UNIT-1

Introduction to NLP:

NLP is a subset of AI deals with the interaction between computers and human (natural) languages. It helps machines understand, interpret, and generate human language. NLP applications include machine translation, sentiment analysis, text summarization, and chatbots. NLP techniques that involves analyzing the structure, meaning, and context of the data.

Uses of NLP:

1. **Language Translation**
    - Example: Google Translate, Microsoft Translator
    - Converts text or speech from one language to another.
2. **Chatbots and Virtual Assistants**
    - Example: Siri, Alexa, ChatGPT
    - Helps in human-like conversations and answering questions.
3. **Text Summarization**
    - Automatically shortens long documents while keeping the main idea.
4. **Sentiment Analysis**
    - Used in social media, reviews, etc., to understand emotions (positive, negative, neutral).
5. **Speech Recognition**
    - Converts spoken language into text.
    - Example: Voice typing, voice search.
6. **Spam Detection**
    - Filters out spam emails by understanding the content.
7. **Search Engines**
    - Improves search accuracy by understanding the user's query.
8. **Text Classification**
    - Categorizes documents, emails, or news articles automatically.

Grammar-based LM

Grammar-based language modelling is a method in NLP where rules of grammar (like subject-verb-object, tenses, etc.) are used to build or analyze sentences. Instead of just predicting the next word based on probability it uses the structure of the language how sentences are formed to understand or generate text.

A language model is a machine learning model LM that predicts upcoming words. More formally, a language model assigns a probability to each possible next word. Language models can also assign a probability to an entire sentence.

How Grammar-Based LMs Work:

1. Define Grammar Rules:

The process begins with defining a set of grammar rules that specify how words and phrases can be combined to form valid sentences.

- Sentence → Noun Phrase + Verb Phrase
- Noun Phrase → Article + Noun
- Verb Phrase → Verb + Noun Phrase
- Article → "the", "a"
- Noun → "boy", "girl", "apple"
- Verb → "eats", "sees"

2. Parse Input Text:

The input text is parsed using the defined grammar rules. This involves identifying the different parts of the sentence and their relationships.

Ex: "The boy / eats / the apple"

3. Apply Rules for Generation:

The rules are then applied to generate new text that sticks to the grammar or to interpret the meaning of existing text.

- " A girls sees the boy"
- " The boy sees a girl"
- " The girl eats the apple"

4. Evaluate Output:

The generated text is evaluated based on its grammaticality and coherence.

- "The boy eats an apple" → grammatically correct and clear meaning.
- "Eats boy the apple" → wrong word order, breaks grammar rules.

Types of Grammar Models

- Context-Free Grammar (CFG) – Most common; rules don't depend on context.
- Dependency Grammar – Focuses on relationships between words (like verb-object).
- Phrase Structure Grammar – Breaks sentences into phrases (noun phrase, verb phrase).
- Feature-based Grammar – Adds grammatical features like number, gender, tense.

Uses of Grammar-Based Models

- Syntax Checking – Ensures sentence is grammatically correct.
- Machine Translation – Helps translate sentences with correct structure.
- Speech Recognition – Understands spoken words in correct order.
- Text Generation – Generates well-formed, grammatically correct sentences.

Advantages:

- Provides high grammatical accuracy.
- Ensures structured and logical sentence formation.
- Easy to explain and debug (rules are visible and editable).
- Good for low-resource languages (when training data is limited).

Disadvantages:

- Rule creation is time-consuming and requires grammatical knowledge.
- Not suitable for slang or informal text (like social media).
- Doesn't adapt to new language patterns automatically.
- Not scalable for very large datasets or dynamic language use.

Statistical LM

Statistical Language Modelling, is the development of probabilistic models that can predict the next word in the sequence given the words that precede it.

A statistical language model learns the probability of word occurrence based on examples of text. Simpler models may look at a context of a short sequence of words, whereas larger models may work at the level of sentences or paragraphs. Most commonly, language models operate at the level of words.

Types of statistical LM

1. N-gram:

This is one of the simplest approaches to language modelling. Here, a probability distribution for a sequence of 'n' is created, where 'n' can be any number and defines the size of the gram. There are different types of N-Gram models such as unigrams, bigrams, trigrams, etc.

Unigram:

A unigram model is the simplest type of statistical language model. It treats each word in a sentence as independent of the words before or after it. This means the model doesn't consider the order of words it only looks at the individual words and how often they appear in a large text.

Example:
Sentence: "I like ice cream"
Unigrams: "I", "like", "ice", "cream"

Bigram:

A bigram model looks at two words at a time to understand how likely a word is to follow another. It takes word order into account, unlike the unigram model. It calculates the probability of a word based on the word that came before it.

Example:
Sentence: "I like to eat "
Bigrams: "I lke", "like to", "to eat"

Trigram:

A trigram model takes it a step further by looking at three words at a time. It predicts the next word based on the previous two words. This gives it a better understanding of sentence structure and context than unigram and bigram models.

Example

Sentence: "She is reading books"
Trigrams: "She is reading", "is reading books"

2. Continuous space:

In this type of statistical model, words are arranged as a non-linear combination of weights in a neural network. The process of assigning weight to a word is known as word embedding. This type of model proves helpful in scenarios where the data set of words continues to become large and include unique words.

Regular Expressions

- Regular expression is also called as Regex.
- Regular expression is a sequence of characters that define a search pattern.
- It is used for pattern matching or string matching.
- Especially it is used for email validation, phone number validation. So if we want password contains a lowercase alphabets and uppercase alphabets and one special character whatever will those we can added using regular expression.

Certain rules for Regex:

[abc] – it is basically means anything a, b or c

[^ abc] – it basically any character except a,b,c

[a - z] – it basically any character a-z

[A - Z] – it is basically any character A-Z

[0 - 9] – it basically any digit 0 to 9

[a-z, A-Z] – it basically a to z, A to Z

Quantifiers:

[ ]? – what are written inside or rule inside it will occur zero or one time.

[ ]+ - what are written inside it will one time or more times.

[ ]* - what are written inside it will zero or more times.

[ ]{n} – what are written inside it will occur n times.

[ ]{n, } – what are written inside it will occur n or more times.

[ ] {y,z} – what are written it will occur at least y times and less than z times.

^ caret – it tells computer that must start at beginning of the string or line.

$ dollar – it tells computer that must occur at end of the string or line.

\ backslash – it is used actual '+','-', '.'etc characters add a backslash before the character this tell computer to treat that following character as a search character.

Regex meta characters:

\s: matches any whitespace character such as space and tab.

\S: matches any non whitespace characters.

\d: matches any digital characters.

\D; matches any non digital characters.

\w: matches any word characters.

\W; matches any non word characters.

Examples:

1. **Mobile number start with 8 or 9 and total digit =10.**

   [8 9] [0 - 9]{9}

2. **First character uppercase, contains lowercase, only one digit allowed in between.**
   [A - Z] [a - z]+ [0 – 9] [a – z, A - Z] +
3. **Email ID – Aditya123@gmail.com**

   First email can be divide into three parts
   Part1 – Aditya123
   Part2 – gmail
   Part3 – com, in

   [A -Z, a – z,0 – 9, _ \-] + [@] [a – z] + [\.] [a – z]{2,3}

Finite State Automate

A Finite State Automaton (FSA), also known as a Finite State Machine (FSM), is a computational model used in computer science to represent and control execution flow. It consists of a finite number of states, transitions between those states, and actions.
Finite automata come in deterministic (DFA) and non-deterministic (NFA), They are widely used in text processing, compilers, and network protocols.

How it works

- Starts in one initial state (called the start state).
- It reads input symbols one at a time (like letters or numbers).
- Based on the current state and the input symbol, it moves to another state using a rule (called a transition).
- After reading all the input, if it ends in a final state (accepting state), the input is said to be accepted.
- Otherwise, the input is rejected.

Formal Definition:

A **Finite State Automaton** is defined as a 5-tuple:

$$FSA = (Q,\Sigma ,q0,F, \delta)$$

Where:

- **Q**: A finite set of states
- **Σ (Sigma)**: A finite set of input symbol
- **q_0**: The initial state
- **F**: A set of final states ($F \subseteq Q$)
- **δ (delta)**: A transition function $\delta: Q \times \Sigma \rightarrow Q$

Types of Finite Automata

There are two types of finite automata:

- Deterministic Finite Automata (DFA)
- Non-Deterministic Finite Automata (NFA)

Deterministic Finite Automata (DFA):

A DFA is represented as {Q, Σ, q, F, δ}. In DFA, for each input symbol, the machine transitions to one and only one state. DFA does not allow any null

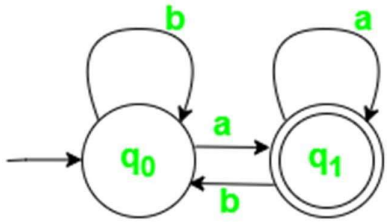
transitions, meaning every state must have a transition defined for every input symbol.

- In DFA given the current state we know that next state will be.
- It has only one unique next state.
- It has no choice.
- It is a simple and easy to design.

Formal Definition:

DFA consists of 5 tuples: {Q, Σ, q, F, δ}.

- **Q:** set of all states.
- **Σ:** set of input symbols.
- **q:** Initial state.
- **F:** set of final state (F ⊆ Q)
- **δ:** Transition Function, defined as δ: Q X Σ --> Q.



Transition table

State\	a	b
q0	q1	q0
q1	q1	q0

Non-Deterministic Finite Automata (NFA):

A Nondeterministic Finite Automaton (NFA) is a type of finite state machine used in computational theory and natural language processing. Unlike a DFA, an NFA allows multiple possible transitions for a given state and input symbol, including transitions without any input (called ε-transitions).

- It can transition to multiple states for the same input.
- It allows null (ϵ) moves, where the machine can change states without consuming any input. The next state may be chosen random.

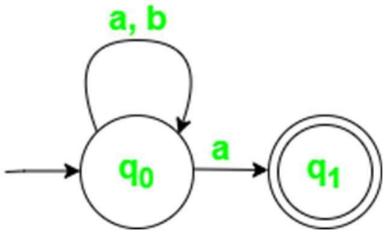
Formal Definition:

An NFA is a 5-tuple:

$$NFA= \{Q, \Sigma, q, F, \delta\}$$

Where:

- **Q**: A finite set of states.
- **Σ (Sigma)**: A finite set of input symbols (alphabet)
- **q**: The initial state.
- **F**: A set of accepting (final) states ($F \subseteq Q$)
- **δ (delta)**: A transition function: $\delta: Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2Q$



Transition table

State	a	b
q0	{q0,q1}	q0
q1	φ	φ

English morphology

Morphology is the study of words morpheme are the minimal units of words that have a meaning and cannot be sub divided further. There are mainly 2 types free and bound. Free morpheme occur alone and bound morpheme must occur with another morpheme.

For ex: free morpheme is “bad” and on ex of bound morpheme is “ly” it is bound although it has meaning it cannot stand alone. It must be attached to another morpheme to produce a word.

Free morpheme: bad

Bound morpheme: ly

Word: badly

When we talk about words, there are two groups lexical or content and function words or grammatical.

Lexical or content: There are open class words and include nouns, verbs, adjectives and adverbs. New word can regularly be added to this group.

Function word or grammatical: There are closed word are conjunctions, prepositions, article and pronouns and new words cannot be (or) rarely added to this class.

Affixes are often the bound morpheme, this group include prefix, suffix, infixes and circumfix.

Prefixes are added to the beginning of another morphemes

Suffixes are added to the end. Following are ex of each of there

Prefix: re- added to produce redo.

Suffix: -or added to edit produce editor.

Infix: -um- added to fikas produce fumikas in Bontoc.

Circumfix: ge--- t is added to lieb to produce geliebt in German.

There are two categories of affixes: derivational and inflectional the main difference between the two is that derivational affix are added to morpheme to form new words that may or may not be the some part of speech and inflectional affixal are added to the end of an existing word for purely grammatical reasons.

