14-06-2023

**FURTHER DEEP INTO NATURAL LANGUAGE PROCESSING**

1. **Introduction:**

* Language is a method of communication with the help of which we can speak, read and write. Natural Language Processing (NLP) is the sub-field of Computer Science especially Artificial Intelligence (AI) that is concerned about enabling computers to understand and process human language.
* Technically, the main task of NLP would be to program computers for analysing and processing huge amount of natural language data. Language is a crucial component for human lives and also the most fundamental aspect of our behavior.
* There are mainly two forms – written and spoken.
* In the written form, it is a way to pass our knowledge from one generation to the next.
* In the spoken form, it is the primary medium for human beings to coordinate with each other in their day-to-day behavior.

1. **History of NLP:**
2. **Machine Translation Phase- Late 1940s to 1960s**

* The research on NLP started
* The publication of the journal MT (Machine Translation) started.
* First international conference on MT was held.

1. **AI Influenced Phase- Late 1960s to late 1970s**

* Work was mainly related to knowledge and its role in construction and manipulation of meaning representations.
* Problems of addressing and constructing data or knowledge base.
* A Baseball question-answering system was developed. (Simple one)
* A much-advanced system was described by Minsky.

1. **Grammatico-logical Phase- Late 1970s to late 1980s**

* Moved towards the use of logic for knowledge representation.
* Powerful general purpose sentence processors like SRI’s Core Language Engine and Discourse Representation Theory were built.
* Practical resources and tools like parsers for more commercial systems were developed.
* The work on lexicon was also directed.

1. **Lexical and Corpus Phase- 1990s**

* Lexicalised approach to grammar.
* Introduction of machine learning algorithms for language processing.

1. **Ambiguity and Uncertainty in Language**

Ambiguity, generally used in natural language processing, can be referred as the ability of being understood in more than one way.

*Types of Ambiguity:*

1. **Lexical Ambiguity:**

Ambiguity of a single word.

**Ex:** Silver as a noun, adjective, or a verb

1. **Syntactic Ambiguity:**

Ambiguity when a sentence is parsed in different ways.

**Ex:** The man saw the girl with the telescope.

It is ambiguous is the man saw the girl carrying a telescope or the man saw the girl through a telescope.

1. **Semantic Ambiguity:**

Ambiguity when the meaning of the words themselves can be misinterpreted.

**Ex:** “I don't like it when my father smokes.” The word “smokes” has more than one meaning, and the significance of the sentence changes dramatically depending on which meaning is intended.

1. **Anaphoric Ambiguity:**

This kind of ambiguity arises due to the use of anaphora entities in discourse.

**Ex:** The horse ran up the hill. It was very steep. It soon got tired.

Here, the anaphoric reference of “it” in two situations cause ambiguity.

1. **Pragmatic Ambiguity:**

This ambiguity refers to the situation where the context of a phrase gives it multiple interpretations.

**Ex:** “I like you too”

Multiple interpretations like I like you (just like you like me), I like you (just like someone else dose) is possible.

1. **NLP Phases:**

Input sentence--🡪 Morphological Processing---🡪 Syntax Analysis (Lexicon and Grammar) ----🡪Semantic Analysis based of Semantic Rules----🡪 Pragmatic Analysis based on Contextual Information---🡪 Target Representation

1. **Morphological Processing:**

Breaks chunks of language input into sets of tokens corresponding to paragraphs, sentences and words.

**Ex:** Uneasy can bebroken into sub-tokens as “un-easy”.

1. **Syntax Analysis:**

-🡪 Check if the sentence is well formed.

-🡪 Break it up into a structure that shows the syntactic relationships between different words.

**Ex: “**The school goes to the boy” gets rejected by the syntax analyser or parser.

1. **Semantic Analysis:**

The purpose of this phase is to draw exact meaning.

Dictionary meaning from the text. The text is checked for meaningfulness.

**Ex:** semantic analyser would reject a sentence like “Hot ice-cream”.

1. **Pragmatic Analysis:**

Pragmatic analysis simply fits the actual objects/events, which exist in a given context with object references obtained during the last phase (semantic analysis).

**Ex:** “Put the banana in the basket on the shelf.”

This can have two semantic interpretations and pragmatic analyser will choose between these two possibilities.

