An advanced guide to NLP analysis with Python and NLTK

opensource.com/article/20/8/nlp-python-nltk

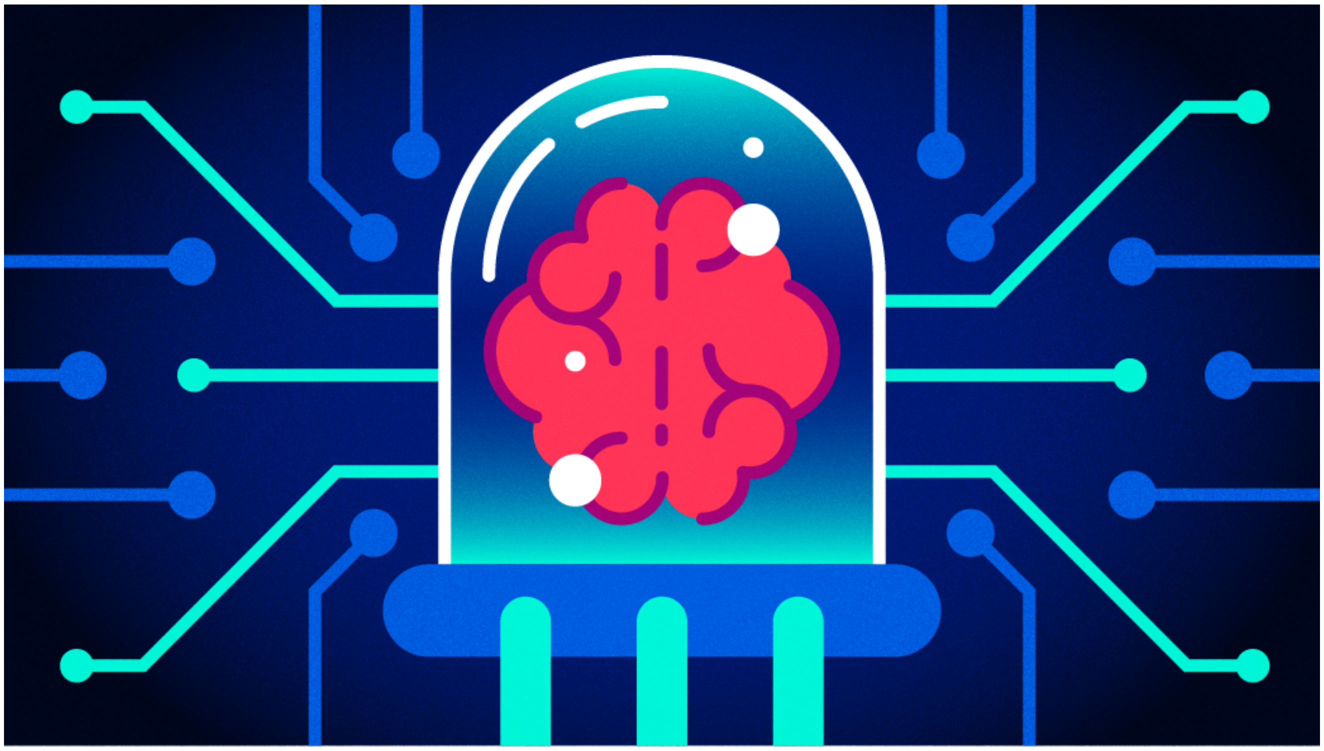


Image by:
opensource.com

In my previous article, I introduced natural language processing (NLP) and the Natural Language Toolkit (NLTK), the NLP toolkit created at the University of Pennsylvania. I demonstrated how to parse text and define stopwords in Python and introduced the concept of a corpus, a dataset of text that aids in text processing with out-of-the-box data. In this article, I'll continue utilizing datasets to compare and analyze natural language.

The fundamental building blocks covered in this article are:

- WordNet and synsets
- Similarity comparison
- Tree and treebank
- Named entity recognition

WordNet and synsets

WordNet is a large lexical database corpus in NLTK. WordNet maintains cognitive synonyms (commonly called synsets) of words correlated by nouns, verbs, adjectives, adverbs, synonyms, antonyms, and more.

WordNet is a very useful tool for text analysis. It is available for many languages (Chinese, English, Japanese, Russian, Spanish, and more), under many licenses (ranging from open source to commercial). The first WordNet was created by Princeton University for English under an MIT-like license.