English morphemes

A: Free

1. open class.
2. closed class.

B: Bound

1. Affix
    a. Derivational
    b. Inflectional.
2. Root

Transducers or lexicon and rules

A lexical transducer is a special type of finite-state machine that is used in language processing to convert inflected or surface forms of words into their basic or lexical forms, and vice versa. It was first introduced by Korhonen, Kaplan, and Zaenen in 1992. In simple terms, it acts like a translator between how a word appears in a sentence (surface form) and its base form with grammatical tags (lexical form). This is useful in morphological analysis and generation.

Lexicon:

The lexicon is a list of base words (root forms) along with information about their meanings and grammatical behaviour. Each word in the lexicon has a canonical form and a set of tags that describe its grammatical features like tense, number, person, etc.

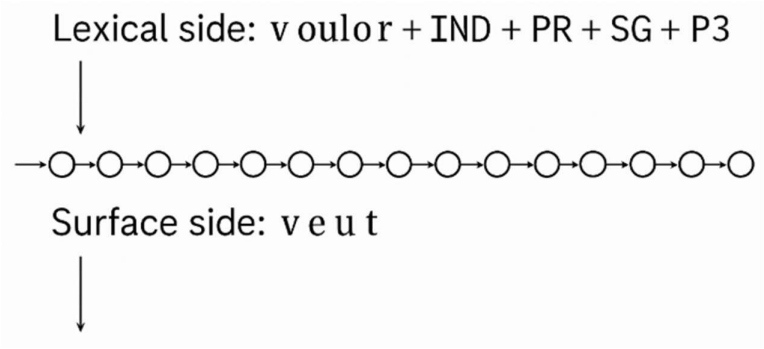
Rules:

Rules define how words change from their base (lexical) form into their surface form. These rules are often morphological, like adding suffixes for tense or changing spellings for conjugation.

Example (French):

For example, the French surface word **"veut"** (which means "wants") can be analysed using a lexical transducer and matched to its lexical form: **vouloir + IND + PR + SG + P3**

- **vouloir** = verb root (to want)
- **IND** = indicative mood
- **PR** = present tense
- **SG** = singular
- **P3** = third person



- Circles represent the states and arcs represents the pair of symbols: a lexical symbol and a surface symbol Sometimes the lexical and surface symbols are the same (e.g., **v:v**), Sometimes they differ (e.g., **o:e**).
- Finite state transducers are bidirectional. The same transducer can be used for analysis (veut -> voulor +IND + PR+SG+P3) as well as generation (voulor +IND + PR+SG+P3 - > veut).
- Analysis and generation different only with the respect to the choices of the input side (surface or lexical).

Tokenization

Tokenization is a fundamental step in Natural Language Processing (NLP). It involves dividing a Textual input into smaller units known as tokens. These tokens can be in the form of words, characters, sub-words, or sentences. used to convert unstructured text into a structured format that machines can easily analyze and understand.

- Involves dividing a string or text into a list of smaller units known as tokens.
- A tokenizer breaks unstructured text into smaller parts, treating each part as a separate piece of information.
- Tokens: Words or Sub-words in the context of natural language processing. Example: A word is a token in a sentence, A character is a token in a word, etc.
- Application: Multiple NLP tasks, text processing, language modelling, and machine translation

Types of Tokenization: Tokenization can be classified into several types based on how the text is segmented

1. Word Tokenization:

Word tokenization is the most commonly used method where text is divided into individual words. It works well for languages with clear word boundaries, like English.

Another word of word tokenization is white space tokenization

For example

Input: ["Machine learning is fascinating"]

Output when tokenized by word: ["Machine", "learning", "is", "fascinating"]

2. **Character Tokenization:**

   In Character Tokenization, the textual data is split and converted to a sequence of individual characters. This is beneficial for tasks that require a detailed analysis, such as spelling correction or for tasks with unclear boundaries. It can also be useful for modelling character-level language**.**

   For example

   Input ["You are helpful"]

   Output when tokenized by characters: ["Y", "o", "u", " ", "a", "r", "e", " ", "h", "e", "l", "p", "f", "u", "l"]

3. **Punctuation-based Tokenization:**

   Punctuation-based tokenization splits text into tokens by separating words and punctuation marks. This method treats punctuation (like commas, periods, question marks, etc.) as individual tokens instead of combining them with words. It helps in preserving the meaning and structure of a sentence during text processing, which is useful in tasks like sentiment analysis and grammar correction.

   For example:

   Input ["Hello, how are you?"]

   Output ["Hello", ",", "how", "are", "you", "?"]

4. **Sub word Tokenization:**

   This strikes a balance between word and character tokenization by breaking down text into units that are larger than a single character but smaller than a full word. This is useful when dealing with morphologically rich languages or rare words.

   For example

   Time table - ["Time", "table"]

   Rain coat - ["Rain", "coat"]

   Run way - ["Run", "way"]

   Sub-word tokenization helps to handle out-of-vocabulary words in NLP tasks and for languages that form words by combining smaller units.

5. **Sentence Tokenization:**

   Sentence tokenization is also a common technique used to make a division of paragraphs or large set of sentences into separated sentences as tokens. This is useful for tasks requiring individual sentence analysis or processing

   For example:

   Input ["He is here. She left."]

   Output ["He is here.", "She left."]

Detecting and correcting spelling errors

Spelling correction in natural language processing involves detecting and correcting misspelled words in text. It is a process of detecting and some times providing suggestions for incorrectly spelled words in the text. In computing spell checker is an application program that flags word in a document that may not be spelled correctly.

This is typically achieved through combination of techniques, including dictionary lookups, error model-based approaches, and machine learning algorithms. The goal is to identify and correct errors like real-word errors (misused words) and non-word errors (typos).

- Real-word errors: Those error words that are acceptable words in the dictionary.
- Non-word errors: Those error words that are cannot be found in the dictionary.

Detection of spelling errors:

1. **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.
2. **Language modelling:** More advanced techniques involve building language models e.g., N-gram language models also widely used in error detection. These models evaluate the probability of a word appearing a given context by analyzing the frequency of a word sequences. If a sequence is statistically unlikely, it may indicate a real-word error.
3. **Part-of-speech (POS):** part-of-speech tagging helps find grammar mistakes by checking how words are used in a sentence. For example, if a verd is used where a noun should be, it can be flagged as a error.

Correction of spelling errors:

1. **Minimum edit distance:** This technique suggests replacement words that are closest in spelling to the detected error, based on the fewest number of edits needed to reach a valid word. While simple, it is very effective for common typographical mistakes.

2. **Contextual spelling correction:** Contextual spelling correction takes this a step further by using advanced language models like BERT or GPT. These models consider the surrounding words and suggest corrections that make sense in the given context, which is especially useful for real-word errors (e.g., “their” vs. “there”).
3. **Similarity key technique:** The Similarity Key Technique is a method where each word (or string) is changed into a special code, called a key. Words that are spelled in a similar way will get similar keys. This helps in finding and correcting spelling mistakes, because if two words have similar keys, they are likely to be similar or related.
4. **Rule-based correction:** Rule-based correction methods use fixed grammar rules to correct common mistakes. This method is often used in specific fields where the types of errors are already known.

Tools and Libraries:

TextBlod: A python library that provides a simple API for common NLP tasks, including spell checking.

SpellChecker: Another python library that focuses specifically on spelling correction.

SymSpell: An efficient algorithm for spelling correction, available in python.

Spark NLP: A library for NLP tasks, including spell checking built on Apache Spark.

Minimum edit distance:

Minimum Edit Distance (MED) is a technique used in Natural Language Processing (NLP) to find how similar two strings (words or sentences) are. It measures the minimum number of operations required to convert one string into another.

Minimum edit distance between two strings str1 and str2 is defined as the number of insert/ delete/ substitute operations required to transforms str1 and str2.

Common Operations:

1. **Insertion** – Insert any character before or after any index.
2. **Deletion** – Remove a character.
3. **Substitution** – Replace one character with another

Example: str1 = “ab”, str2 = “abc” the making on insert operation od char (c) on str1 transforms str l into str 2. Therefore edit distance between str 1 and str 2 is 1.

Input: s1 = "geek", s2 = "gesek"
Output: 1
Explanation: We can convert s1 into s2 by inserting an 's' between two consecutive 'e' in s1.

Input: s1 = "gfg", s2 = "gfg"
Output: 0
Explanation: Both strings are same.

Input: s1 = "abcd", s2 = "bcfe"
Output: 3
Explanation: We can convert s1 into s2 by removing 'a', replacing 'd' with 'f' and inserting 'e' at the end.

UNIT-2

N-grams

N-grams are defined as the contiguous sequence of n items that can be extracted from a given sample of text or speech. The items can be letters, words, or base pairs, according to the application. The N-grams typically are collected from a text or speech corpus.

N-grams are classified into different types depending on the value that n takes. When n=1, it is said to be a unigram. When n=2, it is said to be a bigram. When n=3, it is said to be a trigram.

Unsmoothed N-gram:

Unsmoothed n-gram models calculate the probability of word sequences based only on their observed frequencies in the training data

Unigram: A model that calculates the probability of a single word based on how often it appears in the corpus.

$$P(a) = \frac{\text{Count}(a)}{\text{Total number of words in the corpus}}$$

Data: "The dog barks. The cat sleeps. The dog runs. The cat jumps."

Total words =12

Vocabulary: {"The", "dog", "barks", "cat", "sleeps", "runs", "jumps"} = 7

P (dog) = 2/10 = 0.1666

P (Apple) = 0/12 = 0 (zero probability)

Bi gram:

In the bigram model, we calculate the probability of a word b given the previous word a. It assumes that the current word depends only on the word immediately before it.

$$P(b \mid a) = \frac{\text{Count}(a, b)}{\text{Count}(a)}$$

Data: "The dog barks. The cat sleeps. The dog runs. The cat jumps."

Total words =12

Vocabulary: {"the dog", "dog barks", "barks the", "the cat", "cat sleeps", "sleeps the", "dog runs", "runs the", "cat jumps"} = 9

P (dog | the) = count (The, dog) / count (The) = 2/4 = 0.5

P (jumps | dog) = 0/2 = 0 (zero probability)

Tri gram:

In the trigram model, we calculate the probability of a word c based on the two previous words, a and b. It assumes that a word depends on the last two words only.

$$P(c \mid a, b) = \frac{\text{Count}(a, b, c)}{\text{Count}(a, b)}$$

Data : "The dog barks. The cat sleeps. The dog runs. The cat jumps."

Total words =12

Vocabulary: {"The dog barks", "dog barks the, "barks the cat", "The cat sleeps", "cat sleeps the", "sleeps the dog", "The dog runs", "dog runs the", "runs the cat", "The cat jumps"} = 10

P (barks | the, dog) = Count (the, dog, barks) / Count (the, dog) = ½ = 0.5

P (sleeps | the, dog) = 0/2 = 0 (zero probability)

These models are simple and useful for understanding basic language patterns. However, they have a major drawback: if an n-gram has never appeared in the training data, its probability is zero, even if it's a valid phrase.

Therefore, unsmoothed n-grams are mainly used for educational purposes or as a base model, while practical systems apply smoothing techniques to handle unseen n-grams and improve accuracy.

Smoothing

Smoothing in Natural Language Processing (NLP) is a method used to fix the problem of getting zero probability for word combinations that the model hasn't seen before. When a language model comes across a sentence or phrase that wasn't in its training data, it gives it a probability of zero, which can cause problems in tasks like text prediction or machine translation.

Smoothing helps by slightly adjusting the probabilities so that even unknown or new word combinations still get a small chance, instead of being completely ignored. Without smoothing, unseen word combinations are assigned a probability of zero, which can cause the entire sentence probability to become zero. Overall, smoothing is essential for building accurate and reliable NLP systems.

Types of smoothing

1. **Add one smoothing (Laplace):**

The easiest way to apply smoothing is to add one to all n-gram counts before converting them into probabilities. This means even unseen word combinations will get a count of 1 instead of 0. This method is called Laplace smoothing.

