✅ Topic 1: Unsmoothed N-Grams

🔹 What is an N-gram?

An **N-gram** is a **contiguous sequence of 'n' items (usually words or characters)** from a given text.

It is used to model **how likely a word is to follow previous words** — i.e., **Language Modeling** based on local context.

🔹 Types of N-grams

N	Name	Example (on “I love NLP”)
1	Unigram	[“I”, “love”, “NLP”]
2	Bigram	[“I love”, “love NLP”]
3	Trigram	[“I love NLP”]
N	N-gram	Group of N consecutive words

🔹 N-Gram Probability

To compute the probability of a sentence like:

$$P(w_1, w_2, w_3, \ldots, w_n)$$

Using **chain rule**:

$$P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_1, w_2) \cdots P(w_n|w_1, \ldots, w_{n-1})$$

But this is **computationally expensive**. So we **approximate** using N-gram models:

🔸 Example: Bigram Model

$$P(w_1, w_2, w_3) \approx P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_2)$$

This assumes that each word **depends only on the previous word**.

🔹 How to Calculate Bigram Probability?

$$P(w_n|w_{n-1}) = \frac{\text{Count}(w_{n-1}\ w_n)}{\text{Count}(w_{n-1})}$$

🔸 Example:

Given a corpus:

```
text
CopyEdit
I love NLP. I love AI. NLP is cool.
```

Bigram counts:

- Count("I love") = 2
- Count("love") = 2

Then,

$$P(\text{love} \mid \text{I}) = \frac{\text{Count}(I\ love)}{\text{Count}(I)} = \frac{2}{2} = 1.0$$

🔹 Limitations of Unsmoothed N-Grams

Limitation	Why it's a Problem
Zero probability	If N-gram never appeared in training, prob = 0
Data sparsity	Exponential number of N-grams needed
Context ignorance	Doesn't consider deeper sentence structure

👉 These are later solved by **Smoothing**, **Backoff**, and **Neural Language Models**.

📝 Exam-Oriented Questions & Model Answers

🔸 Q1. (Understand, 4 marks)

What is an N-gram? Give an example of a bigram and trigram.

✅ **Model Answer:**

An **N-gram** is a sequence of **N words** from a text. It helps estimate the probability of a word based on its previous words in NLP.

- **Bigram**: N = 2 → "I love", "love NLP"
- **Trigram**: N = 3 → "I love NLP"

N-gram models are used in language modeling, text prediction, and spell correction.

🔸 Q2. (Apply, 5 marks)

Given the following corpus: "AI is the future. AI will change the world."
Compute the bigram probability: P(will | AI)

✅ **Model Answer:**

Bigram: "AI will"

- Count("AI will") = 1
- Count("AI") = 2

$$P(\text{will} \mid \text{AI}) = \frac{1}{2} = 0.5$$

So, the probability of "will" following "AI" is **0.5**.

🔸 Q3. (Evaluate, 6 marks)

Explain any two limitations of using unsmoothed N-gram models in NLP.

✅ **Model Answer:**

1. **Zero Probabilities**:
   If an N-gram does not appear in training data, its probability becomes 0, which breaks the sentence probability chain.
2. **Data Sparsity**:
   For large N, the number of possible N-grams grows exponentially, requiring huge amounts of data to cover all cases.

🔸 Question:

Explain any two limitations of using unsmoothed N-gram models in NLP.

✅ **Detailed Model Answer (6 marks)**:

Unsmoothed N-gram models are simple statistical models that estimate the probability of a word based on the previous *n-1* words. While they are easy to build and understand, they suffer from major limitations:

---

1. Zero Probability Problem (Out-of-Vocabulary and Sparsity)

Unsmoothed N-gram models assign **zero probability** to any N-gram that was **not seen during training**, even if it is a valid and likely phrase in real-world usage.

❌ **Problem:**

If a test sentence contains even a **single unseen N-gram**, the entire sentence gets a **zero probability**, regardless of how similar it is to known phrases.

