**Probabilistic Context-Free Grammars and Subcategories of Verbs**

In this section, we give examples of two different ideas. The first is the idea of subcategories of verbs. Some of the subcategories in English are:

transitive verbs, such as ‘saw’ and ‘chased’, require an NP direct object

The cat saw the dog.

The dog chased the squirrel.

intransitive verbs do not take any object

The dog barked.

dative verbs have two objects, expressed in grammar as either two objects or a direct object and a prepositional phrase

He gave John the book.

He gave a dog to a man.

sentential verbs are followed by a sentential construct

He said that a dog barked.

In addition, there may be optional modifiers, such as adverbs, and auxiliary verbs (*be, do, have, must, might, etc.*) for some verb tenses, that we won’t go into here.

The squirrel was really frightened.

The man really saw a bear.

The man really thought the bear was angry.

The other idea is that of the probabilistic grammar. In these grammars, each rule is associated with the probability that the left-hand-side symbol is rewritten using that particular rule. The probabilities for each non-terminal symbol must add up to 1. Note that I put in a rule to allow dative verbs to have a NP PP, but I omitted the case of NP NP.

>>>prob\_grammar = nltk.PCFG.fromstring("""

S -> NP VP [0.9]| VP [0.1]

VP -> TranV NP [0.3]

VP -> InV [0.3]

VP -> DatV NP PP [0.4]

PP -> P NP [1.0]

TranV -> "saw" [0.2] | "ate" [0.2] | "walked" [0.2] | "shot" [0.2] | "book" [0.2]

InV -> "ate" [0.5] | "walked" [0.5]

DatV -> "gave" [0.2] | "ate" [0.2] | "saw" [0.2] | "walked" [0.2] | "shot" [0.2]

NP -> Prop [0.2]| Det N [0.4] | Det N PP [0.4]

Prop -> "John" [0.25]| "Mary" [0.25] | "Bob" [0.25] | "I" [0.25]

Det -> "a" [0.2] | "an" [0.2] | "the" [0.2] | "my" [0.2] | "that" [0.2]

N -> "man" [0.15] | "dog" [0.15] | "cat" [0.15] | "park" [0.15] | "telescope" [0.1] | "flight" [0.1] | "elephant" [0.1] | "pajamas" [0.1]

P -> "in" [0.2] | "on" [0.2] | "by" [0.2] | "with" [0.2] | "through" [0.2]

""")

The NLTK provides a parser called ViterbiParser to parse using probabilistic CFGs:

>>>viterbi\_parser = nltk.ViterbiParser(prob\_grammar)

>>>for tree in viterbi\_parser.parse(['John', 'saw', 'a', 'telescope']):

print (tree)

>>>for tree in viterbi\_parser.parse(sent2list):

print (tree)

>>>for tree in viterbi\_parser.parse(sent4list):

print (tree)

**Last week’s parsing exercise (optional)**

This week’s Python file has one solution

**Parsing demos (optional)**

Recall that we mentioned the NLTK parsing demos during the lectures, and these are described in Chapter 8 of the NLTK book, section 8.4. First we look at the parsing demo for the recursive descent parser, which is a top-down, back-tracking parser. [In my experience, these do not run on some versions of Mac OS, but do run in the labs.]

nltk.app.rdparser()

The second shows the shift-reduce parser, which is a bottom-up parser and needs guidance as to what operation (shift or reduce) to apply at some steps.

nltk.app.srparser()

The third shows a chart parser. Section 8.4 also has a description of Chart Parsing, including the chart data structures, called Well-Formed Substring Tables in NLTK. Here is one way to run the chart parser demo. You can omit the first argument to see the parser choices.

nltk.parse.chart.demo()

[Note:

For short examples of all the parsers, see the HOWTO parse module document: <http://www.nltk.org/howto/parse.html>

End Note]

Note that the NLTK has shift-reduce parsers as well, but as we noted in the parser demo, it doesn’t have backtracking so it doesn’t always find parse trees. The shift-reduce parser is also further described in section 8.4 of the NLTK book.