A word is typically associated with multiple synsets based on its meanings and parts of speech. Each synset usually provides these attributes:

Attribute	**Definition**	**Example**
Name	Name of the synset	Example: The word "code" has five synsets with names `code.n.01`, `code.n.02`, `code.n.03`, `code.v.01`, `code.v.02`
POS	Part of speech of the word for this synset	The word "code" has three synsets in noun form and two in verb form
Definition	Definition of the word (in POS)	One of the definitions of "code" in verb form is: "(computer science) the symbolic arrangement of data or instructions in a computer program"
Examples	Examples of word's use	One of the examples of "code": "We should encode the message for security reasons"
Lemmas	Other word synsets this word+POC is related to (not strictly synonyms, but can be considered so); lemmas are related to other lemmas, not to words directly	Lemmas of `code.v.02` (as in "convert ordinary language into code") are `code.v.02.encipher`, `code.v.02.cipher`, `code.v.02.cypher`, `code.v.02.encrypt`, `code.v.02.inscribe`, `code.v.02.write_in_code`
Antonyms	Opposites	Antonym of lemma `encode.v.01.encode` is `decode.v.01.decode`
Hypernym	A broad category that other words fall under	A hypernym of `code.v.01` (as in "Code the pieces with numbers so that you can identify them later") is `tag.v.01`
Meronym	A word that is part of (or subordinate to) a broad category	A meronym of "computer" is "chip"

Attribute	Definition	Example
Holonym	The relationship between a parent word and its subordinate parts	A hyponym of "window" is "computer screen"

There are several other attributes, which you can find in the `nltk/corpus/reader/wordnet.py` source file in `<your python install>/Lib/site-packages`.

Some code may help this make more sense.

This helper function:

```
def synset_info(synset):
    print("Name", synset.name())
    print("POS:", synset.pos())
    print("Definition:", synset.definition())
    print("Examples:", synset.examples())
    print("Lemmas:", synset.lemmas())
    print("Antonyms:", [lemma.antonyms() for lemma in synset.lemmas() if
len(lemma.antonyms()) > 0])
    print("Hypernyms:", synset.hypernyms())
    print("Instance Hypernyms:", synset.instance_hypernyms())
    print("Part Holonyms:", synset.part_holonyms())
    print("Part Meronyms:", synset.part_meronyms())
    print()

synsets = wordnet.synsets('code')
```

shows this:

```
5 synsets:
Name code.n.01
POS: n
Definition: a set of rules or principles or laws (especially written ones)
Examples: []
Lemmas: [Lemma('code.n.01.code'), Lemma('code.n.01.codification')]
Antonyms: []
Hypernyms: [Synset('written_communication.n.01')]
Instance Hpernyms: []
Part Holonyms: []
Part Meronyms: []

...

Name code.n.03
POS: n
Definition: (computer science) the symbolic arrangement of data or instructions in a
computer program or the set of such instructions
Examples: []
Lemmas: [Lemma('code.n.03.code'), Lemma('code.n.03.computer_code')]
Antonyms: []
Hypernyms: [Synset('coding_system.n.01')]
Instance Hpernyms: []
Part Holonyms: []
Part Meronyms: []

...

Name code.v.02
POS: v
Definition: convert ordinary language into code
Examples: ['We should encode the message for security reasons']
Lemmas: [Lemma('code.v.02.code'), Lemma('code.v.02.encipher'),
Lemma('code.v.02.cipher'), Lemma('code.v.02.cypher'), Lemma('code.v.02.encrypt'),
Lemma('code.v.02.inscribe'), Lemma('code.v.02.write_in_code')]
Antonyms: []
Hypernyms: [Synset('encode.v.01')]
Instance Hpernyms: []
Part Holonyms: []
Part Meronyms: []
```

Synsets and lemmas follow a tree structure you can visualize:

```
def hypernyms(synset):
    return synset.hypernyms()

synsets = wordnet.synsets('soccer')
for synset in synsets:
    print(synset.name() + " tree:")
    pprint(synset.tree(rel=hypernyms))
    print()
```

```
code.n.01 tree:
[Synset('code.n.01'),
 [Synset('written_communication.n.01'),
   ...

code.n.02 tree:
[Synset('code.n.02'),
 [Synset('coding_system.n.01'),
   ...

code.n.03 tree:
[Synset('code.n.03'),
   ...

code.v.01 tree:
[Synset('code.v.01'),
 [Synset('tag.v.01'),
   ...

code.v.02 tree:
[Synset('code.v.02'),
 [Synset('encode.v.01'),
   ...
```

WordNet does not cover all words and their information (there are about 170,000 words in English today and about 155,000 in the latest version of WordNet), but it's a good starting point. After you learn the concepts of this building block, if you find it inadequate for your needs, you can migrate to another. Or, you can build your own WordNet!

