UNIT-2

WORD LEVEL ANALYSIS : Introduction, Regular Expression,Finite State Automata,Morphological Parsing,Spelling Error Detection and Correction,Words and Word Classes, Parts-Of-Speech Tagging

SYNTACTIC ANALYSIS : Introduction,Context-Free Grammer,Constituency,Parsing,Probabilistic Parsing,Indian Languages.

INTRODUCTION TO WORLD LEVEL ANALYSIS :

Regular Expressions :

- The language used to specify text search strings is called a regular expression (RE).
- Using a unique syntax that is stored in a pattern, RE aids us in matching or finding other strings or sets of strings.
- Both in UNIX and MS Word, regular expressions are used similarly to search text.

Properties of Regular Expressions :

Followings are some of the important properties of RE –

=> American Mathematician Stephen Cole Kleene formalized the Regular Expression language.

- RE is a formula in a special language, which can be used for specifying simple classes of strings, a sequence of symbols. In other words, we can say that RE is an algebraic notation for characterizing a set of strings.
- Regular expression requires two things, one is the pattern that we wish to search and other is a corpus of text from which we need to search.

Mathematically, A Regular Expression can be defined as follows –

- **ε** is a Regular Expression, which indicates that the language is having an empty string.
- **φ** is a Regular Expression which denotes that it is an empty language.
- If **X** and **Y** are Regular Expressions, then

  X, Y
  X.Y(Concatenation of XY)
  X+Y (Union of X and Y)
  X*, Y* (Kleen Closure of X and Y)

  are also regular expressions.

- If a string is derived from above rules then that would also be a regular expression.

Examples of Regular Expressions

The following table shows a few examples of Regular Expressions –

Regular Expressions	**Regular Set**
(0 + 10*)	{0, 1, 10, 100, 1000, 10000, … }
(0*10*)	{1, 01, 10, 010, 0010, …}
(0 + ε)(1 + ε)	{ε, 0, 1, 01}
(a+b)*	It would be set of strings of a's and b's of any length which also includes the null string i.e. {ε, a, b, aa , ab , bb , ba, aaa…….}
(a+b)*abb	It would be set of strings of a's and b's ending with the string abb i.e. {abb, aabb, babb, aaabb, ababb, ……..}

(11)*	It would be set consisting of even number of 1's which also includes an empty string i.e. {ε, 11, 1111, 111111, …….}

Finite State Automata

- The plural form of the word automaton is automata, and an automaton is an abstract self-propelled computing device that automatically performs a predetermined sequence of operations.

- An automaton having a finite number of states is called a Finite Automaton (FA) or Finite State automata (FSA).

Mathematically, an automaton can be represented by a 5-tuple (Q, Σ, δ, q0, F), where −

Q is a finite set of states.

Σ is a finite set of symbols, called the alphabet of the automaton.

δ is the transition function

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

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

Types of Finite State Automation (FSA)

Finite state automation is of two types. Let us see what the types are.

Deterministic Finite automation (DFA)

It may be defined as the type of finite automation wherein, for every input symbol we can determine the state to which the machine will move. It has a finite number of states that is why the machine is called Deterministic Finite Automaton (DFA).

