**Part-of-speech tagging**

Part-of-speech tagging, often abbreviated as POS tagging or simply tagging, is a natural language processing (NLP) task that involves assigning a specific grammatical category, or part-of-speech, to each word in a sentence. The goal is to automatically analyse and label words in a text based on their syntactic roles within the sentence. POS tagging is a crucial step in various NLP applications, including information retrieval, machine translation, and text-to-speech synthesis.

**1. Rule-based POS Tagging**

One of the oldest techniques of tagging is rule-based POS tagging. Rule-based taggers use dictionary or lexicon for getting possible tags for tagging each word. If the word has more than one possible tag, then rule-based taggers use hand-written rules to identify the correct tag. Disambiguation can also be performed in rule-based tagging by analyzing the linguistic features of a word along with its preceding as well as following words. For example, suppose if the preceding word of a word is article then word must be a noun.

As the name suggests, all such kind of information in rule-based POS tagging is coded in the form of rules. These rules may be either −

* Context-pattern rules
* Or, as Regular expression compiled into finite-state automata, intersected with lexically ambiguous sentence representation.

We can also understand Rule-based POS tagging by its two-stage architecture −

* **First stage** − In the first stage, it uses a dictionary to assign each word a list of potential parts-of-speech.
* **Second stage** − In the second stage, it uses large lists of hand-written disambiguation rules to sort down the list to a single part-of-speech for each word.

Properties of Rule-Based POS Tagging

Rule-based POS taggers possess the following properties −

* These taggers are knowledge-driven taggers.
* The rules in Rule-based POS tagging are built manually.
* The information is coded in the form of rules.
* We have some limited number of rules approximately around 1000.
* Smoothing and language modeling is defined explicitly in rule-based taggers.

Example Rule: Adjective-Noun Combination

Rule: If a word is preceded by an adjective, it is likely to be a noun.

Example:

Sentence: "The big house is beautiful."

Rule-based POS Tagging:

"The" - Determiner

"big" - Adjective

"house" - Noun

"is" - Verb

"beautiful" - Adjective

In this example, the rule suggests that if a word follows an adjective, it is more likely to be a noun. As per the rule, "big" is tagged as an adjective, and "house" is tagged as a noun.

**Stochastic POS Tagging**

Another technique of tagging is Stochastic POS Tagging. Now, the question that arises here is which model can be stochastic. The model that includes frequency or probability (statistics) can be called stochastic. Any number of different approaches to the problem of part-of-speech tagging can be referred to as stochastic tagger.

The simplest stochastic tagger applies the following approaches for POS tagging −

**Word Frequency Approach**

In this approach, the stochastic taggers disambiguate the words based on the probability that a word occurs with a particular tag. We can also say that the tag encountered most frequently with the word in the training set is the one assigned to an ambiguous instance of that word. The main issue with this approach is that it may yield inadmissible sequence of tags.

**Tag Sequence Probabilities**

It is another approach of stochastic tagging, where the tagger calculates the probability of a given sequence of tags occurring. It is also called n-gram approach. It is called so because the best tag for a given word is determined by the probability at which it occurs with the n previous tags.

Properties of Stochastic POST Tagging

* This POS tagging is based on the probability of tag occurring.
* It requires training corpus
* There would be no probability for the words that do not exist in the corpus.
* It uses different testing corpus (other than training corpus).
* It is the simplest POS tagging because it chooses most frequent tags associated with a word in training corpus.

| **Top-Down Parsing** | **Bottom-Up Parsing** |
| --- | --- |
| It is a parsing strategy that first looks at the highest level of the parse tree and works down the parse tree by using the rules of grammar. | It is a parsing strategy that first looks at the lowest level of the parse tree and works up the parse tree by using the rules of grammar. |
| Top-down parsing attempts to find the left most derivations for an input string. | Bottom-up parsing can be defined as an attempt to reduce the input string to the start symbol of a grammar. |
| In this parsing technique we start parsing from the top (start symbol of parse tree) to down (the leaf node of parse tree) in a top-down manner. | In this parsing technique we start parsing from the bottom (leaf node of the parse tree) to up (the start symbol of the parse tree) in a bottom-up manner. |
| This parsing technique uses Left Most Derivation. | This parsing technique uses Right Most Derivation. |
| The main leftmost decision is to select what production rule to use in order to construct the string. | The main decision is to select when to use a production rule to reduce the string to get the starting symbol. |
|  |  |
| Simple to implement | |  | | --- | | Complex to implement | |  | |
| It is less powerful compared to Bottom-up parsing. | |  | | --- | | It is more powerful compared to Top-down parsing. | |  | |
| Example: Recursive Descent parser. | Example: Shift Reduce parser. |
|  |  |

**CYK (Cocke–Younger–Kasami) Parser:**

1. Description:

- Algorithm Type: CYK is a dynamic programming-based parsing algorithm for context-free grammars.