While it's not very effective for modern language models, it helps us understand how smoothing works and is still useful for simpler tasks like text classification.

$$P(b \mid a) = \frac{\text{count}(a, b) + 1}{\text{count}(a) + V}$$

Count(a,b) = how often "a b" appears together

Count(a) = how often "a" appears as the first word in a Unigram

V = vocabulary size (total unique words)

Example:

Data: "The dog barks. The cat sleeps. The dog runs. The cat jumps."

Total words =12

Vocabulary: {"the dog", "dog barks", "barks the", "the cat", "cat sleeps", "sleeps the", "dog runs", "runs the", "cat jumps"} = 9

$$P(\text{"jumps"} \mid \text{"dog"}) = \frac{0+1}{2+9} = \frac{1}{11} \approx 0.0909$$

2. Add-K smoothing:

Add-k smoothing is like add-one smoothing, but instead of adding 1 to each count, we add a smaller value like 0.5 or 0.1. This way, we move less probability to unseen word pairs. The value of k can be chosen by testing different values on a development set. While add-k smoothing works well for some tasks like text classification, it is not very effective for language modelling.

$$P(b \mid a) = \frac{\text{count}(a,b) + k}{\text{count}(a) + k \cdot V}$$

Count(a,b) = how often "a b" appears together

Count(a) = how often "a" appears as the first word in a Unigram

V = vocabulary size (total unique words)

k = a small positive constant (e.g., 0.1 or 0.5)

Example:

$$P(\text{jumps} \mid \text{dog}) = \frac{0+0.5}{2+0.5 \cdot 7} = \frac{0.5}{5.5} \approx \boxed{0.091}$$

Interpolation:

Sometimes, a language model doesn't find a 3-word combination (trigram) in the training data. Instead of giving zero probability, we can look at shorter combinations like 2-word (bigram) or even single words (unigram). This method is called interpolation.

In interpolation, we combine the probabilities of the trigram, bigram, and unigram using weights. This helps the model make better guesses by mixing different levels of context instead of relying on just one.

$$P(b \mid a) = \lambda_1 \cdot \frac{\text{count}(a,b)}{\text{count}(a)} + \lambda_2 \cdot \frac{\text{count}(b)}{N}$$

Count(a,b) = how often "a b" appears together

Count(a) = how often "a" appears as the first word in a Unigram

Count(b) = how often "b" appears as the first word in a Unigram

P_{ML} = maximum likelihood estimate (regular probability from counts)

λ1+λ2 =1 (they are weights)

N = Total number of tokens (words) in the corpus

Example:

$$P(\text{jumps} \mid \text{dog}) = 0.7 \cdot \frac{0}{2} + 0.3 \cdot \frac{1}{12} = 0 + 0.025 = 0.025$$

Interpolation and backoff

Interpolation:

Sometimes, a language model doesn't find a 3-word combination (trigram) in the training data. Instead of giving zero probability, we can look at shorter combinations like 2-word (bigram) or even single words (unigram). This method is called interpolation.

In interpolation, we combine the probabilities of the trigram, bigram, and unigram using weights. This helps the model make better guesses by mixing different levels of context instead of relying on just one.

$$P(b \mid a) = \lambda_1 \cdot \frac{\text{count}(a, b)}{\text{count}(a)} + \lambda_2 \cdot \frac{\text{count}(b)}{N}$$

Count(a,b) = how often "a b" appears together

Count(a) = how often "a" appears as the first word in a Unigram

Count(b) = how often "b" appears as the first word in a Unigram

P_{ML} = maximum likelihood estimate (regular probability from counts)

λ1+λ2 =1 (they are weights)

N = Total number of tokens (words) in the corpus

Example:

Data: “The dog barks. The cat sleeps. The dog runs. The cat jumps.”

Total words =12

Vocabulary: {“the dog”, “dog barks”, “barks the”, “the cat”, “cat sleeps”, “sleeps the”, “dog runs”, “runs the”, “cat jumps”} = 9

$$P(\text{jumps} \mid \text{dog}) = 0.7 \cdot \frac{0}{2} + 0.3 \cdot \frac{1}{12} = 0 + 0.025 = 0.025$$

Backoff:

Backoff is a smoothing technique used when an n-gram (like a trigram) has zero count in the data. In such cases, the model "backs off" to a lower-order n-gram (like a bigram or unigram) until it finds a match.

- If the trigram exists in the data, we use it.
- If not, we "back off" and use the bigram.
- If the bigram is also not found, we finally back off to the unigram.

We only use a lower-order n-gram when the higher-order one is missing or has zero count.

To ensure proper probability distribution, traditional backoff methods apply discounting to higher-order n-grams, saving some probability for lower orders. However, a simpler method called Stupid Backoff skips discounting. Instead, if a higher-order n-gram is missing, it directly backs off to a lower-order n-gram and multiplies its score by a fixed weight (like 0.4). While this doesn't form a true probability distribution, it's fast and works well for large datasets.

$$\text{score}(b \mid a) = \begin{cases} \frac{\text{count}(a,b)}{\text{count}(a)}, & \text{if count}(a,b) > 0 \\ \lambda \cdot \frac{\text{count}(b)}{N}, & \text{otherwise} \end{cases}$$

λ= fixed weight (e.g., 0.4)

N = total number of words in the corpus

Example:

$$\text{score}(\text{jumps} \mid \text{dog}) = 0.4 \cdot \frac{1}{12} = 0.033$$

Word classes

In Natural Language Processing (NLP), word classes (also known as parts of speech (POS)) refer to the grammatical categories that words belong to based on their roles in sentences. Understanding word classes is fundamental in many NLP tasks like POS tagging, syntactic parsing, information extraction, and machine translation.

1. Noun (NN)

Nouns are words that name people, places, things, or ideas. Ex: cat, India, house.
Example: *Ravi went to the* ***market****.*

2. Pronoun (PRP)

Pronouns replace nouns to avoid repetition. Ex: he, she, they, them.
Example: ***He*** *is a doctor.*

3. Verb (VB)

Verbs describe an action, event, or state of being. Ex: run, write, sit, walk.
Example: *She* ***runs*** *every morning.*

4. Adjective (JJ)

Adjectives describe or modify nouns to give more information. Ex: beautiful, large, smart.
Example: *He is a* ***smart*** *boy.*

5. Adverb (RB)

Adverbs modify verbs, adjectives, often showing manner, time. Ex: quickly, slowly, silently.
Example: *She speaks* ***slowly****.*

6. Preposition (IN)

Prepositions show the relationship between a noun/pronoun and other words. Ex: in, on at, under.
Example: *The book is* ***on*** *the table.*

7. Conjunction (CC)

Conjunctions join words, phrases, or clauses. Ex: and, but, or.
Example: *I want tea* ***and*** *snacks.*

8. Interjection (UH)

Interjections express sudden emotions or feelings. Ex: wow, hey, oh.
Example: *"**Wow!** That's amazing."*

9. Determiner (DT)

Determiners are used before nouns to specify quantity or reference. Ex: the, an, a, this, those.
Example: ***This** apple is sweet.*

10. Numeral (CD)

Numerals indicate numbers or order. Ex: one, two, 3rd.
Example: *She has **two** dogs.*

Part of speech tagging

Parts of Speech (PoS) tagging is a core task in NLP, It gives each word a grammatical category such as nouns, verbs, adjectives and adverbs. Through better understanding of phrase structure and semantics, this technique makes it possible for machines to study human language more accurately.

PoS tagging is essential in many NLP applications like machine translation, sentiment analysis and information retrieval. It serves as a link between language and machine understanding, enabling the creation of complex language processing systems.

POS Tag	Meaning	Example
NN	Noun (singular)	*dog, school, computer*
NNS	Noun (plural)	*dogs, schools*
VB	Verb (base)	*go, run, eat*
VBD	Verb (past)	*went, ran, ate*
VBG	Verb (gerund)	*going, running, eating*
JJ	Adjective	*brown, beautiful, quick*
RB	Adverb	*quickly, very*
PRP	Pronoun	*he, she, it, they*
IN	Preposition	*in, on, at, by*
DT	Determiner	*the, a, an, this*

CC	Coordinating Conjunction	*and, but, or*
UH	Interjection	*wow, oh, hey!*

Example: "The quick brown fox jumps over the lazy dog."

- "The" is tagged as determiner (DT)
- "quick" is tagged as adjective (JJ)
- "brown" is tagged as adjective (JJ)
- "fox" is tagged as noun (NN)
- "jumps" is tagged as verb (VB)
- "over" is tagged as preposition (IN)
- "the" is tagged as determiner (DT)
- "lazy" is tagged as adjective (JJ)
- "dog" is tagged as noun (NN)

Workflow of POS Tagging in NLP

1. **Tokenization:**
   The input text is split into individual words or subwords, enabling word-level analysis.
2. **Loading a language model:** Tools like NLTK requires a pre-trained language model to perform POS tagging. These models are trained on large datasets and provide insights into the grammatical rules and structure of the language.
3. **Text Preprocessing**: The text is then cleaned to improve accuracy. Common preprocessing steps include converting text to lowercase, removing special characters and eliminating irrelevant content.
4. **Syntactic Analysis:** The sentence is parsed to understand grammatical roles, helping prepare for accurate POS assignment.
5. **POS tagging:** each token is labelled with its appropriate part of speech using context and syntax.
6. **Result evaluation:** The output is checked for accuracy and any tagging errors are corrected if needed.

Rule based POS tagging

Rule-based POS tagging assigns grammatical tags to words using a predefined set of rules, as opposed to machine learning-based methods that require training on annotated corpora. These rules are crafted based on morphological features (like word endings) and syntactic context, making the approach highly interpretable and transparent.

Example

a rule might specify that words ending in “-tion” or “-ment” should be tagged as nouns, based on common suffix patterns found in English.

- **Rule:** Assign the POS tag "Noun" to words ending in -tion or -ment.
- **Text:** "Her dedication and commitment inspired the entire management team."

Tagged output:

- "Her" is tagged as Pronoun (PRP)
- "dedication" is tagged as Noun (NN)
- "and" is tagged as Conjunction (CC)
- "commitment" is tagged as Noun (NN)
- "inspired" is tagged as Verb (VB)
- "the" is tagged as Determiner (DT)
- "entire" is tagged as Adjective (JJ)
- "management" is tagged as Noun (NN)
- "team" is tagged as Noun (NN)

In this case, the rule-based tagger correctly identifies "dedication," "commitment," and "management" as nouns by applying suffix-based rule. Even though it's simple, this example shows how rule-based systems can understand many language patterns by using clear and organized rules.

Stochastic and Transformation-based tagging

Transformation Based tagging

Transformation-Based Tagging is a method used to improve part-of-speech tags by applying correction rules step by step. It doesn't depend on fixed grammar rules like rule-based taggers or probabilities like statistical taggers. Instead, it starts with basic tags and then makes corrections using rules that look at the context of each word.

Example rule: "If a word is tagged as a verb, but it comes after 'the', change it to a noun"

Example Sentence:

"The walk was long "

Initial tags (before rule is applied):

The – determiner (DT)

Walk – verb (VB)

was – verb (VB)

long – adverb (RB)

Transformation rule applied:

If a word is tagged as VB (verb) but comes after "the", change it to NN (noun).

Final Tags (after rule applied):

The – determiner (DT)

Walk – noun (NN)

Was – verb (VB)

Long – adverb (RB)

Example sentence 2:

"Can birds fly?"

Initial tags (before rule is applied):

Can – modal verb (MD)

Birds – noun (NN)

Fly – noun (NN)

Transformation rule applied:

If a word is tagged as NN (noun) but comes at the end of a question starting with "Can", change it to VB (verb)

Final Tags (after rule applied):

Can – model verb (MD)

Birds – noun (NN)

Fly – verb (VB)

Stochastic POS tagging:

Stochastic POS tagging (also called statistical tagging) is a method in computational Grammatical Science that uses probability to decide the correct part of speech for each word in a sentence, like noun, verb, or adjective. Unlike rule-based methods that use fixed grammar rules, statistical tagging learns from examples. It studies large amounts of already tagged text and finds patterns using machine learning.

