L3: Syntactic Parsing

Frederik Evenepoel & Piotr Bonar - Pair K

22/03/2025

Graded exercise 1:

The aim of this exercise is to tokenize and compute the Part-of-Speech tags for the given sentences:

1. *"The Jamaica Observer reported that Usain Bolt broke the 100m record."*
2. *"While hunting in Africa, I shot an elephant in my pajamas. How an elephant got into my pajamas I'll never know."*

The output of the code is as follows:

Sentence 1:

Tokenized Output:

['The', 'Jamaica', 'Observer', 'reported', 'that', 'Usain', 'Bolt', 'broke', 'the', '100m', 'record', '.']

POS tags:

[('The', 'DT'), ('Jamaica', 'NNP'), ('Observer', 'NNP'), ('reported', 'VBD'), ('that', 'DT'),

('Usain', 'NNP'), ('Bolt', 'NNP'), ('broke', 'VBD'), ('the', 'DT'), ('100m', 'CD'), ('record', 'NN'), ('.', '.')]

Sentence 2:

Tokenized Output:

['While', 'hunting', 'in', 'Africa', ',', 'I', 'shot', 'an', 'elephant', 'in', 'my', 'pajamas', '.',

'How', 'an', 'elephant', 'got', 'into', 'my', 'pajamas', 'I', "'ll", 'never', 'know', '.']

POS tags:

[('While', 'IN'), ('hunting', 'VBG'), ('in', 'IN'), ('Africa', 'NNP'), (',', ','), ('I', 'PRP'),

('shot', 'VBP'), ('an', 'DT'), ('elephant', 'NN'), ('in', 'IN'), ('my', 'PRP$'), ('pajamas', 'NN'), ('.', '.'),

('How', 'WRB'), ('an', 'DT'), ('elephant', 'JJ'), ('got', 'VBD'), ('into', 'IN'), ('my', 'PRP$'), ('pajamas', 'NN'),

('I', 'PRP'), ("'ll", 'MD'), ('never', 'RB'), ('know', 'VB'), ('.', '.')]

We can see that 'elephant' was tagged as JJ (adjective), which is incorrect. It should be NN. Also, 'shot' is in the past tense, so it should be tagged as VBD instead of VBP.

Implementation Code

```
def exercise_1():
    print("\n=== Exercise 1: Tokenization and PoS Tagging ===")
    sentences = [
        "The Jamaica Observer reported that Usain Bolt broke the 100m record.",
        "While hunting in Africa, I shot an elephant in my pajamas. How an elephant got into my pajamas I'll never know."
    ]
```

```
for sentence in sentences:
    tokens = nltk.word_tokenize(sentence)
    pos_tags = nltk.pos_tag(tokens)

    print("\nSentence:", sentence)
    print("Tokens:", tokens)
    print("PoS Tags:", pos_tags)
```

Graded exercise 2:

The goal is to create a CFG grammar that can correctly parse the sentences in *sentences*. The program will loop through each sentence, generates parse trees, and detects any possible ambiguities. We start by updating the basic grammar components such as nouns, determiners, verbs, and prepositions. Then, we build the parsing rules step by step using a bottom-up approach. The final grammar is shown in code below.

Sentence 1:

For the first sentence, *"John saw a man with my telescope"*, the systems return two parse trees, this means that there is an ambiguity. In the first parse trees (Figure 1), the prepositional phrase is attached to the verb phrase. This suggests that *John used a telescope to see the man*. In the second parse tree (Figure 2), the PP is attached within the noun phrase, modifying "man". This means that *the man John saw was carrying a telescope*.

