Natural Language Processing

R18 B.Tech. CSE (AIML) III & IV Year JNTU Hyderabad

Prepared by
K SWAYAMPRABHA
Assistance Professor

UNIT - II

Syntax Analysis:

Parsing Natural Language

Treebanks: A Data-Driven Approach to Syntax

Representation of Syntactic Structure
1 Syntax Analysis Using Dependency Graphs
2 Syntax Analysis Using Phrase Structure Trees

Parsing Algorithms
1 Shift-Reduce Parsing
2 Hypergraphs and Chart Parsing
3 Minimum Spanning Trees and Dependency Parsing

Models for Ambiguity Resolution in Parsing
1 Probabilistic Context-Free Grammars
2 Generative Models for Parsing
3 Discriminative Models for Parsing

Multilingual Issues: What Is a Token?
1 Tokenization, Case, and Encoding
2 Word Segmentation
3 Morphology

Parsing natural language refers to the process of analyzing the structure of a sentence in order to determine its meaning. This is typically done by breaking down the sentence into its constituent parts, such as nouns, verbs, adjectives, and adverbs, and analyzing how these parts are related to each other.

There are several different methods for parsing natural language,

1. Rule-based systems,

2. Statistical models, and

3. Machine learning algorithms.

Rule-based systems rely on sets of predefined rules to analyze sentence structure and identify the relationships between different parts of speech.

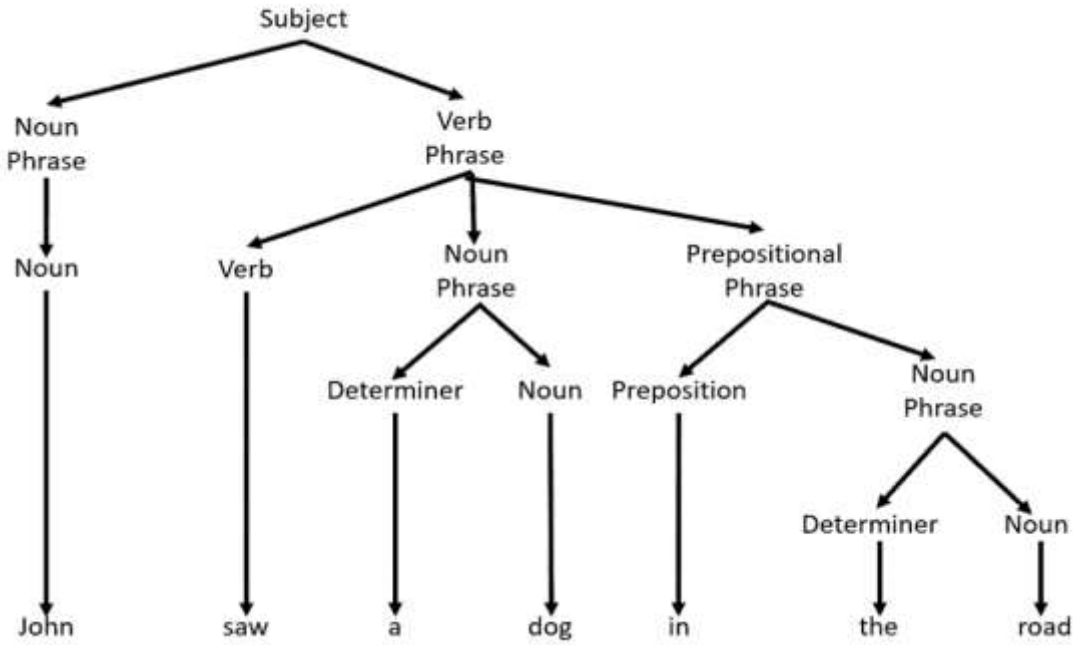
Statistical models use algorithms to analyze large datasets of annotated sentences in order to identify patterns and relationships between different parts of speech.

Machine learning algorithms, such as neural networks, are becoming increasingly popular for parsing natural language. These algorithms use large datasets of annotated sentences to train models that can accurately predict the structure and meaning of new sentences.