These models calculate the chance of a tag being correct for a word based on the words around it. This helps them handle confusing cases and understand complex grammar.

Issues in POS tagging

1. Ambiguity:

Ambiguity happens when a word can have more than one part of speech, depending on the context. This makes it hard for the tagger to choose the correct tag.

Example:

- **“book”**
  - Verb: "Can you **book** a ticket?"
  - Noun: "I read a **book**."
- **“play”**
  - Noun: "The **play** was great."
  - Verb: "They **play** outside."

The tagger needs to look at the surrounding words to decide the correct tag.

2. Out of vocabulary:

Out-of-Vocabulary (OOV) words are words that do not appear in the training data of a POS tagger. These can include new terms, foreign words, slang, or names that the model has never seen before.

When a tagger encounters an OOV word, it struggles to assign the correct part of speech, because it has no past examples to learn from. This often leads to incorrect tagging, which can affect the accuracy of downstream NLP tasks.

Example: “He created a new app called **Zyntora**.”

- OOV Word: “Zyntora” (a made-up product name)
- Issue: The tagger may not know if it's a noun, a verb, or something else.

3. Errors in rule and statistical models:

Rule-based POS taggers can make errors if their rules are too strict or don’t cover all cases. They often fail with new word usages. Statistical taggers make mistakes when they rely too much on training data. If a word is rare or has multiple meanings, the model may choose the wrong tag.

Example:

Sentence: "They can fish in the lake."

- **Rule-based error:** Tags **"can"** as a **verb**, and **"fish"** as a **noun**
- **Statistical error:** Tags **"can"** as a **noun** (like a tin can), and **"fish"** as a **verb** (like swimming).

Correct tags:

- **"can"** – Modal verb
- **"fish"** – Main verb

4. Multi word expressions:

MWEs are phrases made of two or more words that act as one meaning, like idioms or phrasal verbs. Their meaning is often not clear from the individual words. This makes POS tagging harder, because tagging word by word may miss the actual meaning of the whole phrase.

Example: **Sentence:** "He kicked the bucket."

- **He** – Pronoun (PRP)
- **kicked** – Verb (VB)
- **the** – Determiner (DT)
- **bucket** – Noun (NN)

Literally, this makes sense. But **"kick the bucket"** is an idiom meaning *"to die."*

So, the tagger should ideally treat **"kick the bucket"** as a **single verbal expression** rather than tagging each word individually with its literal meaning.

Hidden Markov and maximum entropy model

In Natural Language Processing (NLP), we use different types of models to understand and process human language. Two important models are the Hidden Markov Model (HMM) and the Maximum Entropy Model (MaxEnt). These models help computers to do tasks like part-of-speech (POS) tagging, named entity recognition, speech recognition, and many others.

Hidden Markov Model (HMM):

Hidden Markov Model is a statistical model used to predict hidden patterns or tags based on observed data, like words in a sentence. It is commonly used in NLP tasks such as POS tagging, where the system guesses the correct tag sequence using probabilities.

- It tries to find the **best tag** (like noun or verb) for each word.
- It uses the idea that the **next tag depends on the previous one**.
- It looks at how often a tag follows another tag.
- Useful for tasks like **POS tagging**, **name finding**, and **speech recognition**.
- Easy to understand, but may not work well with complex sentences.

Example:

Sentence **"Birds fly"**

Observed words (input): "Birds", "fly"

Possible tags: Noun (N), Verb (V)
We check all tag combinations:

- Noun → Verb (Birds/N, fly/V)
- Noun → Noun (Birds/N, fly/N)

HMM calculates which tag path has the **highest total probability**.

The best option will be: **Birds/Noun → fly/Verb**

Advantages:

- Simple and easy to implement.

- Works well with small datasets.

Disadvantages:

- Makes strong assumptions (only looks at the previous tag).
- Cannot handle complex context or long-range information.

Maximum Entropy Model:

The Maximum Entropy model is used to predict the most suitable tag or label for a word based on useful clues or features from the sentence. It works on the idea of making no extra assumptions — only using the information available.

- It is a model that directly tries to guess the correct tag or label for a word.
- It looks at useful clues like the word itself, the word before it, capital letters, endings like "-ing" or "-ed", etc.
- It learns which clues are important by using training data.
- It is used in tasks like part-of-speech tagging, text classification, and f Named Entity Recognition (NER).
- It is more accurate and flexible, especially when we have more data to train it.

Example: Sentence: **"Apple is sweet"**

- For the word "Apple", MaxEnt looks at features like: Is the word capitalized? Is it the first word in the sentence?
- Based on these features, it might predict the tag: **Proper Noun (NNP)**
- For "sweet", it might check if it comes after "is", and guess: **Adjective (JJ)**

Advantages:

- Can use **many types of features**.
- More **flexible and accurate** in many cases.

Disadvantages:

- Needs more data and feature design.
- Takes longer to train than HMM.

UNIT-3

Context-Free Grammar

In NLP, a grammar defines the rules of how words and phrases combine to form valid sentences. Just like in English we say *"The cat sits on the mat"* is correct but *"Cat on sits mat the"* is not.

- A Context-Free Grammar is a type of formal grammar that is powerful enough to describe the structure of most natural languages.
- It is called context-free because the rules apply regardless of surrounding words (context).
- To list all strings in a language using a set of rules (production rules). It extends the capabilities of regular expressions and finite automata.

Example: A **noun phrase (NP)** can be defined as Det + Noun, no matter what words come before or after.

Components of a CFG

A CFG is defined as G = (V, Σ, P, S) where:

Variables(V):

Also called non-terminals, they represent syntactic categories like Sentence (S) or Noun Phrase (NP). They can be expanded using rules. Variables are denoted by upper cases.

Terminals (T):

Basic symbols or words that appear in the final strings. They cannot be replaced further. Examples include letters, digits, or actual words like *cat*, *dog*. The terminals are denoted by lower cases.

Production Rules (P):

Define how variables can be rewritten using other variables and terminals, giving the recursive structure of the language.

Start Symbol (S):

The start symbol is the main variable from which the derivation of any string begins. It represents the entire language defined by the grammar.

Every production rule must look like: Variable → (Variables and/or Terminals)

- The left side is always one Variable (not a terminal).
- The right side can have variables, terminals, or both.

Example:

S → aA

A → b

- Variables: {S, A}
- Terminals: {a, b}
- Start Symbol: S
- Productions: {S → aA, A → b}

This grammar generates the string **"ab"**.

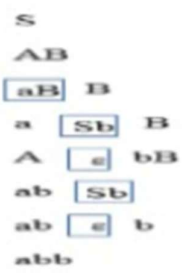
Example:

Grammar:

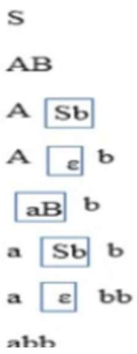
1. S→AB | ε
2. A→aB
3. B→Sb Output string: ‘abb’

There are types of CFG derivation: left most derivation, right most derivation

Left most derivation:



Right most derivation:



Uses of CFG:

- **Parsing:** Parsing helps in generating **parse trees** that represent how words combine to form a sentence. These trees make it easier for computers to understand sentence structure and meaning.
- **Syntax checking:** CFGs are used to ensure that sentences follow grammatical rules. This is useful in applications like grammar checkers and compilers where correctness is important.
- **Machine translation:** By capturing the structure of sentences, CFGs guide translation from one language to another. This helps maintain the meaning and grammar of sentences across languages.
- **Speech recognition:** CFGs assist in predicting the most likely word sequences during speech input. This improves accuracy by filtering out grammatically incorrect possibilities.

Example Rules in English

S → NP VP

NP → Det N

VP → V NP

Det → "the" | "a"

N → "cat" | "dog

V → "chased" | "slept"

"**The dog chased a cat**" is a valid sentence.

- S → NP VP
- NP → Det N → the dog
- VP → V NP → chased a cat

Grammar rules for English

In Natural Language Processing (NLP), English grammar is usually represented through Context-Free Grammar (CFG) rules. These rules capture the hierarchical structure of English sentences.

1. Sentence Rule:

CFG Rule: S → NP VP

Every English sentence typically consists of a Noun Phrase (subject) followed by a Verb Phrase (predicate).

Example:

- *"The boy runs."*
- Here, NP = The boy, VP = runs.

2. Noun Phrase (NP) Rules:

Noun phrases function as the subject or object in sentences.

CFG Rule: NP → Det N,

NP → Det Adj N,

Examples:

- *"The cat"* → Det + N
- *"A big house"* → Det + Adj + N

3. Verb Phrase (VP) Rules:

Verb phrases describe the action or state in a sentence.

CFG Rules: VP → V NP

VP → V NP PP

Examples:

- *"chased the dog"* → V + NP
- *"put the book on the table"* → V + NP + PP

4. Prepositional Phrase (PP) Rules

Prepositional phrases add extra details like place, time, and shows relationship between a noun/ pronouns and other words.

CFG Rule: PP → P NP

Examples:

- *"in the park"*
- *"on the table"*

5. Adverb Phrase (AdvP) Rules

Adverbs modify verbs, adjectives, often showing manner.

CFG Rules: AdvP → Adv

AdvP → Adv Adv

Examples:

- *"quickly"* → Adv
- *"very slowly"* → Adv + Adv

6. Adjective Phrase (AdjP) Rules

Adjectives modify nouns, to give more information.

CFG Rules: AdjP → Adj

AdjP → Adv Adj

Examples:

- *"red"* → Adj
- *"very tall"* → Adv + Adj

Sentence: *"The boy chased the dog in the park"*

- S → NP VP
- NP → Det N → the boy
- VP → V NP PP → chased (NP) (PP)
- NP → Det N → the dog
- PP → P NP → in (NP)
- NP → Det N → the park

Final string = *"The boy chased the dog in the park".*

English grammar in NLP is modeled using CFG rules that break down sentences into Noun Phrases, Verb Phrases, Prepositional Phrases, etc. These rules form the backbone of parsing, translation, and many other language technologies.

Treebanks

A **Treebank** in Natural Language Processing is a collection of sentences that are tagged with their syntactic or grammatical structure, usually represented in the form of parse trees. These trees show how words in a sentence are grouped into phrases and how they relate to each other according to grammar rules. Treebanks are used in NLP to train computers to understand sentence structures, check grammar, and improve applications like translation and chatbots.

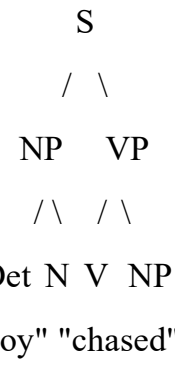
- Treebanks provide real examples of sentences with their grammatical structures, which help in studying and processing natural language.
- They act as training data for parsers and machine learning models in NLP.
- They give a standard reference for testing and comparing different language processing systems.
- Treebanks help in creating linguistic resources for multiple languages, not just English.

Types of treebanks:

1. Syntactic Treebanks

- Focus on the grammatical (syntactic) structure of sentences.
- Sentences are annotated with parse trees, showing how words combine into phrases (like NP, VP, PP).
- They are mainly used for parsing, grammar checking, and machine translation.
- Example: *Penn Treebank* (the most widely used English syntactic treebank).

Sentence: *"The boy chased the dog."*



2. Semantic Treebanks

- Go beyond grammar to represent the meaning (semantics) of sentences.

- Sentences are annotated with logical forms, semantic roles, or meaning representations.
- They are used in question answering, semantic parsing, and natural language understanding.
- They help in capturing relationships between entities and actions, which is crucial for tasks like information extraction and text understanding.
- Semantic treebanks are widely used in dialog systems and chatbots, enabling machines to respond with contextually meaningful answers.