- Parsing Approach: It builds parse trees bottom-up, considering all possible combinations of sub-phrases.

- Algorithm Steps: CYK operates by filling a dynamic programming table with possible non-terminals for sub-phrases in the input sentence.

- Parsing Efficiency: It can efficiently handle ambiguous grammars and multiple parse trees for a given sentence.

Based on Dynamic Programming approach: Build solutions compositionally from sub solutions

It uses the grammar directly

CYK is capable of handling ambiguity in languages, providing all possible parse trees for an ambiguous sentence.

- The dynamic programming approach helps explore and evaluate various combinations efficiently.

2. Time and Space Complexity:

- Time Complexity: O(n^3), where n is the length of the input sentence.

- Space Complexity: O(n^2), due to the dynamic programming table.

3. Advantages:

- Efficiently handles ambiguous grammars and can provide multiple parse trees for a sentence.

- Suitable for parsing natural language sentences with complex structures.

- Well-suited for languages with free word order.

4. Disadvantages:

- Sensitivity to grammar size; the cubic time complexity can be a drawback for longer sentences.

- May produce a large number of parse trees, requiring additional disambiguation strategies.

- Limited in handling semantic or contextual information.

5. Applications:

- Widely used in natural language processing for syntax analysis and parsing.

- Applied in the analysis of programming languages, especially for compilers.

- Used in tasks where a comprehensive understanding of the syntactic structure is required.

7. Grammatical Constraints:

- The algorithm is designed for context-free grammars, and its effectiveness depends on the grammar's structure and ambiguity level.

- It is particularly well-suited for languages with relatively simple syntactic structures.

8. Limitations:

- The cubic time complexity can be a limiting factor for longer sentences.

- Producing a large number of parse trees may require additional post-processing steps for disambiguation.

- May not capture semantic or contextual information as effectively as more advanced models.

The CYK parser is a fundamental tool in natural language processing and syntactic analysis, offering valuable insights into the grammatical structure of sentences. Its efficiency in handling ambiguity and providing multiple parse trees makes it applicable in various language-related tasks.

**PCFG (Probabilistic Context-Free Grammar):**

1. Description:

- Grammar Extension: PCFG is an extension of context-free grammars (CFG) where each production rule is associated with a probability.

- Probability Assignment: The probabilities represent the likelihood of choosing a particular production rule when expanding a non-terminal.

- Ambiguity Handling: PCFGs help in disambiguating sentences by favoring more probable parse trees.

2. Advantages:

- Probabilistic Modeling: Allows for probabilistic modeling of syntactic structures, aiding in disambiguation.

- Flexible Modeling: Can capture variations in sentence structures and preferences for certain parse trees.

- Useful in Language Modeling: PCFGs are widely used in statistical language modeling and parsing.

3. Disadvantages:

- Complex Training: Training PCFGs requires a large and diverse dataset for accurate probability estimation.

- Limited Semantics: Primarily focuses on syntax; may not capture semantic or contextual information.

- Difficulty in Handling Ambiguity: While helpful in many cases, PCFGs may struggle with certain types of ambiguity.

5. Applications:

- Statistical Parsing: PCFGs are extensively used in statistical parsing to estimate the likelihood of different parse trees.

- Machine Translation: In applications where understanding the likelihood of different sentence structures is crucial.

- Syntax-Based Language Modeling: Commonly applied in language modeling tasks, especially for syntactic structure predictions.

7. Probabilistic Parsing:

- Probabilistic Decisions: During parsing, PCFGs make probabilistic decisions about which production rules to apply, considering the likelihood of each rule.

Probabilistic Context-Free Grammars are valuable tools in natural language processing, providing a probabilistic framework for syntactic analysis and aiding in disambiguating language structures. They find applications in various fields where capturing the likelihood of different syntactic structures is essential.

