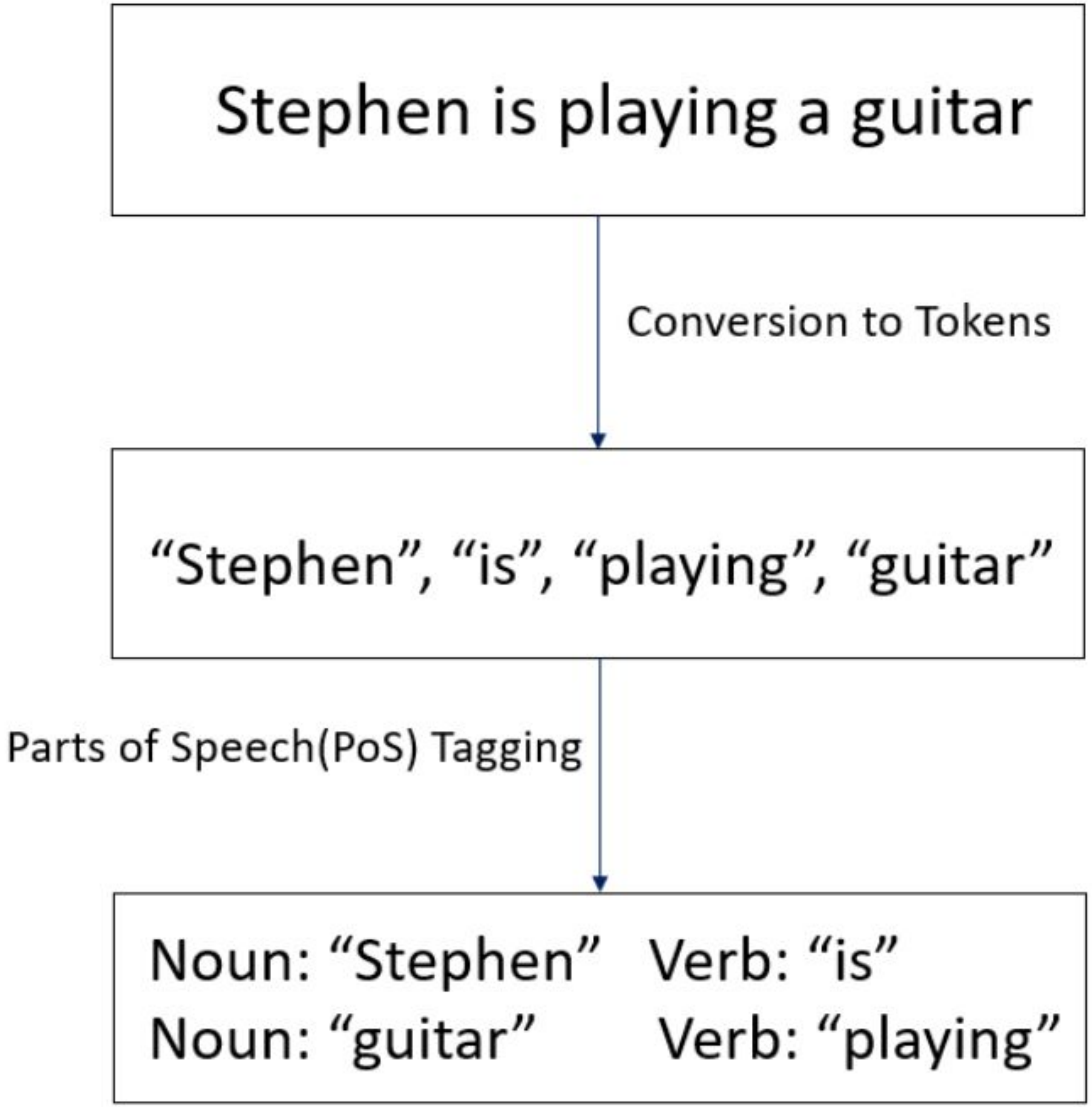
Unit 2

Contents

What is Parsing in NLP?

- Parsing is the process of analyzing a sentence, breaking it down into smaller components, and identifying the grammatical structure of the sentence.
- It is a crucial component of NLP and helps machines understand human language.
- In other words, parsing is the process of analyzing a sentence's syntax and its underlying structure to extract meaning from it.


Stephen is playing a guitar
Conversion to Tokens
“Stephen”, “is”, “playing”, “guitar”
Parts of Speech(PoS) Tagging
Noun: “Stephen” Verb: “is”
Noun: “guitar” Verb: “playing”

- As per the above example, it is evident that parsing a natural language sentence involves ***analyzing the input sentence by breaking it down into its grammatical constituents, identifying the parts of speech, and syntactic relations***.

- It involves analyzing the text to determine the roles of specific words, such as nouns, verbs, and adjectives, as well as their interrelationships.

- Parsers expose the structure of a sentence by constructing parse trees or dependency trees that illustrate the hierarchical and syntactic relationships between words.

- This essential NLP stage is crucial for a variety of language understanding tasks, which allow machines to extract meaning, provide coherent answers, and execute tasks such as machine translation, sentiment analysis, and information extraction.