Sentence: *"The boy chased the dog."*

- Agent (who did the action): The boy
- Action (verb): chased
- Patient (who received the action): the dog

Syntactic Treebanks → Show grammatical structure of sentences (phrases and word order).

Semantic Treebanks → Show meaning of sentences (who did what, to whom).

Normal Forms for Grammar

Normal Forms reduce the grammar into simpler and more consistent patterns, such as rules that always produce only two variables, or rules that always begin with a terminal. By restricting the form of rules in this way, parsing algorithms become easier to design, faster to execute, and more systematic to apply.

When working with Context-Free Grammars (CFGs) in Natural Language Processing, the grammar rules can sometimes be complex. This makes parsing difficult because parsers need to handle too many variations. To solve this, grammars are often converted into Normal Forms, which are special standardized formats for the production rules.

Chomsky Normal Form (CNF):

Chomsky Normal Form (CNF) is a simplified form of context-free grammar where every production rule is either of the form $A \rightarrow BC$ (two non-terminals) or $A \rightarrow a$ (a single terminal). It is mainly used to make parsing algorithms like CYK simpler and more systematic.

- $A \rightarrow BC$ (a variable produces two variables), OR
- $A \rightarrow a$ (a variable produces a single terminal), OR
- $S \rightarrow \varepsilon$ (start symbol can produce empty string).

Example:

$S \rightarrow AB$

A → a

B → b

Generates string “ab”

Example:

S → ASA | aB

A → B | S

B → b

Step 1:

A → B and A → S are unit productions.

- Replace A → B with A → b (since B → b).
- Replace A → S with all productions of S.

So grammar becomes:

S → ASA | aB

A → b | ASA | aB

B → b

Step 2:

In aB, the terminal a is mixed with variable. Introduce new variable X → a.

So grammar looks like

S → ASA | XB

A → b | ASA | XB

B → b

X → a

Step 3:

ASA has three symbols. Break it: Introduce Y → SA.

S → AY | XB

A → b | AY | XB

B → b

X → a

Y → SA

So the production rules This CFG is now in Chomsky Normal Form (CNF).

Dependency Grammar:

Dependency grammar is a way of understanding how words in a sentence are connected to each other. Instead of looking at the whole sentence as big chunks (like noun phrase or verb phrase), it focuses on word-to-word links, showing which word depends on which.

You can think of a sentence like a puzzle, where each word is a piece. Dependency grammar shows how these pieces fit together to form the full picture. This idea is not new — it has been used in linguistics for centuries, even in studying the grammar of languages like Sanskrit.

1. **Word tokens:** In natural language processing, the text is divided into basic units called tokens. A sentence is made up of a group of word tokens. Each of these tokens has a unique function and is a building block of the language. For the sentence "The cat chased the mouse." The tokens are : "The", "cat", "chased", "the", "mouse".
2. **Dependency relations**: show how words in a sentence are connected based on their grammatical roles. For example, in "The cat chased the mouse," the word "cat" depends on "chased" as the subject, while "mouse" depends on "chased" as the object.
3. **Root node**: Every sentence has one main word, usually the main verb, called the root. Example: in *"The cat chased the mouse"*, the root is *"chased"*.
4. **Governor and dependent**: In each link, one word is the governor (main word) and the other is the dependent (the word connected to it). Example: in *"The cat chased the mouse"*, *"chased"* is the governor and *"cat"* is the dependent (subject) and "*mouse*" is the dependent (object).
5. **Dependency labels**: Each connection has a label that tells the role. Example: *"cat"* is labeled subject (nsubj) and *"mouse"* is labeled object (dobj) of the verb *"chased"*.

Example:

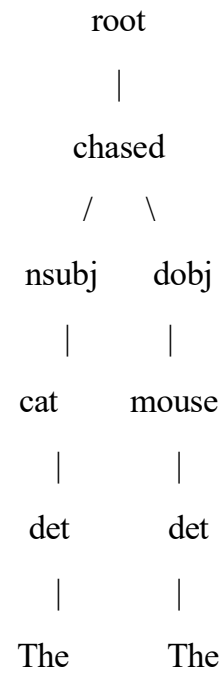
Let's use the dependency grammar framework to represent the following sentence

"The cat chased the mouse."

Dependency relations:

- **chased** → root (main verb)

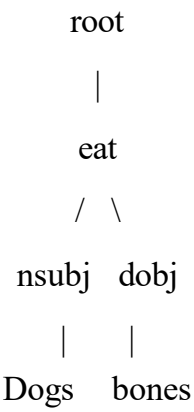
- **cat** → nsubj (subject of "chased")
- **mouse** → dobj (object of "chased")
- **The** (before cat) → det (determiner)
- **The** (before mouse) → det (determiner)



Example: "Dogs eat bones."

Dependency relations:

- **eat** → root (main verb)
- **Dogs** → nsubj (subject of "eat")
- **bones** → dobj (object of "eat")



Dependency grammar is an important idea in NLP that shows how words in a sentence are linked to each other, like puzzle pieces fitting together. It uses arrows or lines to represent these word-to-word connections.

Ambiguity

Ambiguity in Natural Language Processing (NLP) refers to situations where a word, phrase, or sentence can be understood in more than one way. It occurs because natural languages are flexible, and the same word or structure can take on different meanings depending on the context. Ambiguity makes it difficult for computers to correctly interpret text or speech, since they need to figure out the intended meaning among several possibilities.

For example:

- **Word:** "book"
    - *Verb*: "Can you book a ticket?" → Here, book means to reserve.
    - *Noun*: "I read a book." → Here, book is an object you read.
- **Word:** "play"
    - *Noun*: "The play was great." → Here, play means a theatrical performance.
    - *Verb*: "They play outside." → Here, play shows an action.

The POS tagger has to analyze the context and surrounding words to assign the correct tag (noun, verb, etc.).

Syntactic Parsing

Syntactic Parsing in Natural Language Processing (NLP) is the process of analyzing the grammatical structure of a sentence to determine how words are related to each other and how they combine to form meaningful sentences. It involves generating a parse tree or dependency structure according to the grammar of the language, which represents the syntax of the sentence.

Goal: Identify how words group into phrases and how those phrases relate to each other.

Key Aspects

Grammatical Structure:

The core function is to identify the syntactic relationships and grammatical structure of a sentence by breaking it into its parts of speech and phrases.

Parse trees:

Parsers represent the grammatical structure as a tree, where leaf nodes are the words and internal nodes represent phrases and parts of speech, showing how the sentence is formed

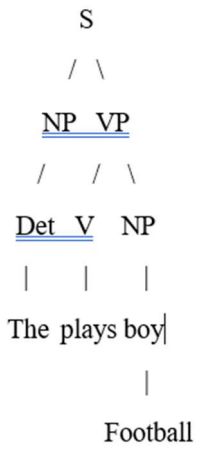
How it Works

1. **Input:** A software component called a parser takes a sentence as input.
2. **Analysis:** The parser analyzes the words and their arrangement, often after lexical analysis (breaking text into words).
3. **Output:** It produces a structured representation, such as a parse tree, that shows the grammatical relationships between the words.

Types of Syntactic Parsing

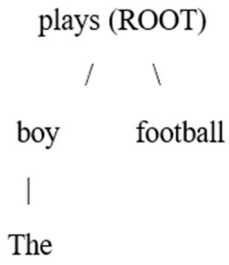
1. Constituency Parsing

- Breaks a sentence into phrases (constituents) like noun phrases (NP) and verb phrases (VP).
- Produces a parse tree that shows the hierarchical structure of the sentence.
- Example: ″
- *The boy plays football.″* → [S [NP The boy] [VP plays [NP football]]]



2. Dependency Parsing

- Focuses on word-to-word relationships (head → dependent).
- Produces a dependency graph instead of a tree of phrases.
- Example: *"The boy plays football."* → "plays" (root), "boy" (subject), "football" (object)



Shallow parsing

Shallow parsing, also called chunking, is the process of identifying and extracting phrases (like noun phrases and verb phrases) from a sentence without building a full parse tree. Instead of analyzing the entire grammatical structure, it groups words into meaningful chunks.

- Provides limited syntactic information but is faster and more efficient than full parsing.
- Focuses on phrases (NP, VP, PP) instead of complete syntax.
- The process usually involves tokenization, POS tagging, and then chunking to form phrases.

Example

Sentence: *"The boy plays football in the park."*

- Noun Phrase (NP): **The boy**
- Verb Phrase (VP): **plays football**

- Prepositional Phrase (PP): **in the park**

Unlike full syntactic parsing, it won't show how these chunks are connected in a hierarchy.

Applications

- **Information Extraction**: Identifying key entities and phrases from text.
- **Text Classification**: Extracting relevant features from sentences.
- **Sentiment Analysis**: Identifying opinionated phrases.
- **Machine Translation**: Providing partial syntactic information for translation.

Shallow parsing (chunking) is a simplified form of parsing that breaks a sentence into chunks like noun or verb phrases, without analyzing the full grammatical hierarchy.

Probabilistic CFG

A Probabilistic Context-Free Grammar (PCFG) is a Context-Free Grammar (CFG) extended with probabilities attached to each production rule. This allows modelling not just whether a string belongs to the language, but also how likely each parse or derivation is. PCFGs are widely used in natural language processing (NLP), parsing, and statistical grammar models.

Formally, a PCFG is a 5-tuple:

$$G=(N,\Sigma,R,S,P)$$

where:

- **N** = Nonterminals (like S, NP, VP)
- **Σ** = Terminals (actual words like dog, eats)
- **R** = Rules (productions like S → NP VP)
- **S** = Start symbol (usually S)
- **P** = Probabilities for each rule

For every nonterminal, probabilities of all its rules must add up to 1.

- P(A→s) means the probability of using the rule A→ when expanding nonterminal A.
- Each probability value is between 0 and 1.
- For every nonterminal A, the total probability of all its rules must add up to 1.

Example PCFG:

Probabilistic Context Free Grammar G = (N, T, S, R, P)

- N = {S, NP, VP, Det, N, V}
- T = {‘’the, ‘an’, ‘dog’, ‘apple’, ‘eats’, ‘barks’}
- S = S
- R = { S → NP VP

NP → Det N | john

VP → V NP | V

Det → ‘the’ | ‘an’
N → ‘dog’ | ‘apple’

V → ‘eats’ | ‘barks’}

- P = R with associated probability as in the table below;

Rule	***Probability***	***Rule***	***Probability***
S → NP VP	1.0	Det → ‘an’	0.6
		Det → ‘the’	0.4
NP → DET N	0.9	N → ‘dog’	0.5
NP → john	0.1	N → ‘apple’	0.5
VP → V NP	0.7	V → ‘eats’	0.5
VP → V	0.3	V → ‘barks’	0.5

Example 1: “The dog barks”

Probability is:

$$P(\text{derivation})=P(S\rightarrow NPVP)\times P(NP\rightarrow DetN)\times P(Det\rightarrow the)\times P(N\rightarrow dog)\times P(VP\rightarrow V)\times P(V\rightarrow barks)$$

$$=1.0\times 0.9\times 0.4\times 0.5\times 0.3\times 0.5$$

$$=0.027$$

Example 2: “John eats an apple”

Probability is:

$$P(\text{derivation})=P(S\rightarrow NPVP)\times P(NP\rightarrow john)\times P(VP\rightarrow VNP)\times P(V\rightarrow eats)\times P(NP\rightarrow$$

$$\text{DetN})\times P(\text{Det}\rightarrow \text{an})\times P(\text{N}\rightarrow \text{apple})$$

$$=1.0\times 0.1\times 0.7\times 0.5\times 0.9\times 0.6\times 0.5$$

$$= 0.00945$$

A PCFG is a CFG with probabilities assigned to each rule, where the probability of a derivation is the product of its rule probabilities. It helps in choosing the most likely parse among many possible parses in natural language processing.

Feature Structure

A feature structure in Natural Language Processing (NLP) is a formal representation of grammar study information using a set of attribute–value pairs, where each attribute (feature) describes a grammatical property (such as number, gender, tense, person, case), and each value specifies the corresponding property of a word, phrase, or sentence.