Mathematically, a DFA can be represented by a 5-tuple (Q, Σ, δ, q0, F), where −

$$\delta: Q \times \Sigma \rightarrow Q$$

Example of DFA

Suppose a DFA be

$Q = \{a, b, c\}$,

$\Sigma = \{0, 1\}$,

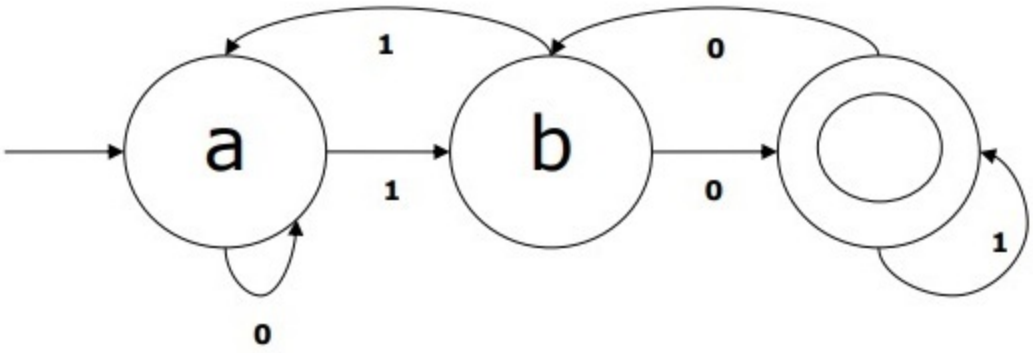
$q_0 = \{a\}$,

$F = \{c\}$,

=> Transition function δ is shown in the table as follows −

Current State	Next State for Input 0	Next State for Input 1
A	a	B
B	b	A
C	c	C

The graphical representation of this DFA would be as follows −



Non-deterministic Finite Automation (NDFA)

It may be defined as the type of finite automation where for every input symbol we cannot determine the state to which the machine will move i.e. the machine can move to any combination of the states. It has a finite number of states that is why the machine is called Non-deterministic Finite Automation (NDFA).

Mathematically, NDFA can be represented by a 5-tuple (Q, Σ, δ, q0, F), where –

$$\delta: Q \times \Sigma \rightarrow 2^{Q}$$

Example of NDFA

Suppose a NDFA be

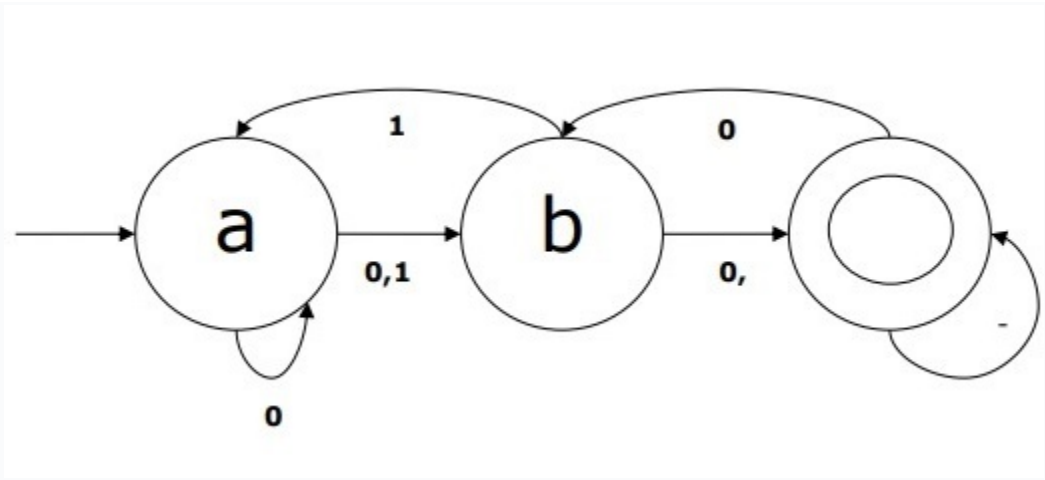
Q = {a, b, c},
Σ = {0, 1},
q0 = {a},
F = {c},

=> Transition function δ is shown in the table as follows –

Current State	Next State for Input 0	Next State for Input 1

A	a,b	B
B	C	a,c
C	b,c	C

The graphical representation of this NDFA would be as follows -



Graphically both DFA and NDFA can be represented by diagraphs called state diagrams where –

- The states are represented by vertices.
- The transitions are shown by labeled arcs.
- The initial state is represented by an empty incoming arc.
- The final state is represented by double circle.

Morphological Parsing

The term morphological parsing is related to the parsing of morphemes.

The challenge of realizing that a word can be broken down into linguistic structures known as morphemes (smaller meaningful units) is known as morphological parsing.

For example, we can break the word foxes into two, fox and -es. We can see that the word foxes, is made up of two morphemes, one is fox and other is -es.

In other sense, we can say that morphology is the study of –

- The formation of words.
- The origin of the words.
- Grammatical forms of the words.
- Use of prefixes and suffixes in the formation of words.
- How parts-of-speech (PoS) of a language are formed.

