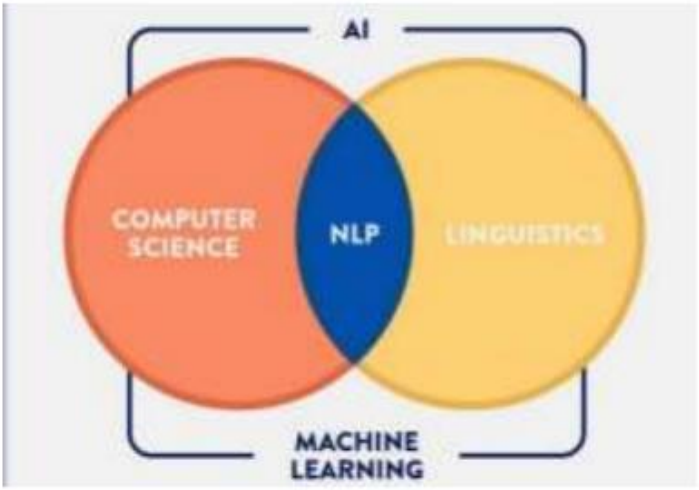
UNIT-I

What is Natural Language Processing (NLP)

Linguistics (Study of Human Speech): is concerned with language, it's formation, syntax, meaning, different kind of phrases (noun or verb).

Computer Science: is concerned with applying linguistic knowledge, by transforming it into computer programs with the help of sub-fields such as Artificial Intelligence (Machine Learning & Deep Learning).Natural language processing (NLP) is the intersection of computer science, linguistics and machine learning.

The field focuses on communication between computers and humans in natural language and NLP is all about making computers understand and generate human language.



- Humans communicate through some form of language either by text or speech.
- To make interactions between computers and humans, computers need to understand natural languages used by humans.
- Natural language processing is all about making computers learn, understand, analyse, manipulate and interpret natural (human) languages.
- NLP stands for Natural Language Processing, which is a part of Computer Science, Human language, and Artificial Intelligence.
- Processing of Natural Language is required when you want an intelligent system like robot to perform as per your instructions, when you want to hear decision from a dialogue based clinical expert system, etc.
- The ability of machines to interpret human language is now at the core of many applications that we use every day - chatbots, Email classification and spam filters, search engines, grammar checkers, voice assistants, and social language translators.
- The input and output of an NLP system can be Speech or Written Text

Advantages of NLP

- NLP helps users to ask questions about any subject and get a direct response within seconds.
- NLP offers exact answers to the question means it does not offer unnecessary and unwanted information.
- NLP helps computers to communicate with humans in their languages.
- It is very time efficient.
- Most of the companies use NLP to improve the efficiency of documentation processes, accuracy of documentation, and identify the information from large databases.

Why NLP

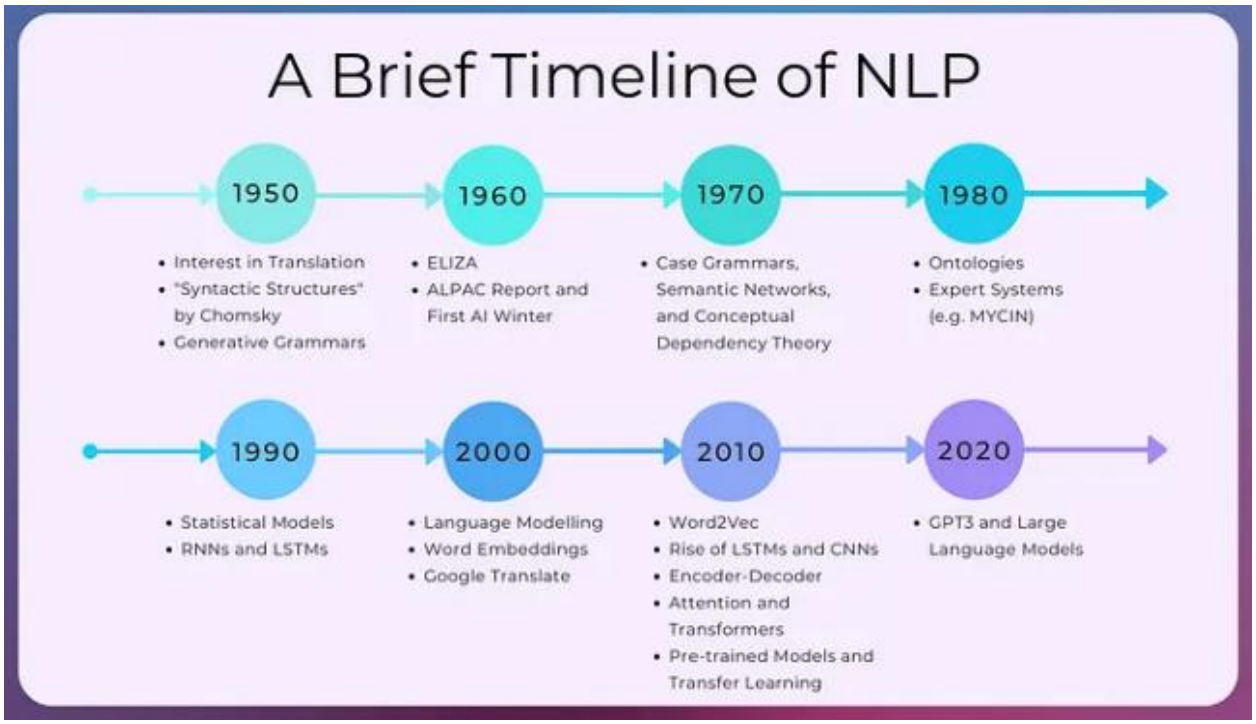
•Huge data from tweets, reviews, chats, queries etc.

•Most of them are unstructured.

•Hard for Human's to handle and manage them.

•To have a deep understanding of broad natural language.

Origins of NLP or History of NLP



(1940-1960) - Focused on Machine Translation (MT)The Natural Languages Processing started in the year 1940s.

1948 - In the Year 1948, the first recognisable NLP application was introduced in Birkbeck College, London.

1950s - In the Year 1950s, there was a conflicting view between linguistics and computer science. Now, Chomsky developed his first book syntactic structures and claimed that language is generative in nature.

In 1957, Chomsky also introduced the idea of Generative Grammar, which is rule based descriptions of syntactic structures.

(1960-1980) - Flavored with Artificial Intelligence (AI)

In the year 1960 to 1980, the key developments were:

Augmented Transition Networks (ATN)Augmented Transition Networks is a finite state machine that is capable of recognizing regular languages.

Case Grammar

Case Grammar was developed by Linguist Charles J. Fillmore in the year 1968. Case Grammar uses languages such as English to express the relationship between nouns and verbs by using the preposition.

In Case Grammar, case roles can be defined to link certain kinds of verbs and objects.

Ex:

"Neha broke the mirror with the hammer". In this example case grammar identify Neha as an agent, mirror as a theme, and hammer as an instrument. In the year 1960 to 1980, key systems were:

SHRDLU

SHRDLU is a program written by Terry Winograd in 1968-70. It helps users to communicate with the computer and moving objects. It can handle instructions such as "pick up the green boll" and also answer the questions like "What is inside the blackbox." The main importance of SHRDLU is that it shows those syntax, semantics, and reasoning about the world that can be combined to produce a system that understands a natural language.