Types of Parsing in NLP

Syntactic Parsing

- Syntactic parsing deals with **a sentence's grammatical structure**. It involves **looking at the sentence to determine parts of speech, sentence boundaries**, and word relationships. The two most common approaches included are as follows:

Constituency Parsing: Constituency Parsing **builds parse trees** that break down a sentence into its constituents, such as noun phrases and verb phrases.

- It displays a sentence's hierarchical structure, demonstrating how words are arranged into bigger grammatical units.

Dependency Parsing: Dependency parsing depicts grammatical links between words by **constructing a tree structure** in which **each word in the sentence is dependent on another.**

- It is frequently used in tasks such as information extraction and machine translation because it focuses on word relationships such as subject-verb-object relations.

Semantic Parsing

- Semantic parsing goes beyond syntactic structure to **extract a sentence's meaning or semantics.**
- It attempts to **understand the roles of words in the context** of a certain task and how they interact with one another.
- Semantic parsing is utilized in a variety of NLP applications, such as question answering, knowledge base populating, and text understanding.
- It is essential for activities requiring the extraction of actionable information from text.

Parsing Techniques in NLP

- Top-down Parse Tree
- Bottom-up Parse Tree

Parsing Algorithms

- **Recursive Descent Parser**
  - A **top-down** parser that **iteratively breaks down the highest-level grammar rule into subrules** is known as a recursive descent parser.
  - It is frequently implemented as a set of recursive functions, each of which handles a certain grammatical rule.
  - This style of parser is frequently employed **in hand-crafted parsers** for simple programming languages and domain-specific languages.

- Consider a simple grammar for arithmetic expressions:

$$E \longrightarrow T + E \mid T$$

$$T \longrightarrow int \mid (E)$$

- Input String:-3 + (4 + 5)
- Tokens from the Lexer:-[int: 3, +, (, int: 4, +, int: 5,)]
- Parser functions

```
def parse_E():
    parse_T()  # Parse a Term (T)
    if next_token() == '+':  # Check if the next token is '+'
        consume_token('+')  # Consume '+'
        parse_E()  # Parse another Expression (E)
```

```
def parse_T():
    if next_token().type == 'int':  # Check if the token is an integer
        consume_token('int')  # Consume the integer token
    elif next_token() == '(':
        consume_token('(')  # Consume '('
        parse_E()  # Parse an Expression (E) inside parentheses
        consume_token(')')  # Consume ')'
    else:
        raise SyntaxError("Expected int or '('")
```

The parse tree generated for the input 3 + (4 + 5)

```
     E
    /|\
   T + E
   |   |
  int  T
   |   |
   3  (E)
       |
       T + E
       |   |
      int  T
       |   |
       4  int
           |
           5
```

- **Shift-Reduce Parser**
  - A shift-reduce parser is a sort of **bottom-up parser** that starts with the input and builds a parse tree by performing a series of shift (**transfer data to the stack**) and reduction **(apply grammar rules)** operations.
  - Shift-reduce parsers are **used in programming language parsing** and are frequently used with LR (Left-to-right, Rightmost derivation) or LALR (Look-Ahead LR) parsing techniques

- **Chart Parser**
  - A chart parser is a sort of parsing algorithm that efficiently **parses words by using dynamic programming** and chart data structures.
  - To reduce unnecessary work, it **stores and reuses intermediate parsing results.**
  - Early parser is a type of chart parser that is commonly utilized for **parsing context-free grammars.**
- **Regexp Parser**
  - A **regexp (regular expression) parser is used to match patterns** and **extract text**. It scans a larger text or document for substrings that match a specific regular expression pattern.
  - Text processing and information retrieval tasks make extensive use of regexp parsers.

Probabilistic Parsing

- **Essential for NLP**:
  - Human languages are inherently ambiguous, requiring probabilistic methods for disambiguation.
- **Examples**:
  - **PCFG**: Assigns probabilities to each production rule, allowing the parser to choose the most likely parse tree.
  - **Dependency Parsing**: Probabilistic dependency parsers are widely used in modern NLP.
- **Modern Implementations**:
  - Machine learning models, such as Conditional Random Fields (CRFs) and neural networks, are used to predict probabilities of parse trees.

Dependency Parsing

- **Core NLP Parsing Approach**:
  - Focuses on identifying grammatical relationships (e.g., subject-verb, object-verb) rather than building full parse trees.
  - Dependency parsers **are faster and often more suitable** for practical NLP tasks like information extraction or machine translation.

- **Algorithms**:
  - **Transition-Based Parsing**: Combines machine learning and shift-reduce parsing for efficiency.
  - **Graph-Based Parsing**: Models dependency parsing as finding the maximum spanning tree in a graph.

Statistical and Neural Parsing

- **Modern NLP Parsing Techniques**:
  - Leverages machine learning to improve parsing accuracy.
  - **Transition-based parsers** like **MaltParser** and **graph-based parsers** like **Stanford Dependency Parser** are common.
- **Neural Models**:
  - Neural networks, especially **transformers**, power state-of-the-art parsers.
  - Example: **BERT-based parsers** that combine syntactic and semantic parsing.

