Module 3

Q. What is POS Tagging? What are Challenges Faced by POS Tagging ?

=>

1. POS tagging stands for **Part-of-Speech tagging**.
2. It is the process of **assigning grammatical categories** such as noun, verb, adjective, adverb, etc., to each word in a sentence.
3. It helps computers understand **how each word functions** in a sentence.
4. POS tagging is a fundamental step in **Natural Language Processing (NLP)** tasks such as text analysis, translation, and information retrieval.
5. Each word in a sentence can play **different roles** depending on the context.
6. For example, the word **"book"** can be a **noun** ("I read a book") or a **verb** ("Please book a ticket").
7. POS tagging helps identify the **correct role** based on its position and surrounding words.
8. A POS tagger uses **linguistic rules** and **statistical models** to assign the right tag.
9. The common POS tags include **NN (Noun)**, **VB (Verb)**, **JJ (Adjective)**, **RB (Adverb)**, **PRP (Pronoun)**, **IN (Preposition)**, and others.
10. POS tagging is used in many applications like **machine translation**, **chatbots**, **speech recognition**, and **text summarization**.
11. There are three main approaches to POS tagging: **Rule-based**, **Statistical**, and **Hybrid** methods.
12. **Rule-based taggers** use predefined grammar rules to decide the tag.
13. **Statistical taggers** use probability models like **Hidden Markov Models (HMM)** or **Conditional Random Fields (CRF)**.
14. **Hybrid taggers** combine both rule-based and statistical approaches for better accuracy.
15. Example: In the sentence *"She runs fast,"*

    → "She" = Pronoun (PRP)

    → "runs" = Verb (VBZ)

    → "fast" = Adverb (RB).

16. Thus, POS tagging provides **grammatical structure** that helps in understanding sentence meaning.

Challenges Faced by POS Tagging

1. **Ambiguity of Words:**
    a. Many words in English have multiple meanings.

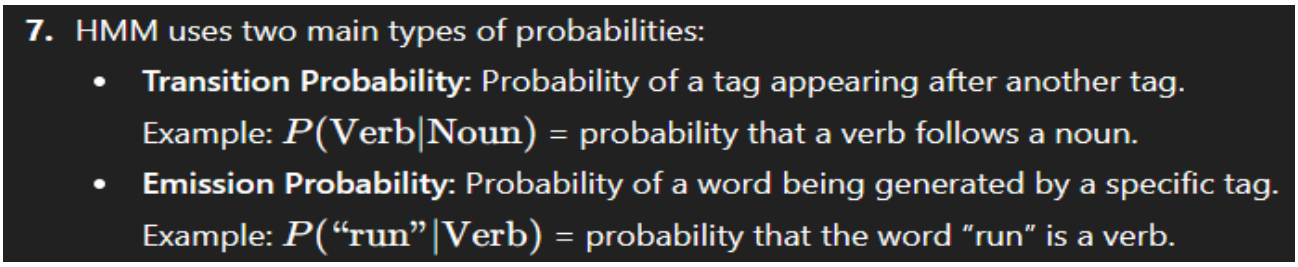
   b. Example: The word **"play"** can be a noun ("I watched a play") or a verb ("Children play outside").
   c. Choosing the right tag becomes difficult without context.
2. **Unknown or Out-of-Vocabulary Words:**
   a. POS taggers may encounter new or rare words not present in the training data.
   b. Example: A new word like **"selfie"** might not have a known tag, making tagging inaccurate.
3. **Context Sensitivity:**
   a. The same word can have different tags depending on the sentence context.
   b. Example: "Light" in "The room has light walls" (adjective) vs. "Turn on the light" (noun).
4. **Proper Noun Recognition:**
   a. It can be difficult to identify **names of people, places, or brands** correctly.
   b. Example: "Amazon" could mean a **company (proper noun)** or a **river (common noun)**.
5. **Idiomatic and Colloquial Expressions:**
   a. Informal phrases and idioms do not follow standard grammar rules.
   b. Example: "Kick the bucket" means "to die," not literally a "kick" action, which confuses taggers.
6. **Morphological Variations:**
   a. Words may appear in different forms (plural, tense, gender, etc.).
   b. Example: "Run," "runs," "running," and "ran" all represent the same verb but in different forms.
7. **Errors in Tokenization:**
   a. If a sentence is not split into words properly before tagging, errors occur.
   b. Example: "don't" may be split incorrectly into "do" and "n't," affecting tagging.
8. **Language Diversity and Dialects:**
   a. POS taggers trained on standard English may fail for **regional dialects** or **slang**.
   b. Example: "gonna" (going to) or "wanna" (want to) are hard to tag correctly.
9. **Compound and Hyphenated Words:**
   a. Phrases like "well-known," "ice-cream," or "mother-in-law" pose difficulty in assigning correct tags.
   b. The tagger must understand whether they act as adjectives or nouns.
10. **Domain-Specific Vocabulary:**
   a. Technical fields like medicine or law have unique terms.
   b. Example: "Virus" in biology vs. "virus" in cybersecurity.
11. **Free Word Order in Some Languages:**
   a. Languages like Hindi or Sanskrit allow flexible word order, making POS tagging harder than in English.
   b. Example: "राम आम खाता है" and "आम राम खाता है" mean the same but have different word orders.

