1. Discuss the significance of language models in NLP.

Language models play a foundational role in Natural Language Processing (NLP). They are designed to understand, represent, and predict human language in a computationally efficient way. A language model assigns a probability to a sequence of words and helps in **determining the likelihood of sentences**, thereby improving the fluency and correctness of NLP applications.

The primary function of a language model is to capture the **statistical structure of language**. By learning which word sequences are more likely, it can anticipate what comes next in a sentence, detect grammatical errors, and disambiguate word meanings based on context. For instance, in predictive text, when the user types "I want to eat…", the model might suggest "pizza" or "dinner" based on the highest probability learned from training data.

Language models are crucial in applications like:

- **Machine Translation** (predicting word order in the target language)
- **Speech Recognition** (converting spoken words to the most likely text)
- **Text Generation** (chatbots, story generators)
- **Spelling Correction** (finding the most likely word substitution)

Earlier, rule-based and n-gram models were popular, but modern NLP heavily relies on **neural network-based models** like GPT, BERT, and Transformer-based architectures that provide much better context understanding and language generation capabilities.

Thus, language models serve as the **backbone of all intelligent text-processing systems**.

---

2. Difference between Grammar-based LM and Statistical LM

Language models can be broadly categorized into **grammar-based** and **statistical-based** models, each with distinct methodologies and applications.

A **grammar-based language model** uses **formal grammatical rules** derived from linguistics to define valid sentence structures. These rules are expressed using frameworks like **Context-Free Grammars (CFGs)** or **Probabilistic CFGs (PCFGs)**. The system generates or validates sentences based on rule sets without relying on real-world data. Such models are more rigid and deterministic but offer **complete syntactic accuracy**. For example, a grammar rule like `S → NP VP` helps ensure syntactically valid sentence structures such as "The boy eats an apple."

On the other hand, a **statistical language model** relies on **large corpora of text** to learn word sequences and their probabilities. N-gram models, for example, calculate the probability of a word given its preceding words. These models are **data-driven**, flexible, and can handle natural variations and errors in real-world language use. A bigram model would estimate P("go") after "let's" using frequency counts from training data.

Feature	Grammar-based LM	Statistical LM
Method	Rule-based syntax	Probability-based on corpus
Flexibility	Rigid, predefined	Adaptive, handles variations
Data Requirement	None or minimal	Requires large corpora
Example Output	Valid but rare sentences	Fluent, real-world sentence flow
Use Cases	Parsing, rule checking	Translation, prediction, speech

While grammar-based models are useful in parsing and teaching, **statistical models dominate modern NLP** due to their scalability and ability to model real-world usage.

---

3. What is regex with example?

Regular Expressions (Regex) are symbolic notations used to define **patterns for matching strings**. In NLP, regex is extensively used for **text search, validation, tokenization, data extraction, and preprocessing**.

Regex acts like a filter that identifies whether a string (or part of it) matches a defined pattern. The pattern is written using a specific syntax that includes metacharacters like:

- `.` – Matches any character
- `*` – Matches 0 or more repetitions
- `+` – Matches 1 or more
- `[a-z]` – Matches a lowercase alphabet
- `\d` – Matches a digit (0–9)
- `\s` – Matches any whitespace

Example:
To extract an email address from text:

```
regex
CopyEdit
[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}
```

This will match strings like: `john.doe23@example.com`.

Regex is especially helpful in **preprocessing pipelines**, such as:

- Removing punctuation
- Detecting dates (`\d{2}/\d{2}/\d{4}`)
- Extracting phone numbers (`\(\d{3}\)\s\d{3}-\d{4}`)

While regex is powerful and flexible, it can be **hard to debug or maintain**, and doesn't handle nested or recursive structures (like grammars) well. Still, it is a crucial tool in the **first stages of text processing**.