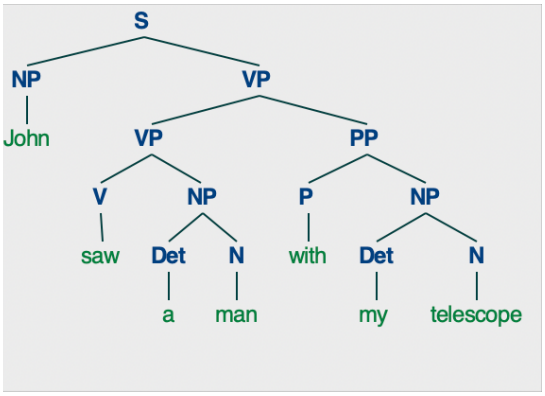

Figure 1: Parse tree 1 (Sentence 1)

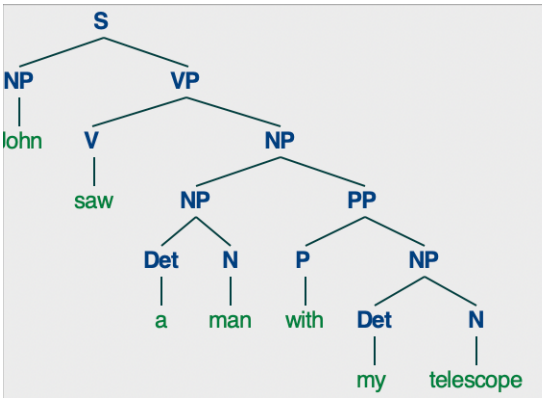

Figure 2: Parse tree 2 (Sentence 1)

Sentence 2:

For the second sentence: "*Alex kissed the dog*", there is only a single parse tree (Figure 3), meaning there is no ambiguity attached to this sentence.

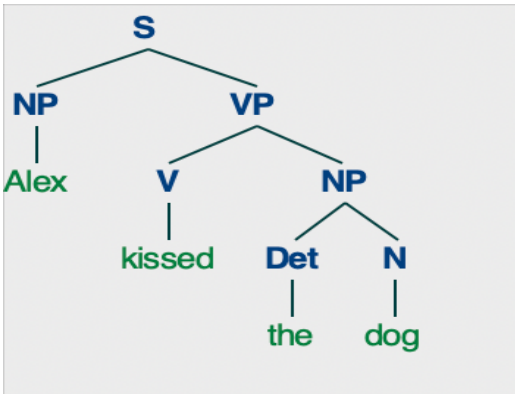

Figure 3: Parse tree 1 (Sentence 2)

Sentence 3:

For the third sentence, *"The man with the telescope ate a sandwich in the park"*, there are two different parse trees, indicating ambiguity. The ambiguity in the sentence arises from the attachment of the prepositional phrase "in the park." In the first parse tree (Figure 4), "in the park" modifies the verb phrase "ate a sandwich," meaning the eating happened in the park, while in the second parse tree (Figure 5), it modifies "a sandwich," implying that the sandwich itself was located in the park.

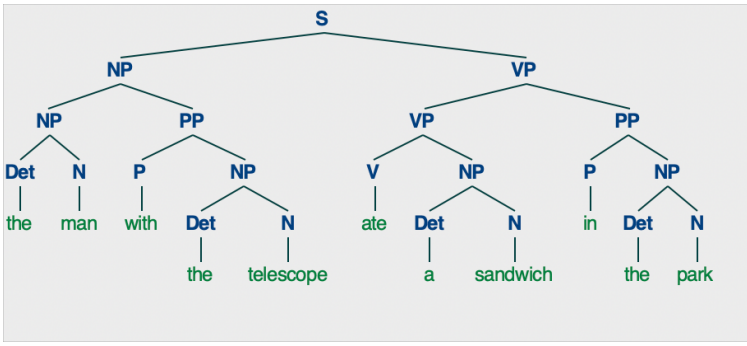

Figure 4: Parse tree 1 (Sentence 3)

