TA5

Arun Kumar Rajasekaran

“Parsing”,” “syntactic analysis,” “syntax analysis”

-> The process of analyzing the strings of symbols in natural language in accordance with the norms of formal grammar is one way to describe it.

-> A Parser is What You Need to Report Any Errors with the Syntax

It assists in recovering from errors that are often experienced, allowing the processing of the remaining portions of the program to continue uninterrupted.

-> A Parser is Used to Assist in the Generation of a Parse Tree

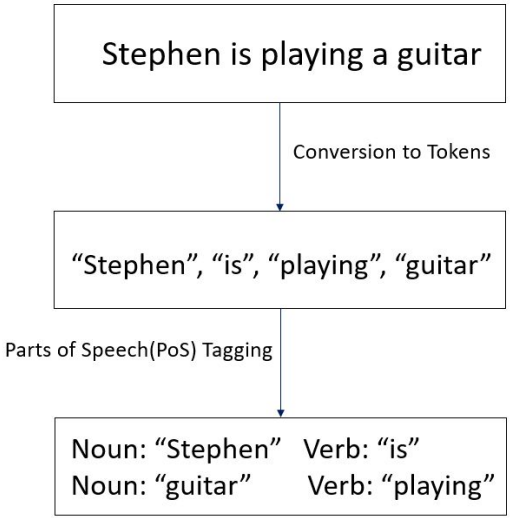
In natural language processing (NLP), the symbol table that is generated by the parser is an essential component. In addition to that, the parser is used to create intermediate representations (IR).

Shallow Parsing

“ analyzing the input sentence by breaking it down into its grammatical constituents, identifying the parts of speech, and syntactic relations ”

Shallow parsing is fairly limited (shallow) levels of parsing such as POS tagging, chunking, etc. We will learn chunking in the subsequent lessons.

Shallow parsing is a limited form of parsing that cannot check the grammatical structure of the sentence, i.e. whether a sentence is grammatically correct, or understand the dependencies between words in a sentence.



Deep vs Shallow parsing

Deep Parsing	Shallow Parsing
In deep parsing, the search strategy will give a complete syntactic structure to a sentence.	It is the task of parsing a limited part of the syntactic information from the given task.
It is suitable for complex NLP applications.	It can be used for less complex NLP applications.
Dialogue systems and summarization are the examples of NLP applications where deep parsing is used.	Information extraction and text mining are the examples of NLP applications where deep parsing is used.
It is also called full parsing.	It is also called chunking.

Dependency parsing vs POS tagging

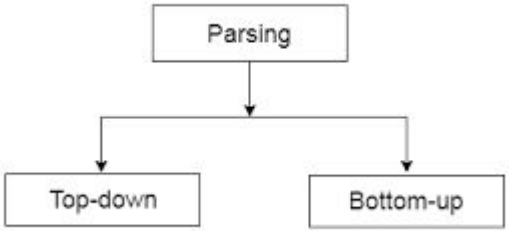
Part of Speech tagging or POS tagging specifies the property or attribute of the word or token. Each word in a sentence is associated with a part of speech tag such as nouns, verbs, adjectives, adverbs. The POS tags define the usage and function of a word in the sentence.

The part of speech only tags individual words and not phrases, hence it is not sufficient to create a parse tree. Parse trees are where we tag the phrases as noun Phrase, verb phrase, or prepositional phrase and it needs to be in that particular order.

Lets see some code examples

Top-down & Bottom-up

The definition of a parser may be summed up as a procedural interpretation of grammar. After looking through the area occupied by a number of trees, it arrives at the best tree option for the statement that was provided.



Top down parsing

The top-down parsing approach follows the left-most derivation technique. In the top-down parsing approach, the construction of the parse tree starts from the root node. So, first, the root node is formed and then the generation goes subsequently down by generating the leaf node (following the top to down approach).

Some of the important points regarding top-down parsing are:

- The top-down parsing uses the leftmost derivation approach to construct the parsing tree of the text.
- The top-down parser in NLP does not support the grammar having common prefixes.
- The top-down parser in NLP also guarantees that the grammar is free from all ambiguity and has followed the left recursion.