Parsing natural language is an important task in many areas, including natural language processing, machine translation, and speech recognition. It can also be used in applications such as chatbots, virtual assistants, and search engines, where understanding the meaning of user queries is essential for providing accurate and relevant responses.

TreeBank

It may be defined as **linguistically parsed text corpus that annotates syntactic or semantic sentence structure**. Geoffrey Leech coined the term 'treebank', which represents that the most common way of representing the grammatical analysis is by means of a tree structure.



Treebanks are a data-driven approach to syntax analysis in natural language processing. A treebank is a collection of sentences that have been parsed and annotated with their syntactic structures, typically represented as syntax trees.

In a treebank, each word in a sentence is labeled with its part of speech, such as noun, verb, or adjective. The words are then connected together in a tree structure that represents the relationships between them.

For example, a simple sentence like "The cat chased the mouse" might be represented as a tree with "cat" and "mouse" as noun phrases, "chased" as a verb, and "the" and "the" as determiners.

Treebanks are created by linguists and other experts who manually annotate the sentences with their syntactic structures. The process of creating a treebank is time-consuming and requires a lot of expertise, but once a treebank has been created, it can be used to train machine learning algorithms to automatically parse new sentences.

Treebanks are an important resource in natural language processing because they provide a large corpus of annotated sentences that can be used to train and evaluate syntactic parsers. They can also be used to study patterns in syntax across different languages and to develop new theories of syntax.

Some examples of well-known treebanks include the Penn Treebank, which is a collection of over 4 million words of English text, and Prague

Dependency Treebank, which is a collection of annotated sentences in Czech.

Representation of Syntactic Structure

The representation of syntactic structure in natural language processing refers to how the grammatical structure of a sentence is captured and represented. There are several different ways to represent syntactic structure

1. Syntax Analysis Using Dependency Graphs

Syntax analysis using dependency graphs is a common approach in natural language processing that represents the grammatical structure of a sentence as a directed graph. In this approach, each word in the sentence is represented as a node in the graph, and the relationships between the words are represented as directed edges between the nodes.

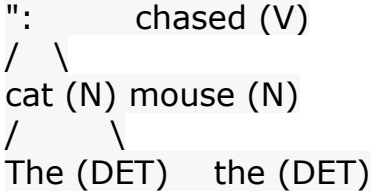
The edges in a dependency graph represent the syntactic relationships between words in the sentence.

For example, a subject-verb relationship is represented by an edge from the subject word to the verb word, while an object-verb relationship is represented by an edge from the object word to the verb word. Other relationships that can be represented in dependency graphs include adverbial modifiers, conjunctions, and prepositions.

Dependency graphs can be used for a variety of syntax analysis tasks, including dependency parsing, named entity recognition, and sentiment analysis. In dependency parsing, the goal is to automatically generate a dependency graph for a given sentence. This can be done using a variety of algorithms, including transition-based and graph-based parsers. Once a dependency graph has been generated, it can be used to identify the grammatical structure of the sentence, extract information about the relationships between words, and perform other syntactic analysis tasks.

Named entity recognition is another task that can be performed using dependency graphs. In this task, the goal is to identify and classify named entities in a sentence, such as people, places, and organizations. This can be done by analyzing the dependency relationships between words in the sentence and looking for patterns that indicate the presence of named entities.

Here is an example of a dependency graph for the sentence "The cat chased the mouse":

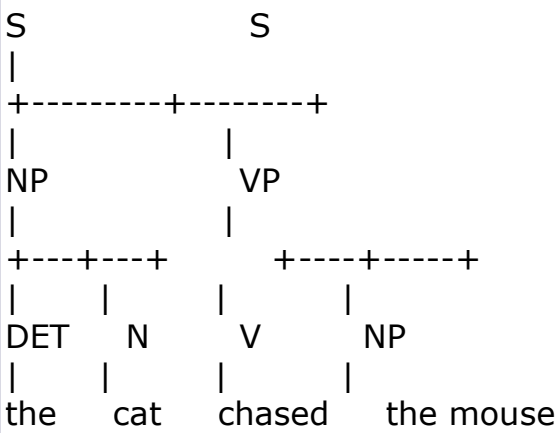


In this example, the words "cat" and "mouse" are connected to the verb "chased" by directed edges that indicate their syntactic relationships. Specifically, "cat" and "mouse" are both direct objects of the verb, while "chased" is the head or root of the sentence. The determiners "the" preceding both "cat" and "mouse" are also included in the dependency graph as dependents of their respective nouns. This dependency graph captures the syntactic structure of the sentence and can be used to perform a variety of syntactic analysis tasks.