LUNAR

LUNAR is the classic example of a Natural Language database interface system that is used ATNs and Woods' Procedural Semantics. It was capable of translating elaborate natural language expressions into database queries and handle 78% of requests without errors.

1980 – Current Till the year 1980, natural language processing systems were based on complex sets of hand-written rules. After 1980, NLP introduced machine learning algorithms for language processing.

In the beginning of the year 1990s, NLP started growing faster and achieved good process accuracy, especially in English Grammar. In 1990 also, an electronic text introduced, which provided a good resource for training and examining natural language programs. Other factors may include the availability of computers with fast CPUs and more memory. The major factor behind the advancement of natural language processing was the Internet.

Disadvantages of NLP

- NLP may not show context.
- NLP is unpredictableoNLP may require more keystrokes.
- NLP is unable to adapt to the new domain, and it has a limited function that's whyNLP is built for a single and specific task only.

Components of NLP

There are the following two components of NLP -

1. Natural Language Understanding (NLU)
   Natural Language Understanding (NLU) helps the machine to understand and analyse human language by extracting the metadata from content such as concepts, entities, keywords, emotion, relations, and semantic roles.NLU mainly used in Business applications to understand the customer's problem in both spoken and written language.
   NLU involves the following tasks –
   - It is used to map the given input into useful representation.
   - It is used to analyze different aspects of the language.
2. Natural Language Generation (NLG)
   Natural Language Generation (NLG) acts as a translator that converts the computerized data into natural language representation. It mainly involves Text planning, Sentence planning, and Text Realization. Note: The NLU is difficult than NLG

Applications of NLP

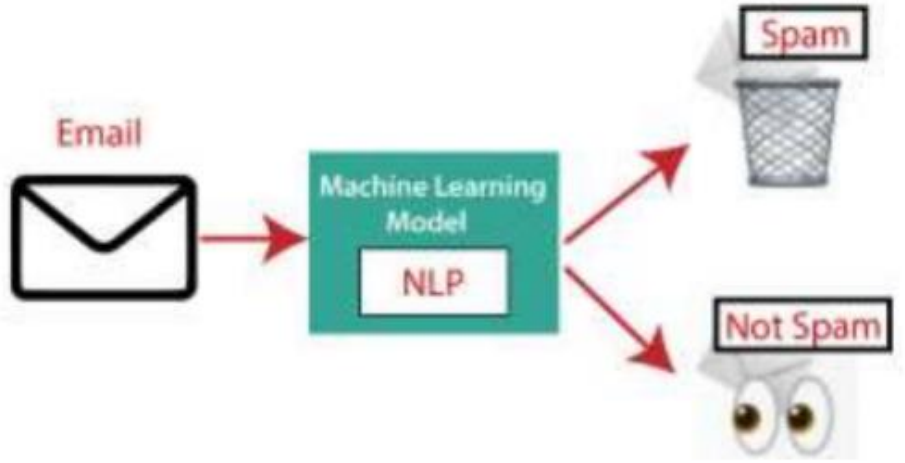
There are the following applications of NLP –

1. Question Answering

   Question Answering focuses on building systems that automatically answer the questions asked by humans in a natural language.

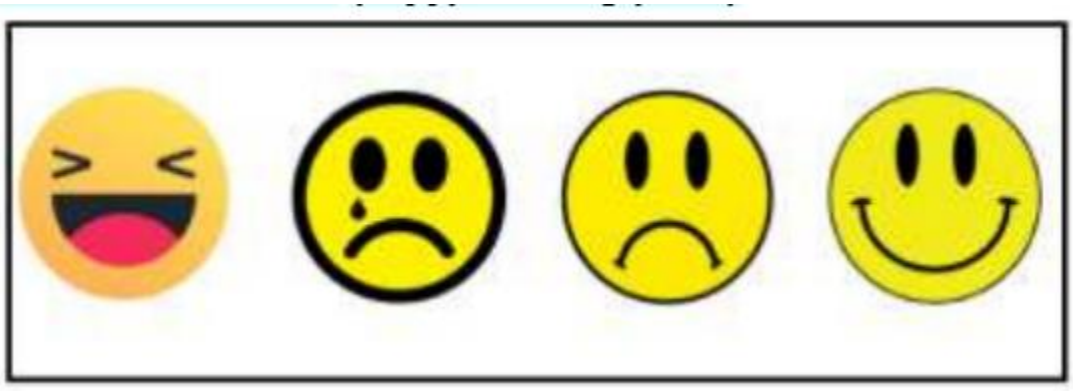
2. Spam Detection

   Spam detection is used to detect unwanted e-mails getting to a user's inbox.



3. Sentiment Analysis

   Sentiment Analysis is also known as opinion mining. It is used on the web to analyse the attitude, behaviour, and emotional state of the sender. This application is implemented through a combination of NLP (Natural Language Processing) and statistics by assigning the values to the text (positive, negative, or natural), identify the mood of the context (happy, sad, angry, etc.)



4. Machine Translation Machine translation is used to translate text or speech from one natural language to another natural language.

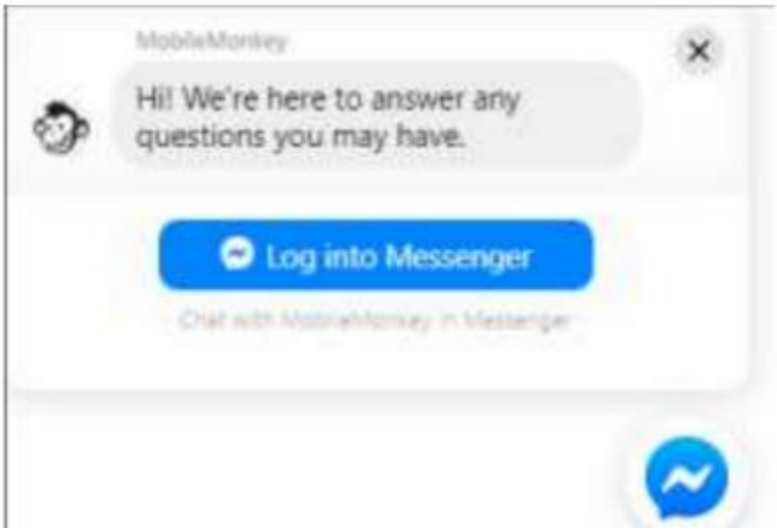
Example: Google Translator

5. Spelling correction Microsoft Corporation provides word processor software like MS-word, PowerPoint for the spelling correction.

6. Speech Recognition

Speech recognition is used for converting spoken words into text. It is used in applications, such as mobile, home automation, video recovery, dictating to Microsoft Word, voice biometrics, voice user interface, and so on.

7. Chatbot
   Implementing the Chatbot is one of the important applications of NLP. It is used by many companies to provide the customer's chat services.



8. Information extraction
   Information extraction is one of the most important applications of NLP. It is used for extracting structured information from unstructured or semi-structured machine readable documents.
9. Natural Language Understanding (NLU).
   It converts a large set of text into more formal representations such as first-order logic structures that are easier for the computer programs to manipulate notations of the natural language processing.

Language and Knowledge

i. How to build an NLP pipeline. There are the following steps to build an NLP pipeline.

Step1: Sentence Segmentation

Sentence Segment is the first step for building the NLP pipeline. It breaks the paragraph into separate sentences.

Example: Consider the following paragraph -Independence Day is one of the important festivals for every Indian citizen. It is celebrated on the 15th of August each year ever since India got independence from the British rule. The day celebrates independence in the true sense.