1. **Linguistic Resources in Natural Language Processing**
2. **Corpus:**

A corpus is a large and structured set of machine-readable texts that have been produced in a natural communicative setting. They can be derived in different

ways like text that was originally electronic, transcripts of spoken language and optical character recognition, etc.

1. **Elements of Corpus Design:**

Language is infinite, but a corpus must be finite in size. To achieve a good corpus design, we need to sample a wide range of text types proportionally.

**Corpus representativeness** is a crucial aspect, defined by Leech and Bibber as the ability to generalize findings to a language variety and include the full range of variability in a population, respectively.

**Balance**, referring to the genres included, and sampling, involving the selection of chunks for each genre, determine the representativeness of a corpus. Corpus balance is important for representativeness and relies on estimation and intuition rather than a scientific measure. **Sampling** is unavoidable in corpus building and is closely tied to representativeness and balance. Considerations in sampling include the sampling unit (e.g., newspaper, journal, book), sampling frame (list of sampling units), and population (assembly of all sampling units).

**Corpus size** is another key element influenced by the intended purpose, user query expectations, methodology, and data availability.

1. **TreeBank Corpus:**

The term "treebank" was coined by Geoffrey Leech, and it signifies that the most common method of representing grammatical analysis is through a tree structure.

* Treebanks are typically built on top of an existing corpus that has already been annotated with part-of-speech tags. This means that the original corpus has undergone a process where each word in the text has been labelled with its corresponding part of speech (noun, verb, adjective, etc.). Building on this annotated corpus, a treebank then adds further linguistic annotations that represent the hierarchical structure of sentences, often in the form of a parse tree.
* By utilizing a tree structure, a treebank provides a visual representation of how words in a sentence relate to each other syntactically or semantically. This kind of annotation is valuable for various natural language processing tasks, such as grammar analysis, parsing, and machine learning algorithms that require a deeper understanding of sentence structure.
* Overall, a treebank is a linguistic resource that enhances a corpus by adding syntactic or semantic annotations in the form of a tree structure to aid in the analysis and understanding of sentence-level linguistic phenomena.

1. **Types of TreeBank Corpus:**

There are two common types of Treebanks in linguistics: Semantic Treebanks and Syntactic Treebanks.

1. **Semantic TreeBanks:**

* Semantic Treebanks focus on the formal representation of a sentence's semantic structure.
* These Treebanks can vary in the depth of their semantic representation, meaning some may provide more detailed semantic annotations than others.
* Examples of Semantic Treebanks include the Robot Commands Treebank, Geoquery, Groningen Meaning Bank, and RoboCup Corpus.
* These resources capture and annotate the semantic relationships and meanings expressed within sentences.

1. **Syntactic TreeBank:**

* On the other hand, Syntactic Treebanks take a different approach.
* Instead of emphasizing the semantic structure, Syntactic Treebanks work with inputs that are expressions of a formal language.
* These inputs are obtained through the conversion of parsed Treebank data. The outputs of Syntactic Treebank systems are based on predicate logic and represent the meaning of sentences in a syntactic manner.
* Different languages have their own Syntactic Treebanks.
* For instance, the Penn Arabic Treebank and Columbia Arabic Treebank are examples of Syntactic Treebanks created for the Arabic language, while the Sininca syntactic Treebank is for Chinese, and Lucy, Susane, and BLLIP WSJ syntactic corpus are for English.

In summary, Semantic Treebanks focus on capturing semantic structures and relationships in sentences, while Syntactic Treebanks concentrate on formal language expressions and provide predicate logic-based representations of sentence meaning. Both types of Treebanks play a crucial role in linguistic research, language processing, and computational analysis of natural language.