Types of Morphemes

Morphemes, the smallest meaning-bearing units, can be divided into two types –

- Stems
- Word Order

Stems

It is the core meaningful unit of a word. We can also say that it is the root of the word. For example, in the word foxes, the stem is fox.

Affixes – As the name suggests, they add some additional meaning and grammatical functions to the words. For example, in the word foxes, the affix is – es.

Further, affixes can also be divided into following four types –

Prefixes – As the name suggests, prefixes precede the stem. For example, in the word unbuckle, un is the prefix.

Suffixes – As the name suggests, suffixes follow the stem. For example, in the word cats, -s is the suffix.

Infixes – As the name suggests, infixes are inserted inside the stem. For example, the word cupful, can be pluralized as cupsful by using -s as the infix.

Circumfixes – They precede and follow the stem. There are very less examples of circumfixes in English language. A very common example is 'A-ing' where we can use -A precede and -ing follows the stem.

WORD ORDER

The order of the words would be decided by morphological parsing. Let us now see the requirements for building a morphological parser –

Lexicon

The very first requirement for building a morphological parser is lexicon, which includes the list of stems and affixes along with the basic information about them. For example, the information like whether the stem is Noun stem or Verb stem, etc.

Morphotactics

It is basically the model of morpheme ordering. In other sense, the model explaining which classes of morphemes can follow other classes of morphemes inside a word. For example, the morphotactic fact is that the English plural morpheme always follows the noun rather than preceding it.

Orthographic rules

These spelling rules are used to model the changes occurring in a word. For example, the rule of converting y to ie in word like city+s = cities not citys.

SPELLING ERROR DETECTION AND CORRECTION

- Spelling Correction is a very important task in NLP.

- It is used in various tasks like Search Engines, Sentiment Analysis, Text Summarization, etc..
- As the name suggests, we try to detect and correct spelling errors in spelling correction.
- In real-world NLP tasks,we often deal with data having typos, and their spelling correction comes to the rescue to improve model performance.
- For example, if we want to search apple and type “aple”, we will wish that the search engine suggests “apple” instead of giving no results.

SPELLING CORRECTION

- It is a process of detecting and sometimes providing suggestions for incorrectly spelled words in a text.
- Spelling errors can be divided into two categories: Real-word errors and Non-word errors. Real-word errors are those error words that are acceptable words in the dictionary. Non-word errors are those error words that cannot be found in the dictionary.

Different Approaches to Spelling Correction

NORVIG’S APPROACH

Peter Norvig (director of research at Google) described the following approach to spelling correction.

Let’s take a word and brute force all possible edits, such as delete, insert, transpose, replace and split. Eg. for word ***abc*** *possible candidates will be:* ***ab ac bc bac cba acb a_bc ab_c aabc abbc acbc adbc aebc*** *etc.*

Every word is added to a candidate list. We repeat this procedure for every word for a second time to get candidates with bigger edit distance (for cases with two errors).

Each candidate is estimated with unigram language model. For each vocabulary word frequencies are pre-calculated, based on some big text collections. The candidate word with highest frequency is taken as an answer.

- **ADDING MORE CONTEXT**
  We can use n-gram models with n>1, instead of the unigram language model, providing better accuracy. But, at the same time, the model becomes heavy due to a higher number of calculations.

- **INCREASING SPEED :** SymSpell Approach (Symmetric Delete Spelling Correction)
  Here we pre-calculate all delete typos, unlike extracting all possible edits every time we encounter any error. This will cost additional memory consumption.

- **IMPROVING MEMORY CONSUMPTION**

We require large datasets (at least some GBs) to obtain good accuracy. If we train the n-gram language models on small datasets like a few MBs, it leads to high memory consumption. The symspell index and the language models occupy half-half of the size. This increased usage is because we store frequencies instead of simple text.