2. Syntax Analysis Using Phrase Structure Trees with example

Syntax analysis using phrase structure trees is another approach in natural language processing that represents the grammatical structure of a sentence as a tree structure. In this approach, a sentence is broken down into its constituent phrases, and each phrase is represented as a node in the tree. The words in the sentence are then assigned to the appropriate nodes based on their syntactic roles.

Here is an example of a phrase structure tree for the sentence "The cat chased the mouse":



In this example, the sentence is represented as a tree structure with three levels: the root node S (representing the sentence), the second-level nodes

NP and VP (representing noun phrases and verb phrases), and the third-level nodes DET, N, and V (representing determiners, nouns, and verbs). The words in the sentence are assigned to the appropriate nodes based on their syntactic roles. For example, "cat" and "mouse" are both assigned to the NP node, while "chased" is assigned to the V node. The determiners "the" preceding both "cat" and "mouse" are also included in the tree structure as children of their respective noun nodes.

This phrase structure tree captures the syntactic structure of the sentence and can be used to perform a variety of syntactic analysis tasks, including parsing, translation, and text-to-speech synthesis.

Parsing Algorithms

Parsing algorithms are algorithms used in computer science to analyze the structure of a string of symbols in a particular formal language, typically represented as a context-free grammar. The process of analyzing the structure of a string is called parsing

---

1.Shift-reduce parsing is a type of bottom-up parsing technique used in computer science to analyze and understand the structure of a string or a sequence of tokens based on a given grammar. The technique starts with an empty stack and an input string, and repeatedly applies two operations: shift and reduce.

Shift moves the next input token onto the stack, while reduce applies a grammar rule to reduce the top of the stack to a non-terminal symbol. The parser continues to apply these operations until it reaches the end of the input string and the stack contains only the start symbol of the grammar.

The shift-reduce parsing algorithm can be implemented using

a finite state machine or a push-down automaton, which enables it to handle a wide range of context-free grammars, including ambiguous grammars. However, it may fail to recognize some valid input strings, which can be resolved by using look ahead techniques or by modifying the grammar to eliminate ambiguities.

Shift-reduce parsing is widely used in compiler design and natural language processing, as it provides an efficient and effective method for parsing and analyzing large volumes of structured data. Some common algorithms for shift-reduce parsing include LR parsers, SLR parsers, and LALR parsers, each

of which has its own strengths and weaknesses depending on the complexity and structure of the grammar.

here's an example of shift-reduce parsing:

Suppose we have the following grammar:

```
S -> E
E -> E + T | T
T -> T * F | F
F -> ( E ) | id
```

And we want to parse the input string "id * (id + id)". We can use a shift-reduce parser to build a parse tree for this string as follows:

1. Start with an empty stack and the input string "id * (id + id)".
2. Shift the first token "id" onto the stack.
3. Reduce the top of the stack to "F" using the rule "F -> id".
4. Shift the next token "*" onto the stack.
5. Shift the next token "(" onto the stack.
6. Shift the next token "id" onto the stack.
7. Reduce the top of the stack to "F" using the rule "F -> id".
8. Reduce the top of the stack to "T" using the rule "T -> F".
9. Reduce the top of the stack to "E" using the rule "E -> T".
10. Shift the next token "+" onto the stack.
11. Shift the next token "id" onto the stack.
12. Reduce the top of the stack to "F" using the rule "F -> id".
13. Reduce the top of the stack to "T" using the rule "T -> F".
14. Reduce the top of the stack to "E" using the rule "E -> E + T".
15. Shift the next token ")" onto the stack.
16. Reduce the top of the stack to "F" using the rule "F -> (E)".
17. Reduce the top of the stack to "T" using the rule "T -> F".
18. Reduce the top of the stack to "E" using the rule "E -> T".
19. The stack now contains only the start symbol "S", indicating that the input string has been successfully parsed.