12. **Incomplete or Grammatically Incorrect Sentences:**
    a. Social media texts often contain spelling mistakes or missing words.
    b. Example: “U goin?” or “gr8 work” confuse taggers trained on formal grammar.
13. **Lack of Annotated Data:**
    a. Training statistical or neural taggers needs large annotated corpora, which are not available for all languages.
14. **Transfer Between Languages:**
    a. POS taggers trained on one language may not perform well on another due to grammar differences.
15. **Computational Complexity:**
    a. Some algorithms like CRF and deep learning models require heavy computation and large memory.

Q. Explain Hidden Markov Model (HMM) for POS Tagging ? What are the Limitations of Hidden Markov Model

=>

1. **Hidden Markov Model (HMM)** is a **statistical model** used in **Natural Language Processing (NLP)** to predict the most likely sequence of hidden states (in this case, POS tags) for a given sequence of words.
2. The word "**Hidden**" means that the **actual tags are not directly visible**, but can be inferred from the words we see.
3. HMM assumes that there is a **sequence of states (POS tags)** and **observations (words)** that are related probabilistically.
4. It works on the **Markov assumption**, which says that **the current state depends only on the previous state**.
5. In POS tagging, the **states** are the **parts of speech** (like noun, verb, adjective), and the **observations** are the **words** in the sentence.
6. The goal of HMM is to find the **most probable sequence of POS tags** for the given sequence of words.
7. HMM uses two main types of probabilities:



8. HMM assumes both these probabilities can be estimated from a **tagged corpus (training data)**.
9. Using these probabilities, the model predicts which sequence of tags is **most likely** to produce the given words.
10. The **Viterbi Algorithm** is commonly used to find the **best possible sequence** of POS tags in HMM.
11. Example:
    a. Sentence: "The dog barks."
    b. Words: "The", "dog", "barks."
    c. Possible tags:
        i. "The" → Determiner (DT)
        ii. "dog" → Noun (NN)
        iii. "barks" → Verb (VBZ)
    d. HMM will calculate the **transition and emission probabilities** to find the most likely sequence: DT → NN → VBZ.

12. The system selects the tag sequence that has the **highest overall probability** among all possible sequences.
13. HMM is one of the **earliest and most widely used** methods for automatic POS tagging.
14. It performs well when trained on a **large and clean corpus** of labeled sentences.
15. HMM also forms the **foundation of more advanced models** such as Conditional Random Fields (CRF) and Neural Sequence Models.
16. It is based on the idea that **language follows patterns**, and those patterns can be represented mathematically using probabilities.
17. The model assumes that the **probability of a tag depends only on the previous tag**, which makes the computation simpler.
18. Hence, for a sequence of words ($w_1, w_2, \ldots, w_n$), we find tags ($t_1, t_2, \ldots, t_n$) that maximize:

$$P(t_1, t_2, \ldots, t_n \mid w_1, w_2, \ldots, w_n)$$

$$\approx P(t_1) \times P(t_2|t_1) \times \ldots \times P(t_n|t_{n-1}) \times P(w_1|t_1) \times \ldots \times P(w_n|t_n)$$

19. This simplification allows efficient computation of the most likely tag sequence.
20. Thus, HMM provides a **probabilistic approach** to predict tags that best explain the observed words.