1. **PropBank Corpus:**

* PropBank, also known as "Proposition Bank," which is a corpus that contains annotations of verbal propositions and their associated arguments. It is a resource that focuses on verbs and provides syntactic-level annotations. The PropBank corpus was developed by Martha Palmer and her colleagues at the Department of Linguistics, University of Colorado Boulder.
* PropBank serves as a verb-oriented linguistic resource that annotates sentences with propositions, which represent the semantic relationships between verbs and their arguments. These arguments can include various syntactic components such as subjects, objects, and modifiers. The annotations in PropBank are primarily concerned with the syntactic structure and the roles played by different constituents in relation to the verb.
* In the field of Natural Language Processing (NLP), the PropBank project has played a significant role, particularly in the area of semantic role labelling. Semantic role labelling is the task of identifying and labelling the different roles that nouns and pronouns play in relation to the verb in a sentence. PropBank provides valuable annotations that assist in this process by mapping verb senses and argument structures to specific roles.
* It's important to note that the term "PropBank" can be used more broadly to refer to any corpus that has been annotated with propositions and their associated arguments, following a similar annotation scheme. The original PropBank corpus developed by Martha Palmer and her team serves as a prominent example of such a resource, but there may be other PropBanks for different languages or domains that adhere to the same annotation principles.

1. **VerbNet (VN):**

VerbNet (VN) is a comprehensive and domain-independent lexical resource in English. VN incorporates both semantic and syntactic information about its contents, making it a valuable tool for linguistic analysis. It is considered the largest verb lexicon available in English and has connections to other lexical resources like WordNet, Xtag, and FrameNet.

VerbNet is organized into hierarchical verb classes, which extend Levin classes. This organization involves refining and adding subclasses to ensure both syntactic and semantic coherence among the members of each class. This hierarchical structure allows for a detailed and systematic representation of verb behavior and characteristics.

Each VerbNet class consists of two main components:

1. **Syntactic descriptions or syntactic frames:** These describe the possible surface realizations of the argument structure for different constructions. Examples include transitive and intransitive constructions, prepositional phrases, resultatives, and various diathesis alternations. Syntactic frames provide a blueprint for how verbs can combine with different arguments in a sentence.

2. **Semantic descriptions:** These provide information about the semantic properties associated with the verb class. They include categories such as animate, human, organization, which help constrain the types of thematic roles allowed by the arguments. Semantic descriptions provide insights into the expected syntactic behavior and the likely constituents associated with specific thematic roles.

By combining syntactic frames and semantic descriptions, VerbNet enables a detailed analysis of verb behavior, argument structures, and thematic roles. It offers a rich source of information for studying the syntax and semantics of verbs in English, allowing for more accurate and comprehensive language processing and understanding.

1. **Regular Expressions: (ReGex)**

A regular expression (RE) is a language for specifying text search strings. RE helps us to match or find other strings or sets of strings, using a specialized syntax held in a pattern. Regular expressions are used to search texts in UNIX as well as in MS WORD in identical way. We have various search engines using a number of RE features.

*Properties of Regular Expressions:*

1. Regular Expressions were formalized by American mathematician Stephen Cole Kleene.

2. RE is a formula in a special language used to specify simple classes of strings, which are sequences of symbols. It is an algebraic notation for characterizing sets of strings.

3. To use regular expressions, two things are needed: a pattern to search for and a corpus of text to search within.

4. Mathematically, a Regular Expression can be defined as follows:

- ε represents an empty string.

- φ represents an empty language (no strings).

- If X and Y are Regular Expressions, the following operations can be performed:

- X, Y (simple expressions)

- X.Y (concatenation of X and Y)

- X+Y (union of X and Y)

- X\*, Y\* (Kleene closure of X and Y)

5. If a string is derived using the above rules, it is also considered a regular expression.

Regular Expressions provide a powerful and flexible way to describe and manipulate patterns within strings. They are widely used in various fields, including programming, text processing, and data validation, to search, match, and manipulate text efficiently and effectively.

1. **Regular Sets and Their Properties:**

It may be defined as the set that represents the value of the regular expression and consists specific properties.

*Properties of Regular Sets:*

1. Union: If we take the union of two regular sets, the resulting set would also be regular. In other words, if we combine two regular sets, the set containing all the elements from both sets will still be regular.

2. Intersection: If we take the intersection of two regular sets, the resulting set would also be regular. This means that if we find the common elements between two regular sets, the resulting set will still be regular.

3. Complement: If we take the complement of a regular set, the resulting set would also be regular. The complement of a set consists of all elements that are not in the original set. In the case of regular sets, the resulting set will still be regular.