The resulting parse tree for this input string is:

```
S
|
E
/ \
T   +
```

/ \ |
F id E
/ \
T F
/ \ |
F id id

2. Hypergraphs and Chart Parsing

Hypergraphs and chart parsing are both techniques used in natural language processing and computational linguistics to analyze the structure and meaning of natural language sentences.

A hypergraph is a graph-like data structure in which hyper edges connect multiple nodes instead of just two nodes as in a conventional graph. In the context of natural language processing, hypergraphs can be used to represent the syntactic and semantic structures of sentences. Each node in the hypergraph represents a word or a phrase in the sentence, and each hyper edge represents a grammatical relationship between those words or phrases.

Chart parsing is a type of parsing algorithm that uses a dynamic programming approach to build a chart or table that represents the different possible syntactic and semantic structures of a sentence. The chart contains cells that represent the different combinations of words and phrases in the sentence, and the algorithm uses a set of grammar rules to fill in the cells with possible syntactic and semantic structures.

Chart parsing can be used with hypergraphs to build a more complex representation of sentence structure and meaning. The hypergraph can be used to represent the full range of possible syntactic and semantic structures for a sentence, while the chart can be used to efficiently explore and evaluate those structures.

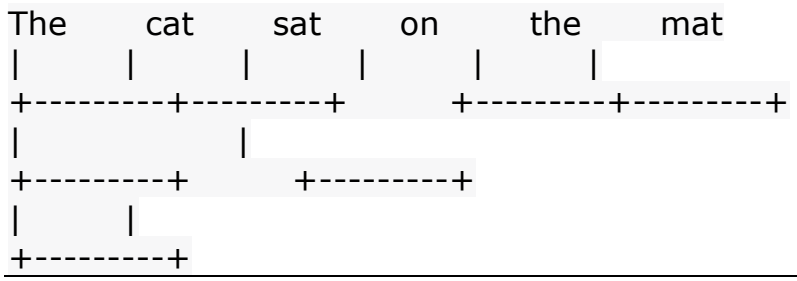
One common type of chart parsing algorithm is the Earley parser, which uses a bottom-up approach to construct the chart. Another common algorithm is the CYK parser, which uses a top-down approach and is based on context-free grammars.

Chart parsing with hypergraphs is widely used in natural language processing applications such as machine translation, text-to-speech synthesis, and information extraction. By representing the structure and meaning of sentences in a formal and precise way, these techniques can help computers to better understand and generate natural language text.

Suppose we have the following sentence:

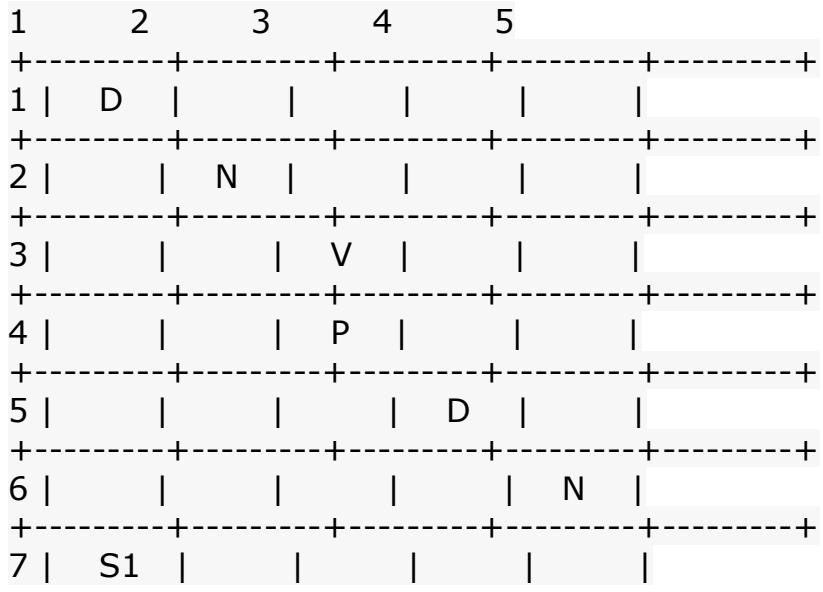
"The cat sat on the mat."