Limitations of Hidden Markov Model

1. **Strong Independence Assumption:**
    a. HMM assumes that the current tag depends only on the **previous tag**, not on any other tags.
    b. This is not always true in natural language, where context may depend on multiple previous words.
2. **Limited Context Handling:**
    a. It cannot handle **long-range dependencies** between words in a sentence.
    b. Example: In "The boy who played football is tall," the tag of "is" depends on "boy," not the previous word "football."
3. **Emission Probability Limitation:**
    a. HMM assumes that each word depends **only on its tag**, not on neighboring words.
    b. This can lead to wrong tagging in ambiguous cases.
    c. Example: The word "bank" could mean a **river bank (noun)** or **financial bank (noun)**, which needs more context.
4. **Data Sparsity Problem:**
    a. It requires a large, well-annotated training corpus to estimate probabilities accurately.
    b. Rare words or unseen combinations cause errors.
5. **Unknown Word Handling:**

   a. HMM struggles with **new or unseen words** that are not present in the training data.
   b. Example: Newly coined words like "metaverse" or "cryptotoken" may be mistagged.
6. **Fails on Irregular Grammar:**
   a. It does not perform well on **social media text**, **informal writing**, or grammatically incorrect sentences.
   b. Example: "u r awesome" or "luv dis song" confuse the model.
7. **Need for Labeled Data:**
   a. Training an HMM requires a **large manually tagged dataset**, which is time-consuming and costly to prepare.
8. **Ambiguity in Similar Words:**
   a. It often confuses words with **similar frequencies and probabilities**.
   b. Example: "can" (modal verb) vs. "can" (noun) are difficult to separate statistically.
9. **No Use of Semantic Meaning:**
   a. HMM focuses on statistical probabilities and ignores **word meanings** or **semantic relationships**.
10. **Computational Complexity:**
   a. For long sentences with many possible tags, HMM can become **computationally expensive**, even though Viterbi helps optimize it.
11. **Fixed Transition Probabilities:**
   a. The model assumes that the transition probabilities between tags are **fixed**, which may not hold true in real-world language variations.
12. **Limited Adaptability:**
   a. HMM trained on one domain (e.g., news articles) may not perform well in another domain (e.g., medical or social media texts).
13. **Cannot Capture Non-linear Dependencies:**
   a. Complex sentence structures, idioms, or context shifts are not effectively modeled by linear HMM assumptions.
14. **Outperformed by Modern Models:**
   a. With the rise of **neural networks and transformers (like BERT)**, HMMs have become less accurate for large-scale tagging tasks.

Q. Explain Maximum Entropy Model for POS Tagging and Sequence Labelling

=>

1. The **Maximum Entropy Model (MEM)** is a **probabilistic model** used in Natural Language Processing (NLP) for **POS tagging** and **sequence labeling** tasks.
2. The term "Maximum Entropy" means **choosing the most uniform model possible** that fits the observed data — i.e., making **no extra assumptions** beyond what the data supports.
3. It follows the **principle of maximum entropy**, which states:

→ *When predicting something uncertain, choose the distribution that is most uniform while still consistent with known facts.*

4. MEM belongs to the family of **log-linear models**, where the output probabilities are calculated using **weighted feature functions**.
5. It predicts the **most likely tag (or label)** for a given word based on **contextual features** like neighboring words, suffixes, prefixes, and capitalization.
6. The model aims to assign a **POS tag** or **sequence label** that maximizes the overall conditional probability (P(tag | context)).
7. Example:

   Sentence: "The cat sleeps."

   Tags: "The/DT", "cat/NN", "sleeps/VBZ."

   The model learns which words or word patterns usually get which tags, based on training data.

8. MEM does not assume any independence among features, unlike HMM.
9. It can easily combine **different types of linguistic features** in the same model, giving it flexibility.
10. The model is particularly useful in NLP tasks like **POS tagging**, **Named Entity Recognition (NER)**, **Chunking**, and **Text Classification**.

Working of Maximum Entropy Model

1. MEM is based on the **conditional probability** of a tag given a word and its context.
   Mathematically:
   $P(t \mid w, c) = (1/Z) \times \exp(\Sigma \lambda_i \times f_i(t, w, c))$
   where:
   - *t* = tag (like noun, verb, etc.)
   - *w* = current word
   - *c* = context (neighboring words, previous tags)

- f_i = feature functions (binary indicators, 1 or 0)
- λ_i = weight of each feature
- Z = normalization factor (ensures probabilities sum to 1)