- Feature structures are mostly used to handle agreement between nouns and verbs (subject–verb agreement).
- They can be simple (with atomic values) or complex (with nested structures).
- They are useful in parsing, where sentence structure is analyzed.
- They also help to represent syntactic and semantic information in computational grammar study.

Representation of Feature Structures

Feature structures in NLP are usually represented using attribute–value pairs, where each attribute specifies a grammatical property and its value gives the corresponding feature. These can be represented in different formats such as:

Noun: Nouns are represented using attributes such as Category (CAT), Number (NUM), Gender, Case, Person.

Example 1: "dogs"

[CAT noun
NUM plural
CASE nominative]

Example 2: "girl"

[CAT noun
NUM singular
GENDER feminine
CASE accusative]

Verb: Verbs are represented with attributes like Category (CAT), Tense, Number, Person, Aspect.

Example 1: "runs"	**Example 2: "played"**
[CAT verb	[CAT verb
TENSE present	TENSE past
NUM plural	ASPECT perfective
PERS third]	NUM singular
	PERS third]

Types of Feature structure

1. Attribute–Value Matrix (AVM) Representation

The Attribute–Value Matrix (AVM) is a way to represent feature structures in a simple, tabular form. Each attribute (feature) describes a grammatical property like category, number, or tense, and its value specifies the property of the word, phrase, or sentence. AVMs are widely used in NLP to capture morphological and syntactic information clearly and concisely.

Example: For the noun **"dogs"**:

[CAT noun

NUM plural

CASE nominative]

2. Nested feature structure

A nested feature structure represents grammar study information using sentences, showing how smaller units (like nouns or verbs) are part of larger structures (like phrases or sentences). It allows capturing internal relationships such as subject–verb agreement or modifier relationships.

Example (sentence: "**The boy runs**"):

[S

SUBJ: [CAT: noun, NUM: singular, CASE: nominative]

PRED: [CAT: verb, TENSE: present, NUM: singular , PERS: third]

]

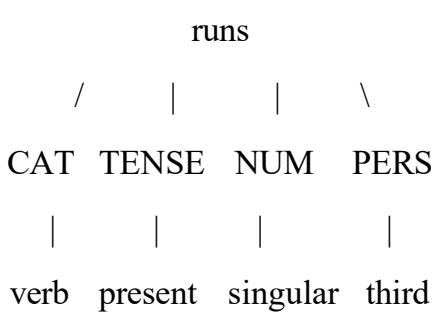
- SUBJ → "The boy"
- PRED → "runs"

- SUBJ and PRED each have their own feature structures nested within the sentence.
- This helps in parsing, grammar checking, and agreement constraints.

3. Tree-like representation

In tree-like representation, feature structures are shown as a diagram or tree, where the main node represents the word or phrase and branches represent attributes with their corresponding values. This makes it easy to visualize grammatical properties and hierarchical relationships in sentences.

Example: Verb "runs"



Applications in Feature Structure

- **Grammar Checking** – Detect errors in subject–verb agreement, tense, or case.
- **Information Extraction** – Extract structured information from text using syntactic and grammatical features.
- **Machine Translation (MT)** – Map linguistic properties from source to target language correctly.
- **Natural Language Understanding (NLU)** – Encode syntax and semantics for chatbots and QA systems.

Named entity Recognition (NER)

Named Entity Recognition (NER) in NLP focuses on identifying and categorizing important information known as entities in text. These entities can be names of people, places, organizations, dates, etc. It helps in transforming unstructured text into structured information which helps in tasks like text summarization, knowledge graph creation and question answering.

- NER is used to find important words or phrases in text and put them into fixed categories.
- It supports other NLP tasks like POS tagging and parsing by giving more meaning to text.

Common Entity Types (with examples):

- **Person** → *Albert Einstein*
- **Organization** → *TCS*
- **Location** → *Paris*
- **Date/Time** → *5th May 2025*
- **Quantity/Percentage** → *50%, $100*

It helps in handling ambiguity by analyzing surrounding words, structure of sentence and the overall context to make the correct classification. It means context can change based on entity's meaning.

Unambiguous NER Examples

1. Mahatma Gandhi led India's freedom movement → Person
2. Microsoft develops computer software → Organization

Ambiguous NER Examples

1. **Apple**
    - Apple launched the iPhone 16 → Organization
    - I ate a red apple this morning → Object / Fruit
2. **Amazon**
    - Amazon is expanding rapidly → Organization
    - The Amazon is the largest rainforest → Location

Working of Named Entity Recognition (NER)

1. **Text Analysis**: Process the text to find possible entities. This gives a rough idea of where important information might be present.
2. **Sentence Boundaries**: Detect where sentences start and end (using punctuation, capitalization). Helps preserve meaning and avoids mixing entities across sentences.
3. **Tokenization & POS Tagging**: Split text into words (tokens) and mark their grammatical role. Parts of speech like nouns or proper nouns act as clues for entities.
4. **Entity Detection & Classification**: Identify tokens/groups as entities and assign categories (Person, Organization, Location, etc.). This is the main step where raw text turns into structured information.
5. **Model Training**: Train models on labeled data to learn patterns. The more data it sees, the better the model recognizes entities.
6. **Adaptation**: Models improve and handle new languages, styles, or unseen entities using context. This makes NER flexible and useful across domains.

Methods of Named Entity Recognition (NER)

1. Rule-based Approach

- Uses hand-written rules, regular expressions, and dictionaries (gazetteers).
- Example: If a word starts with a capital letter and follows "Mr.", classify it as a Person.

2. Dictionary Approach

- Matches words or phrases with pre-compiled lists of names (cities, companies, etc.).
- Works well for known entities, but misses new/unlisted ones.

3. Machine Learning-based Approach

- Uses models like Hidden Markov Models (HMMs), Maximum Entropy, or Conditional Random Fields (CRFs).
- Learns from features like capitalization, part-of-speech tags, context words.

Unit-4

Essays

1. Explain Word Sense Disambiguation (WSD) and its methods.

Word Sense Disambiguation

Word Sense Disambiguation (WSD) is the process of figuring out which meaning of a word is meant in a given context. Many words have more than one meaning (polysemy). For example, in the sentence "I went to the bank," the word "bank" could mean a financial institution or the side of a river. We use the surrounding words to know the correct meaning.

WSD is important in natural language processing (NLP) because wrong word meanings can cause problems:

- Machine translation may give incorrect translations.
- Information retrieval may return unrelated results.
- Question-answering systems may provide inaccurate answers.

Why WSD is important

- Machine translation: To translate Sentences correctly, the machine must understand the correct meaning of each word Woong meaning wrong translation.
- Information Retrieval: Search engines can give better results if they know the correct word Sense. Searching for "apple" it is fruit or company?
- Chatbots and Al: To respond accurately, the System must Understand the context of words.

How WSD works:

1. look at the surrounding words (context):
   - Example = "river" near "bank" = bank →river side
   - Example = "bank" near "river" = bank → financial institution
2. check dictionary of knowledge base:
   - Match meanings with the context words.
3. choose the most likely meaning:
   - Based on probability or patterns

Types of Word Sense Disambiguation

1. Knowledge-Based WSD:

- Knowledge-based approaches utilize lexical resources such as dictionaries and semantic networks to determine word meanings. The Lesk Algorithm works over this approach.
- Compare context words with dictionary definitions of candidate senses.
- Calculate overlap between contextual words and definitional content.
- Select the sense with maximum overlap score.

Advantages:

- Does not require annotated training data
- Leverages existing linguistic knowledge bases
- Provides interpretable disambiguation decisions

Example: check dictionary meaning of "light" and match with sentence context.

2. Supervised Learning WSD:

Supervised approaches treat WSD as a classification problem, training machine learning models on datasets where word instances have been manually annotated with correct senses.

Key characteristics:

- Treat WSD as a classification task. Train machine learning models (e.g., SVM, decision trees, neural networks) on sense-annotated data.
- Process: Extract features (surrounding words, syntax, etc.), Train a classifier, Apply model to new text.
- While supervised methods achieve high accuracy, they face the challenge of obtaining sufficient annotated data for all word-sense combinations.

Example: Teach the computer with sentences where "bank" is labelled as financial institution or river side.

3. Unsupervised Learning WSD:

Unsupervised approaches operate without un-labelled training data, instead relying on distributional patterns in large text corpora.

Principle: Words used in similar contexts usually share meanings.

Modern methods:

- Use word embeddings and contextual representations
- Apply clustering algorithms
- Leverage large language models

Best for: Situations where annotated data is limited or unavailable.

Example: using clustering methods to identify senses of “but” in different sentences.

Applications of WSD:

- Machine translation → google translate
- Search engines → google
- Spell checkers → current word suggestions
- Voice assistants → siri, alexa
- Text analysis → sentiment analysis, content understanding

2. Describe the Transformer Architecture and its impact on semantic analysis.

The Transformer architecture, introduced by Vaswani et al. in 2017 through the paper “Attention is All You Need”, marked a major breakthrough in the field of Natural Language Processing (NLP). Unlike earlier models such as recurrent neural networks (RNNs) and long short-term memory networks (LSTMs), the Transformer does not rely on sequential processing of text.

Instead, it makes use of a self-attention mechanism, which allows it to capture the relationships between words in a sentence regardless of their distance from one another. This innovation has not only improved efficiency but also transformed the way machines understand semantic meaning in language.

1. Architecture of the Transformer

The Transformer uses an encoder-decoder structure, with both parts made up of multiple layers.

- Encoder: Takes input text and creates contextual embeddings. Each encoder layer has:
  - Multi-Head Self-Attention: Lets the model focus on different words in a sentence at the same time. Example: in “The cat sat on the mat because it was tired,” it links “it” to “cat.”
  - Feed-Forward Network: A fully connected layer that improves word representations.
- Decoder: Produces the output sequence (e.g., a translation). It uses self-attention and encoder-decoder attention so the output depends on both the encoder’s context and previous outputs.
- Positional Encoding: Adds word order information since the model processes words in parallel.
- Residual Connections & Layer Normalization: Make training stable and help build deep models.

2. Advantages over Previous Models

Before Transformers, RNNs and LSTMs dominated NLP. However, they suffered from problems like vanishing gradients, slow training, and difficulty in capturing long-distance dependencies. The Transformer overcomes these limitations by:

- Allowing parallel processing of text, making training faster and more scalable.
- Using self-attention to capture dependencies between words across long sentences.
- Scaling up effectively to very large models such as BERT and GPT, which are now central to NLP.

3. Impact on Semantic Analysis

Semantic analysis means extracting meaning from text by understanding how words, phrases, and sentences relate. Transformers have greatly improved this process:

- Better Contextual Understanding: Self-attention looks at the whole sentence when deciding a word's meaning, helping solve ambiguity (e.g., "bank" as money vs. riverbank).
- Word Sense Disambiguation (WSD): Transformers create context-aware embeddings. Unlike older models with one fixed vector per word, models like BERT represent different senses based on context.
- Rich Semantic Representations: Pre-trained models (BERT, RoBERTa, GPT) capture deep relationships in text. These can be fine-tuned for tasks like sentiment analysis, summarization, or question answering.
- Cross-Lingual Semantics: Transformers improve translation by understanding meaning across languages, handling idioms and context better.
- Scalability and Transfer Learning: Training on massive text corpora allows Transformers to learn general knowledge, which can be adapted to specific domains with less effort.

4. Applications in Semantic Analysis

The impact of Transformers on semantic analysis can be seen in multiple NLP applications:

- Machine Translation – Producing contextually accurate translations across languages.
- Information Retrieval – Improving search engines by understanding query intent.
- Question Answering – Providing precise answers by interpreting the meaning of both the question and text passages.
- Text Summarization – Capturing key ideas while preserving meaning.

- Sentiment Analysis – Identifying not just positive or negative sentiment, but also subtle emotional tones.