📌 **Example:**

If “I love pizza” exists in the training set, but “I love pasta” does not, then:

P("pasta" | "love")=0P(\text{"pasta" | "love"}) = 0P("pasta" | "love")=0

→ This causes issues in **text generation, spell correction, and machine translation**.

---

2. Data Sparsity and Combinatorial Explosion

N-gram models rely heavily on the availability of **large amounts of data** to reliably estimate probabilities. As `n` increases, the number of possible combinations increases exponentially, leading to:

- **Sparser counts**
- Many zero-frequency events
- Poor generalization to new or rare contexts

📌 **Example:**

For a vocabulary of 10,000 words:

- Number of **bigrams** = $10^4 \times 10^4 = 10^8$
- Number of **trigrams** = $10^4 \times 10^4 \times 10^4 = 10^{12}$
  Most of these combinations will **never occur**, even in large corpora.

This makes **training, storing, and computing** N-gram probabilities inefficient and **inaccurate** without smoothing.

---

3. *(Optional if more marks)* Lack of Long-Term Context Understanding

Unsmoothed N-gram models **only look at the last (n–1) words**, which limits their ability to understand long-range dependencies in language.

📌 **Example:**

In the sentence:

“If the weather is nice tomorrow, we will go to the beach.”

A bigram/trigram model can't connect “If” with “we will go...” — it lacks **semantic depth**.

This is a key reason why **neural language models** (like RNNs, BERT, GPT) outperform traditional N-grams.

✅ Topic 2: Evaluating N-Gram Models

🔹 Why Evaluate N-Gram Models?

To know **how well a language model predicts text**, we must evaluate its performance.
This is crucial when comparing:

- Different N-gram sizes (bigram vs trigram)
- Smoothed vs unsmoothed models
- N-gram models vs neural models

🔹 Common Evaluation Metric: Perplexity

Perplexity measures how **confused** a language model is when predicting the next word.
Lower perplexity = better model.

🔹 What is Perplexity?

$$\text{Perplexity}(W) = P(w_1, w_2, ..., w_N)^{-\frac{1}{N}}$$

Or equivalently:

$$\text{Perplexity}(W) = \exp\left(-\frac{1}{N}\sum_{i=1}^{N}\log P(w_i|w_{i-n+1}^{i-1})\right)$$

Where:

- $W = w_1, w_2, ..., w_N$: sentence or test corpus
- $P(w_i|\cdot)$: probability of each word from the model
- N: number of tokens

🔹 Intuition Behind Perplexity

- Measures **average branching factor** of the model:
  i.e., how many choices the model thinks it has at each word.

Perplexity Value	Interpretation
Low (≈ 10–100)	Model is **confident** and accurate
High (1000+)	Model is **uncertain**, performs poorly

🔸 Example Calculation

Suppose a trigram model gives:

- P("I | <s>") = 0.5
- P("love | <s>, I") = 0.25
- P("NLP | I, love") = 0.125

$$\text{Perplexity} = (0.5 \times 0.25 \times 0.125)^{-\frac{1}{3}} = (0.015625)^{-\frac{1}{3}} \approx 4$$

✅ Lower perplexity = better prediction ability.

🔹 Other Metrics (Rare in N-gram evaluation)

Metric	Description
Cross-entropy	Measures average log probability per word
BLEU	Mostly for machine translation

But for **language modeling**, **Perplexity** is the gold standard.

🔹 Limitations of Perplexity

1. **Sensitive to vocabulary size**
2. Doesn't directly reflect task-specific performance (e.g., translation, classification)
3. Can be misleading if overfit on training data

📝 Exam-Oriented Questions & Model Answers

🔸 Q1. (Understand, 4 marks)

What is perplexity in language modeling? How is it interpreted?