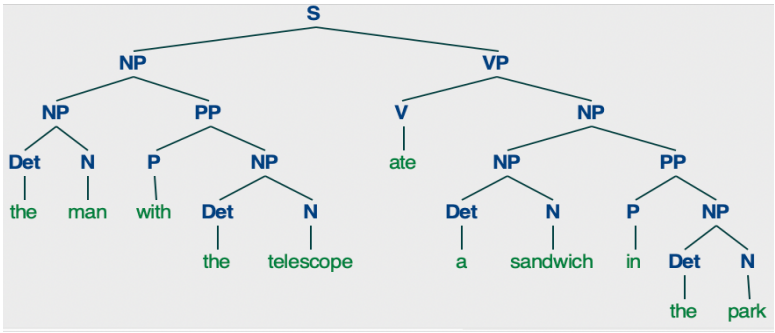

Figure 5: Parse tree 2 (Sentence 3)

Implementation Code

```
def exercise_2():
    """Parse sentences using a CFG grammar."""
    print("\n=== Exercise 2: Parsing with CFG ===")
   # Define a CFG grammar
    grammar = CFG.fromstring("""
    S -> NP VP
    PP -> P NP
    NP -> Det N | NP PP | 'John' | 'Alex'
    VP -> V NP | VP PP
    Det -> 'a' | 'my'| 'the'
    N -> 'man' | 'telescope'| 'dog'| 'sandwich'| 'park'
    V -> 'saw'| 'kissed'| 'ate'
    P -> 'in'| 'with'
    """)
    print("START grammar:")
    print("PRODUCTIONS grammar:", grammar.start(), grammar.productions())

    sentences = [
        "John saw a man with my telescope",
        "Alex kissed the dog",
        "the man with the telescope ate a sandwich in the park"
    ]
    parser = ChartParser(grammar)
    for sentence in sentences:
        print(f"\nParsing: {sentence}")
        tokens = nltk.word_tokenize(sentence)
        trees = list(parser.parse(tokens))
        if not trees:
            print("No valid parse tree found.")
            continue
        for tree in trees:
            print(tree)
            tree.pretty_print()
        if len(trees) > 1:
            print("Ambiguity detected")
```

Graded exercise 3:

An example of a sentence that is syntactically correct in English but cannot be parsed by the **initial** grammar is: “I shot”. The given grammar requires every VP to follow the rule VP → V NP or VP → VP PP, meaning the verb "shot" must have a NP as its object. Since "I" is a valid NP but "shot" alone does not form a valid VP under these rules, the grammar cannot generate the sentence “I shot.”

Graded exercise 4:

We can integrate a spelling corrector, like in the previous exercise, to handle spelling errors. The sentence first goes through a pre-processing step where errors are detected and corrected using a trained language model. The corrected sentence is then passed to the CFG parser, and if it still cannot be parsed, a feedback mechanism can possibly suggest alternative corrections or request user input.

Graded exercise 5:

The goal of this exercise is to analyse what different trace values mean and observe the probability assigned to different parse trees for the sentence "*I saw the man with a telescope.*" By modifying the grammar probabilities, we can force one parse tree to be more likely than the other. We can then compare the changes in probability and parse tree structure.

A trace level of 0 will generate no tracing output, while higher trace levels produce increasingly more detailed output. Essentially, increasing the trace level gives more insight into how the parser constructs the most probable parse tree based on the probabilities in the PCFG.

The parse tree of initial grammar (given in the assignment) is given below (Figure 6), the parsing probability of this parse tree is p=0.000104081. The parse tree shows that "with a telescope" is attached to "the man" within the NP, meaning it describes the man rather than the action of seeing. This indicates that the correct interpretation of this sentence is "*I saw a man who had a telescope*," rather than "*I used a telescope to see the man.*"