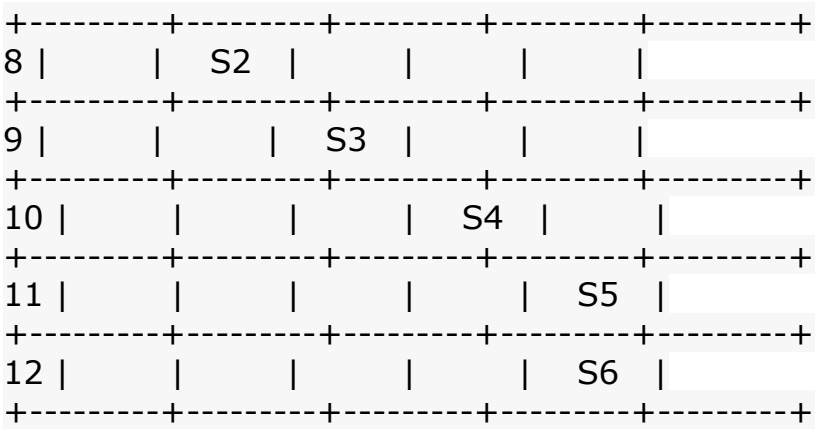
We can use a hypergraph to represent the different possible syntactic and semantic structures for this sentence. Each node in the hypergraph represents a word or a phrase, and each hyperedge represents a grammatical relationship between those words or phrases. Here's an example hypergraph for this sentence:



In this hypergraph, the words "the", "cat", "sat", "on", and "mat" are represented as nodes, and the hyperedges represent the grammatical relationships between those words. For example, the hyperedge connecting "cat" and "sat" represents the fact that "cat" is the subject of the verb "sat".

We can then use chart parsing to build a chart that represents the different possible syntactic and semantic structures for this sentence. Each cell in the chart represents a combination of words or phrases, and the chart is filled in with possible structures based on a set of grammar rules. Here's an example chart for this sentence:





In this chart, the rows represent the start and end positions of phrases, and the columns represent the different phrase types (D for determiner, N for noun, V for verb, and P for preposition). The cells are filled in with the possible structures based on the grammar rules. For example, cell (2,2) represents the phrase "cat", cell (3,3) represents the verb phrase "sat", and cell (4,4) represents the prepositional phrase "on the mat".

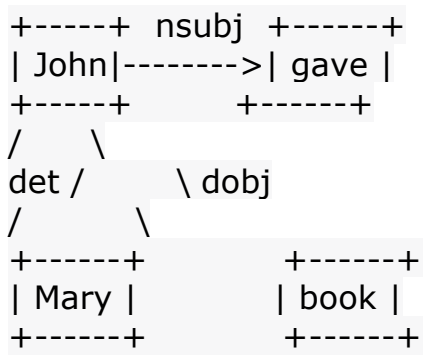
By combining hypergraphs and chart parsing, we can build a more complete Minimum spanning trees and dependency parsing are techniques used in natural language processing to analyze the grammatical structure of sentences.

3. A minimum spanning tree (MST) is a tree-like structure that connects all the nodes in a weighted graph with the minimum possible total edge weight. In the context of natural language processing, an MST can be used to represent the most likely grammatical structure for a sentence, with the nodes representing the words in the sentence and the edges representing the grammatical relationships between those words.

Dependency parsing is a type of parsing algorithm that uses syntactic dependency relationships to analyze the structure of a sentence. In a dependency tree, the nodes represent the words in the sentence and the edges represent the grammatical relationships between those words, such as subject-verb or object-preposition.