✅ **Model Answer:**

Perplexity is a metric used to evaluate how well a language model predicts a sample. It is the **inverse probability of the test set**, normalized by the number of words.
It tells how **"surprised"** the model is by the data.

$$\text{Perplexity}(W) = P(w_1, ..., w_N)^{-1/N}$$

A **lower perplexity** indicates better performance, meaning the model assigns **higher probabilities** to the correct next words.

🔸 Q2. (Apply, 5 marks)

A model predicts the sentence: "AI is powerful" with probabilities:

- P(“AI”) = 0.4
- P(“is” | “AI”) = 0.2
- P(“powerful” | “AI is”) = 0.1

Calculate the perplexity.

✅ **Solution:**

$$P(\text{sentence}) = 0.4 \times 0.2 \times 0.1 = 0.008$$

$$\text{Perplexity} = (0.008)^{-1/3} \approx 5$$

✅ So, the model’s perplexity on this sentence is **5**.

🔸 Q3. (Evaluate, 6 marks)

Why is perplexity used to evaluate language models? What are its advantages and limitations?

✅ **Model Answer:**

Perplexity is used because it reflects how well a model **predicts text**.
Lower perplexity means the model assigns **higher probabilities to correct words**, indicating better performance.

Advantages:

- Easy to compute
- Standard metric in language modeling
- Allows model comparison

Limitations:

- Sensitive to vocabulary size
- Can’t measure task-specific performance
- Doesn’t account for grammatical correctness or meaning

Thus, while useful, perplexity should be complemented with other metrics in downstream tasks.

✅ Topic 3: Smoothing Techniques in NLP

🔹 Why Do We Need Smoothing?

In unsmoothed N-gram models, **unseen word combinations** get a probability of **zero**, even if they are valid in real-world language.

This leads to two major problems:

- The total sentence probability becomes zero (breaks the chain)
- The model **fails to generalize** to new contexts

✅ **Smoothing adjusts** these probabilities so that:

- No valid word sequence has zero probability
- Probabilities are more evenly distributed

🔹 How Smoothing Works in Word-Level Analysis

In **word-level N-gram models**, smoothing modifies the way **word probabilities are calculated**, especially when a word or word sequence is:

- **Rare**
- **Absent from training data**

✅ Smoothing helps models to **"guess better"** and assign small but non-zero probabilities to unseen N-grams, improving performance on new data.

🔹 Types of Smoothing Techniques

Smoothing Type	Key Idea	Formula (for bigram)
Add-1 (Laplace)	Add 1 to all counts (including unseen)	[P(w_n
Add-k Smoothing	Add small value k < 1 (more flexible than Laplace)	Same as above, with +k and +kV
Good-Turing	Re-estimates count of N-grams based on frequency of frequencies	Adjusts: Count(0) → based on Count(1)
Backoff Models	If trigram not found, backoff to bigram or unigram	Tries smaller context if large-N fails
Interpolation	Combine all levels (unigram, bigram, trigram) using weighted average	$P = \lambda_1 P_{unigram} + \lambda_2 P_{bigram} + \lambda_3 P_{trigram}$

🔸 Let's Understand Add-1 (Laplace) Smoothing with an Example:

Suppose corpus:

"I love AI. I love NLP."

Now:

- Count("love AI") = 1
- Count("love") = 2
- Vocabulary size (V) = 4 (I, love, AI, NLP)

Without smoothing:

$$P(\text{AI} \mid \text{love}) = \frac{1}{2} = 0.5$$

$$P(\text{cool} \mid \text{love}) = \frac{0}{2} = 0 \text{❌}$$

With Laplace smoothing:

$$P(\text{cool} \mid \text{love}) = \frac{0+1}{2+4} = \frac{1}{6} \approx 0.167$$

Now **"love cool"** has **non-zero** probability, which helps the model generalize better!

🔸 How Smoothing Helps in Word-Level NLP Tasks

Word-Level Task	Role of Smoothing
Spell correction	Assigns probability to unseen typos (e.g., "fren")
Text prediction	Predicts rare next words using smoothed N-gram probabilities
POS tagging	Handles rare tag sequences or unknown words
Language generation	Prevents generation from breaking on unseen N-grams

---

📝 Exam-Oriented Questions & Model Answers

🔸 Q1. (Understand, 4 marks)

What is smoothing in N-gram language models? Why is it needed?