Try it yourself

Using the Python libraries, download Wikipedia's page on open source and list the synsets and lemmas of all the words.

Similarity comparison

Similarity comparison is a building block that identifies similarities between two pieces of text. It has many applications in search engines, chatbots, and more.

For example, are the words "football" and "soccer" related?

```
syn1 = wordnet.synsets('football')
syn2 = wordnet.synsets('soccer')

# A word may have multiple synsets, so need to compare each synset of word1 with
synset of word2
for s1 in syn1:
    for s2 in syn2:
        print("Path similarity of: ")
        print(s1, '(', s1.pos(), ')', '[', s1.definition(), ']')
        print(s2, '(', s2.pos(), ')', '[', s2.definition(), ']')
        print("   is", s1.path_similarity(s2))
        print()
```

```
Path similarity of:
Synset('football.n.01') ( n ) [ any of various games played with a ball (round or
oval) in which two teams try to kick or carry or propel the ball into each other's
goal ]
Synset('soccer.n.01') ( n ) [ a football game in which two teams of 11 players try to
kick or head a ball into the opponents' goal ]
   is 0.5

Path similarity of:
Synset('football.n.02') ( n ) [ the inflated oblong ball used in playing American
football ]
Synset('soccer.n.01') ( n ) [ a football game in which two teams of 11 players try to
kick or head a ball into the opponents' goal ]
   is 0.05
```

The highest path similarity score of the words is 0.5, indicating they are closely related.

What about "code" and "bug"? Similarity scores for these words used in computer science are:

```
Path similarity of:
Synset('code.n.01') ( n ) [ a set of rules or principles or laws (especially written
ones) ]
Synset('bug.n.02') ( n ) [ a fault or defect in a computer program, system, or
machine ]
   is 0.1111111111111111
...
Path similarity of:
Synset('code.n.02') ( n ) [ a coding system used for transmitting messages requiring
brevity or secrecy ]
Synset('bug.n.02') ( n ) [ a fault or defect in a computer program, system, or
machine ]
   is 0.09090909090909091
...
Path similarity of:
Synset('code.n.03') ( n ) [ (computer science) the symbolic arrangement of data or
instructions in a computer program or the set of such instructions ]
Synset('bug.n.02') ( n ) [ a fault or defect in a computer program, system, or
machine ]
   is 0.09090909090909091
```

These are the highest similarity scores, which indicates they are related.

NLTK provides several similarity scorers, such as:

- path_similarity
- lch_similarity
- wup_similarity
- res_similarity
- jcn_similarity
- lin_similarity

See the Similarity section of the WordNet Interface page to determine the appropriate one for your application.

Try it yourself

Using Python libraries, start from the Wikipedia Category: Lists of computer terms page and prepare a list of terminologies, then see how the words correlate.

Tree and treebank

With NLTK, you can represent a text's structure in tree form to help with text analysis.

Here is an example:

A simple text pre-processed and part-of-speech (POS)-tagged:

```
import nltk

text = "I love open source"
# Tokenize to words
words = nltk.tokenize.word_tokenize(text)
# POS tag the words
words_tagged = nltk.pos_tag(words)
```

You must define a grammar to convert the text to a tree structure. This example uses a simple grammar based on the Penn Treebank tags.

```
# A simple grammar to create tree
grammar = "NP: {<JJ><NN>}"
```

Next, use the grammar to create a tree:

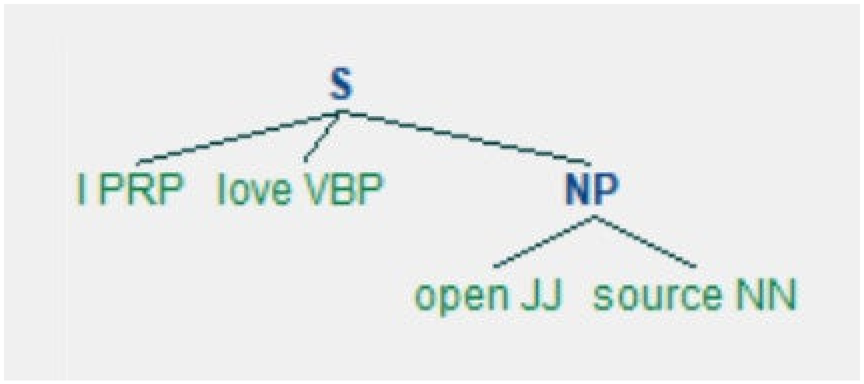
```
# Create tree
parser = nltk.RegexpParser(grammar)
tree = parser.parse(words_tagged)
pprint(tree)
```

