Experiment 8

Parsing and context free grammers

Aim:

Develop parsing and context free grammars.

Theory:

The first thing we'll do in this lab is create our own CFG and use it to parse a sentence. The goal of is to help us better understand the recursive nature of Context-Free Grammars. By definition, these grammars have to be recursive, because valid sentences can contain other smaller, valid sentences. Consider the following example, pulled from the NLTK book's chapter on Analyzing Sentence Structure. We've highlighted the top-level Verb Phrase in each sentence.

a. Usain Bolt broke the 100m record

b. The Jamaica Observer reported that Usain Bolt broke the 100m record

c. Andre said The Jamaica Observer reported that Usain Bolt broke the 100m record

d. I think Andre said the Jamaica Observer reported that Usain Bolt broke the 100m record

What do we notice?

In the examples above, each sentence is easily recognizable as being grammatically valid. And yet, we can see that sentence 'd' contains 3 other possible valid sentences. This is because of the recursive structure of grammar. Recall that a sentence (S) is made up of a Noun Phrase (NP) and a Verb Phrase (VP). This gets interesting when we realize that both NPs and VPs are allowed to be made up of other NPs and VPs! In each of the sentences above, we can see that the smaller sentences are just constituent pieces of the top-level Verb Phrase. For example, in sentence b, the top-level Noun Phrase is "The Jamaica Observer". The top-level Verb Phrase is 'reported that Usain Bolt broke the 100m record'. However, this Verb Phrase can be broken down into a verb 'reported', and a Noun Phrase 'that Usain Bolt broke the 100m record.' This noun phrase can itself be broken down into the smaller, valid parse tree from the first sentence, with 'Usain Bolt' as a Noun Phrase, and 'broke the 100m record' as a Verb Phrase. All this recursion can be a bit confusing! Luckily, computers are really, really good at it.

To really drive this point home about sentences being made up of recursive units of words, take a look at the following diagram (also from the NLTK ch. 8 provided above):

This diagram shows the various valid combinations words from the sentences "He ran" and "The little bear saw the fine fat trout in the brook". As we can see from the diagram, as long as we're switching out grammatical units that are the same type (e.g. a Noun Phrase for a Noun Phrase), the result is a completely valid sentence--both "the little bear saw the fine fat trout in the brook" and "he saw the fine fat trout in the brook" are grammatically correct. This means that we don't actually need to care what the actual words are--just their grammatical part of speech!

Now that we've seen some examples, let's get to creating our own CFG and using it to create a parse tree!

Generating a CFG

NLTK makes it extremely easy to generate a CFG by providing a helper function. Here's the example we saw in the last lesson, which was also pulled from the NLTK book in chapter 8:

Code:

```
import nltk

groucho_grammar = nltk.CFG.fromstring("""

S -> NP VP

PP -> P NP

NP -> Det N | Det N PP | 'I'

VP -> V NP | VP PP

Det -> 'an' | 'my'

N -> 'elephant' | 'pajamas'

V -> 'shot'

P -> 'in'

""")
```

Once we run the cells above, we'll have created a CFG, which we can then feed into a parser object. For this lesson, we'll use NLTK's ChartParser, which we can find in nltk.ChartParser().

In the cell below, create a parser by passing in groucho_grammar.

```
parser = nltk.ChartParser(groucho_grammar)
```

Now that we have a parser, we can use it to parse sentences by passing in a tokenized sentence to the parser's .parse() method. This will return 0 or more Parse Trees for that sentence. If the

sentence cannot be parsed, it will return no trees. If there is more than one grammatically valid way (according to the rules defined in our CFG) to parse the sentence, then it will return them all.

Run the cell below to complete the example from the NLTK book:

```
sent = ['I', 'shot', 'an', 'elephant', 'in', 'my', 'pajamas']
for tree in parser.parse(sent):
    print(tree)
```

```
(S
  (NP I)
  (VP
    (VP (V shot) (NP (Det an) (N elephant)))
    (PP (P in) (NP (Det my) (N pajamas)))))
(S
  (NP I)
  (VP
    (V shot)
    (NP (Det an) (N elephant) (PP (P in) (NP (Det my) (N pajamas))))))
```

Great! Now that we have a working example, let's create own own, to parse the first sentence from the Usain Bolt example!

To keep things simple, we've already labeled the words as their given parts of speech inside the grammar below. This way, we only need to focus on what sorts of rules we give the CFG to define the valid structures for Noun Phrases, Verb Phrases, and Prepositional Phrases.

Parsing a Sentence

Run the cells below. This will:

Create a sample grammar. The grammar can be broken down into two sections.

Rules defining valid compositions for Sentences (S), Prepositional Phrases (PP), Noun Phrases (NP), and Verb Phrases (VP). These are incomplete, and we'll need to modify them to get the sentences to parse correctly.

Mappings that define the POS tag for each word we'll be working with, as seen from the line Det -> and all the lines after it. These will be provided for you at each step, so that we only have to worry about the rules.

Cell 2 defines the sentences we'll be working with as strings.

Cell 3 creates a tokenized version of each sentence.

Cell 4 creates a new parser with our grammar CFG.

Cell 5 tries to parse the tokenized version of the sentence.

Run these cells now.

```
grammar = nltk.CFG.fromstring("""
S -> NP VP
PP -> P NP
NP -> Det N | Det N PP |
VP -> V NP | VP PP
Det -> 'the'
Adj -> '100m'
N -> 'usain_bolt' | 'record' |
V -> 'broke'
P ->
""")
from nltk import word_tokenize
sent = 'usain_bolt broke the 100m record'
tokenized_sent = word_tokenize(sent)
parser = nltk.ChartParser(grammar)
for tree in parser.parse(tokenized_sent):
    print(tree)
```

This means our grammar should now look like:

```
grammar = nltk.CFG.fromstring("""
S -> NP VP
PP -> P NP
NP -> Det N PP | N | Det NP | Adj NP
VP -> V NP | VP PP
Det -> 'the'
Adj -> '100m'
N -> 'usain_bolt' | 'record' |
V -> 'broke'
P ->
""")
```

Let's try reloading our grammar, and reparsing our sentence.

In the cell below:

Modify and recreate our CFG using our updated rules.

Recreate our parser with the updated rules.

Try to parse tokenized_sent1 and see if we get any output.

```
grammar = nltk.CFG.fromstring("""
S -> NP VP
PP -> P NP
NP -> Det N PP | N | Det NP | Adj NP
VP -> V NP | VP PP
Det -> 'the'
Adj -> '100m'
N -> 'usain_bolt' | 'record' |
```

```
V -> 'broke'

P ->

""")

parser = nltk.ChartParser(grammar)

for tree in parser.parse(tokenized_sent):

    print(tree)
```

Conclusion:

Thus, we have studied what is meant by Parsing and context free grammars. We have also implemented them in Python language.