✅ **Model Answer:**

Smoothing is a technique used in N-gram models to assign **non-zero probabilities** to **unseen word sequences** in the training data.
It solves the **zero-probability problem** in unsmoothed models, allowing the system to generalize better to new data.

Without smoothing, any unknown N-gram makes the entire sentence probability zero. Techniques like **Laplace**, **Good-Turing**, and **backoff** are used to smooth N-gram probabilities.

🔸 Q2. (Apply, 5 marks)

Using Laplace smoothing, calculate P("world" | "hello") given:

- Count("hello") = 3
- Count("hello world") = 0
- Vocabulary size = 5

✅ **Solution:**

$$P(\text{world} \mid \text{hello}) = \frac{0+1}{3+5} = \frac{1}{8} = 0.125$$

✅ This assigns **non-zero probability** to “hello world”, even if it wasn’t seen in training.

🔸 Q3. (Evaluate, 6 marks)

Compare Laplace, Good-Turing, and Interpolation smoothing in N-gram models.

✅ **Model Answer:**

- **Laplace Smoothing (Add-1)**:
  Adds 1 to all counts. Simple, but **over-smooths** frequent words.
- **Good-Turing Smoothing**:
  Adjusts probabilities based on the number of N-grams that occur once (frequency of frequencies).
  More accurate for rare events, but harder to compute.
- **Interpolation**:
  Combines multiple models (unigram, bigram, trigram) using weights (λ).
  More **flexible and powerful**, but requires tuning.

Each technique balances **accuracy** and **generality**, and is used based on the task and data size.

✅ Topic 4: Backoff and Interpolation in N-Gram Models

These are **advanced smoothing techniques** used to handle the **data sparsity problem** more effectively than basic methods like Add-1.

🔹 1. Backoff Smoothing

Backoff means:
🔁 If a higher-order N-gram (like trigram) has **zero count**, **back off** to a lower-order model (like bigram or unigram).

📌 **Example:**

If we want:

$$P(\text{”NLP”} \mid \text{”love to”})$$

and "love to NLP" is **unseen**, then:

- Back off to bigram: $P(\text{”NLP”} \mid \text{”to”})$
- If still not seen, back off to unigram: $P(\text{”NLP”})$

🔄 **Fall back to less specific context** if data is missing.

🔹 2. Interpolation Smoothing

Interpolation combines multiple N-gram models (unigram, bigram, trigram) by **assigning weights (λ)** and **blending their probabilities**.

✅ Formula:

$$P(w_n|w_{n-2}, w_{n-1}) = \lambda_3 P_{\text{trigram}} + \lambda_2 P_{\text{bigram}} + \lambda_1 P_{\text{unigram}}$$

Where:

- $\lambda_1 + \lambda_2 + \lambda_3 = 1$
- Weights are tuned based on validation data.

📌 Example:

If we have:

- Trigram: $P(w_n|w_{n-2}, w_{n-1}) = 0.1$
- Bigram: $P(w_n|w_{n-1}) = 0.2$
- Unigram: $P(w_n) = 0.05$

And:

- $\lambda_3 = 0.6, \lambda_2 = 0.3, \lambda_1 = 0.1$

Then:

$$P(w_n) = 0.6 \times 0.1 + 0.3 \times 0.2 + 0.1 \times 0.05 = 0.06 + 0.06 + 0.005 = 0.125$$

🔹 Comparison: Backoff vs Interpolation

Feature	Backoff	Interpolation
Uses lower-order model only when higher is missing	✅	❌
Combines all orders (even if higher exists)	❌	✅
Faster but less accurate	✅	❌
More flexible and stable	❌	✅
Better generalization	❌	✅

🔹 Applications of Backoff & Interpolation

Task	Role of Backoff/Interpolation
Spell Correction	Estimate rare corrections using lower N-grams
Text Prediction	Predict next words using partial context
Speech Recognition	Handles noisy inputs with fallback predictions

Chatbots	Improves robustness when input is incomplete or unseen

---

📝 Exam-Oriented Questions & Model Answers

🔸 Q1. (Understand, 4 marks)

What is backoff in N-gram models? Explain with an example.