4. Difference: If we take the difference of two regular sets, the resulting set would also be regular. The difference between two sets consists of all the elements in the first set that are not present in the second set. Even after taking the difference, the resulting set will remain regular.

5. Reversal: If we reverse a regular set, the resulting set would also be regular. Reversing a set means reversing the order of its elements. Regular sets maintain their regularity even after being reversed.

6. Closure: If we take the closure (also known as the Kleene closure) of a regular set, the resulting set would also be regular. The closure of a set consists of all possible concatenations of the elements in the set, including the empty string. Regular sets preserve their regularity even after taking the closure.

7. Concatenation: If we concatenate two regular sets, the resulting set would also be regular. Concatenation involves combining the elements of one set with the elements of another set in all possible ways. The resulting set, formed by joining the elements from both sets, will still be regular.

1. **Finite State Automata:**

Automata are abstract self-propelled computing devices that perform a predetermined sequence of operations automatically.

An automaton with a finite number of states is referred to as a Finite Automaton (FA) or Finite State Automaton (FSA). Mathematically, an automaton can be represented using a 5-tuple (Q, ∑, δ, q0, F), where each element serves a specific purpose:

1. Q: It is a finite set of states that the automaton can be in. The automaton can transition between these states based on its input and the defined rules.

2. ∑: It is a finite set of symbols, known as the alphabet of the automaton. These symbols represent the valid inputs or symbols that the automaton can process.

3. δ: It represents the transition function, which defines how the automaton moves from one state to another based on the current state and the input symbol. It specifies the rules for state transitions.

4. q0: It is the initial state from which the automaton starts processing the input. The automaton begins its operation from this state.

5. F: It is a set of final state(s) from the set of states Q. When the automaton reaches any state in this set after processing the input, it is considered a valid or accepted input. The automaton's task may involve recognizing patterns or determining if the input belongs to a specific language or satisfies certain conditions.

By defining these elements, an automaton can be used to model and solve various computational problems. Finite Automata have been extensively studied in computer science and play a fundamental role in areas such as formal languages, automata theory, and compiler design.

1. **Relation between Finite Automata, Regular Grammars and Regular Expressions**:

• As we know that finite state automata are the theoretical foundation of computational work and regular expressions is one way of describing them.

• We can say that any regular expression can be implemented as FSA and any FSA can be described with a regular expression.

• On the other hand, regular expression is a way to characterize a kind of language called regular language. Hence, we can say that regular language can be described with the help of both FSA and regular expression.

• Regular grammar, a formal grammar that can be right-regular or left-regular, is another way to characterize regular language.

1. **Types of Finite State Automation: (FSA)**

There are two types of Finite State Automaton: Deterministic Finite Automaton (DFA) and Non-deterministic Finite Automaton (NDFA).

**1. Deterministic Finite Automaton (DFA):**

A DFA is a type of finite automaton in which, for every input symbol, the state to which the machine will move can be determined. It is called "deterministic" because the transition from one state to another is uniquely defined. A DFA is represented by a 5-tuple (Q, ∑, δ, q0, F), where:

- Q is a finite set of states.

- ∑ is a finite set of symbols, the alphabet of the automaton.

- δ is the transition function, δ: Q × ∑ → Q.

- q0 is the initial state from where any input is processed (q0 ∈ Q).

- F is a set of final states of Q (F ⊆ Q).

Graphically, a DFA is represented using state diagrams:

- States are represented by vertices.

- Transitions are shown by labelled arcs.

- The initial state is represented by an empty incoming arc.

- The final state is represented by a double circle.

**2. Non-deterministic Finite Automaton (NDFA):**

An NDFA is a type of finite automaton in which, for every input symbol, the state to which the machine will move cannot be uniquely determined. The machine can move to any combination of states. Similar to DFA, an NDFA has a finite number of states. It is represented by a 5-tuple (Q, ∑, δ, q0, F), where:

- Q is a finite set of states.

- ∑ is a finite set of symbols, the alphabet of the automaton.

- δ is the transition function, δ: Q × ∑ → 2^Q (the power set of Q).

- q0 is the initial state from where any input is processed (q0 ∈ Q).

- F is a set of final states of Q (F ⊆ Q).

The graphical representation of an NDFA is similar to a DFA, using state diagrams.