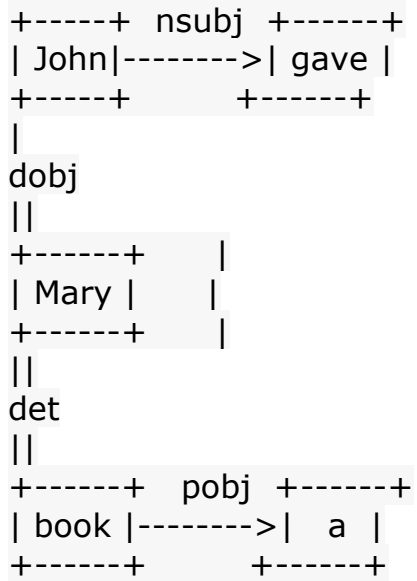
Here's an example of how minimum spanning trees and dependency parsing can be used to analyze a sentence:

Consider the sentence "John gave Mary a book". We can use dependency parsing to identify the syntactic dependencies between the words in the sentence:



In this dependency tree, the nodes represent the words in the sentence, and the edges represent the syntactic dependencies between those words. For example, the "nsubj" edge connects "John" to "gave" and represents the fact that "John" is the subject of the verb "gave". The "dobj" edge connects "book" to "gave" and represents the fact that "book" is the direct object of the verb "gave".

We can then use an MST algorithm to find the minimum spanning tree that connects all the nodes in the dependency tree with the minimum possible total edge weight. The resulting MST represents the most likely grammatical structure for the sentence.



In this MST, the nodes still represent the words in the sentence, but the edges represent the most likely grammatical relationships between those words. For example, the "nsubj" and "dobj" edges are the same as in the original dependency tree, but the "det" edge connecting "a" to "book" represents the fact that "a" is a determiner for "book".

By analyzing the grammatical structure of sentences with minimum spanning trees and dependency parsing, we can gain insights into the meaning and structure of natural language text. These techniques are widely used in applications such as machine translation, sentiment analysis, and text classification.

Models for Ambiguity Resolution in Parsing

Parsing is the process of analyzing a sentence and determining its syntactic structure. However, natural language sentences can often be ambiguous, and different parsing models may assign different syntactic structures to the same sentence. In order to resolve ambiguity in parsing, various models have been proposed. Here are some of the most common models for ambiguity resolution in parsing:

1. Probabilistic context-free grammars (PCFGs) are a type of context-free grammar where each production rule is assigned a probability. These probabilities represent the likelihood of generating a particular string of symbols using the given rule. PCFGs are often used in natural language processing tasks such as parsing, where they can be used to assign probabilities to different parse trees for a given sentence.

Here is an example of a PCFG for generating simple arithmetic expressions:

S -> E
E -> E + E [0.4]
E -> E - E [0.3]
E -> E * E [0.2]
E -> E / E [0.1]
E -> (E) [0.0]
E -> num [0.0]

In this grammar, S is the start symbol and E represents an arithmetic expression. The production rules for E indicate that an arithmetic expression can be generated by adding two expressions with probability 0.4, subtracting two expressions with probability 0.3, multiplying two expressions with probability 0.2, dividing two expressions with probability 0.1, or enclosing an expression in parentheses with probability 0.0. Finally, an arithmetic expression can also be a number (num) with probability 0.0.

2 Generative Models for Parsing

Generative models for parsing are statistical models that aim to generate or simulate sentences that follow the same distribution as a given training corpus. These models use probabilistic context-free grammars (PCFGs) or other similar techniques to generate sentences based on their grammar and probability distributions.