We can compress our n-gram model by using hashing. A perfect hash is a hash that can never have collisions. We can use a perfect hash to store the counts of n-grams. As we don't have any collisions, we store the count frequencies instead of the original n-grams. We need to make sure that the hash of unknown words does not match with those of known words, so for that, we use a bloom filter with known words. A bloom filter utilizes space efficiently to check whether an element belongs to a set or not. We can reduce the memory usage by using a nonlinear quantization to pack the 32-bit long count frequencies into 16-bit values.

- **IMPROVING ACCURACY**

  We can use machine learning algorithms to improve accuracy further. We can start with a machine learning classifier that decides whether a given the word has an error or not. Then we can use a regression model to rank the words. This technique mimics the role of smoothing in language models as it uses all the grams as input, and then a classifier decides the rank, meaning how powerful each gram is.

  For word ranking, we will train a catboost(gradient boosted decision trees) ranking model with multiple features like word frequency, word length, edit distance between the source word and modified word(i.e., number of changes required to convert the source to modified), n-gram language model prediction, the frequencies of neighboring words, etc.

- We can also gather a large corpus with errors and their corresponding corrected text and then train a model to improve accuracy.

ERROR DETECTION TECHNIQUES

A. Dictionary Lookup Technique :

In this, Dictionary lookup technique is used which checks every word of input text for its presence in dictionary. If that word present in the dictionary, then it is a correct word. Otherwise it is put into the list of error words. The most common technique for gaining fast access to a dictionary is the use of a Hash Table. To look up an input string, one simply computes its hash addresses and retrieves the word stored at that address in the pre- constructed hash table. If the word stored at the hash address is different from the input string or is null, a misspelling is indicated.

B. N-gram Analysis Technique :

In this, N-grams are n-letter sub sequences of words or strings where n usually is one, two or three. One letter ngrams are referred to as unigrams or monograms; two letter n-grams are referred to as bi-grams and three letter n-grams as trigrams. In general, n-gram detection technique work by examining each n-gram is an input string and looking it up in a precompiled table of n-gram statistics to as certain either its existence or its frequency of words or strings that are found to contain nonexistence or highly infrequent n-grams are identified as either misspellings.

ERROR CORRECTION TECHNIQUES

A. Minimum Edit Distance Technique : The minimum edit distance is the minimum number of operations (insertions, deletions and substitutions) required to transform one text string into another. In its original form, minimum edit distance algorithms require m comparisons between misspelled string and the dictionary of m words. After comparison, the words with minimum edit distance are chosen as correct alternatives. Minimum edit distance has different algorithms are Levenshtein algorithm, Hamming, Longest Common Subsequence.

1) **The Levenshtein algorithm**: This algorithm is a weighting approach to appoint a cost of 1 to every edit operations (Insertion, deletion and substitution). For instance, the Levenshtein edit distance between “dog” and “cat” is 3 (substituting d by c, o by a, g by t).

2) **The Hamming algorithm**: This algorithm is measure the distance between two strings of equal length. For instance, the hamming distance between “sing” and “song” is 1 (changing i to o).

3) **The Longest Common Subsequence algorithm**: This algorithm is a popular technique to find out the difference between two words. The longest common subsequence of two strings is the mutual subsequence.

B. Similarity Key Technique: In this, Similarity key technique is to map every string into a key such that similarly spelled strings will have similar keys. Thus when key is computed for a misspelled string it will provide a pointer to all similarly spelled words in the lexicon.

1) **Soundex Algorithm**: This algorithm is used for indexing words based on their phonetic sound. Words with similar pronunciation but different meaning are coded similarly so that they can be matched regardless of trivial differences in their spelling.

2) **The SPEEDCOP System**: It is a way of automatically correcting spelling errorspredominantly typing errors in a very large database of scientific abstracts. A key

was computed for each word in the dictionary. This consisted of the first letter, followed by the consonants letters of the word, in the order of their occurrence in the word, followed by the vowel letters, also in the order of their occurrence, with each letter recorded only once.

C. Rule Based Technique : Rule Based Techniques are algorithms that attempt to represent knowledge of common spelling errors patterns in the form of rules for transforming misspellings into valid words. The candidate generation process consists of applying all applicable rules to a misspelled string and retaining every valid dictionary word those results.