This produces:

```
Tree('S', [('I', 'PRP'), ('love', 'VBP'), Tree('NP', [('open', 'JJ'), ('source',
'NN')])])
```

You can see it better graphically.

```
tree.draw()
```



(Girish Managoli, CC BY-SA 4.0)

This structure helps explain the text's meaning correctly. As an example, identify the subject in this text:

```
subject_tags = ["NN", "NNS", "NP", "NNP", "NNPS", "PRP", "PRP$"]
def subject(sentence_tree):
    for tagged_word in sentence_tree:
        # A crude logic for this case -  first word with these tags is considered
subject
        if tagged_word[1] in subject_tags:
            return tagged_word[0]

print("Subject:", subject(tree))
```

It shows "I" is the subject:

```
Subject: I
```

This is a basic text analysis building block that is applicable to larger applications. For example, when a user says, "Book a flight for my mom, Jane, to NY from London on January 1st," a chatbot using this block can interpret the request as:

Action: Book

What: Flight

Traveler: Jane

From: London

To: New York

Date: 1 Jan (of the next year)

A treebank refers to a corpus with pre-tagged trees. Open source, conditional free-for-use, and commercial treebanks are available for many languages. The most commonly used one for English is Penn Treebank, extracted from the *Wall Street Journal*, a subset of which is included in NLTK. Some ways of using a treebank:

```
words = nltk.corpus.treebank.words()
print(len(words), "words:")
print(words)

tagged_sents = nltk.corpus.treebank.tagged_sents()
print(len(tagged_sents), "sentences:")
print(tagged_sents)
```

```
100676 words:
['Pierre', 'Vinken', ',', '61', 'years', 'old', ',', ...]
3914 sentences:
[[('Pierre', 'NNP'), ('Vinken', 'NNP'), (',', ','), ('61', 'CD'), ('years', 'NNS'),
('old', 'JJ'), (',', ','), ('will', 'MD'), ('join', 'VB'), ('the', 'DT'), ('board',
'NN'), ('as', 'IN'), ('a', 'DT'), ('nonexecutive', 'JJ'), ('director', 'NN'), ...]
```

See tags in a sentence:

```
sent0 = tagged_sents[0]
pprint(sent0)

[('Pierre', 'NNP'),
 ('Vinken', 'NNP'),
 (',', ','),
 ('61', 'CD'),
 ('years', 'NNS'),
...
```

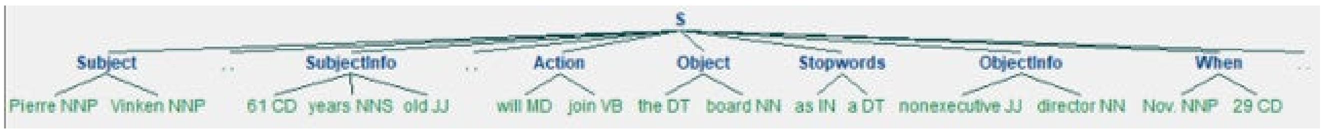
Create a grammar to convert this to a tree:

```
grammar = '''
    Subject: {<NNP><NNP>}
    SubjectInfo: {<CD><NNS><JJ>}
    Action: {<MD><VB>}
    Object: {<DT><NN>}
    Stopwords: {<IN><DT>}
    ObjectInfo: {<JJ><NN>}
    When: {<NNP><CD>}
'''
parser = nltk.RegexpParser(grammar)
tree = parser.parse(sent0)
print(tree)

(S
  (Subject Pierre/NNP Vinken/NNP)
  ,/,
  (SubjectInfo 61/CD years/NNS old/JJ)
  ,/,
  (Action will/MD join/VB)
  (Object the/DT board/NN)
  as/IN
  a/DT
  (ObjectInfo nonexecutive/JJ director/NN)
  (Subject Nov./NNP)
  29/CD
  ./.)
```

See it graphically:

```
tree.draw()
```



(Girish Managoli, CC BY-SA 4.0)

The concept of trees and treebanks is a powerful building block for text analysis.