---

4. What is Finite State Automata? DFA vs NFA

A **Finite State Automaton (FSA)** is a mathematical model used to represent **regular languages**. It consists of a finite number of states, transitions between those states, an initial state, and one or more accepting (final) states. FSAs are used in **text pattern matching, morphological analysis, and lexicon processing**.

There are two main types:

Deterministic Finite Automaton (DFA):

- Each state has exactly **one transition per input symbol**.
- No ε-transitions (empty transitions).
- Easier and faster to simulate in practice.

Non-Deterministic Finite Automaton (NFA):

- A state can have **multiple transitions** for the same symbol.
- Can include **ε-transitions**, where the machine moves without consuming input.
- Easier to design and more compact but harder to simulate directly.

Example:
To match strings ending in “ab”:

- **DFA** will need multiple states tracking "a" and "b"
- **NFA** can branch and guess which transition will lead to a valid end state

Both DFA and NFA can recognize the same class of languages—**regular languages**—but NFAs are more flexible in design, while DFAs are more efficient in execution.

FSAs are fundamental in **regex engines, compilers, and NLP tools** like morphological analyzers.

---

5. What do you mean by tokens?

In NLP, **tokens** are the **smallest units of text** derived during the preprocessing step called **tokenization**. A token can be a **word, punctuation mark, number, or subword unit**, depending on the tokenizer.

Tokenization is essential because it breaks down raw text into manageable and meaningful units for further processing like **POS tagging, parsing, named entity recognition, and text classification**.

Types of Tokens:

- **Word-level**: "The quick brown fox"
- **Subword-level**: "unbelievable" → "un", "believ", "able"
- **Character-level**: "dog" → "d", "o", "g"

Example:

Sentence: `"NLP is fun!"`
Tokens: `["NLP", "is", "fun", "!"]`

Challenges in tokenization include:

- Handling contractions (e.g., “don’t” → “do”, “not”)
- Handling punctuations and quotes
- Language-specific rules (e.g., no whitespace in Chinese)

Advanced tokenizers (e.g., WordPiece or BPE used in BERT/GPT) combine rule-based and statistical methods for better accuracy. Proper tokenization is **foundational for every NLP task**.

---

6. What is Minimum Edit Distance?

Minimum Edit Distance (MED) is a metric used to determine the **similarity between two strings**. It calculates the minimum number of operations required to transform one string into another. The allowable operations typically include:

- **Insertion**
- **Deletion**
- **Substitution**

Types of Edit Distance:

- **Levenshtein Distance**: Allows all three operations.
- **Damerau-Levenshtein Distance**: Also includes **transposition** of characters.

Example:

Convert “intention” to “execution”:

Step	Operation	Result
1	Replace ‘i’ → ‘e’	*e*ntention
2	Replace ‘n’ → ‘x’	ex*t*ention
3	Replace ‘t’ → ‘c’	execention
4	Replace ‘n’ → ‘u’	execuition
5	Replace ‘i’ → ‘t’	executton
6	Replace ‘o’ → ‘i’	execution

MED = 5

Applications:

- **Spell correction**: Suggest “receive” for “recieve”
- **Speech recognition**: Match recognized phrases with intended words
- **Plagiarism detection**: Compare document similarity
- **DNA sequence analysis**: In bioinformatics

Efficient algorithms (e.g., dynamic programming) are used to compute MED in O(n*m) time, where n and m are string lengths.

---

7. What are the origins and challenges of Natural Language Processing (NLP)?

Origins of NLP:

NLP originated as a subfield of **linguistics and artificial intelligence**. In its early stages during the 1950s–1970s, it focused on **rule-based approaches**, where language was modeled using grammars, dictionaries, and formal logic. One of the earliest efforts was **machine translation**, like the Georgetown-IBM experiment (1954).

By the 1980s, the rise of **statistical models** brought data-driven methods into NLP. These methods used large corpora and probabilistic techniques (e.g., n-grams) for language modeling. In the 2000s and beyond, **machine learning and deep learning**, especially models like **BERT**, **GPT**, and **Transformers**, revolutionized NLP by enabling systems to understand language contextually.