**Shift-Reduce Parser:**

1. Description:
   * Parsing Approach: Shift-reduce parsing is a bottom-up parsing method that builds a parse tree by shifting input tokens onto a stack and then reducing them based on predefined grammar rules.
   * Action Selection: At each step, the parser decides whether to shift the next input token onto the stack or reduce a portion of the stack using a grammar rule.
2. Shift and Reduce Actions:
   * Shift Action: Adds the next input token to the stack.
   * Reduce Action: Replaces a group of symbols on the stack with a non-terminal symbol, following a grammar rule.
3. Advantages:
   * Efficiency: Shift-reduce parsers are often more memory-efficient than other parsing methods.
   * Simplicity: Conceptually simpler to implement compared to some other parsing techniques.
   * Applicability: Suitable for a wide range of grammars, including ambiguous grammars.
4. Disadvantages:
   * Ambiguity Handling: May struggle with certain types of ambiguity in languages.
   * Limited Lookahead: The parser's decisions are often based on a limited lookahead, which may lead to conflicts.
   * Not Always Optimal: In some cases, it might not find the most globally optimal parse tree.
5. Applications:
   * Compiler Construction: Shift-reduce parsing is commonly used in the development of compilers for programming languages.
   * Syntactic Analysis: Applied in syntactic analysis tasks, including the parsing of sentences in natural language processing.
6. State Transition Diagram:
   * Parser States: Shift-reduce parsers can be represented by a state transition diagram, where each state corresponds to a configuration of the stack and input.
   * Transition Actions: Transitions between states are determined by shift and reduce actions.

Shift-reduce parsers are popular in compiler construction due to their efficiency and simplicity. They are versatile and applicable to a broad range of grammars, making them suitable for various syntactic analysis tasks in different domains.

**Early Parser:**

1. **Description:**
   * **Parsing Approach:** Early parser is a predictive, top-down parsing algorithm that employs a chart-based approach. It predicts and constructs possible parse tree structures.
   * **Chart-Based Parsing:** The parsing process involves filling a chart table with predicted and completed parse items.
   * **Non-Terminal Predictions:** Predicts non-terminal symbols based on the grammar rules.
2. **Time and Space Complexity:**
   * **Time Complexity:** O(n^3), where n is the length of the input sentence. The cubic time complexity arises from the chart-based approach.
   * **Space Complexity:** O(n^2), as the chart table has a size proportional to the square of the input length.
3. **Advantages:**
   * **Efficient Parsing:** Effective for grammars with a limited lookahead and is suitable for natural language processing.
   * **Dynamic Programming:** Uses dynamic programming principles to efficiently explore and fill the chart table.
   * **Flexibility:** Can be extended to handle ambiguous grammars.
4. **Disadvantages:**
   * **Sensitivity to Grammar Size:** Can be sensitive to the size of the grammar, and the cubic time complexity may become a limitation for longer sentences.
   * **Left Recursion:** Struggles with left-recursive grammars without appropriate modifications.
   * **Limited Semantic Information:** Primarily focuses on syntax, and capturing semantic or contextual information is challenging.
5. **Applications:**
   * **Natural Language Processing:** Applied in syntactic analysis for natural language processing tasks, such as parsing sentences.
   * **Compiler Construction:** Utilized in the development of compilers for programming languages.
   * **Syntax Analysis:** Commonly used in tasks where a comprehensive understanding of the syntactic structure is required.
6. **Chart Table Structure:**
   * **States and Items:** The chart table contains states and items representing the parsing progress.
   * **Predicted and Completed Items:** The table is filled with predicted and completed parse items based on the input and grammar.
7. **Handling Ambiguity:**
   * **Ambiguous Grammars:** Can handle ambiguous grammars by allowing multiple items in the chart table corresponding to different possible parse trees.
   * **Disambiguation Strategies:** Additional strategies may be required for disambiguating between multiple parse trees.

The Early parser is a valuable tool in natural language processing and compiler construction. Its chart-based approach and dynamic programming principles contribute to efficient parsing, especially for grammars with limited lookahead. However, considerations regarding grammar size and complexity should be taken into account.