D. **Probabilistic Techniques :** In this, two types of Probabilistic technique have been exploited.
1) **Transition Probabilities**: They represent that a given letter will be followed by another given letter. These are dependent. They can be estimated by collecting n-gram frequency statistic on a large corpus of text from the discourse.
2) **Confusion Probabilities**: They are estimates of how often a given letter is mistaken or substituted for another given letter. Confusion probabilities are source dependent because different OCR devices use different techniques and features to recognize characters, each device will have a unique confusion probability distribution.

E. N-gram Based Techniques : Letter n-grams, including tri-grams, bi-grams and unigrams have been used in a variety of ways in text recognition and spelling correction techniques. They have been used by OCR correctors to capture the lexical syntax of a dictionary and to suggest legal corrections.

F. Neural Net Techniques : Neural nets are likely candidates for spelling correctors because of their inherent ability to do associative recall based on incomplete or noisy input.
Back Propagation Algorithm : This algorithm is the most widely used algorithm for training a neural net. A typical back propagation net consists of three layers of node: input layer, an intermediate layer, an output layer. Each node in the input layer is connected by a weighted link to every node in the hidden layer. Similarly each node in the hidden layer is denoted by a weighted link to every node in the output layer. Input and output information is represented by on-off patterns of activity on the input and output nodes of the net. A 1 indicates that a node is turned on and 0 indicates that a node is turned off.

Word Classes:

In grammar, a part of speech or part-of-speech (POS) is known as **word class or grammatical category**, which is a category of words that have similar grammatical properties.

The English language has four major word classes: Nouns, Verbs, Adjectives, and Adverbs.

Commonly listed English parts of speech are nouns, verbs, adjectives, adverbs, pronouns, prepositions, conjunction, interjection, numeral, article, and determiners.

These can be further categorized into open and closed classes.

Closed Class:

Closed classes are those with a relatively fixed/number of words, and we rarely add new words to these POS, such as prepositions. Closed class words are generally functional words like of, it, and, or you, which tend to be very short, occur frequently, and often have structuring uses in grammar.

Example of closed class-

Determiners: a, an, the Pronouns: she, he, I, others Prepositions: on, under, over, near, by, at, from, to, with

Open Class:

Open Classes are mostly content-bearing, i.e., they refer to objects, actions, and features; it's called open classes since new words are added all the time.

By contrast, nouns and verbs, adjectives, and adverbs belong to open classes

Example of open class-

Nouns: computer, board, peace, school Verbs: say, walk, run, belong Adjectives: clean, quick, rapid, enormous Adverbs: quickly, softly, enormously, cheerfully

Part-of-Speech Tagging in NLP:

Tagging:

Tagging is a kind of classification that may be defined as the automatic assignment of description to the tokens. Here the descriptor is called tag, which may represent one of the part-of-speech, semantic information and so on

Part-of-speech (POS) tagging is an important Natural Language Processing (NLP) concept that categorizes words in the text corpus with a particular part of speech tag (e.g., Noun, Verb, Adjective, etc.)

POS tagging could be the very first task in text processing for further downstream tasks in NLP, like speech recognition, parsing, machine translation, sentiment analysis, etc.

Simply put, In Parts of Speech tagging for English words, we are given a text of English words we need to identify the parts of speech of each word.

Example Sentence : Learn NLP from Scaler

Learn -> ADJECTIVE NLP -> NOUN from -> PREPOSITION Scaler -> NOUN

Although it seems easy, Identifying the part of speech tags is much more complicated than simply mapping words to their part of speech tags.

Why Difficult ? :

Words often have more than one POS tag. Let's understand this by taking an easy example.

In the below sentences focus on the word "back" :

Sentence	Pos tag of Word "back"
The "back" door	Adjective
On my "back"	Noun

Win the voters “back”	Adverb
Promise to “back” the bill	Verb

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

Rules:

- 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:

- These taggers are knowledge-driven taggers.
- The rules in Rule-based POS tagging are built manually.
- We have some limited number of rules approximately around 1000.

- Smoothing and language modeling is defined explicitly in rule-based taggers.