Challenges in NLP:

1. **Ambiguity**
   - **Lexical ambiguity**: “bank” can mean a financial institution or a riverbank.
   - **Syntactic ambiguity**: “I saw the man with the telescope” – who has the telescope?
2. **Context and World Knowledge**
   - NLP systems struggle to interpret sarcasm, cultural context, or implied meaning.
3. **Multilinguality**
   - Developing NLP tools for all languages requires huge datasets, which may not be available.
4. **Data Noisiness**
   - Social media text is filled with typos, abbreviations, and slang, making it hard to process.
5. **Morphology and Inflection**
   - Handling plural forms, tenses, gender forms, and derivations across different languages.
6. **Code-Switching**
   - Mixed language use (e.g., Hinglish) poses a problem for standard NLP pipelines.

Despite these challenges, modern NLP continues to evolve through **hybrid approaches** that combine linguistics, machine learning, and real-world data.

---

8. What is English Morphology in NLP?

Morphology is the branch of linguistics that studies the **structure and formation of words**. In NLP, English morphology deals with analyzing and processing **word forms, roots, affixes (prefixes/suffixes), and inflections**.

Understanding morphology allows systems to:

- Normalize words (e.g., "cats" → "cat")
- Handle variations in tense or number
- Improve tasks like **stemming, lemmatization, POS tagging**, and **information retrieval**

Two Main Types of Morphology:

1. **Inflectional Morphology**
   - Changes form to express grammatical properties (tense, number, etc.)
   - Does **not change** the word category
   - Examples:
     - walk → walks → walked → walking
     - cat → cats
2. **Derivational Morphology**
   - Creates **new words** by adding prefixes/suffixes
   - Often **changes the part of speech**
   - Examples:
     - happy → happiness (adj → noun)
     - develop → development

Application in NLP:

In information retrieval, both “run” and “running” should match a search query. Morphological analysis ensures that such **morphological variants** are treated as the same base word.

Advanced tools use **finite state transducers** and **morphological analyzers** to parse words into morphemes and restore their root forms.

---

9. What are transducers in NLP?

In NLP, a **transducer** is a device or algorithm that performs a **mapping from input strings to output strings**. Specifically, a **finite-state transducer (FST)** is a machine that reads input characters and produces corresponding output characters, often used for modeling **morphological processes** and **lexical transformations**.

Definition:

A transducer is an extension of a Finite-State Automaton (FSA), where each transition has:

- An **input symbol**
- An **output symbol**
- A transition from one state to another

It is used when a system must convert or interpret a form of text into another.

Example Use Cases:

1. **Morphological Generation**
   Input: `walk + PAST`
   Output: `walked`
2. **Morphological Analysis**
   Input: `walked`
   Output: `walk + PAST`
3. **Spell Correction**
   FSTs can map commonly misspelled words to correct forms.
4. **Speech Recognition**
   Mapping phonemes (spoken input) to words.

Benefits of Transducers:

- Can model **both rule-based** and **data-driven** transformations.
- Compact and efficient.
- Widely used in morphological analyzers, especially in **low-resource languages**.

Transducers help **bridge the gap between linguistic theory and real-world applications**, making them vital in NLP pipelines.

---

10. How are spelling errors detected and corrected in NLP?

Spelling error correction is a key preprocessing step in NLP used in **search engines, autocorrect, grammar checkers**, and **OCR systems**. It involves two main tasks:

---

1. Error Detection

The system identifies whether a given word exists in a **vocabulary or dictionary**. If the word is unknown, it is flagged as potentially misspelled.

---

2. Error Correction

This step tries to replace the incorrect word with the most likely intended word.