Sentence Segment produces the following result:

1. "Independence Day is one of the important festivals for every Indian citizen."

2. "Itis celebrated on the 15th of August each year ever since India got independence from the British rule."

3. "This day celebrates independence in the true sense."

Step2: Word Tokenization

Word Tokenizer is used to break the sentence into separate words or tokens.

Example: Java Tpoint offers Corporate Training, Summer Training, Online Training, and Winter Training.

Word Tokenizer generates the following result:

"JavaTpoint", "offers", "Corporate", "Training", "Summer", "Training", "Online","Training", "and", "Winter", "Training", "."

Step3: Stemming

Stemming is used to normalize words into its base form or root form. For example, celebrates, celebrated and celebrating, all these words are originated with a single root word "celebrate." The big problem with stemming is that sometimes it produces the root word which may not have any meaning.

For Example, intelligence, intelligent, and intelligently, all these words are originated with a single root word "intelligen." In English, the word "intelligen" do not have any meaning.

Step 4: Lemmatization

Lemmatization is quite similar to the Stemming. It is used to group different inflected forms of the word, called Lemma. The main difference between Stemming and lemmatization is that it produces the root word, which has a meaning.

For example: In lemmatization, the words intelligence, intelligent, and intelligently has a root word intelligent, which has a meaning.

Step 5: Identifying Stop Words In English, there are a lot of words that appear very frequently like "is", "and", "the", and "a". NLP pipelines will flag these words as stop words.

Stop words might be filtered out before doing any statistical analysis.

Example: He is a good boy. Note: When you are building a rock band search engine, then you do not ignore the word "The."

Step 6: Dependency Parsing Dependency Parsing is used to find that how all the words in the sentence are related to each other.

Step 7:

POS tags

POS stands for parts of speech, which includes Noun, verb, adverb, and Adjective. It indicates that how a word functions with its meaning as well as grammatically with in the sentences. A word has one or more parts of speech based on the context in which itis used.

Example: "Google" something on the Internet. In the above example, Google is used as a verb, although it is a proper noun.

Step 8:

Named Entity Recognition (NER)

Named Entity Recognition (NER) is the process of detecting the named entity such as person name, movie name, organization name, or location.

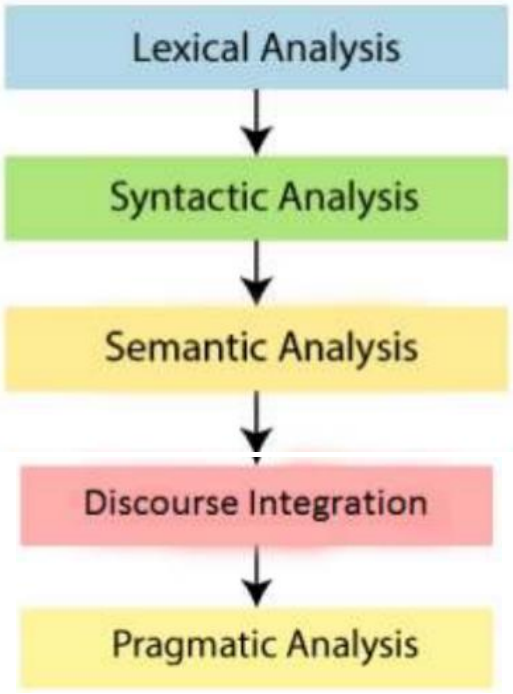
Example: Steve Jobs introduced iPhone at the Macworld Conference in San Francisco, California.

Step 9: Chunking

Chunking is used to collect the individual piece of information and grouping them intobigger pieces of sentences.

Phases of NLP

Natural Language Processing is separated into five primary stages or phases, starting with simple word processing and progressing to identifying complicated phrase meanings.



1. Lexical or Morphological Analysis

Lexical or Morphological Analysis is the initial step in NLP. It entails recognizing and analyzing word structures. The collection of words and phrases in a language is referred to as the lexicon.

Lexical analysis is the process of breaking down a text file into paragraphs, phrases, and words. The source code is scanned as a stream of characters and converted into intelligible lexemes in this phase.

It includes techniques as follows:

•Stop word removal (removing 'and', 'of', 'the' etc. from text)

•Tokenization (breaking the text into sentences or words)

a. Word tokenizer
b. Sentence tokenizer
c. Tweet tokenizer

•Stemming (removing 'ing', 'es', 's' from the tail of the words)

•Lemmatization (converting the words to their base forms)

2 Syntax Analysis or Parsing

Syntactic or Syntax analysis is a technique for checking grammar, arranging words,and displaying relationships between them. It entails examining the syntax of the words in the phrase

and arranging them in a way that demonstrates the relationship between them. Syntax analysis guarantees that the structure of a particular piece of text is proper. It tries to parse the sentence in order to ensure that the grammar is correct at the sentence level.

A syntax analyzer assigns POS tags based on the sentence structure given the probable POS created in the preceding stage. Some of the techniques used in this phase are:

- Dependency Parsing
- Parts of Speech (POS) tagging

3. Semantic Analysis

Semantic analysis is the process of looking for meaning in a statement. It concentrates mostly on the literal meaning of words, phrases, and a sentence is the main focus. It also deals with putting words together to form sentences. It extracts the text's exact meaning or dictionary definition. The meaning of the text is examined. It is accomplished by mapping the task domain's syntactic structures and objects.

Ex:

"The guava ate an apple." The line is syntactically valid, yet it is illogical because guavas cannot eat.

4. Discourse Integration

The term "discourse integration" refers to a feeling of context. The meaning of any sentence is determined by the meaning of the sentence immediately preceding it. In addition, it establishes the meaning of the sentence that follows. The sentences that come before it play a role in discourse integration. That is to say, that statement or word is dependent on the preceding sentence or words. It's the same with the use of proper nouns and pronouns.

Example: "John got ready at 9 AM. Later he took the train to California"

Here, the machine is able to understand that the word "he" in the second sentence is referring to "John".

5. Pragmatic Analysis

The fifth and final phase of NLP is pragmatic analysis. The overall communicative and social content, as well as its impact on interpretation, are the focus of pragmatic analysis.

Pragmatic Analysis uses a set of rules that describe cooperative dialogues to help you find the intended result. It covers things like word repetition, who said what to whom, and soon. It comprehends how people communicate with one another, the context in which they converse,

and a variety of other factors. It refers to the process of abstracting or extracting the meaning of a situation's use of language.

It translates the given text using the knowledge gathered in the preceding stages. "Switch on the TV" when used in a sentence, is an order or request to switch the TV on.

Some more examples:"Thank you for coming so late, we have wrapped up the meeting" (Contains sarcasm)"Can you share your screen?" (here the context is about computer's screen share during a remote meeting)

Challenges of NLP

Ambiguity in computational linguistics is a situation where a word or a sentence may have more than one meaning. That is, a sentence may be interpreted in more than one way. This leads to uncertainty in choosing the right meaning of a sentence especially while processing natural languages by computer.

•Ambiguity is a challenging task in natural language understanding (NLU).

•The process of handling the ambiguity is called as disambiguation.

•Ambiguity presents in almost all the steps of natural language processing. (Steps of NLP – lexical analysis, syntactic analysis, semantic analysis, discourse analysis, and pragmatic analysis).

Ex:

"Raj tried to reach his friend on the mobile, but he didn't attend"

In this sentence, we have the presence of lexical, syntactic, and anaphoric ambiguities.