Stochastic POS Tagging:

Stochastic POS Tagger uses probabilistic and statistical information from the corpus of labeled text (where we know the actual tags of words in the corpus) to assign a POS tag to each word in a sentence.

This tagger can use techniques like Word frequency measurements and Tag Sequence Probabilities. It can either use one of these approaches or a combination of both.

Word Frequency Measurements:

The tag encountered most frequently in the corpus is the one assigned to the ambiguous words(words having 2 or more possible POS tags).

Let's understand this approach using some example sentences :

Ambiguous Word = "play"

Sentence 1 : I play cricket every day. POS tag of play = VERB

Sentence 2 : I want to perform a play. POS tag of play = NOUN

The word frequency method will now check the most frequently used POS tag for "play". Let's say this frequent POS tag happens to be VERB; then we assign the POS tag of "play" = VERB

The main drawback of this approach is that it can yield invalid sequences of tags.

Tag Sequence Probabilities:

In this method, the best tag for a given word is determined by the probability that it occurs with “n” previous tags.

Simply put, assume we have a new sequence of 4 words, *w*1*w*2*w*3 *w4*

And we need to identify the POS tag of *w*4.

If n = 3, we will consider the POS tags of 3 words prior to w4 in the labeled corpus of text

Let’s say the POS tags for

*w*1= NOUN, *w*2= VERB , *w*3= DETERMINER

In short, N, V, D: NVD

Then in the labeled corpus of text, we will search for this NVD sequence.

Let’s say we found 100 such NVD sequences. Out of these -

10 sequences have the POS of the next word is NOUN 90 sequences have the POS of the next word is VERB

Then the POS of the word *w*4= VERB

The main drawback of this technique is that sometimes the predicted sequence is not Grammatically correct.

Some properties and limitations of the Stochastic tagging approach :

1. This POS tagging is based on the probability of the tag occurring (either solo or in sequence)
2. It requires labeled corpus, also called training data in the Machine Learning lingo

3. There would be no probability for the words that don't exist in the training data
4. It uses a different testing corpus (unseen text) other than the training corpus
5. It is the simplest POS tagging because it chooses the most frequent tags associated with a word in the training corpus

Transformation-Based Learning Tagger: TBL:

Transformation based tagging is also called Brill tagging. It is an instance of the transformation-based learning (TBL), which is a rule-based algorithm for automatic tagging of POS to the given text. TBL, allows us to have linguistic knowledge in a readable form, transforms one state to another state by using transformation rules.

It draws the inspiration from both the previous explained taggers − rule-based and stochastic. If we see similarity between rule-based and transformation tagger, then like rule-based, it is also based on the rules that specify what tags need to be assigned to what words. On the other hand, if we see similarity between stochastic and transformation tagger then like stochastic, it is machine learning technique in which rules are automatically induced from data.

Working of Transformation Based Learning(TBL):

It involves 3 steps:

- **Start with the solution** − The TBL usually starts with some solution to the problem and works in cycles.
- **Most beneficial transformation chosen** − In each cycle, TBL will choose the most beneficial transformation.
- **Apply to the problem** − The transformation chosen in the last step will be applied to the problem.

The algorithm will stop when the selected transformation in step 2 will not add either more value or there are no more transformations to be selected. Such kind of learning is best suited in classification tasks.

Advantages of Transformation-based Learning (TBL):

- Development as well as debugging is very easy in TBL because the learned rules are easy to understand.
- Complexity in tagging is reduced because in TBL there is interlacing of machine learned and human-generated rules.
- Transformation-based tagger is much faster than Markov-model tagger.

Disadvantages of Transformation-based Learning (TBL):

- Transformation-based learning (TBL) does not provide tag probabilities.
- In TBL, the training time is very long especially on large corpora.

CONTEXT -FREE GRAMMAR:

Context Free Grammar is formal grammar, the syntax or structure of a formal language can be described using context-free grammar (CFG), a type of formal grammar. Context-free grammar can also be seen as the list of rules that define the set of all well-formed sentences in a language.It is a notation for describing languages and a superset of Regular grammar.

