1. Identify larger syntactic structure.
2. Extract syntactic features from trees
3. Implement key aspects of these algorithms.
4. Use statistics to disambiguate language to find the best parse.
5. Evaluate parsing output relative to gold standard treebanks.

Different nltk syntactic parser :

* The Stanford Parser
* The Chart Parser - disambiguate language : multiple meaning
* The Shift-Reduce Parser
* The Recursive Descent Parser

Nltk regex parser is not syntactic parser. It is normal parser.

# Tree class constructor:

PRP = Tree("PRP", ["I"])

VBD = Tree("VBD", ["met"])

N1 = Tree("NN", ["bill"])

NP1 = Tree("NP", [PRP])

NP = Tree("NP", [N1])

VP = Tree("VP", [VBD, NP])

S= Tree("s", [NP1, VP])

print(S)

#(s (NP (PRP I)) (VP (VBD met) (NP (NN bill))))

# Bracketed string

S = Tree.fromstring("(S (NP (PRP I)) (VP (VBD met) (NP (NN Bill))))")

print(S)

#(S (NP (PRP I)) (VP (VBD met) (NP (NN Bill))))

# Alternative representation

np = Tree("NP", [("I", "PRP")])

vp = Tree("VP", [("met", "VBD"), ("NP", [("Bill", "NN")])])

S= Tree("s", [NP1, VP])

print(S)

#(s (NP I/PRP) (VP (VBD met/VBD) (NP bill/NN)))

The cat sat on the mat

First “The cat” is NP-SBJ

Second ‘the mat’ is only NPs

# nltk.tree.Tree

from nltk.corpus import treebank

from nltk.tree import Tree

# The function parsed\_sents() returns the syntax trees in the treebank.

syntax\_trees = treebank.parsed\_sents()

## tree.label

tree = syntax\_trees[0]

print(tree.label())

for phrase in tree:

print(phrase)

print("---")

## tree.subtrees

for phrase in tree.subtrees():

print(phrase)

print("---")

print(tree.leaves())

## Tree.fromstring

tree = Tree.fromstring("(S (NP (PRP I)) (VP (VBD met) (NP (NNP Bill))))")

print(tree.height())

print(tree.leaves())

print(tree.height())

print(tree.flatten())

# nltk.RegexpParser

* can define a set of grammar

from nltk import RegexpParser

simple\_chunker = RegexpParser("NP: {<DT|PRP\$><NN>}")

**parsed\_sent = simple\_chunker.parse(tagged\_sent)**

chunks = []

for phrase in parsed\_sent.subtrees():

if phrase.label() == "NP":

words = [word for word, tag in phrase]

chunks.append(" ".join(words))

"NP: {<DT|PRP\$><NN>}"

The structure of the grammar is:

* "NP:" is label, in this case "NP" for noun phrase.
* "{<DT|PRP$><NN>}" is the actual grammar definition.

The definition consists of multiple tags that are separated by < > :

* <DT> - Defines a determiner such as "the", "a", "an".
* <PRP$> - Defines possessive pronoun such as "my","his","hers"
* <NN> - Defines a noun such as "fox", "dog"

1. \ character is used to escape the following character.
2. for this case if we want to make $ as regex inside {} then we have to use \ otherwise it will be cosidered as the $ in string.
3. if $ as literal character means as regex

# Chunk dataset, GOLD standard chunk

conll2000 chunking shared task dataset.

from nltk.corpus import conll2000

gold\_chunked\_sents = conll2000.chunked\_sents(chunk\_types=["NP"])

tagged\_sent = conll2000.tagged\_sents()

print(gold\_chunked\_sents[0])

print(tagged\_sent)

print(simple\_chunker.parse(tagged\_sent[0]))

PRP : personal pronoun

I," "you," "he," "she," "it," "we," "they," "me," "him," "her," "us,"

JJ: adjective