Dependency Parsing in Natural Language Processing with Examples

Pure Language Processing is an interdisciplinary concept that uses the fundamentals of computational linguistics and Synthetic Intelligence to understand how human languages interact with technology.

To apply NLP to real-world scenarios, it is necessary to have a thorough grasp of various terminology and ideas. Dependency parsing and syntactic parsing serve as linguistic analysis methods in natural language processing. Dependency parsing focuses on revealing grammatical relationships between words in a sentence, portraying how words depend on each other. It constructs a tree structure that illustrates these dependencies, aiding in understanding sentence structure. Syntactic parsing, broader in scope, aims to uncover the overall syntactic structure of a sentence, encompassing phrase boundaries, constituents, and grammatical rules. Both techniques play a crucial role in extracting meaning and insights from text data, benefiting various language processing tasks. some of the important concepts are Half-of-Speech (POS) Tagging, Statistical Language Modeling, Syntactic, Semantic, and Sentiment Evaluation, Normalization, Tokenization, Dependency Parsing, and Constituency Parsing.

We will examine the principles of Dependency Parsing in this article to gain a better understanding of its application in Natural Language Processing.

Dependency Parsing

The term Dependency Parsing (DP) refers to the process of examining the dependencies between the phrases of a sentence in order to determine its grammatical structure. Dependency parsing divides a sentence into many sections based mostly on this. The process is based on the assumption that there is a direct relationship between each linguistic unit in a sentence. These hyperlinks are called dependencies.

Consider the following statement: “I prefer the morning flight through Denver.”

The diagram below explains the sentence’s dependence structure:

In a written dependency structure, directed arcs express the relationships between each linguistic unit or phrase in the sentence. The root of the tree “prefer” varies the pinnacle of the preceding sentence, as labelled within the illustration.

A dependence tag indicates the relationship between two phrases. For example, the word “flight” changes the meaning of the noun “Denver.” As a result, you may identify a dependence from

flight -> Denver, where flight is the pinnacle and Denver is the kid or dependent. It’s represented by nmod, which stands for the nominal modifier.

Scenario for Dependency

This distinguishes the scenario for dependency between the two phrases, where one serves as the pinnacle and the other as the dependent. Currently, the Common Dependency V2 taxonomy consists of 37 common syntactic relationships, as shown in the table below:

|  |  |
| --- | --- |
| **Dependency Tag** | **Description** |
| acl | clausal modifier of a noun (adnominal clause) |
| acl:relcl | relative clause modifier |
| advcl | adverbial clause modifier |
| advmod | adverbial modifier |
| advmod:emph | emphasizing phrase, intensifier |
| advmod:lmod | locative adverbial modifier |
| amod | adjectival modifier |
| appos | appositional modifier |
| aux | auxiliary |
| aux:move | passive auxiliary |
| case | case-marking |
| cc | coordinating conjunction |
| cc:preconj | preconjunct |
| ccomp | clausal complement |
| clf | classifier |
| compound | compound |
| compound:lvc | gentle verb building |
| compound:prt | phrasal verb particle |
| compound:redup | reduplicated compounds |
| compound:svc | serial verb compounds |
| conj | conjunct |
| cop | copula |
| csubj | clausal topic |
| csubj:move | clausal passive topic |
| dep | unspecified dependency |
| det | determiner |
| det:numgov | рrоnоminаl quаntifier gоverning the саse оf the nоun |
| det:nummod | рrоnоminаl quаntifier agreeing with the саse оf the nоun |
| det:poss | possessive determiner |
| discourse | discourse ingredient |
| dislocated | dislocated parts |
| expl | expletive |
| expl:impers | impersonal expletive |
| expl:move | reflexive pronoun utilized in reflexive passive |
| expl:pv | reflexive clitic with an inherently reflexive verb |
| mounted | mounted multiword expression |
| flat | flat multiword expression |
| flat:overseas | overseas phrases |
| flat:title | names |
| goeswith | goes with |
| iobj | oblique object |
| checklist | checklist |
| mark | marker |
| nmod | nominal modifier |
| nmod:poss | possessive nominal modifier |
| nmod:tmod | temporal modifier |
| nsubj | nominal topic |
| nsubj:move | passive nominal topic |
| nummod | numeric modifier |
| nummod:gov | numeriс mоdifier gоverning the саse оf the nоun |
| obj | object |
| obl | indirect nominal |
| obl:agent | agent modifier |
| obl:arg | indirect argument |
| obl:lmod | locative modifier |
| obl:tmod | temporal modifier |
| orphan | orphan |
| parataxis | parataxis |
| punct | punctuation |
| reparandum | overridden disfluency |
| root | root |
| vocative | vocative |
| xcomp | open clausal complement |

**Dependency Parsing using NLTK**

The Pure Language Toolkit (NLTK) package facilitates Dependency Parsing, providing a set of libraries and codes for statistical Natural Language Processing (NLP) of human language.