Probabilistic context-free grammars (PCFGs)

- A Probabilistic Context-Free Grammar **(PCFG)** is an **extension** of a **Context-Free Grammar (CFG)** that **assigns probabilities to its production rules**.
- This allows the grammar to handle **ambiguity** by ranking possible parse trees and selecting the most likely one.

1. **Context-Free Grammar (CFG)**:
    - A CFG consists of:
        - **Non-Terminals (N)**: Symbols representing categories or constituents (e.g., S, NP, VP).
        - **Terminals (T)**: Symbols representing actual words in the language.
        - **Start Symbol (S)**: The initial non-terminal symbol.
        - **Production Rules (P)**: Rules of the form $A \rightarrow \alpha$, where A is a non-terminal and α is a string of terminals and/or non-terminals.

- $S \longrightarrow NP\ VP$
- $NP \longrightarrow Det\ N$
- $VP \longrightarrow V\ NP$
- Det $\longrightarrow$ the | a
- N $\longrightarrow$ cat | dog
- V $\longrightarrow$ sees | pets

Probabilistic Extension:

- In a PCFG, **each production rule** is assigned a **probability**.
- The probability of a production rule $A \rightarrow \alpha$ is denoted as $P(A \rightarrow \alpha)$.
- The probabilities for all productions of a given non-terminal must sum to 1

$$\sum P(A \rightarrow \alpha) = 1 \quad \text{(for all } \alpha \text{ derived from A)}$$

Example PCFG

- $S \longrightarrow NP\ VP$ [P = 1.0]
- $NP \longrightarrow Det\ N$ [P = 0.8]
- $NP \longrightarrow N$ [P = 0.2]
- $VP \longrightarrow V\ NP$ [P = 0.9]
- $VP \longrightarrow V$ [P = 0.1]
- Det $\longrightarrow$ the [P = 0.6]
- Det $\longrightarrow$ a [P = 0.4]
- N $\longrightarrow$ cat [P = 0.5]
- N $\longrightarrow$ dog [P = 0.5]
- V $\longrightarrow$ sees [P = 0.7]
- V $\longrightarrow$ pets [P = 0.3]

- **Parsing with a PCFG**

1. **Ambiguity in Parsing**:
   - For a given sentence, multiple parse trees may be possible.
   - A PCFG assigns probabilities to these parse trees based on the production rules used.
2. **Tree Probability**:

   The probability of a parse tree is the product of the probabilities of all production rules used in that tree.

$$P(\text{Tree}) = \Pi\ P(A \rightarrow \alpha)$$

Most Likely Parse:

- The **most probable parse tree** is the one with the highest P(Tree).

Algorithm: Parsing with Probabilities

The algorithm works by:

1. **Constructing all possible parse trees** according to the grammar rules.
2. **Calculating the probability for each parse tree**:
    - Multiply the probabilities of all rules used in the tree.
3. **Selecting the most probable tree**.

Input Sentence:

the cat sees a dog

Tokens:
- the (Det)
- cat (N)
- sees (V)
- a (Det)
- dog (N)

Step 1: Generate Parse Trees

- **Parse Tree 1:**

(S (NP (Det the) (N cat)) (VP (V sees) (NP (Det a) (N dog))))

- NP → Det N
- VP → V NP
- NP → Det N

- **Parse Tree 2:**

(S (NP (Det the) (N cat)) (VP (V sees)))

- NP → Det N
- VP → V

- **Step 2: Calculate Probabilities**
- **For Tree 1:**

P(Tree 1) = P(S → NP VP) * P(NP → Det N) * P(Det → the) * P(N → cat) *

P(VP → V NP) * P(V → sees) * P(NP → Det N) *

P(Det → a) * P(N → dog)

P(Tree 1) = 1.0 * 0.8 * 0.6 * 0.5 * 0.9 * 0.7 * 0.8 * 0.4 * 0.5

= 0.0486

- For **Tree 2**:

$$P(\text{Tree 2}) = P(S \longrightarrow NP\ VP) * P(NP \longrightarrow Det\ N) * P(Det \longrightarrow the) * P(N \longrightarrow cat) * P(VP \longrightarrow V) * P(V \longrightarrow sees)$$

$$P(\text{Tree 2}) = 1.0 * 0.8 * 0.6 * 0.5 * 0.1 * 0.7 = 0.0168$$

Step 3: Compare Probabilities

- Tree 1: P(Tree 1) = 0.0486
- Tree 2: P(Tree 2) = 0.0168

Most Probable Parse Tree: Tree 1

How the Algorithm Works

1. **Tokenize the Sentence**: Break the input into parts of speech (Det, N, V).
2. **Apply Grammar Rules**: Start from the start symbol S and recursively expand using grammar rules.
3. **Assign Probabilities**: Multiply the probabilities of the rules used to construct the tree.
4. **Rank Trees**: Compare probabilities of all valid parse trees.
5. **Select the Most Probable Tree**: The tree with the highest probability is chosen as the final parse.