Bottom up Parsing

The bottom-up parsing approach follows the leaves-to-root approach or technique. In the bottom-up parsing approach, the construction of the parse tree starts from the leaf node. So, first, the leaf node is formed and then the generation goes subsequently up by generating the parent node, and finally, the root node is generated (following the bottom-to-up approach). The main aim of bottom-up parsing is to reduce the input string of text to get the start symbol by using the rightmost derivation technique.

Some of the important points regarding bottom-up parsing are:

- The bottom-up parsing approach is used by various programming language compilers such as C++, Perl, etc.
- This technique can be implemented very efficiently.
- The bottom-up parsing technique detects the syntactic error faster than other parsing techniques.

Types of Parsing

Recursive Descent Parser

Recursive descent among the many types of parsing, parsing is one of the easiest to understand. The recursive descent parser is discussed in more depth below, along with some key considerations.

- It works from the highest level down to the lowest.
- It makes an effort to validate the correctness or incorrectness of the syntax of the input stream.
- It reads the sentence from left to right, as it was entered.
- In order for a recursive descent parser to function properly, one of its required operations is to read characters from the input stream and then match those characters with the terminals from the grammar.

Types of Parsing

Shift-Reduce Parser

The shift-reduce parser will be discussed further after the following crucial points:

- It is a straightforward procedure working from the ground up.
- It searches for a string of words and phrases that match to the right-hand side of a grammatical production and then attempts to replace those words and phrases with the material that belongs on the left-hand side of the production.
- The effort to locate a sequence of words that was just described will continue until the whole phrase has been cut down.
- To put it another way, the shift-reduce parser begins with the input symbol and works its way toward the start symbol while attempting to build the parser tree.

Types of Parsing

Chart Parser

The following is a list of key information about chart parser:

- It is most helpful or appropriate for ambiguous grammars, especially the grammars of natural languages.
- In order to solve the parsing issues, it utilizes dynamic programming.
- Framework known as a ‘chart’ is used to hold the partially postulated outcomes that are generated as a consequence of dynamic programming.
- The ‘chart’ may also be utilized in a different way.

Types of Parsing

Regep Parser

- The regular expression (Regexp) parsing approach is one of the most frequently used parsing techniques. The following is a list of key information about the Regexp parser:
-
- On top of a string that has been POS-tagged, it makes use of a regular expression that is specified in the form of grammar, as the name suggests.
- It simply works by applying these regular expressions to the phrases that are supplied in order to construct a parse tree network from the data that is provided.

Types of parsing

Dependency Parsing

Dependency Parsing, often known as DP, is a relatively new parsing process. The key idea behind DP is that every linguistic unit, or word, has a direct connection to every other word in the language. In linguistic parlance, we refer to these direct connections as “dependencies.”

Types

1. Recursive descent parsing
    - top-down algorithm
    - pro: finds all successful parses.
    - con: inefficient. will try all rules brute-force, even the ones that do not match the input. Goes into an infinite loop when handling a left-recursive rule.
    - `nltk.RecursiveDescentParser()`
    - Interactive parsing demo: `nltk.app.rdparser()`
2. Shift-reduce parsing
    - bottom-up algorithm
    - pro: efficient. only works with the rules that match input words.
    - con: may fail to find a legitimate parse even when there is one.
    - `nltk.ShiftReduceParser()`
    - Interactive parsing demo: `nltk.app.srparser()`
3. The left-corner parser
    - a top-down parser with bottom-up filtering
    - `nltk.LeftCornerChartParser()`: combines left-corner parsing and chart parsing
4. Chart parsing
    - utilized **dynamic programming**: builds and refers to **well-formed substring tables (WFST)**
    - pro: efficient.
    - con: may take up a big memory space when dealing with a long sentence.
    - `nltk.ChartParser()`
    - Interactive parsing demo: `nltk.app.chartparser()`