2. The **feature functions (f_i)** represent linguistic clues, e.g.,
   - "Does the word end with '-ing'?"
   - "Is the previous word a determiner?"
   - "Is the word capitalized?"
3. Each feature has a **weight (λ)** that shows its importance in predicting the tag.
4. During training, the model **learns these weights** to maximize the probability of correct tagging for training examples.
5. MEM estimates probabilities so that the **expected value of each feature** matches its value in the training data.
6. This ensures the model is **consistent with known data** but doesn't assume unnecessary dependencies.
7. Once trained, the model can tag unseen sentences by computing the **most probable tag sequence**.
8. Example:
   Sentence: "He is running fast."
   Features might include:
   - Word suffix "-ing" → likely a Verb (VB).
   - Previous word "is" → next word often Verb (VB).
   Model combines these clues to assign the tag "running/VBG."
9. The **Maximum Entropy Classifier** selects the tag with the highest conditional probability for each word.
10. It is often combined with the **Viterbi decoding algorithm** for sequence labeling tasks to ensure globally consistent tag sequences.

Example Summary

1. Example Sentence: *"She loves playing piano."*
2. Features:
   - Word suffix "-ing" → Verb indicator.
   - Previous tag "VBZ" → next tag likely "VBG."
   - Word capitalization → Possible start of sentence.
3. MEM combines all these clues to tag correctly:
   "She/PRP loves/VBZ playing/VBG piano/NN."
4. This shows that MEM can intelligently combine different clues to produce accurate tagging results.

Q. Compare top-down and bottom-up approach of parsing with example

=>

Feature	Top-Down Parsing	Bottom-Up Parsing
Definition	Top-down parsing starts from the **root (start symbol)** and tries to derive the input string using grammar rules.	Bottom-up parsing starts from the **input symbols (leaves)** and tries to build up the parse tree to reach the start symbol.
2. Approach Direction	Proceeds from **high-level (start symbol)** to **low-level (terminals)**.	Proceeds from **low-level (terminals)** to **high-level (start symbol)**.
Starting Point	Begins with the **start symbol (S)** of the grammar.	Begins with the **input string** (sequence of tokens).
Goal	The goal is to **predict what could generate** the input string.	The goal is to **reduce the input** to the start symbol using grammar rules.
Working Method	Uses **derivation rules** to expand non-terminals until the input string is formed.	Uses **reduction rules** to replace substrings with non-terminals until only the start symbol remains.
Type of Derivation	Follows a **Leftmost derivation** (expands leftmost non-terminal first).	Follows a **Rightmost derivation in reverse** (reduces the rightmost handle first).
Example Grammar	Grammar: S → NP VP NP → Det N VP → V NP Det → “the” N → “dog” V → “chased”	Same Grammar: S → NP VP NP → Det N VP → V NP Det → “the” N → “dog” V → “chased”
Example Sentence	Input: “the dog chased”	Input: “the dog chased”
Parsing Steps (Simplified)	1. Start with **S**. 2. Expand: S → NP VP. 3. Expand NP → Det N → “the dog”. 4. Expand VP → V → “chased”. 5. Successfully matches input “the dog chased”.	1. Start with **input**: “the dog chased”. 2. Reduce “the dog” → Det N → NP. 3. Reduce “chased” → V. 4. Combine NP + V → NP VP. 5. Reduce NP VP → S.
Nature of Processing	**Predictive:** It predicts what could come next based on grammar.	**Constructive:** It constructs higher-level phrases from the actual input.

Parser Examples	Recursive Descent Parser, LL(1) Parser.	Shift-Reduce Parser, LR Parser.
Backtracking	Often requires **backtracking** if the prediction fails.	Usually **no backtracking** as reductions are data-driven.
Error Detection	Detects errors **later**, as it predicts first and verifies later.	Detects errors **early**, while reducing mismatched patterns.
Implementation Complexity	**Simpler** to implement but less powerful.	**More complex** but can handle a wider range of grammars.
Suitable Grammar Type	Works for **LL grammars** (Left-to-right, Leftmost derivation).	Works for **LR grammars** (Left-to-right, Rightmost derivation in reverse).
Handling of Left Recursion	**Cannot handle left recursion** directly (causes infinite loops).	**Can handle left recursion** effectively.
Speed and Efficiency	Slower due to prediction and backtracking.	Faster and more deterministic for large grammars.
Example of Parser Movement	Moves **from root to leaves** in the parse tree.	Moves **from leaves to root** in the parse tree.
Example Parse Tree Direction	Builds the parse tree **top-down** from S → NP → Det N.	Builds the parse tree **bottom-up**, starting from Det N → NP → S.
Real-World Usage	Used in simple compilers, interpreters, and educational tools.	Used in most **modern compilers and parsing tools** due to higher accuracy.