Correction Techniques:

1. **Minimum Edit Distance**
   Calculates how many operations (insert, delete, substitute) are needed to convert one word to another.
   Example: "teh" → "the" (one substitution)
2. **Phonetic Similarity**
   Suggests corrections based on pronunciation (e.g., Soundex algorithm)
3. **Confusion Matrices**
   Learns common misspelling patterns (e.g., "ie" instead of "ei")
4. **Contextual Spell Correction**
   Uses **language models** to suggest corrections based on sentence context.
   Example: "I have two much homework" → "too" instead of "two"

Example:

Wrong: "recieve"
Detected: Not in dictionary
Corrected: "receive" (using edit distance and frequency)

Spelling correction is especially important in **search engines** where users often type queries with errors. Modern NLP systems use deep learning models like BERT to handle context-sensitive corrections effectively.

1. What is an N-gram?

An **N-gram** is a **contiguous sequence of 'n' words** from a given text or speech. N-grams are a type of **statistical language model** used to predict the likelihood of a word given the previous words, helping machines understand patterns and context in natural language.

Types of N-grams:

- **Unigram (n=1)**: Single word
- **Bigram (n=2)**: Sequence of two words
- **Trigram (n=3)**: Sequence of three words
- **n-gram (general)**: Sequence of n words

Formula:

The probability of a sentence using bigrams:
$P(w1, w2, ..., wn) \approx \prod P(wi \mid wi\text{-}1)$

Example:

For sentence: “I love NLP”

- Unigrams: “I”, “love”, “NLP”
- Bigrams: “I love”, “love NLP”

Applications:

- Text prediction and autocomplete
- Speech recognition
- Spell correction
- Machine translation

Limitation:

- Higher n increases accuracy but also **data sparsity**
- Assumes limited context (Markov assumption)

N-gram models are foundational in NLP and are widely used in basic language processing tasks before neural models took over.

---

2. How can smoothing be performed at word-level analysis?

In **word-level language modeling**, **smoothing** is used to handle the problem of **zero probabilities** for unseen word combinations (n-grams) in training data. Without smoothing, any unseen n-gram would be assigned zero probability, which can break many NLP applications.

Why Smoothing?

Even large corpora miss valid word sequences. Smoothing adjusts probability distributions to assign **non-zero probabilities** to unseen events while slightly decreasing the probabilities of seen ones.

Common Smoothing Techniques:

1. **Add-One (Laplace) Smoothing**
   Add 1 to all word counts before calculating probabilities.
   P(wi | wi-1) = (Count(wi-1, wi) + 1) / (Count(wi-1) + V), where V = vocabulary size.
2. **Add-k Smoothing**
   A generalization of Laplace where $k < 1$ (e.g., 0.5) to avoid over-penalizing frequent terms.
3. **Good-Turing Discounting**
   Adjusts frequencies based on the number of n-grams that occur once, twice, etc. Useful when there are many rare events.
4. **Backoff Models**
   Use a lower-order n-gram (e.g., bigram or unigram) if the higher-order n-gram (e.g., trigram) is missing.
5. **Interpolation**
   Combines multiple n-gram models (e.g., unigram + bigram + trigram) using weights:
   P(wi|...) = λ1 × trigram + λ2 × bigram + λ3 × unigram

Application:

Used in **chatbots**, **autocomplete**, and **speech recognition** to ensure robustness when encountering new word sequences.

---

3. What is meant by word classes?

Word classes, also known as **parts of speech (POS)**, are categories that group words based on their syntactic roles and grammatical behavior in sentences.

Common Word Classes:

- **Noun**: Represents people, places, or things (e.g., "book")
- **Verb**: Describes action or state (e.g., "run")
- **Adjective**: Modifies a noun (e.g., "beautiful")
- **Adverb**: Modifies a verb or adjective (e.g., "quickly")
- **Preposition**: Shows relation (e.g., "on", "in")
- **Pronoun**: Replaces a noun (e.g., "he", "she")
- **Conjunction**: Connects clauses (e.g., "and", "but")

Importance in NLP:

- Helps in **POS tagging**, **syntax parsing**, and **semantic analysis**
- Provides grammatical structure and meaning to sentences
- Helps resolve **ambiguity** (e.g., "can" as noun or verb)

Example:

Sentence: “Time flies like an arrow.”