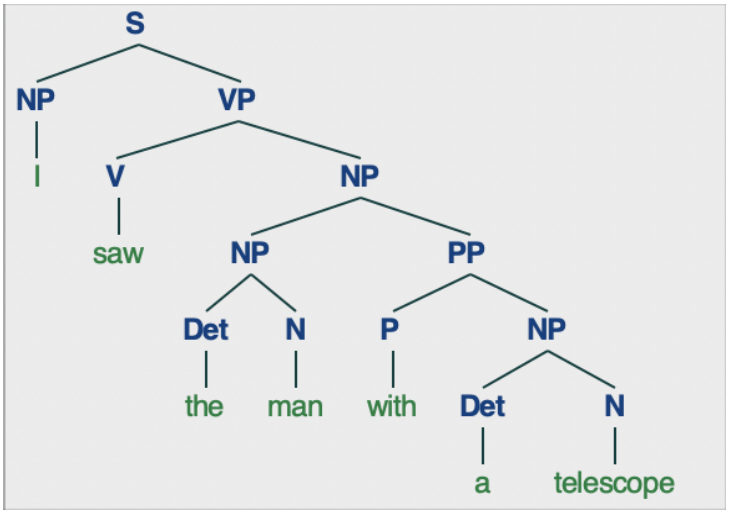

Figure 6: Parse tree 1 PCFG

To force the parser to prefer the alternative parse tree where "with a telescope" modifies "saw" instead of "the man," we adjusted the grammar probabilities (which resulted in the parse tree given in Figure 7). We decreased the probability of NP → NP PP from 0.25 to 0.1, making it less likely that the PP attaches to the noun. We increased the probability of VP → VP PP from 0.1 to 0.5, encouraging the PP to modify the verb instead. Additionally, we balanced the probabilities of VP → V NP and VP → V to prevent direct verb-object attachment from being overly dominant. This balance allows the parser to favour interpreting "saw with a telescope" over "the man with a telescope.", while maintaining the correct probability distribution. In every case we have to make sure that probabilities sum up to 1.

As a result, the parser now prefers the interpretation "I used a telescope to see the man" rather than "I saw a man who had a telescope.". The probability of this tree is p=0.000171288. The parse tree is given below (Figure 7).

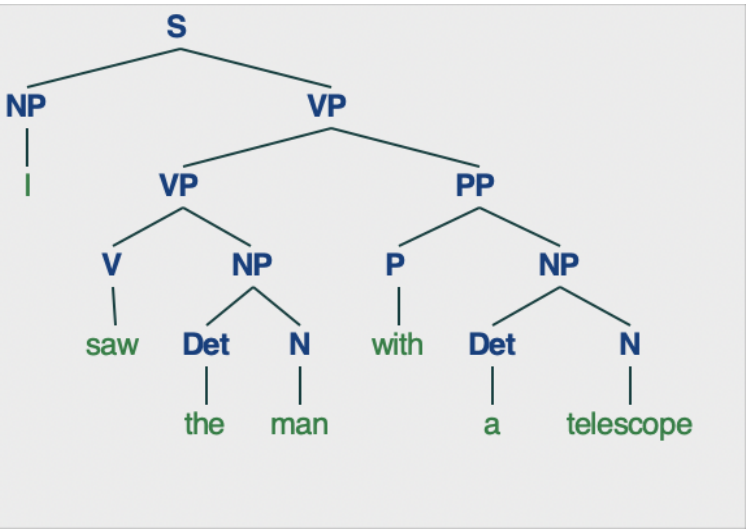

Figure 7: Parse tree 2 PCFG

Implementation Code

```
def exercise_5_1():
    pcfg1 = PCFG.fromstring("""
    S -> NP VP [1.0]
    NP -> Det N [0.5] | NP PP [0.25] | 'John' [0.1] | 'I' [0.15]
    Det -> 'the' [0.8] | 'a' [0.2]
    N -> 'man' [0.5] | 'telescope' [0.5]
    VP -> VP PP [0.1] | V NP [0.7] | V [0.2]
    V -> 'ate' [0.35] | 'saw' [0.65]
    PP -> P NP [1.0]
    P -> 'with' [0.61] | 'in' [0.39]
    """)

    print(pcfg1)
    text = "I saw the man with a telescope"
    text_tokens = nltk.word_tokenize(text)
    viterbi_parser = nltk.ViterbiParser(pcfg1, trace=2)
    trees = viterbi_parser.parse(text_tokens)
    for tree in trees:
        tree.pretty_print()
        print(tree)
        tree.draw()

def exercise_5_2():
    pcfg1 = PCFG.fromstring("""
    S -> NP VP [1.0]
    NP -> Det N [0.6] | NP PP [0.1] | 'John' [0.15] | 'I' [0.15]
    Det -> 'the' [0.8] | 'a' [0.2]
    N -> 'man' [0.5] | 'telescope' [0.5]
    VP -> VP PP [0.5] | V NP [0.4] | V [0.1]
    V -> 'ate' [0.35] | 'saw' [0.65]
    PP -> P NP [1.0]
    P -> 'with' [0.61] | 'in' [0.39]
    """)

    print("Modified PCFG Grammar:\n", pcfg1)
```

```
text = "I saw the man with a telescope"
text_tokens = nltk.word_tokenize(text)

viterbi_parser = nltk.ViterbiParser(pcfg1)
trees = list(viterbi_parser.parse(text_tokens))

for tree in trees:
    print("\nModified Parse Tree:")
    tree.pretty_print()
    print(tree)
    tree.draw()
```