The grammar has four tuples: (V,T,P,S).

V - It is the collection of variables or nonterminal symbols.

T - It is a set of terminals.

P - It is the production rules that consist of both terminals and nonterminals.

S - It is the Starting symbol.

A grammar is said to be the Context-free grammar if every production is in the form of :

$G \rightarrow (V \cup T)^*$, where $G \in V$

- And the left-hand side of the G, here in the example can only be a Variable, it cannot be a terminal.
- But on the right-hand side here it can be a Variable or Terminal or both combination of Variable and Terminal.

Above equation states that every production which contains any combination of the ‘V’ variable or ‘T’ terminal is said to be a context-free grammar.

For example the grammar A = { S, a,b, P,S} having production :

- Here S is the starting symbol.
- {a,b} are the terminals generally represented by small characters.
- P is variable along with S.

S-> aS

S-> bSa

but

a->bSa, or

a->ba is not a CFG as on the left-hand side there is a variable which does not follow the CFGs rule.

Context-free grammars are frequently used, especially in the areas of formal language theory, compiler development, and natural language processing. It is also used for explaining the syntax of programming languages and other formal languages.

Capabilities of CFG:

There are the various capabilities of CFG:

- Context free grammar is useful to describe most of the programming languages.

- If the grammar is properly designed then an efficient parser can be constructed automatically.
- Using the features of associatively & precedence information, suitable grammars for expressions can be constructed.
- Context free grammar is capable of describing nested structures like: balanced parentheses, matching begin-end, corresponding if-then-else's & so on.

What is Constituency Parsing in NLP?

Constituency parsing is an important concept in Natural Language Processing that involves analyzing the structure of a sentence grammatically by identifying the constituents or phrases in the sentence and their hierarchical relationships.
As we know that understanding natural language is a very complex task as we have to deal with the ambiguity of the natural language in order to properly understand the natural language.

Working of Constituency Parsing

For understanding natural language the key is to understand the grammatical pattern of the sentences involved. The first step in understanding grammar is to segregate a sentence into groups of words or tokens called constituents based on their grammatical role in the sentence.

Let's understand this process with an example sentence:

"The lion ate the deer." Here, "The lion" represents a *noun phrase*, "ate" represents a *verb phrase*, and "the deer" is another *noun phrase*.

Let's look at some of the more examples below:

The dog	ran	on the ground
The squirrel	jumped	across the fence
He	walked	Up the stairs
NP	**VBD**	**PP**

Applications of Constituency Parsing

Constituency parsing is a process of identifying the constituents (noun phrases, verbs, clauses, etc.) in a sentence and grouping them into a tree-like structure that represents the grammatical relationships among them.

The following are some of the applications of constituency parsing:

- Natural Language Processing (NLP) – It is used in various NLP tasks such as text summarization, machine translation, question answering, and text classification.

- Information Retrieval – It is used to extract information from large corpora and to index it for efficient retrieval.
- Text-to-Speech – It helps in generating human-like speech by understanding the grammar and structure of the text.
- Sentiment Analysis – It helps in determining the sentiment of a text by identifying positive, negative, or neutral sentiments in the constituents.
- Text-based Games and Chatbots – It helps in generating more human-like responses in text-based games and chatbots.
- Text Summarization – It is used to summarize large texts by identifying the most important constituents and representing them in a compact form.
- Text Classification – It is used to classify text into predefined categories by analyzing the constituent structure and relationships.

What is Parsing ?

Parsing is the process of examining the grammatical structure and relationships inside a given sentence or text in natural language processing (NLP).

- It involves analyzing the text to determine the roles of specific words, such as nouns, verbs, and adjectives, as well as their interrelationships.
- This analysis produces a structured representation of the text, allowing NLP computers to understand how words in a phrase connect to one another.
- Parsers expose the structure of a sentence by constructing parse trees or dependency trees that illustrate the hierarchical and syntactic relationships between words.

Types of Parsing in NLP :

1. Syntactic Parsing
2. Semantic Parsing

Syntactic Parsing :

- Syntactic parsing deals with a sentence's grammatical structure.
- It involves looking at the sentence to determine parts of speech, sentence boundaries, and word relationships.