Try it yourself

Using the Python libraries, download Wikipedia's page on open source and represent the text in a presentable view.

Named entity recognition

Text, whether spoken or written, contains important data. One of text processing's primary goals is extracting this key data. This is needed in almost all applications, such as an airline chatbot that books tickets or a question-answering bot. NLTK provides a named entity recognition feature for this.

Here's a code example:

```
sentence = 'Peterson first suggested the name "open source" at Palo Alto, California'
```

See if name and place are recognized in this sentence. Pre-process as usual:

```
import nltk

words = nltk.word_tokenize(sentence)
pos_tagged = nltk.pos_tag(words)
```

Run the named-entity tagger:

```
ne_tagged = nltk.ne_chunk(pos_tagged)
print("NE tagged text:")
print(ne_tagged)
print()

NE tagged text:
(S
  (PERSON Peterson/NNP)
  first/RB
  suggested/VBD
  the/DT
  name/NN
  ``/``
  open/JJ
  source/NN
  ''/''
  at/IN
  (FACILITY Palo/NNP Alto/NNP)
  ,/,
  (GPE California/NNP))
```

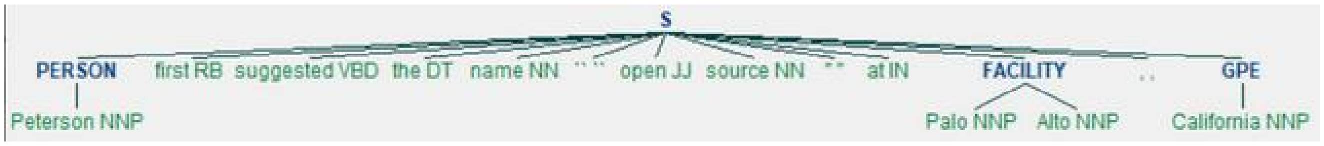
Name tags were added; extract only the named entities from this tree:

```
print("Recognized named entities:")
for ne in ne_tagged:
    if hasattr(ne, "label"):
        print(ne.label(), ne[0:])

Recognized named entities:
PERSON [('Peterson', 'NNP')]
FACILITY [('Palo', 'NNP'), ('Alto', 'NNP')]
GPE [('California', 'NNP')]
```

See it graphically:

```
ne_tagged.draw()
```



(Girish Managoli, CC BY-SA 4.0)

NLTK's built-in named-entity tagger, using PENN's Automatic Content Extraction (ACE) program, detects common entities such as ORGANIZATION, PERSON, LOCATION, FACILITY, and GPE (geopolitical entity).

NLTK can use other taggers, such as the Stanford Named Entity Recognizer. This trained tagger is built in Java, but NLTK provides an interface to work with it (See nltk.parse.stanford or nltk.tag.stanford).

Try it yourself

Using the Python libraries, download Wikipedia's page on open source and identify people who had an influence on open source and where and when they contributed.

Advanced exercise

If you're ready for it, try building this superstructure using the building blocks discussed in these articles.

Using Python libraries, download Wikipedia's Category: Computer science page and:

- Identify the most-occurring unigrams, bigrams, and trigrams and publish it as a list of keywords or technologies that students and engineers need to be aware of in this domain.
- Show the names, technologies, dates, and places that matter in this field graphically. This can be a nice infographic.
- Create a search engine. Does your search engine perform better than Wikipedia's search?

What's next?

NLP is a quintessential pillar in application building. NLTK is a classic, rich, and powerful kit that provides the bricks and mortar to build practically appealing, purposeful applications for the real world.

In this series of articles, I explained what NLP makes possible using NLTK as an example. NLP and NLTK have a lot more to offer. This series is an inception point to help get you started.

If your needs grow beyond NLTK's capabilities, you could train new models or add capabilities to it. New NLP libraries that build on NLTK are coming up, and machine learning is being used extensively in language processing.

Girish Managoli

Girish has over 20 years' experience in technology and software at a global IT Services organization based in India. Girish is architect of "I Got" cloud platform to uplift the bottom of the pyramid built with open source stack and contemporary architectural patterns such as microservices, containerisation and multi tenancy. Girish writes on open source and tech topics.

More about me

Comments are closed.

These comments are closed, however you can Register or Login to post a comment on another article.