**Predictive Parser:**

1. **Description:**
   * **Parsing Approach:** Predictive parser is a top-down parsing method that uses a recursive descent approach. It predicts and expands non-terminal symbols based on a set of production rules.
   * **LL Parsing:** Predictive parsers are often referred to as LL parsers because they process the input from Left to right and construct Leftmost derivations.
2. **Time and Space Complexity:**
   * **Time Complexity:** Typically linear, O(n), where n is the length of the input sentence.
   * **Space Complexity:** Linear, O(n), proportional to the length of the input sentence.
3. **Advantages:**
   * **Simplicity:** Conceptually simpler to implement compared to some other parsing techniques.
   * **Efficiency:** Linear time complexity makes it efficient for a broad range of grammars.
   * **LL(1) Grammar Handling:** Well-suited for LL(1) grammars, where the prediction is unambiguous.
4. **Disadvantages:**
   * **Limited Grammar Class:** Limited to LL(1) grammars, meaning it struggles with certain types of left-recursion and ambiguity.
   * **Left Recursion Handling:** Requires modifications to handle left-recursive grammars.
   * **Lookahead Limitation:** The parser's decisions are based on a single-token lookahead, which can be limiting.
5. **Applications:**
   * **Compiler Construction:** Commonly used in the development of compilers for programming languages.
   * **Syntax Analysis:** Applied in syntax analysis tasks, especially when the grammar is LL(1).
   * **Educational Tool:** Often used in educational settings to teach principles of parsing.
6. **Grammar Constraints:**
   * **LL(1) Requirement:** Effective for LL(1) grammars where the parsing decision is uniquely determined by the first input token.
   * **Parsing Table:** Relies on a predictive parsing table to determine the production rule to apply based on the current input and lookahead token.
7. **Handling Ambiguity:**
   * **Ambiguous Grammars:** Struggles with ambiguous grammars as the predictive nature may lead to conflicts.
   * **Conflict Resolution:** Conflict resolution strategies or modifications to the grammar may be necessary for ambiguous cases.
8. **Recursive Descent Implementation:**
   * **Parsing Functions:** Recursive descent parsers are implemented using parsing functions, each corresponding to a non-terminal symbol.
   * **Recursive Calls:** Recursive calls correspond to the expansion of non-terminals, following the production rules.

Predictive parsers offer a straightforward approach to parsing, especially for LL(1) grammars, and are commonly employed in educational contexts and smaller language projects. While limited in handling certain types of grammars, their simplicity makes them a valuable tool in certain parsing scenarios.

**c Describe open class words and closed class words in English with examples. [5M]**

In linguistics, words in a language can be categorized into two main classes: open class words and closed class words. These classes differ in terms of their characteristics, functions, and the ease with which new words can be added.

1. **Open Class Words:**
   * **Definition:** Open class words are lexical categories of words that readily accept new members. These classes are open to expansion, and new words can be added over time. Open class words play a significant role in expressing content and carrying meaning in a sentence.
   * **Examples:**
     + **Nouns:** Words that represent people, places, things, or ideas.
       - *Examples:* dog, city, happiness, computer
     + **Verbs:** Words that denote actions, processes, or states.
       - *Examples:* run, eat, sleep, think
     + **Adjectives:** Words that describe or modify nouns.
       - *Examples:* tall, blue, happy, intelligent
     + **Adverbs:** Words that modify verbs, adjectives, or other adverbs.
       - *Examples:* quickly, very, well, slowly
2. **Closed Class Words:**
   * **Definition:** Closed class words are grammatical categories of words that have a more stable membership and do not readily accept new additions. The classes are considered closed because there are only a limited number of words in each category, and it is rare for new words to be added.
   * **Examples:**
     + **Pronouns:** Words that replace nouns and other pronouns.
       - *Examples:* he, she, it, they
     + **Prepositions:** Words that indicate relationships between other words.
       - *Examples:* in, on, under, between
     + **Conjunctions:** Words that connect words, phrases, or clauses.
       - *Examples:* and, but, or, because
     + **Articles:** Words that specify a noun as definite or indefinite.
       - *Examples:* a, an, the
     + **Auxiliary Verbs:** Words that assist the main verb in a sentence.
       - *Examples:* is, am, are, have

**Key Differences:**

* **Flexibility:** Open class words are flexible and allow for the addition of new words, while closed class words have a more stable set of members.
* **Function:** Open class words carry the main content and meaning in a sentence, while closed class words often serve grammatical or functional roles.
* **Commonality:** Open class words are more common in everyday language, contributing to the substance of communication, while closed class words provide structural elements and aid in sentence construction.

Understanding the distinction between open class and closed class words is crucial for analyzing the structure and composition of sentences in English and other languages.