

Parsing Text

You now have a preprocessed, clean list of words. Now what? It may be helpful to know how the words relate to each other and the underlying syntax (grammar). **Parsing** is an NLP process concerned with segmenting text based on syntax.

You probably do not want to be doing any parsing by hand and NLTK has a few tricks up its sleeve to help you out:

Part-of-speech tagging (POS tagging) identifies parts of speech (verbs, nouns, adjectives, etc.). NLTK can do it faster (and maybe more accurately) than your grammar teacher.

Named entity recognition (NER) helps identify the proper nouns (e.g., "Natalia" or "Berlin") in a text. This can be a clue as to the topic of the text and NLTK captures many for you.

Dependency grammar trees help you understand the relationship between the words in a sentence. It can be a tedious task for a human, so the Python library spaCy is at your service, even if it isn't always perfect.

In English we leave a lot of ambiguity, so syntax can be tough, even for a computer program. Take a look at the following sentence:

I saw a cow under a tree with binoculars

Do I have the binoculars? Does the cow have binoculars? Does the tree have binoculars?

Regex parsing, using Python's `re` library, allows for a bit more nuance. When coupled with POS tagging, you can identify specific phrase chunks. On its own, it can find you addresses, emails, and many other common patterns within large chunks of text.

Instructions

Run the code to see the silly squid sentences parsed into dependency trees visually!

Change `my_sentence` to a sentence of your choosing and run the code again to see it parsed out as a tree!

script.py

```
import spacy
from nltk import Tree
from squids import squids_text

dependency_parser = spacy.load('en')

parsed_squids = dependency_parser(squids_text)

# Assign my_sentence a new value:
my_sentence = "I want to be an expert in artificial intelligence!"
my_parsed_sentence = dependency_parser(my_sentence)

def to_nltk_tree(node):
    if node.n_lefts + node.n_rights > 0:
        parsed_child_nodes = [to_nltk_tree(child) for child in node.children]
        return Tree(node.orth_, parsed_child_nodes)
    else:
        return node.orth_

for sent in parsed_squids.sents:
    to_nltk_tree(sent.root).pretty_print()

for sent in my_parsed_sentence.sents:
    to_nltk_tree(sent.root).pretty_print()
```