The two most common approaches included are as follows:

1) Constituency Parsing:

- Constituency Parsing builds parse trees that break down a sentence into its constituents, such as noun phrases and verb phrases
- It displays a sentence's hierarchical structure, demonstrating how words are arranged into bigger grammatical units.

2) Dependency Parsing:

- Dependency parsing depicts grammatical links between words by constructing a tree structure in which each word in the sentence is dependent on another.
- It is frequently used in tasks such as information extraction and machine translation because it focuses on word relationships such as subject-verb-object relations.

Semantic Parsing :

- Semantic parsing goes beyond syntactic structure to extract a sentence's meaning or semantics.
- It attempts to understand the roles of words in the context of a certain task and how they interact with one another.
- Semantic parsing is utilized in a variety of NLP applications, such as question answering, knowledge base populating, and text understanding.

Parsing Techniques in NLP :

Natural Language processing provides us with two basic parsing techniques:

1. Top-Down parsing

2. Bottom –Up Parsing

1.Top-Down Parsing:

- A parse tree is a tree that defines how the grammar was utilized to construct the sentence. Using the top-down approach, the parser attempts to create a parse tree from the root node S down to the leaves.
- The procedure begins with the assumption that the input can be derived from the selected start symbol S.
- The next step is to find the tops of all the trees that can begin with S by looking at the grammatical rules with S on the left-hand side, which generates all the possible trees.
- Top-down parsing is a search with a specific objective in mind.
- It attempts to replicate the initial creation process by rederiving the sentence from the start symbol, and the production tree is recreated from the top down.
- Top-down, left-to-right, and backtracking are prominent search strategies that are used in this method.
- The search begins with the root node labeled S, i.e., the starting symbol, expands the internal nodes using the next productions with the left-hand side equal to the internal node, and continues until leaves are part of speech (terminals).
- If the leaf nodes, or parts of speech, do not match the input string, we must go back to the most recent node processed and apply it to another production.

2.Bottom-Up Parsing:

- Bottom-up parsing begins with the words of input and attempts to create trees from the words up, again by applying grammar rules one at a time.
- The parse is successful if it builds a tree rooted in the start symbol S that includes all of the input. Bottom-up parsing is a type of data-driven search. It attempts to reverse the manufacturing process and return the phrase to the start symbol S.
- It reverses the production to reduce the string of tokens to the beginning Symbol, and the string is recognized by generating the rightmost derivation in reverse.
- The goal of reaching the starting symbol S is accomplished through a series of reductions; when the right-hand side of some rule matches the substring of the input string, the substring is replaced with the left-hand side of the matched production, and the process is repeated until the starting symbol is reached.
- Bottom-up parsing can be thought of as a reduction process. Bottom-up parsing is the construction of a parse tree in postorder.

21pa1a5465

INTRODUCTION TO PROBABILISTIC PARSING:

Probabilistic parsing, in distinction to lexicalized parsing, is a type of parsing that uses probabilities and statistical models to determine the most likely parse for a sentence.

- The parser uses a set of rules and probabilities in probabilistic parsing to determine the most likely syntactic structure for a sentence.

- This kind of structure allows the parser to take into account the context of the sentence and the likelihood of different syntactic structures and select the most probable parse given the context.

Both lexicalized and probabilistic parsing can improve the accuracy and efficiency of natural language processing systems and are commonly used in many applications in NLP.

Probabilistic parsing has contributed to strengthening the field of NLP by showing that complex problems like parsing can be solved using data and machine learning.

Probabilistic parsing is a technique used to analyze language phenomena and determine the most likely parse for a given sentence.

Issues of Probabilistic Parsing :

Insufficient lexical conditioning
- Present in pre-terminal rules

Independence assumptions:
- Rule expansion is context-independent
  - Allows us to multiply probabilities

PREPARED BY:

21pa1a5465

21pa1a5492

21pa1a5494

21pa1a5497

21pa1a54a6

21pa1a54b3

21pa1a54b5

21pa1a54b9