Here is an example of a generative model for parsing that uses a PCFG:

Suppose we have a small training corpus of three sentences:

1. The cat sat on the mat.
2. The dog chased the cat.
3. The bird flew away.

We can use this corpus to learn a PCFG that represents the grammar of these sentences. Here is a sample PCFG:

S -> NP VP [1.0]
NP -> Det N [0.67] | N [0.33]
VP -> V NP [0.67] | V [0.33]
Det -> the [1.0]
N -> cat [0.33] | dog [0.33] | bird [0.33]
V -> sat [0.33] | chased [0.33] | flew [0.33]
This grammar allows us to generate new sentences that are similar to the sentences in the training corpus. For example, we can use the following parse tree to generate the sentence "The cat chased the bird":
S
|
NP VP
| | |
Det N V
| | |
the cat chased
| | |
N
bird

To generate this sentence, we start with the S symbol and apply the production rule **S -> NP VP**. We then randomly choose between the two possible expansions of NP and VP based on their probabilities. In this case, we choose **NP -> Det N** and **VP -> V NP**. We then randomly choose the expansions of Det, N, and V based on their probabilities. Finally, we combine the resulting strings to get the sentence "The cat chased the bird".

We can generate other sentences in the same way, by randomly choosing expansion rules based on their probabilities. Note that this approach allows us to generate sentences that may not have appeared in the training corpus, but are still grammatically correct according to the PCFG.

3. Discriminative Models for Parsing with example

Discriminative models for parsing are statistical models that aim to predict the correct parse tree for a given input sentence. These models use features of the input sentence and their context to make this prediction.

Here is an example of a discriminative model for parsing using a linear support vector machine (SVM):

Suppose we have the following input sentence:

"The cat sat on the mat."

We can use a set of hand-crafted features to represent the input sentence and its context, such as:

- The current word "cat"
- The previous word "The"
- The next word "sat"
- The part of speech (POS) tag of "cat"
- The POS tag of "The"
- The POS tag of "sat"
- The dependency relation between "cat" and "sat"
- The head word of "cat"
- The head word of "sat"
-

- We can then use these features as input to a linear SVM, which learns to predict the correct parse tree based on these features. The SVM is trained on a set of annotated sentences, where each sentence is represented by its features and the correct parse tree.

- During testing, the SVM predicts the correct parse tree for a given input sentence by computing a weighted sum of the features, and then applying a threshold to this sum to make a binary classification decision. The predicted parse tree can then be converted into a more readable format, such as a bracketed string.

- Here is an example parse tree that could be predicted by the SVM for the input sentence "The cat sat on the mat":

```
(S
(NP (DT The) (NN cat))
(VP (VBD sat) (PP (IN on) (NP (DT the) (NN mat))))
(. .))
```

Note that discriminative models can be trained on a variety of feature sets, including hand-crafted features as shown in this example, or features learned automatically from the input data using techniques such as neural networks. Discriminative models can also incorporate additional information, such as lexical semantic knowledge or discourse context, to improve their accuracy.

Multilingual Issues: What Is a Token?

In natural language processing, a token refers to a sequence of characters that represent a single unit of meaning. Typically, tokens correspond to words or punctuation marks in a sentence.

However, in multilingual settings, the definition of a token can become more complex. This is because different languages may use different writing systems, character encodings, or word segmentation conventions, which can affect how tokens are defined and processed.

For example, consider the following sentence in Chinese:

我爱北京天安门。

This sentence consists of six characters, which could be considered tokens in a Chinese language processing pipeline. However, the sentence could also be segmented into four words, corresponding to the following tokens:

我 (I) 爱 (love) 北京 (Beijing) 天安门 (Tiananmen)

Similarly, in languages that use non-Latin scripts, such as Arabic or Hebrew, the definition of a token can be more complex due to the presence of diacritics or ligatures, which may affect how words are represented and processed.

In multilingual natural language processing, it is important to carefully define and standardize the tokenization process in order to ensure that input text is processed consistently and accurately across different languages and scripts. This may involve developing language-specific tokenization rules or using machine learning techniques to automatically segment text into tokens.

Tokenization, Case, and Encoding

Tokenization, case, and encoding are important concepts in natural language processing that are often applied to text data prior to modeling or analysis.