Lexical ambiguity – The word "tried" means "attempted" not "judged" or "tested". Also, the word "reach" means "establish communication" not "gain" or "pass" or "strive".

Syntactic ambiguity – The phrase "on the mobile" attached to "reach" and thus means "using the mobile". It is not attached to "friend".

Anaphoric ambiguity – The anaphor "he" refers the "friend" not "Raj".

i. Lexical ambiguity

It is class of ambiguity caused by a word and its multiple senses especially when the word is part of sentence or phrase. A word can have multiple meanings under different part of speech categories. Also, under each POS category they may have multiple different senses.

Lexical ambiguity is about choosing which sense of a particular word under a particular POS category. In a sentence, the lexical ambiguity is caused while choosing the right sense of a word under a correct POS category.

For example, let us take the sentence “I saw a ship”. Here, the words “saw” and “ship”would mean multiple things as follows;

Saw = present tense of the verb saw (cut with a saw) OR past tense of the verb see(perceive by sight) OR a noun saw (blade for cutting) etc.

According to WordNet, the word “saw” is defined under 3 different senses in NOUN category and under 25 different senses in VERB category.

Ship = present tense of the verb ship (transport commercially) OR present tense of the verb ship (travel by ship) OR a noun ship (a vessel that carries passengers) etc.

As per WordNet, the word “ship” is defined with 1 sense under NOUN category and 5 sensesunder VERB category.

Due to multiple meanings, there arises an ambiguity in choosing the right sense of “saw”and “ship”.

Handling lexical ambiguity

Lexical ambiguity can be handled using the tasks like POS tagging and Word Sense Disambiguation.

ii. Syntactic ambiguity

It is a type of ambiguity where the doubt is about the syntactic structure of the sentence. That is, there is a possibility that a sentence could be parsed in many syntactical forms (a sentence may be interpreted in more than one way). The doubt is about which one among different syntactical forms is correct.

Ex:

The sentence “old men and women” is ambiguous. Here, the doubt is that whether the adjective old is attached with both men and women or men alone. Phrase attachment. "Mary ate a salad with spinach from Califonia for lunch on Tuesday." "with spinach" can attach to "salad" or "ate".

"from California" can attach to "spinach", "salad", or "ate"."for lunch" can attach to "California", "spinach", "salad", or "ate" and "on Tuesday" can attach to "lunch", "California", "spinach", "salad" or "ate".(Crossovers are not allowed, so you cannot both attach "on Tuesday" to "spinach" and attach "for lunch" to salad. Nonetheless there are 42 possible different parse trees.)`

iii. Semantic ambiguity

Even after the syntax and the meanings of individual words are resolved, still there aremore than one way of reading a sentence.

For example, the sentence “the dog has beendomesticated for more than 1000 years” could be semantically interpreted as follows;

1. A particular dog has been domesticated or

2. The dog species has been domesticated.

Semantic ambiguity is an uncertainty that occurs when a word, phrase or sentence hasmore than one interpretation.

iv. Anaphoric ambiguity

Anaphora in linguistics is about referring backwards (or an entity in another context) ina text.“Suresh kicked the ball. It went out of the stadium”

In this sentence, the pronoun “it” is an anaphor. This anaphor refers to the entity “theball” in the previous sentence. The entity is described as the antecedent of anaphor “it”.

This example is simple and does not show any ambiguity.

•Anaphors are mostly pronouns, or noun phrases in some cases.

Example: “Darshan plays keyboard. He loves music”. In this sentence, the anaphor “He”is a pronoun.

“A puppy drank the milk. The cute little dog was satisfied”. In this sentence, the anaphor “The cute little dog” is a noun pharse.

•Anaphoric references may not explicitly present in the previous sentence. Instead itmay refer a part of an entity (antecedent) in the previous sentence.

Example: “I went to the hospital, and they told me to go home and rest”. Here, the anaphor “they” refers not to the “hospital” directly, instead to the “hospital staff”.

Anaphors may not be in the immediately previous sentence. They may present in the sentences before the previous one or may present in the same sentence.

Example: “The horse ran up the hill. It was very steep. It soon got tired”. Here, the anaphor “it” of the third sentence refers the “horse” in the first sentence

v. Pragmatic ambiguity

Pragmatics focuses on conversational implicature. Conversational implicature is a process in which the speaker implies and a listener infers. Simply, it is a study about the sentences that are not directly spoken. It is the study of how people use language. The pragmatic level of linguistic processing deals with the use of real-world knowledge and understanding how this impacts the meaning of what is being communicated. By analyzing the contextual dimension of the documents and queries, a more detailed representation is derived.

Difference between semantics and pragmatics

•Semantics is about literal meaning of the words and their interrelations, where as pragmatics focuses on the inferred meaning that the speakers and listeners perceive.

•Semantics is the study of meaning, or more precisely, the study of the relation between linguistic expressions and their meanings. Pragmatics is the study of context, or moreprecisely, a study of the way context can influence our understanding of linguistic utterances.

Example:

Sentence	*Direct meaning (semantic meaning)*	*Other meanings (pragmatic meanings)*
Do you know what time is it?	Asking for the current time	Expressing anger to someone who missed the due time or something `
Will you crack open the door? I am getting hot	To break	Open the door just a little
The chicken is ready to eat	The chicken is ready to eat its breakfast, for example.	The cooked chicken is ready to be served

---

Processing Indian Languages

Natural language processing has the potential to broaden the online access for Indian citizens due to significant advancements in high computing GPU machines, high-speed internet availability and increased use of smart phones.

According to a survey, the consumers pointed out the benefits of the chatbots, among which 55% of people thought getting answers to simple questions was one of the significant benefits. Still, when it comes to India, that's challenging as languages in India aren't that simple.

As Indian languages pose many challenges for NLP like ambiguity, complexity, language grammar, translation problems, and obtaining the correct data for the NLP algorithms, it creates a lot of opportunities for NLP projects in India.

Top NLP libraries for Indian Languages

NLTK (Natural Language Toolkit for Indic Languages)

- NLTK provides support for various NLP applications in Indic languages.

The languages supported are Hindi (hi), Punjabi (pa), Sanskrit (sa), Gujarati (gu), Kannada (kn), Malayalam (ml), Nepali (ne), Odia (or), Marathi (mr), Bengali (bn), Tamil (ta), Urdu (ur), English (en).

- NLTK is like the NLTK Python package. It provides the feature for NLP tasks such as tokenisation and vector embedding for input text with an easy API interface.

Indic NLP Library:

The Indian languages have some difficulties which come from sharing a lot of similarity in terms of script, phonology, language syntax, etc., and this library provides a general solution.

Indic NLP Library provides functionalities like text normalisation, script normalisation, tokenization, word segmentation, romanistion, indicisation, script conversion, transliteration and translation.

Languages supported:

- Indo-aryan:

Assamese (asm), Bengali (ben), Gujarati (guj), Hindi/Urdu (hin/urd), Marathi (mar), Nepali (nep), Odiaa (ori), Punjabi (pan).

- Dravidian:

Sindhi (snd), Sinhala (sin), Sanskrit (san), Konkani (kok), Kannada (kan), Malayalam (mal), Teugu (tel), Tami (tam).

- Others:

English (eng).