- “Time” (noun), “flies” (verb), “like” (preposition), “arrow” (noun)

Understanding word classes is foundational for tasks like **machine translation, text-to-speech, and sentiment analysis**.

---

4. What is POS tagging and what is Bag of Words?

Part-of-Speech (POS) Tagging:

POS tagging is the process of assigning **parts of speech** (noun, verb, adjective, etc.) to each word in a sentence based on its **context** and **usage**.

Example:

Sentence: “He plays cricket.”

- He/PRP (pronoun), plays/VBZ (verb), cricket/NN (noun)

Techniques:

- **Rule-Based**: Uses hand-crafted rules (e.g., Brill Tagger)
- **Stochastic**: Uses probabilistic models like **HMMs**
- **Transformation-Based**: Hybrid of rule and statistical methods

POS tagging is crucial for **syntactic parsing**, **disambiguation**, and **information extraction**.

Bag of Words (BoW):

BoW is a **simple feature extraction model** used in NLP where a document is represented as an unordered collection of words, **ignoring grammar and word order**, but preserving **frequency**.

Steps:

1. Create vocabulary of all words
2. Represent each document as a vector of word counts

Example:

Doc 1: “I love NLP”
Doc 2: “NLP loves me”

Vocabulary = [“I”, “love”, “NLP”, “loves”, “me”]
Vectors:

- Doc1: [1, 1, 1, 0, 0]
- Doc2: [0, 0, 1, 1, 1]

BoW is widely used in **text classification, sentiment analysis**, and **spam detection**, though it lacks contextual information.

---

5. What is a word cloud?

A **word cloud** (or tag cloud) is a **visual representation** of text data in which **frequently occurring words** appear **larger and bolder** than less frequent ones. It provides a quick overview of the **most prominent keywords or themes** in a document or corpus.

Features:

- No grammar or structure involved
- Often generated from BoW or TF-IDF data
- Common in exploratory data analysis

Example:

For reviews of a product, a word cloud might emphasize terms like “fast”, “battery”, “price”, showing what users frequently mention.

Applications:

- Text summarization
- Sentiment and theme analysis
- Visualization in social media analytics or surveys

While visually appealing, word clouds lack context or sentiment analysis. Hence, they are used mainly for **preliminary exploration**, not deep NLP tasks.

---

6. What are problems of POS tagging and how can they be resolved?

POS tagging is essential in NLP, but it faces several **linguistic and computational challenges**:

Challenges:

1. **Ambiguity**
   Words can belong to multiple parts of speech depending on context.
   Example: “Book a flight” vs “Read a book”
2. **Unknown Words (Out-of-Vocabulary)**
   New or rare words not seen during training pose difficulties.
3. **Multiword Expressions**
   Idioms or named entities (e.g., "New York") may need special tagging.
4. **Free Word Order in Some Languages**
   In languages like Hindi, grammatical roles aren’t tied to strict positions.

5. **Code Switching**
   Mixed languages (e.g., Hinglish) make consistent tagging difficult.

Solutions:

1. **Rule-Based Tagging**
   Hand-crafted rules based on syntax help disambiguate (e.g., noun after "a").
2. **Stochastic Tagging**
   Use probabilistic models like **Hidden Markov Models (HMMs)** which assign tags based on context and likelihood.
3. **Transformation-Based Learning**
   Combines rule-based and statistical methods (e.g., Brill tagger).
4. **Deep Learning Models**
   Models like **BiLSTM + CRF** or **Transformer-based models** can model complex dependencies and handle ambiguous words better.

Resolving POS tagging issues improves **syntactic parsing, named entity recognition, and machine translation**.

