged #> [('Derek', 'NNP'), ('Jeter', 'NNP'), ('met', 'VBD'), ('Mariano',
'NNP'), ('Rivera', 'NNP'), ('in', 'IN'), ('New York', 'NNP'), ('.', '.')]

# Finally, we use the ne_chunk function to detect proper nouns.
chunk = ne_chunk(tagged)
# Each Named Entity is assigned a type; New York is identified as a GPE
(geopolitical entity)
chunk #> Tree('S', [Tree('PERSON', [('Derek', 'NNP')]), Tree('PERSON',
[('Jeter', 'NNP')]), ('met', 'VBD'), Tree('PERSON', [('Mariano', 'NNP'),
('Rivera', 'NNP')]), ('in', 'IN'), Tree('GPE', [('New York', 'NNP')]), ('.',
'.')])

# Can you think of a way to clean up the output?
```

The `nltk` package contains many modules with different functionalities. Consult the NLTK book as well as other online guides to explore its many uses.