Tasks handled:

- It handles bilingual tasks like Script conversions for languages mentioned above except Urdu and English.
- Monolingual tasks
- This language supports languages like Konkani, Sindhi, Telugu and some others which aren't supported by NLTK library.
- Transliteration amongst the 18 above mentioned languages.
- Translation amongst ten languages.

Top datasets for NLP (Indian languages)

- Semantic Relations from Wikipedia: Contains automatically extracted semantic relations from multilingual Wikipedia corpus.
- HC Corpora (Old Newspapers): This dataset is a subset of HC Corpora newspapers containing around 16,806,041 sentences and paragraphs in 67 languages including Hindi.
- Sentiment Lexicons for 81 Languages: This dataset contains positive and negative sentiment lexicons for 81 languages which also includes Hindi.
- IIT Bombay English-Hindi Parallel Corpus: This dataset contains parallel corpus for English-Hindi and monolingual Hindi corpus. This dataset was developed ar the Center for Indian Language Technology.
- Indic Languages Multilingual Parallel Corpus: This parallel corpus covers 7 Indic languages (in addition to English) like Bengali, Hindi, Malayalam, Tamil, Telugu, Sinhalese, Urdu.
- Microsoft Speech Corpus (Indian languages)(Audio dataset): This corpus contains conversational, phrasal training and test data for Telugu, Gujarati and Tamil.
- Hindi Speech Recognition Corpus(Audio Dataset): This is a corpus collected in India consisting of voices of 200 different speakers from different regions of the country. It also contains 100 pairs of daily spontaneous conversational speech data.

Early NLP Systems

1950s

The Birth of NLP: In the 1950s, computer scientists began to explore the possibilities of teaching machines to understand and generate human language. One prominent example from this era is the "Eliza" program developed by Joseph Weizenbaum in 1966. Eliza was a simple chatbot designed to simulate a conversation with a psychotherapist. While Eliza's responses were pre-scripted, people found it surprisingly engaging and felt like they were interacting with an actual human.

1960s-1970s

Rule-based Systems: During the 1960s and 1970s, NLP research focused on rule-based systems. These systems used a set of predefined rules to analyse and process text. One notable example is the "SHRDLU" program developed by Terry Winograd in 1970. SHRDLU was a natural language understanding system that could manipulate blocks in a virtual world. Users could issue commands like "Move the red block onto the green block," and SHRDLU would execute the task accordingly. This demonstration highlighted the potential of NLP in understanding and responding to complex instructions.

1980s-1990s

Statistical Approaches and Machine Learning: In the 1980s and 1990s, statistical approaches and machine learning techniques started gaining prominence in NLP. One groundbreaking example during this period is the development of Hidden Markov Models (HMMs) for speech recognition. HMMs allowed computers to convert spoken language into written text, leading to the development of speech-to-text systems. This breakthrough made it possible to dictate text automatically and have it transcribed, revolutionising fields like transcription services and voice assistants.

2000s-2010s

Deep Learning and Neural Networks: The 2000s and 2010s witnessed the rise of deep learning and neural networks, propelling NLP to new heights. One of the most significant breakthroughs was the development of word embeddings, such as Word2Vec and GloVe. These models represented words as dense vectors in a continuous vector space, capturing semantic relationships between words. For example, words like "king" and "queen" were represented as vectors that exhibited similar geometric patterns, showcasing their relational meaning.

2017

In 2017, Google introduced Google Translate's neural machine translation (NMT) system, which used deep learning techniques to improve translation accuracy. The system provided more fluent

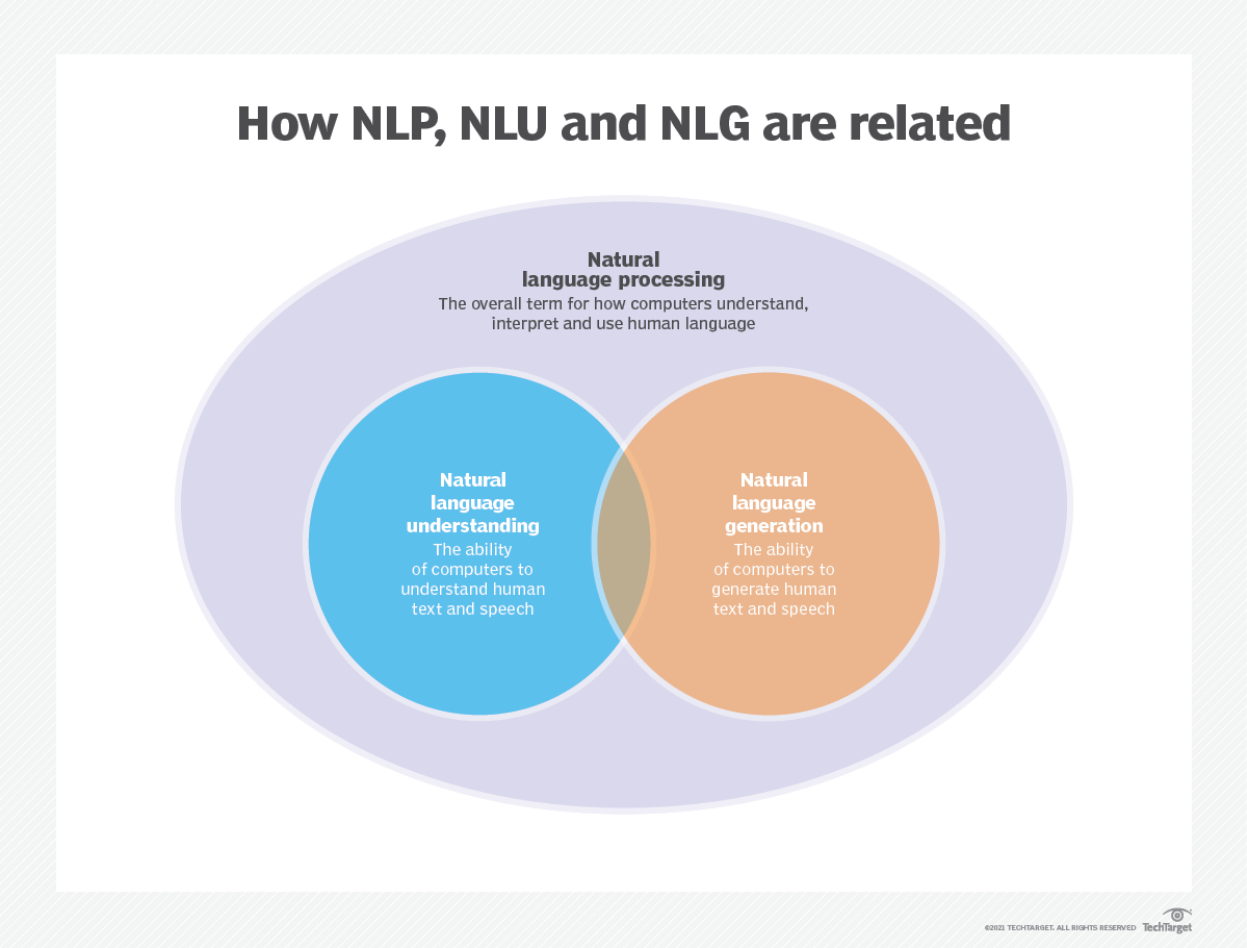
and accurate translations compared to traditional rule-based approaches. This development made it easier for people to communicate and understand content across different languages.