---

7. Minimum Entropy, Maximum Entropy and Trade-Off

Entropy in NLP refers to the **uncertainty or disorder in predicting outcomes**. In probabilistic modeling, it helps evaluate how predictable a word or class is in context.

Minimum Entropy:

- Represents a system with **maximum certainty**
- Predictable sequences with little variation
- Often found in **rule-based systems**
- May cause **overfitting** and lack generalization

Maximum Entropy (MaxEnt):

- A statistical model that **maximizes entropy** under given constraints
- It assumes **no bias** beyond the known information
- Ensures the model is **most uniform** given the evidence
- Used for **POS tagging, text classification, and parsing**

Example:

In POS tagging, MaxEnt assigns probabilities to all possible tags, choosing the tag with the **highest conditional probability** given the features.

Trade-Off:

- **Too little entropy** → rigid, rule-bound model (low flexibility)
- **Too much entropy** → overly uncertain model (poor accuracy)
- The goal is to **balance** entropy to achieve both **accuracy and generalization**

Modern models like **MaxEnt classifiers** and **neural networks** implicitly manage this trade-off during training.

8. How are N-grams evaluated in NLP?

Evaluating N-gram models is essential to measure how well a language model predicts word sequences. The most common metrics used for evaluation are **perplexity** and **cross-entropy**, both derived from information theory.

1. Cross-Entropy:

Cross-entropy measures the **average number of bits** required to encode the data based on the model's probability distribution. Lower cross-entropy implies better prediction.

Formula:
$H(P) = -(1/N) \sum \log_2 P(wi \mid w1, ..., wi-1)$

2. Perplexity:

Perplexity is the **inverse probability of the test set**, normalized by the number of words. It measures how "confused" a model is when predicting the next word.

Formula:
Perplexity = $2^{H(P)}$

A **lower perplexity** score indicates better performance.

Example:

For a test sentence like "I love NLP", if the model assigns high probability to each word in sequence, the perplexity is low, indicating good performance.

Other Evaluation Methods:

- **Intrinsic Evaluation**: Based on probability scores
- **Extrinsic Evaluation**: Measures performance in downstream tasks like translation or speech recognition

Evaluating N-gram models helps determine whether to use **unigram, bigram, or trigram models**, and if **smoothing, backoff, or interpolation** should be applied to improve results.

---

9. What is Interpolation and Backoff in N-gram models?

In real-world NLP, N-gram models often face the **data sparsity problem**: many valid word sequences may not appear in the training data, leading to zero probability predictions. **Interpolation** and **backoff** are smoothing techniques to handle such cases.

---

1. Interpolation:

Interpolation **combines probabilities** from higher-order and lower-order N-grams (e.g., trigram, bigram, unigram) using **weighted averages**.

Formula:
$P(wi \mid wi-2, wi-1) = \lambda 1\ P(wi \mid wi-2, wi-1) + \lambda 2\ P(wi \mid wi-1) + \lambda 3\ P(wi)$

Where:

- $\lambda 1 + \lambda 2 + \lambda 3 = 1$
- Weights are tuned to optimize prediction

This allows the model to **smooth over rare events** by leveraging more reliable, lower-order statistics.

---

2. Backoff:

Backoff uses the **highest possible order N-gram** available. If a trigram is not found, it "backs off" to a bigram; if that's missing too, it uses a unigram.

Example:
If P("is going to") is missing, backoff to P("going to") or even P("to").

Backoff models like **Katz Backoff** include discounting to redistribute probabilities of seen events to unseen ones.

Both techniques **prevent zero probability issues** and allow N-gram models to handle unseen data more gracefully.

---

10. What is Rule-Based POS Tagging?

Rule-based Part-of-Speech (POS) Tagging assigns grammatical categories to words using a set of **hand-crafted linguistic rules** and dictionaries. It was among the earliest methods used in NLP for tagging words with their parts of speech.