The Transformer has transformed semantic analysis by using attention-based parallel processing instead of sequential models. It captures long-range word relationships, handles ambiguity, and creates strong semantic embeddings.

Models like BERT and GPT, built on this architecture, have made semantic analysis more accurate, efficient, and scalable, leading to big improvements in translation, search, and dialogue systems.

3. Discuss syntax-driven semantic analysis and thematic roles.

Syntax-driven semantic analysis is the process of deriving the meaning (semantics) of sentence based on its syntactic structure (grammar / parse tree). The syntax (grammar rules) guide flow we build meaning semantics.

Syntax-driven semantic analysis in NLP involves determining a sentence's meaning by first analyzing its grammatical structure (syntax) and then applying lexical and grammatical rules to derive the meaning. It assigns meaning based on static knowledge of words and grammar, producing context-independent and inference-free representations, unlike broader semantic analysis that considers context and nuances.

How it works:

1. **Syntactic Analysis (Parsing):** The process begins with analyzing the sentence's structure to create a parse tree or similar representation.
2. **Mapping to Meaning:** The syntactic structure is then used to determine the meaning. This involves:
    - **Lexical Semantics:** Analyzing the dictionary meanings of individual words.
    - **Grammatical Rules:** Applying rules of formal grammar to understand how the words and phrases relate to each other

Example:

- **Sentence:** "The dog chases the ball."
- **Syntax-Driven Analysis:**
    - **Syntactic analysis**: reveals "dog" is the subject, "chases" is the verb, and "ball" is the object.

- **Syntax-driven semantic analysis**: assigns an agent role to "dog" and a patient role to "ball" based on their placement and relationship to the verb

Techniques

1. Syntax-Directed Translation

- Associates semantic rules or actions with grammar productions.
- Each rule defines how meaning is built as the syntax is parsed.

Example:

- S → NP VP
- S.semantic = combine(NP.semantic, VP.semantic)

Meaning is constructed as the parse tree is generated.

2. Attribute Grammars

- Extends context-free grammars with attributes that carry semantic information.
- Two types:
  - **Synthesized attributes:** Pass information upward (e.g., meaning of a phrase).
  - **Inherited attributes:** Pass information downward (e.g., context or expected type).
- Used in both compiler design and natural language processing.

3. Compositional Semantics

- Follows Frege's principle: meaning of a sentence comes from meanings of parts + their combination.
- Often uses lambda calculus or logical forms for structured representation.

Key Characteristics:

- **Static Knowledge:** Relies on pre-defined rules from the lexicon and grammar, not dynamic context or inferences.
- **Structure-Based:** Meaning is derived directly from the identified syntactic structure of the sentence.
- **Context-Independent:** The analysis of a sentence's meaning is free from external contextual information.

Thematic Roles:

A thematic role is the semantic role that a noun phase plays in the situation described by a sentence.

It helps answer questions like:

- Who did the action?
- What was affected?
- With what instrument?
- Where and when did it happen?

This is achieved using thematic roles (also called sematic roles)

Major types of thematic roles

- **Agent**: initiator of the action

  ex: The boy opened the door.

- **Theme/patient:** The entity that is affected by the action.

  Ex: Lova opened the door. (door-theme)

- **Experience:** The one who feels or perceives something

  Ex: Usha felt happy. (Usha=experience)

- **Instrument:** The means used to perform the action.

  Ex: He opened the door with key. (Key=instrument)

- **Beneficiary / recipient:** The who benefits or receives something.

  Ex: He gave a book to Lova. (Lova=recipient)

- **Source:** The starting point of movement.

  Ex: She came from Hyderabad. (Hyderabad=source)

- **Location:** The place where the action happens.

  Ex: Children are playing in the park. (park=location)

It explain sentence meaning beyond grammar. Helps in understanding who is doing what to whom, useful in analysing syntax and semantics.

Shorts

1. What are semantic attachments

In Natural Language Processing (NLP)**,** semantic attachments link syntactic structures to their semantic roles (like agent, patient, or instrument). They help map sentence elements to conceptual representations or logical forms, enabling machines to understand and represent meaning in semantic networks or predicate–argument structures**.**

How Semantic Attachments Work

Defining Roles: Semantic attachments assign specific functions to words within a sentence's meaning. For instance, in "John ate the pizza with a fork," "John" is the agent (who acts), "pizza" is the patient (what is acted upon), and "fork" is the instrument (the tool used).

Contextual Meaning: These attachments help resolve ambiguity, particularly for words with multiple meanings (polysemy). By understanding the context and the semantic attachments of surrounding words, NLP systems can determine the correct sense of a word (word sense disambiguation).

Formal Representation: Semantic attachments can be shown using logical forms (like predicate logic) to give a structured meaning of a sentence.

Key Concepts and Applications

- **Lexical Semantics:** Semantic attachments are central to lexical semantics, which studies the meaning of individual words and their relationships to each other.
- **Semantic Networks:** These are knowledge graphs that represent words, concepts, and their relationships, with semantic attachments forming the basis of these connections.
- **Semantic Analysis:** The overall process of understanding meaning in language, which relies on identifying semantic attachments to analyze words, phrases, and sentences accurately.

Applications:

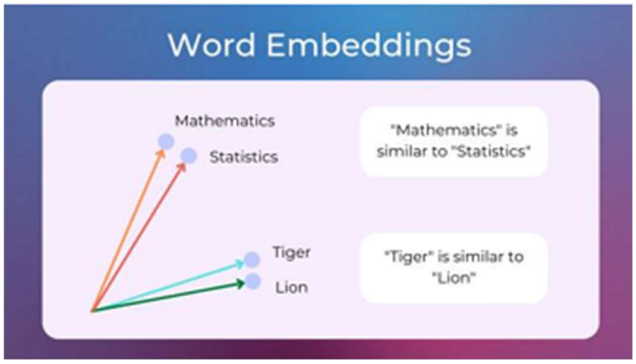
Semantic analysis and attachments are used in various applications, including:

- **Information Retrieval:** To understand context and improve search accuracy.
- **Text Summarization:** To grasp the core meaning for condensation.
- **Machine Translation:** To accurately convey meaning across languages.

2. Define Word Embeddings.

Word embedding is a fundamental concept in Natural Language Processing (NLP) used to represent words as dense, low-dimensional vectors (real-valued numbers) in such a way that words with similar meanings are located close to each other in the vector space.

- Word embedding is a technique in Natural Language Processing (NLP) that maps words into a continuous vector space where words with similar meaning or context are represented by vectors that are close to each other.
- Unlike traditional sparse methods, embeddings are dense, low-dimensional, and semantic-rich representations.



Characteristics

- Dense Representation: Each word represented by ~50–300 dimensions instead of thousands.
- Semantic Closeness: Similar words (e.g., doctor, nurse) are placed close in vector space.
- Linear Relationships: Vector arithmetic can represent analogies:

$$King - Man + Woman \approx Queen$$

- Generalization: Learns from large corpora, allowing unseen but related words to be positioned meaningfully.

Why Word Embeddings?

- Traditional methods like one-hot encoding represent words as sparse, high dimensional vectors (with many zeros), which fail to capture semantic relationships (e.g., "king" and "queen" would be completely unrelated).
- Word embeddings solve this by learning continuous vector representations that preserve semantic and syntactic similarities.

Applications

- Sentiment Analysis (positive/negative review classification).
- Machine Translation (Google Translate).
- Information Retrieval & Search Engines (semantic similarity for better ranking).
- Question Answering & Chatbots (context-aware answers).

3. What are selectional restrictions?

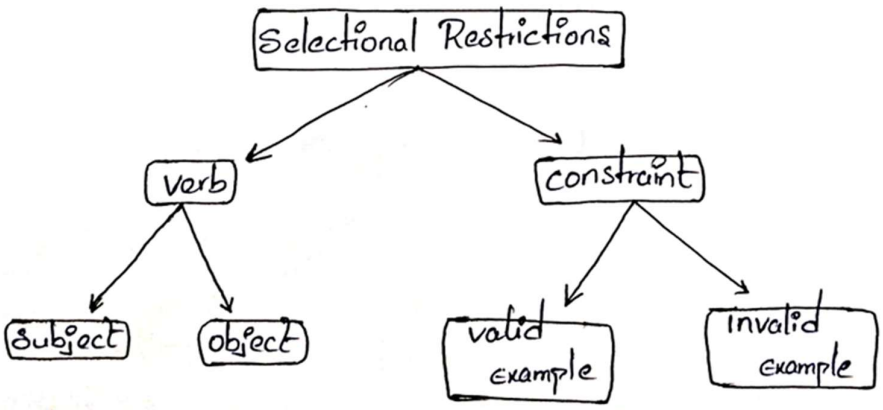
Selectional restriction are rules about which about words can go together in a meaningful way. Especially, verbs control what kind of subjects and objects they can take.

Purpose: To make sure sentences are semantically correct (not just grammatically correct).

Example:

Grammar Ok: The rock drank water

Semantic WRONG: rocks cannot drink → breaks



Types of restrictions:

Subject restriction:

- Who/want can do the action.
- Speak → subject must be human/ animate.
- The teacher spoke. ✔
- The car spoke. ❌ (unless metaphorical)

Object restriction.

- What the action is done to.
- Eat → object must be edible.
- She ate rice ✔
- She ate a dream ❌

Predicate / complement restriction:

- What follows after verb phrases.
- Give → needs giver (animate), receiver (animate), thing (transferable).
- He gave her a gift. ✔
- The chair gave him advice. ❌

Examples:

Verb: kill

- Subject must be animate (someone doing killing)
- Object must be animate (someone being killed).
- The hunter killed the tiger. ✔
- The wind killed the rock. ❌

Why important in NLP:

- Natural language understanding → helps computer understand meaning.
- Machine translation → avoids nonsense outputs.
- Chatbots/ voice assistants → ensures logical answers.
- Information extraction → only picks correct facts.

4. Mention any two WSD techniques.

Word Sense Disambiguation (WSD) is the process of figuring out which meaning of a word is meant in a given context. Many words have more than one meaning (polysemy). For example, in the sentence "I went to the bank," the word "bank" could mean a financial institution or the side of a river. We use the surrounding words to know the correct meaning.

1. Knowledge-Based WSD:

- Knowledge-based approaches utilize lexical resources such as dictionaries and semantic networks to determine word meanings. The Lesk Algorithm works over this approach.
- Compare context words with dictionary definitions of candidate senses.
- Calculate overlap between contextual words and definitional content.
- Select the sense with maximum overlap score.

Advantages:

- Does not require annotated training data
- Leverages existing linguistic knowledge bases
- Provides interpretable disambiguation decisions

Example: check dictionary meaning of "light" and match with sentence context.

2. Supervised Learning WSD:

Supervised approaches treat WSD as a classification problem, training machine learning models on datasets where word instances have been manually annotated with correct senses.

Key characteristics:

- Treat WSD as a classification task. Train machine learning models (e.g., SVM, decision trees, neural networks) on sense-annotated data.
- Process: Extract features (surrounding words, syntax, etc.), Train a classifier, Apply model to new text.
- While supervised methods achieve high accuracy, they face the challenge of obtaining sufficient annotated data for all word-sense combinations.

Example: Teach the computer with sentences where "bank" is labelled as financial institution or river side.

Unit-5

Essays

1. Explain coherence and reference phenomena in discourse analysis.

Coherence and Reference Phenomena in Discourse Analysis

Discourse analysis is the study of how language is used inp texts and conversations to convey meaning beyond individual sentences. It examines how sentences are connected to form meaningful communication. Two important concepts in discourse analysis are coherence and reference. Both are essential for understanding how texts make sense as unified wholes rather than random collections of sentences.

Coherence: The Logical Connection of Ideas

Coherence refers to the logical and meaningful connection between sentences or parts of a discourse that make the text understandable as a whole. It is what makes a piece of writing or speech *"make sense."*

While *cohesion* deals with grammatical and lexical links between sentences (like conjunctions and pronouns), *coherence* deals with the *underlying meaning* and logical flow of ideas.