1. Tokenization refers to the process of breaking down a piece of text into individual units, called tokens. In English, tokens typically correspond to words or punctuation marks, and can be extracted using simple rules based on whitespace and punctuation. For example, the sentence "The quick brown fox jumps over the lazy dog" can be tokenized into the following tokens:

["The", "quick", "brown", "fox", "jumps", "over", "the", "lazy", "dog"]

Case refers to the capitalization of words in a piece of text. In many cases, it is useful to convert all words to lowercase in order to reduce the number of distinct tokens and simplify subsequent analysis. However, there may be cases where preserving the original case of words is important for downstream tasks, such as named entity recognition. For example, the sentence "New York is a city in the United States" can be converted to lowercase as follows:

Encoding refers to the process of converting text into a numerical representation that can be processed by machine learning algorithms. One common encoding scheme is one-hot encoding, where each token is represented as a binary vector with a 1 in the position corresponding to the token's index in a fixed vocabulary, and 0s elsewhere. For example, the sentence "The quick brown fox jumps over the lazy dog" can be encoded as a matrix of shape (9, 9), where each row corresponds to a token and each column corresponds to a position in the vocabulary:

```
[[1, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 1, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 1, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 1, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 1, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 1, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 1, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 1]]
```

Other encoding schemes include word embeddings, which map each token to a low-dimensional vector that captures its semantic and syntactic properties, and character-level encodings, which represent each character in a token as a separate feature.

Overall, tokenization, case, and encoding are critical preprocessing steps that help transform raw text data into a format that can be effectively analyzed and modeled.

2. Word Segmentation

Word segmentation refers to the process of identifying individual words in a piece of text, especially in languages where words are not explicitly separated by spaces or punctuation marks. Word segmentation is an important task in natural language processing and can be challenging in languages such as Chinese, Japanese, and Thai.

For example, consider the following sentence in Chinese:

我爱北京天安门。

This sentence consists of six characters, which could be considered tokens in a Chinese language processing pipeline. However, the sentence could also be segmented into four words, corresponding to the following tokens:

我 (I) 爱 (love) 北京 (Beijing) 天安门 (Tiananmen)

The segmentation of Chinese text into words is typically performed using a combination of statistical and rule-based methods. For example, one common approach is to use a dictionary or corpus of Chinese words as a reference, and then apply statistical models or rule-based heuristics to identify likely word boundaries. Other approaches use machine learning algorithms, such as conditional random fields or neural networks, to learn the boundaries between words from annotated training data.

In languages such as Japanese and Thai, where words may be written without spaces, word segmentation can be even more challenging. In these cases, additional linguistic and contextual information may be required to disambiguate word boundaries. For example, in Japanese, the use of different writing systems (kanji, hiragana, katakana) can provide cues for word segmentation, while in Thai, the tone and pronunciation of individual characters can help identify word boundaries.

Overall, word segmentation is an important task in natural language processing that is essential for accurate analysis and modeling of text data, particularly in languages where words are not explicitly separated by spaces or punctuation marks.

3. Morphology

Morphology refers to the study of the structure of words and the rules that govern the formation of words from smaller units known as morphemes. Morphemes are the smallest units of meaning in a language and can be either free (can stand alone as words) or bound (must be attached to other morphemes to form words).

For example, consider the word "unhappily." This word consists of three morphemes:

1. "un-" is a prefix that means "not"
2. "happy" is the root or base word
3. "-ly" is a suffix that means "in a particular way"

Each of these morphemes has a specific meaning and function, and their combination in the word "unhappily" changes the meaning and grammatical function of the root word "happy."

Morphology is important in natural language processing because it can help identify the meaning and grammatical function of individual words, and can also provide insights into the structure and patterns of a language. Some common applications of morphology in NLP include:

1. Stemming: the process of reducing a word to its root or stem form, which can help reduce the number of unique words in a text corpus and improve efficiency in language modeling and information retrieval systems.
2. Morphological analysis: the process of breaking down words into their constituent morphemes, which can help identify word meanings and relationships, as well as identify errors or inconsistencies in text data.
3. Morphological generation: the process of creating new words from existing morphemes, which can be useful in natural language generation tasks such as machine translation or text summarization.