Graded exercise 6:

The exercise code uses a PCFG induced from the Penn Treebank to parse the sentence 'the boy jumps over the board' using the Viterbi Parser. The trace parameter again controls the verbosity of the parsing process. Setting trace=1 provides a summary of the parser's decision-making, while trace=2 gives a more detailed step-by-step breakdown of probability calculations. The probability score of this parse tree is 1.25001e-20. The probability is low because this sentence structure is either rare or not well-represented in the training data. (Figure 8).

Interpretation tree:

- S (Sentence) is the root node.
- NP-SBJ (Noun Phrase – Subject) contains ‘the boy’.
- NP-PRD (Noun Phrase – Predicate) contains the verb ‘jumps’ and a prepositional phrase.
- PP is ‘over the board’ where ‘over’ is a preposition and the noun phrase is ‘the board’.

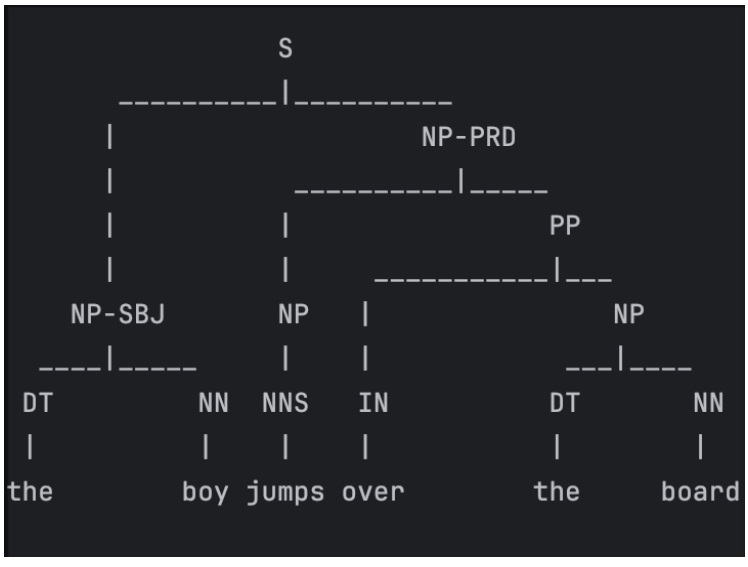

Figure 8: Parse tree exercise 5

Implementation Code

```
def exercise_7():
    productions = []
    S = nltk.Nonterminal('S')
    for f in treebank.fileids():
        for tree in treebank.parsed_sents(f):
            productions += tree.productions()
    grammar = nltk.induce_pcfg(S, productions)
    for p in grammar.productions()[1:25]:
        print(p)
    myparser = nltk.ViterbiParser(grammar, 1)
    text = "the boy jumps over the board"
    mytokens = nltk.word_tokenize(text)
    myparsing, = myparser.parse(mytokens)
    print(myparsing)
    # Parse the sentence
    trees = list(myparser.parse(mytokens))
    if trees:
        print("\nParsed Tree:\n")
        trees[0].pretty_print()
        tree.draw()
```