Example:

- *"I was late to class. The traffic was heavy."*

The connection between the two sentences is logical; the second explains the reason for the first. Hence, the discourse is coherent.

Features of Coherence:

1. **Logical Sequencing:**

   The ideas in a text must follow a logical order. For example, cause should precede effect, and problem should be followed by solution.

2. **Consistency of Topic:**

   The sentences or paragraphs should be centered around one main theme or topic.

3. **Relevance:**

   Each sentence should contribute meaningfully to the central idea of the text.

4. **Reader's Knowledge and Inference:**

   Coherence also depends on the reader's ability to infer missing information and connect ideas based on world knowledge.

Reference Phenomena: Linking Expressions in Discourse

Reference in discourse analysis refers to the way linguistic expressions (such as pronouns, definite articles, or noun phrases) relate to other elements in the text or to things in the real world. Reference helps maintain coherence by linking different parts of a text together.

It answers the question: *"Who or what are we talking about?"*

Types of Reference:

1. **Anaphoric Reference:**
   When a word or phrase refers back to something mentioned earlier in the discourse.
   Example:
   "Ravi bought a car. He drives it every day."
   – "He" refers back to "Ravi," and "it" refers to "car."
2. **Cataphoric Reference:**
   When a word refers to something mentioned later in the discourse.
   Example:
   "When he arrived, Ravi was very tired."
   – "He" refers to "Ravi," which appears later in the sentence.
3. **Exophoric Reference:**
   When a word refers to something outside the text, usually within the physical or situational context.
   Example:
   "Look at that!" – The reference ("that") depends on the speaker's situation or gesture, not on the text.
4. **Endophoric Reference:**
   When the reference is within the same text. Both anaphoric and cataphoric references are forms of endophoric reference.

Examples

Example:
"Mary dropped the plate. It shattered into pieces."

- *Reference:* "It" refers to "the plate."
- *Coherence:* Cause and effect relationship makes the sentences logically connected.

Coherence and reference are important in making language clear and connected. Coherence means the ideas in a text fit together and make sense. Reference uses words like "he," "she," or "it" to link sentences and avoid repeating the same words. If a text

has no coherence, it will be confusing. If it has no reference, it will sound repetitive. Both together make communication easy to understand and meaningful.

2. Describe lexical resources used in NLP with examples.

Natural Language Processing (NLP) is a branch of Artificial Intelligence that helps computers understand, interpret, and generate human language. To process language effectively, NLP systems need access to lexical resources databases or collections that store information about words, their meanings, relationships, and usage.

A lexical resource is a structured collection of words and their linguistic information. It contains details such as:

- Word meanings (semantics)
- Part of speech (noun, verb, adjective, etc.)
- Synonyms and antonyms
- Morphological forms (e.g., run, runs, running)
- Relations between words (e.g., "cat" is a type of "animal")

These resources provide the necessary linguistic knowledge that NLP systems use to process and understand text.

Major uses of lexical resources in NLP

a) WordNet

WordNet is a common database for English words. It puts words with similar meanings into groups called synsets. It also shows how words are related, like synonyms (same meaning), antonyms (opposites), and other word links. This helps in understanding the meanings of words and their connection

Example:

- *Synonym set for "happy"*: {happy, glad, joyful, pleased}.
- *Relation:* "Car" is a type of "vehicle" (hypernym–hyponym relation).

Uses: Word sense disambiguation, semantic search, and machine translation.

b) FrameNet

FrameNet is a linguistic database based on frame semantics, which groups words by the situations or "frames" they describe. Each frame represents a typical event or

concept with its participants and roles. It helps understand how word meanings relate to real-world contexts and actions.

Example:

- In the frame "Buying," the roles include *Buyer*, *Seller*, *Goods*, and *Money.*
- Sentence: "Ravi bought a car from Ramesh for ₹5 lakhs." → Buyer = Ravi, Seller = Ramesh, Goods = Car, Money = ₹5 lakhs.

Uses: Semantic role labeling, event extraction, and understanding sentence meaning.

c) VerbNet

VerbNet is a database that groups English verbs with similar meanings and sentence patterns. It explains how verbs are used in sentences (grammar) and what they mean. VerbNet also builds on WordNet and FrameNet by giving details about verb roles and how actions and participants are related.

Example:

- The verb "give" belongs to the class of verbs expressing *transfer*.
- Pattern: [Agent gives Theme to Recipient]
- "Ravi gave a book to Meena." → Agent = Ravi, Theme = book, Recipient = Meena.

Uses: Machine translation, syntactic parsing, and text understanding.

d) PropBank (Proposition Bank)

PropBank is a linguistic database that annotates verbs in text with their predicate argument structures. It labels each participant's role in an action, such as *who did what to whom*. This helps in understanding sentence meaning for semantic analysis and NLP.

Example:

- Sentence: "The chef cooked a meal for the guests."
- Roles: ARG0 = The chef (doer), ARG1 = a meal (thing done), ARG2 = for the guests (beneficiary).

Uses: Semantic role labeling and event extraction.

e) ConceptNet

ConceptNet is a knowledge base that connects words and ideas using everyday facts and meanings. It helps computers understand real-world relationships, like "a dog is a pet" or "ice is cold." It is used in AI to give machines simple common sense like humans.

Example:

- "A dog is a pet."
- "Coffee is made from beans."

Uses: Understanding natural language meaning, chatbots, and reasoning tasks.

3. Discuss Anaphora Resolution and its applications in NLP.

Natural Language Processing (NLP) is a part of AI that enables computers to understand and process human language. In language, words like *he*, *she*, *it*, or *they* refer to something mentioned earlier—identifying these is called Anaphora Resolution. It helps computers understand sentence meaning and maintain the flow of ideas.

Anaphora

The term anaphora comes from Greek, meaning "carrying back." In linguistics, anaphora refers to the use of words (usually pronouns) that depend on other words or phrases (called *antecedents*) for their meaning.

Example:

"Ravi bought a new car. He drives it every day."

- Here, He refers to Ravi, and it refers to car.

Anaphora Resolution

Anaphora Resolution is the process of identifying which word or phrase, known as the antecedent, a pronoun or referring expression (the anaphor) refers to in a sentence or dialogue. It helps maintain coherence and meaning in text by linking references like *he*, *she*, or *it* to the correct entities.

Types of Anaphora

1. **Pronominal Anaphora:**

When a pronoun refers to a noun mentioned earlier.

Example: "Sita lost her book. She is looking for it."
→ *She* → *Sita*, *it* → *book*

2. **Lexical Anaphora:**

When a word or phrase refers back to another word using similar meaning.

Example: “Ramu bought a car. The vehicle is red.”
→ *vehicle* refers to *car.*

3. **Event Anaphora:**

When a phrase refers to an earlier event or action.

Example: “He passed the exam. This made his parents happy.”
→ *This* refers to *passing the exam.*

Working of Anaphora Resolution

Identify Anaphors: Find words like *he, she, it, they, this,* etc., that refer to something else.

Find Possible Antecedents: Look for nouns or noun phrases earlier in the text that these words might refer to.

Apply Grammar Rules: Match gender, number, and person. *Example:* “Ravi lost his pen.” → *his* refers to *Ravi*, not *pen.*

Check Meaning and Context: See if the reference makes sense in the sentence. *Example:* “Ravi told Kiran that he passed.” → Context helps decide who *he* refers to.

Choose the Best Match: Select the noun that fits both grammatically and logically.

Applications of Anaphora Resolution in NLP

1. **Machine Translation:** Helps translate pronouns correctly between languages.

Example: In English, “she” and “he” must be correctly mapped in languages with gendered nouns.

2. **Information Extraction:** Used to identify entities and their relationships.

Example: “Ravi bought a car. He sold it later.” → Recognizing *Ravi* and *car* connections.

3. **Question Answering Systems:** Enables systems to understand and respond correctly.

Example:

- Q: “Who won the match?”
- A: “Virat Kohli. He scored a century.” → “He” → “Virat Kohli.”

4. **Text Summarization:** Ensures pronouns in the summary clearly refer to the correct entities.
5. **Sentiment Analysis:** Helps correctly associate opinions with their subjects.

Example: “The phone is great, but its battery is weak.” → “its” → “phone.”

Anaphora resolution helps computers understand language like humans by finding what pronouns and references refer to. It improves NLP tasks such as translation, summarization, chatbots, and information retrieval. With advanced AI and deep learning, systems can now resolve anaphora more accurately, making communication with computers more natural and smart.

Shorts

1. What is FrameNet?

FrameNet is a linguistic resource based on the theory of frame semantics, which organizes the meanings of words according to the conceptual structures, or "frames," that represent typical situations, events, or experiences. Each frame describes a scenario involving various participants and elements called frame elements that play specific roles in that situation. For example, in the *Buying* frame, there are roles like *Buyer*, *Seller*, *Goods*, and *Money*.

Example:

- In the frame "Buying," the roles include *Buyer*, *Seller*, *Goods*, and *Money.*
- Sentence: "Ravi bought a car from Ramesh for ₹5 lakhs." → Buyer = Ravi, Seller = Ramesh, Goods = Car, Money = ₹5 lakhs.

Uses: Semantic role labeling, event extraction, and understanding sentence meaning.

2. Define Anaphora.

The term anaphora originates from the Greek word meaning "carrying back." In linguistics, anaphora refers to the use of words—usually pronouns or referring expressions—that depend on earlier words or phrases, called antecedents, to complete their meaning. It helps connect sentences and maintain continuity in communication by avoiding repetition.

Example:
"Ravi bought a new car. He drives it every day."

→ *He* refers to *Ravi*, and *it* refers to *car.*

Anaphora plays an important role in natural language understanding and text cohesion, helping both humans and computers interpret references correctly.

3. List any two lexical resources used in NLP.

WordNet

WordNet is a common database for English words. It puts words with similar meanings into groups called synsets. It also shows how words are related, like synonyms (same meaning), antonyms (opposites), and other word links. This helps in understanding the meanings of words and their connection

Example:

- *Synonym set for "happy"*: {happy, glad, joyful, pleased}.
- *Relation:* "Car" is a type of "vehicle" (hypernym–hyponym relation).

Uses: Word sense disambiguation, semantic search, and machine translation.

VerbNet

VerbNet is a database that groups English verbs with similar meanings and sentence patterns. It explains how verbs are used in sentences (grammar) and what they mean. VerbNet also builds on WordNet and FrameNet by giving details about verb roles and how actions and participants are related.

Example:

- The verb "give" belongs to the class of verbs expressing *transfer*.
- Pattern: [Agent gives Theme to Recipient]
- "Ravi gave a book to Meena." → Agent = Ravi, Theme = book, Recipient = Meena.

Uses: Machine translation, syntactic parsing, and text understanding.

4. What is Question Answering (QA) in NLP?

Question Answering (QA) is a specialized area of Natural Language Processing (NLP) that focuses on building computer systems capable of automatically answering questions posed by humans in natural language. Unlike simple keyword based search, a QA system aims to understand the meaning and intent behind a question, identify relevant information, and generate a precise and contextually correct answer.

A QA system combines multiple NLP techniques such as information retrieval, named entity recognition, semantic analysis, and machine learning to locate and extract answers from structured data (like databases) or unstructured text (like web pages, articles, or documents).

Working:

1. **Question Processing:** The system analyzes the question to identify its type (who, what, where, etc.) and key terms.
2. **Information Retrieval:** It searches documents, databases, or text sources to find relevant information.

3. **Answer Extraction:** The system extracts and ranks the most appropriate answer from the retrieved text.
4. **Answer Generation:** Finally, it presents the answer in a clear, natural-language format.

Applications:

- Search Engines (e.g., Google's featured snippets)
- Chatbots and Virtual Assistants (e.g., Siri, Alexa)
- Customer Support Systems
- Educational Tools and E-learning Platforms
- Healthcare and Legal Information Retrieval