How it works:

1. A **lexicon** provides the most likely tags for known words.

2. A **set of disambiguation rules** is applied to resolve context-based ambiguity.

Example Rules:

- If a word follows a determiner (e.g., "the") and is a noun or verb, tag it as a **noun**.
- If a word ends in "-ing" and follows a verb, tag it as **verb (VBG)**.

Example:

Sentence: "The dogs bark"

- "The" → DT
- "dogs" → NN (based on dictionary)
- "bark" → VB (using rule: verb follows noun)

Advantages:

- Transparent and interpretable
- Doesn't need large training data

Limitations:

- Fails with unknown or ambiguous words
- Not adaptable to new domains
- Rules can become overly complex

Despite being outdated, rule-based taggers are still useful in **low-resource languages or hybrid systems** where accuracy and control are important.

---

11. What is Stochastic POS Tagging?

Stochastic POS tagging uses **probability and statistics** to assign the most likely part-of-speech tag to each word in a sentence. Unlike rule-based systems, it learns from **annotated corpora**.

The most common stochastic approach uses a **Hidden Markov Model (HMM)**.

HMM-Based POS Tagging:

An HMM is a probabilistic model with:

- **States**: POS tags (hidden)
- **Observations**: Words (visible)
- **Transition Probabilities**: Likelihood of moving from one tag to another
- **Emission Probabilities**: Likelihood of a word being generated by a tag

Example:

Sentence: "He can fish."

- "can" could be a **verb** or **noun**
- Using training data, HMM determines that "can" is most likely a **verb** if it follows a pronoun and precedes a noun

Advantages:

- Data-driven, scalable
- Effective at handling ambiguity

Limitations:

- Requires large tagged corpora
- Cannot model long-distance dependencies

Still, stochastic taggers are the basis for **modern sequence labeling tasks** and have been largely replaced by more advanced neural models today.

12. What is Transformation-Based Tagging?

Transformation-Based Tagging (TBL) is a hybrid approach combining **rule-based and statistical methods**. The most well-known TBL system is the **Brill Tagger**.

Working Principle:

1. **Initial Assignment**: Each word is given its most probable tag based on training data.
2. **Transformation Rules**: A set of context-sensitive rules is learned to correct initial errors.

Example Rules:

- If a word is tagged as NN but comes after "to", change to VB
- If a word is tagged as VB and ends in "-ing", change to VBG

Rules are learned automatically from a tagged corpus by identifying the highest accuracy-improving transformations.

Advantages:

- Combines **interpretability** of rule-based systems with **accuracy** of statistical models

- Performs well even with **limited data**

Limitations:

- Rule learning can be computationally expensive
- Not as accurate as modern deep learning models

TBL is valuable for **low-resource languages**, educational tools, and systems where rule transparency is needed.

---

13. What is a Hidden Markov Model (HMM) and how is it used in POS tagging?

A **Hidden Markov Model (HMM)** is a **probabilistic model** that assumes an underlying hidden sequence (states) generates observable events (outputs). In POS tagging, the **states** are part-of-speech tags, and the **observed events** are words.

Components of HMM:

- **States**: POS tags (NN, VB, DT, etc.)
- **Observations**: Words like "dog", "runs", etc.
- **Transition Probability**: $P(tag_i \mid tag_{i-1})$
- **Emission Probability**: $P(word_i \mid tag_i)$

Working Mechanism:

Given a sentence, HMM uses the **Viterbi algorithm** to find the **most likely sequence of POS tags** that maximizes the joint probability of the observed words and tag transitions.

Example:

Sentence: “Time flies like an arrow”

- “flies” could be verb or noun
- Based on transition probabilities (e.g., DT → NN more likely), the model decides the best tag sequence.

Advantages:

- Good at handling **sequential dependencies**
- Learns from real data

- Robust to noise

Limitations:

- First-order Markov assumption limits context awareness
- Doesn’t capture long-range dependencies

HMMs have paved the way for more advanced models like **CRFs** and **LSTMs**, but remain a **foundational concept in sequence modeling**.