We may use NLTK to do dependency parsing in one of several ways:

**1. Probabilistic, projective dependency parser**: These parsers predict new sentences by using human language data acquired from hand-parsed sentences. They’re known to make mistakes and work with a limited collection of coaching information.

**2. Stanford parser**: It is a Java-based pure language parser. You would want the Stanford CoreNLP parser to perform dependency parsing. The parser supports a number of languages, including English, Chinese, German, and Arabic.

Here’s how you should use the parser:

from nltk.parse.stanford import StanfordDependencyParser

path\_jar = ‘path\_to/stanford-parser-full-2014-08-27/stanford-parser.jar’

path\_models\_jar = ‘path\_to/stanford-parser-full-2014-08-27/stanford-parser-3.4.1-models.jar’

dep\_parser = StanfordDependencyParser(

path\_to\_jar = path\_jar, path\_to\_models\_jar = path\_models\_jar

)

consequence = dep\_parser.raw\_parse(‘I shot an elephant in my sleep’)

dependency = consequence.subsequent()

checklist(dependency.triples())

The following is the output of the above program:

[

((u’shot’, u’VBD’), u’nsubj’, (u’I’, u’PRP’)),

((u’shot’, u’VBD’), u’dobj’, (u’elephant’, u’NN’)),

((u’elephant’, u’NN’), u’det’, (u’an’, u’DT’)),

((u’shot’, u’VBD’), u’prep’, (u’in’, u’IN’)),

((u’in’, u’IN’), u’pobj’, (u’sleep’, u’NN’)),

((u’sleep’, u’NN’), u’poss’, (u’my’, u’PRP$’))

]Copy Code

Constituency Parsing

Constituency Parsing is based on context-free grammars. Constituency context-free grammars parse text. Here, the parse tree includes sentences that break down into sub-phrases, each belonging to a different grammar class. A terminal node is a linguistic unit or phrase that has a mother or father node and a part-of-speech tag.

Fоr exаmрle, “A cat” and “a box beneath the bed”, are noun phrases, while “write a letter” and “drive a car” are verb phrases.

Consider the following example sentence: “I shot an elephant in my pajamas.” The constituency parse tree is shown graphically as follows:

The parse tree on the left represents catching an elephant carrying pyjamas, while the parse tree on the right represents capturing an elephant in his pyjamas.

The entire sentence breaks down into sub-phrases until only terminal phrases remain. VP stands for verb phrases, whereas NP stands for noun phrases.

Dependency Parsing vs Constituency Parsing

The Stanford parser will also be used to do constituency parsing. It begins by parsing a phrase using the constituency parser and then transforms the constituency parse tree into a dependency tree.

In case your main objective is to interrupt a sentence into sub-phrases, it is ideal to implement constituency parsing. However, dependency parsing is the best method for discovering the dependencies between phrases in a sentence.

Let’s look at an example to see what the difference is:

A constituency parse tree denotes the subdivision of a text into sub-phrases. The tree’s non-terminals represent different sorts of phrases, the terminals denote the sentence’s words, and the edges remain unlabeled. A constituency parse for the simple statement “John sees Bill” would be:

A dependency parse links words together based on their connections. Each vertex in the tree corresponds to a word, child nodes to words that are reliant on the parent, and edges to relationships. The dependency parse for “John sees Bill” is as follows:

You should choose the parser type that is most closely related to your objective. If you seek sub-phrases within a sentence, you definitely want to explore the constituency parse. If you want to understand the connection between words, you probably want to examine the dependency parse.

Conclusion

Organizations are seeking new methods to make use of computer technology as it advances beyond its artificial limits. A significant rise in computing speeds and capacities has resulted in the development of new and highly intelligent software systems, some of which are ready to replace or enhance human services.

One of the finest examples is the growth of natural language processing (NLP), with smart chatbots prepared to change the world of customer service and beyond.

In Summary, Human language is awe-inspiringly complex and diverse. NLP is significant not only because it helps resolve linguistic ambiguity but also because it provides a strong mathematical foundation for numerous applications like voice recognition and text analytics. To understand NLP, a solid grasp of the basics is essential, with Dependency Parsing being one of them.