Present Day

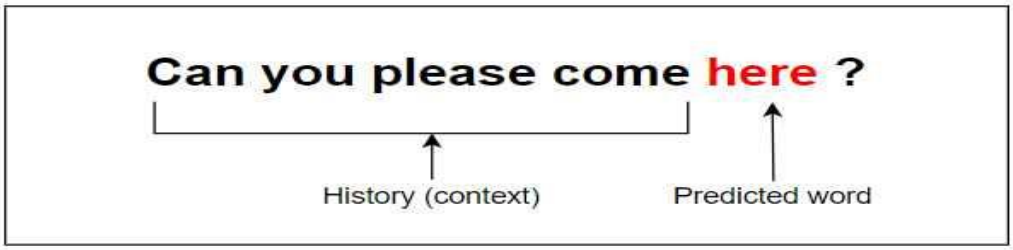
Transformer Models and Large Language Models: In recent years, transformer models like OpenAI's GPT (Generative Pre-trained Transformer) have made significant strides in NLP. These models can process and generate human-like text by capturing the contextual dependencies within large amounts of training data. GPT-3, released in 2020, demonstrated the ability to generate coherent and contextually relevant text across various applications, from creative writing to customer support chat bots.

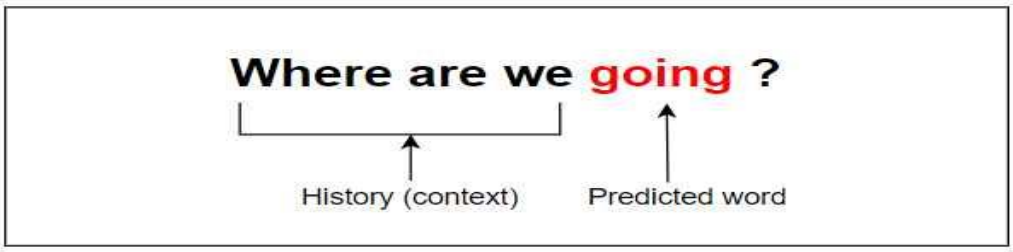
Introduction to Language Modeling

Language modelling (LM) **analyses bodies to text** to provide a foundation for word prediction. These models use statistical and probabilistic techniques to determine the **probability of a particular word sequence** occurring in a sentence.



- In text generation, a language model completes a sentence by generating text based on the incomplete input sentence. This is the idea behind the autocomplete feature when texting on a phone or typing in a search engine. The model will give suggestions to complete the sentence based on the words it predicts with the highest probabilities.



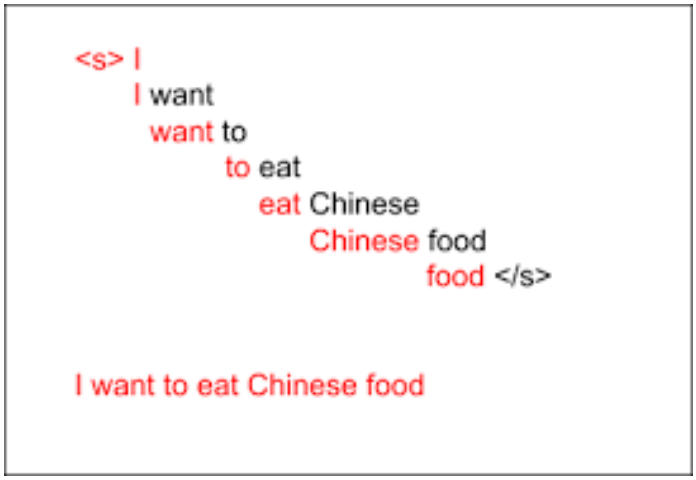


Various Grammar-based Language Models

There are two categories that Language Models fall under:

a) Statistical Language Models:

These models use traditional statistical techniques like N-grams, Hidden Markov Models (HMM), and established linguistic rules to learn the probability distribution of words. Statistical Language Modeling involves the development of probabilistic models that can predict the next word in the sequence given the words that precede it.



Statistical language models are as follows

N-gram Language Models:

The n-gram model is a probabilistic language model that can predict the next item in a sequence using the (n − 1)–order Markov model. Let's understand that better with an example. Consider the following sentence:

"I love reading blogs on Educative to learn new concepts"

A **1-gram** is a one-word sequence. For the above sentence, the unigrams would simply be: "I", "love", "reading", "blogs", "on", "Educative", "and", "learn", "new", "concepts".

A **2-gram** (or bigram) is a two-word sequence of words, like "I love", "love reading", "on Educative" or "new concepts".

A **3-gram** (or trigram) is a three-word sequence of words, like "I love reading", "blogs on Educative", or "learn new concepts".

An N-gram language model predicts the probability of a given N-gram within any sequence of words in the language. If we have a good N-gram model, we can predict $p(w|h)$, or the probability of seeing the word w given a history of previous words h, where the history contains n-1 words.

Example: "I love reading ___". Here, we want to predict what word will fill the dash based on the probabilities of the previous words.

We must estimate this probability to construct an N-gram model. We compute this probability in two steps:

Apply the chain rule of probability

We then apply a very strong simplification assumption to allow us to compute p(w1…ws) in an easy manner.

The chain rule of probability is:

$$p(w1...ws) = p(w1).p(w2|w1).p(w3|w1w2).p(w4|w1w2w3).....p(wn|w1...wn-1)$$

What is the chain rule? It tells us how to compute the joint probability of a sequence by using the conditional probability of a word given previous words.

Here, we do not have access to these conditional probabilities with complex conditions of up to n-1 words.So, how do we proceed? This is where we introduce a simplification assumption. We

can assume for all conditions, that:

$$p(wk|w1...wk-1) = p(wk|wk-1)$$

Here, we approximate the history (the context) of the word wk by looking only at the last word of the context. This assumption is called the Markov assumption. It is an example of the Bigram model. The same concept can be enhanced further for example for trigram model the formula will be:

$$p(wk|w1...wk-1) = p(wk|wk-2wk-1)$$

These models have a basic problem: they give the probability to zero if an unknown word is seen, so the concept of smoothing is used. In smoothing we assign some probability to the unseen words. There are different types of smoothing techniques such as Laplace smoothing, Good Turing, Kneser-ney smoothing.

b) Neural Language Models:

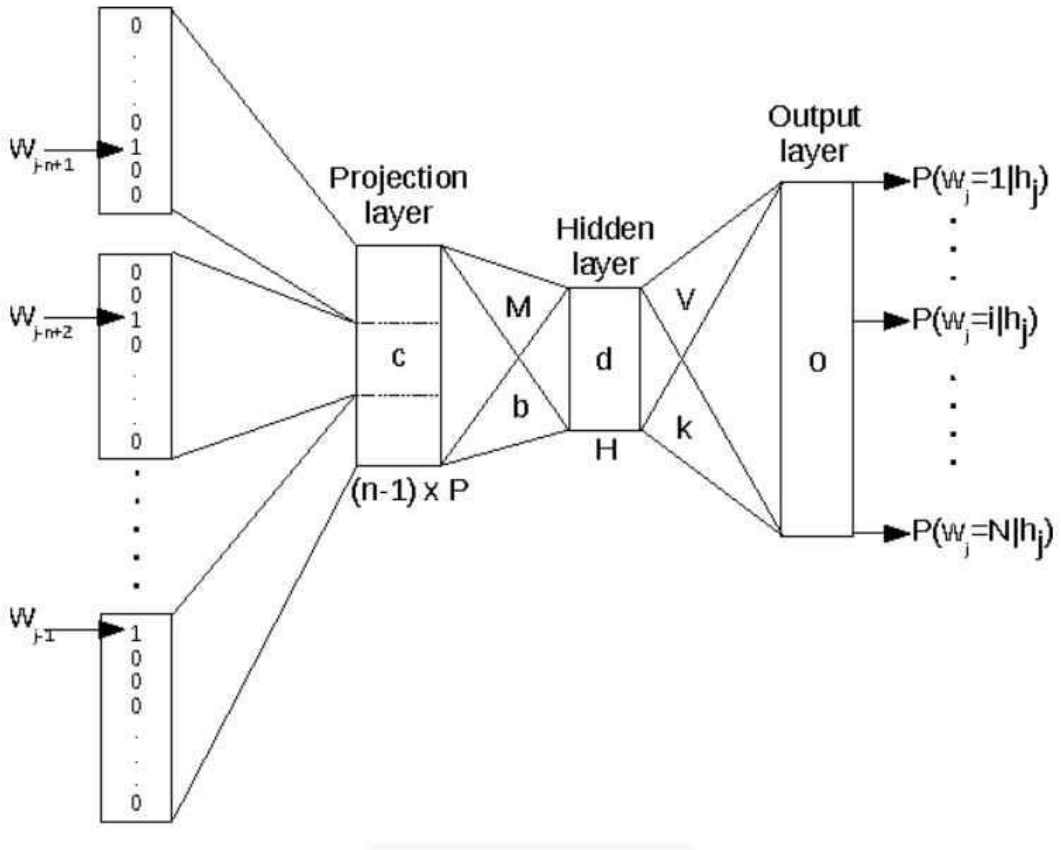
These models are new players in the NLP world and have surpassed the statistical language models in their effectiveness. They use different kinds of Neural Networks to model language. The use of neural networks in the development of language models has become so popular that it is now the preferred approach for challenging tasks like speech recognition and machine translation.

Neural language models have some advantages over probabilistic models. For example, they don't need smoothing, they can handle much longer histories, and they can generalise over contexts of similar words

For a training set of a given size, a neural language model has much higher predictive accuracy than an n-gram language model.

On the other hand, there is a cost for this improved performance: neural net language models are strikingly slower to train than traditional language models, and so for many tasks an N-gram language model is still the right tool.

In neural language models, the prior context is represented by embeddings of the previous words. This allows neural language models to generalise unseen data much better than N-gram language models.



Word embeddings are a type of word representation that allow words with similar meaning to have a similar representation. Word embeddings are, in fact, a class of techniques where individual words are represented as real-valued vectors in a predefined vector space.

Each word is mapped to one vector, and the vector values are learned in a way that resembles a neural network. Each word is represented by a real-valued vector, often tens or hundreds of dimensions.

The Neural language models were first based on RNNs and word embeddings. Then the concept of LSTMs, GRUs and Encoder-Decoder came along. The recent advancement is the discovery of Transformers, which has changed the field of Language Modelling drastically.

The RNNs were then stacked and used bidirectionally, but they were unable to capture long term dependencies. LSTMs and GRUs were introduced to counter this drawback.

The transformers form the basic building blocks of the new neural language models. The concept of transfer learning was introduced which was a major breakthrough. The models were pre-trained using large datasets.

For example, BERT is trained on the entire English Wikipedia. Unsupervised learning was used for training of the models. GPT-2 is trained on a set of 8 million web pages. These models are then fine-tuned to perform different NLP tasks.

Grammar based Language models

Due to the smoothing techniques, bigram and trigram language models are robust and have been successfully used more widely in speech recognition than conventional grammars like context free or even context sensitive grammars. Although these grammars are expected to better capture the inherent structures of the language, they have a couple of problems:

- *robustness*: the grammar must be able to handle a vocabulary of 10000 or more words, and ultimately a non-zero probability must be assigned to each possible word sequence.
- *ambiguity*: while *m*-gram language models avoid any ambiguity in parsing, context free grammars are typically ambiguous and thus produce more than a single parse tree .

When using context free grammars , the terminals or symbols that can be observed are the words of the vocabulary .We use the notation:

- *A*,*B*,*C*,..: non-terminals or abstract syntactic categories;
- *u*,*v*,*w*,..: terminals, i.e. the spoken words $w_1 \ldots w_n \ldots w_N$;
- α: an arbitrary nonempty string of terminals and non-terminals.

We typically attach probabilities to the context free rules. For a production rule like

$$A \rightarrow \alpha$$

we have a conditional probability $p(A \rightarrow \alpha | A)$ that a given non-terminal *A* will be rewritten as α. As usual, these probabilities must be normalised:

$$\sum_{\alpha} p(A \rightarrow \alpha | A) = 1$$

As usual in the framework of context free grammars , one often uses so-called normal forms like Chomsky and Greibach normal form for which standard introductions to formal languages in compiler construction theory can be consulted. Typically, these normal forms simplify the analysis of the problem under consideration, and there are automatic techniques for converting arbitrary context free grammars into these normal forms.

For each parse tree of a given string, we can compute its probability as the product over all its production rules. The probability of the given word string is then obtained by summing these probabilities over all possible parse trees.

For the application of stochastic context free grammars, stochastic grammar we consider four questions or problems:

- *parsing problem*: How to find the best parse tree along with its probability? How to efficiently calculate the total probability of all parse trees for a given sentence? The solution is given by the stochastic extension of the CYK algorithm, which is sometimes referred to as *inside algorithm*.
- *predicting the next word*: How can we use a grammar to predict the next word, i.e. given the word sequence $w_1 \ldots w_{n-1}$, how can we compute the conditional probability for the word w_n in position *n*:

  $$Pr(w = w_n | w_1 \ldots w_{n-1}) ?$$

  The solution is provided by the so-called *left corner algorithm* by Jelinek and Lafferty .
- *learning the rule probabilities*: Given the rules of a grammar, how can we estimate the rule probabilities from training sentences ? The solution is provided by the corresponding version of the EM algorithm , which is called the *inside-outside algorithm*
- *grammatical inference*: How can we learn the grammar as a whole from a set of training data .It is this problem for which no good solutions exist yet.

The main stumbling block in using grammar based language models so far seems to be that the grammar rules as such are not available, either because handwritten grammars are simply not good enough or because the problem of grammatical inference is too big.

To mitigate these problems, there have been attempts to use special types of context free grammars which are referred to as lexicalised or link grammars.

Their non-terminals are assumed to depend on the words of the lexicon.

These grammars include the bigram and trigram language models as special cases and thus provide a smooth transition from the *m*-gram language models to context free grammars. Although so far there are only preliminary results, we will consider these approaches in more detail because they offer promising extensions from *m*-gram models and they provide a good framework to show the relationship between *m*-gram language models, finite state grammars and context free grammars.

To describe this type of grammar, we will use a special version of the Greibach normal form. Each context free grammar can be converted into a grammar whose rules are of the following three types:

$$A \rightarrow w$$

$$A \rightarrow wB$$

$$A \rightarrow wBC$$

The third rule type describes a branching process, which is required for the full power of context free grammars. Without it, we are restricted to regular grammars which can be conveniently represented by finite state networks. Each network state is identified with a non-terminal of the grammar.

To obtain a bigram language model in this framework, we use a separate non-terminal A_w for each word *w*. The bigram model itself is expressed by the following rule type for the word pair (*vw*):

$$A_v \rightarrow wA_w$$

The bigram rules can be represented as a finite state network as illustrated in Figure 7.5 for a given word sequence. Unlike the conventional representation, the nodes stand for the observed words, i.e. the terminals, whereas the links represent the non-terminals. For a complete description of the grammar, we have to include the rules for sentence begin and end. For a trigram model, we have to use a separate non-terminal for each word pair (*vw*), and the rules depend on the word triples (*uvw*):

$$A_{uv} \rightarrow wA_{vw}$$

The trigram rules are illustrated in Figure 7.6.

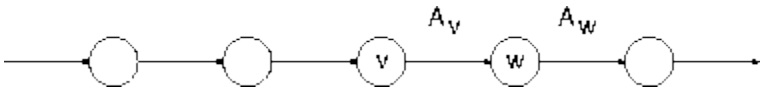


Figure 7.5: Bigram as a finite state grammar

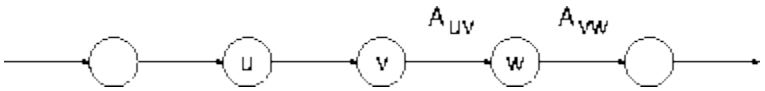


Figure 7.6: Trigram as a finite state grammar

To go beyond the power of *m*-gram and finite state language models , we need more than one non-terminal on the right-hand side. To simplify the presentation, we restrict ourselves to the special case of pairwise word dependencies, which can be viewed as an extended bigram model. In addition to the standard bigram non-terminals now denoted by L_w, we have additional non-terminals for the branching process, which are denoted by R_w for word *w*:

$$L_v \rightarrow w \quad (halt\ rule)$$

$$L_v \rightarrow wL_w \quad (step\ rule)$$

$$L_v \rightarrow wL_wR_w \quad (branch\ rule)$$

The production rule for non-terminal R_w is different from the rules for L_w and, for example, could have this form:

$$R_w \rightarrow yL_y$$

This type of rule is capable of handling long range dependencies such as those in the following English examples [Della Pietra et al. (1994)]:

between ...and ...

neither ...nor ...

to describe ...as ...

to prevent ...from ...

A conventional bigram or trigram language model cannot capture these dependencies. In general, for a given word string, context free grammars define a hierarchical structure by the parsing process.

Figure 7.7 shows the parsing structure in the conventional form.

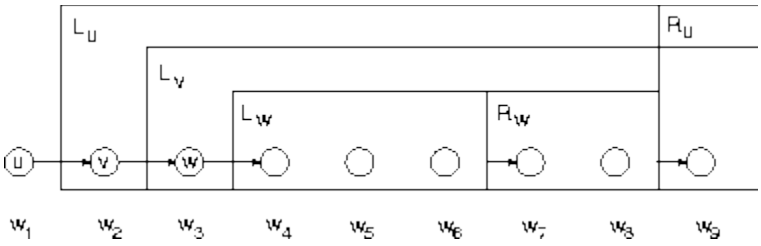


Figure 7.7: Parsing in the conventional form

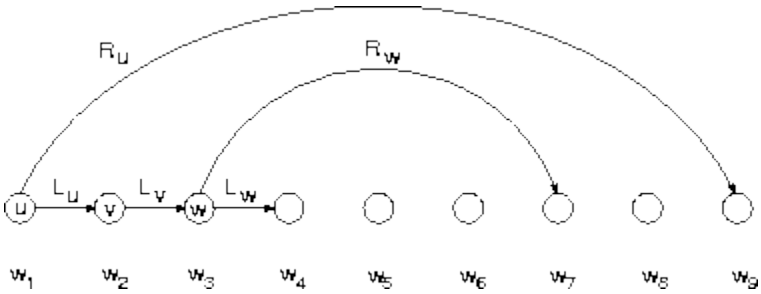


Figure 7.8: Parsing in link grammar form

UNIT-II

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

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

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

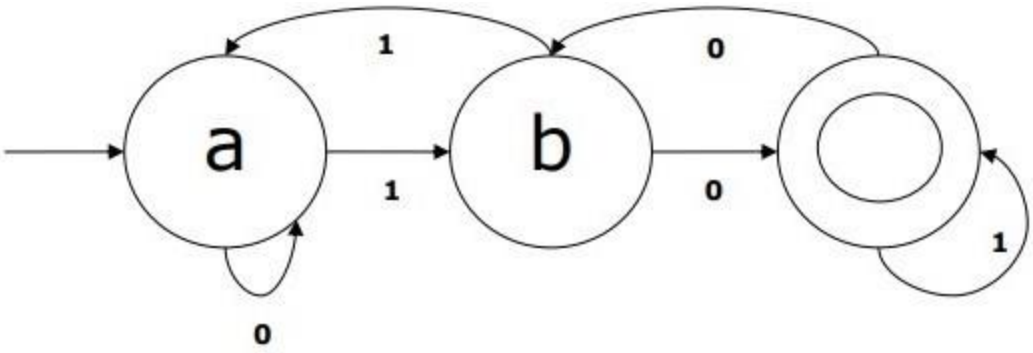
$q_0 = \{a\}$,

$F = \{c\}$,

=> 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

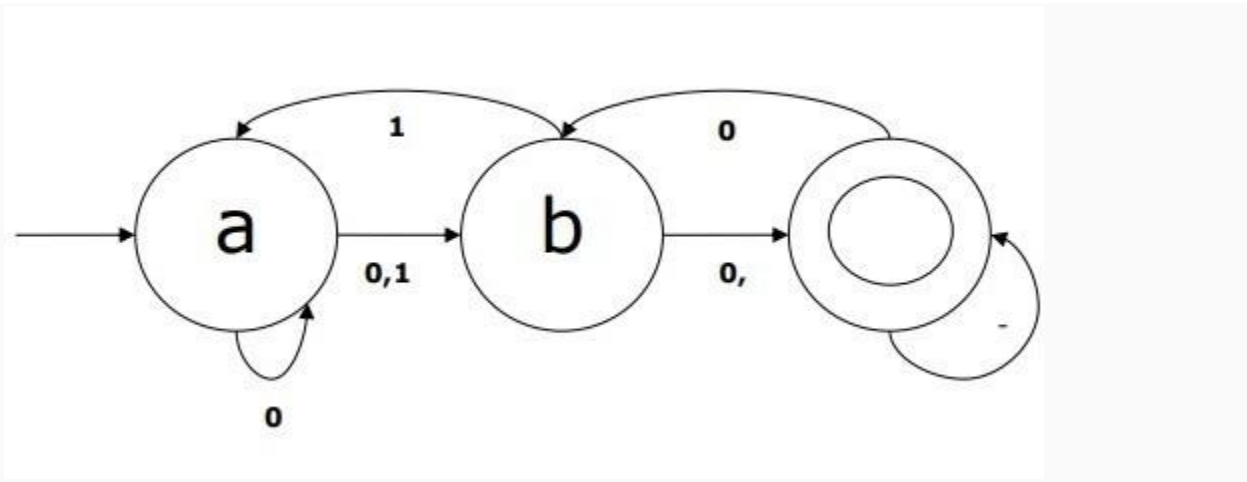
Q = {a, b, c},
Σ = {0, 1},
q0 = {a},
F = {c},

=> 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 -> $(V \cup T)^*$, 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 beconstructed automatically.
- Using the features of associatively & precedence information, suitablegrammars 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 isto 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 parsingtechniques:

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 createtrees 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.

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