✅ **Model Answer:**

Backoff is a smoothing technique where the model **falls back to lower-order N-gram models** when higher-order probabilities are zero.

Example:
If trigram "love to NLP" is not found, the model backs off to bigram "to NLP". If that's also missing, it uses unigram "NLP".

This allows the model to still assign probabilities to **unseen N-grams**, improving robustness.

🔸 Q2. (Apply, 5 marks)

Given the following:

- $P_{\text{trigram}} = 0.1$
- $P_{\text{bigram}} = 0.2$
- $P_{\text{unigram}} = 0.05$
- $\lambda_3 = 0.6, \lambda_2 = 0.3, \lambda_1 = 0.1$

Calculate the interpolated probability.

✅ **Solution:**

$$P = 0.6 \times 0.1 + 0.3 \times 0.2 + 0.1 \times 0.05 = 0.125$$

✅ The interpolated probability is **0.125**.

🔸 Q3. (Evaluate, 6 marks)

Compare backoff and interpolation smoothing. Which one is more effective and why?

✅ **Model Answer:**

Backoff uses lower-order N-grams **only if higher-order ones are missing**, while **interpolation** combines all N-gram orders using **weighted averages**, even if higher-order exists.

Backoff	Interpolation
Simpler, faster	More flexible
May lose accuracy	Better generalization
Conditional use	Always blends info

Interpolation is more effective in practice as it considers **all levels of context**, making predictions more stable and accurate, especially with limited training data.

✅ Topic 5: Word Classes in NLP

🔹 What are Word Classes?

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

Examples include:

- Nouns
- Verbs
- Adjectives
- Prepositions
- Pronouns
- Conjunctions, etc.

Understanding word classes is essential for tasks like:

- Part-of-speech tagging
- Syntax parsing
- Named Entity Recognition (NER)
- Machine translation

🔹 Major Categories of Word Classes

Type	Examples	Function
Noun	dog, India, car	Names person, place, thing
Verb	run, eat, is	Shows action or state
Adjective	big, beautiful	Describes noun
Adverb	quickly, very	Modifies verb/adjective
Pronoun	he, she, it	Replaces noun
Preposition	on, in, by	Shows relationship
Conjunction	and, but, or	Connects words/clauses
Interjection	oh!, wow!	Expresses emotion

🔹 Open vs Closed Word Classes

Word Class Type	Description	Examples
Open Classes	Continuously accept new words	Nouns, Verbs, Adj
Closed Classes	Rarely accept new words, serve grammatical purpose	Pronouns, Prepositions, Conjunctions

🔍 **Open classes** evolve (e.g., "selfie", "hashtag").
📘 **Closed classes** are more fixed (e.g., "in", "of", "and").

🔹 Importance of Word Classes in NLP

Application	How Word Classes Help
POS Tagging	Labeling words based on their class
Parsing	Building grammar trees using word roles
Information Extraction	Identifying entities based on noun phrases
Speech synthesis	Stress and intonation depend on word class

🔹 Real-Life Example

Sentence:

“The quick brown fox jumps over the lazy dog.”

POS tags:

- The (Determiner)
- quick, brown (Adjective)
- fox, dog (Noun)
- jumps (Verb)
- over (Preposition)
- lazy (Adjective)

This POS structure helps models **understand grammar**, not just words.

📝 Exam-Oriented Questions & Model Answers

🔸 Q1. (Understand, 4 marks)

What are word classes in NLP? Differentiate between open and closed word classes with examples.

✅ **Model Answer:**

Word classes are grammatical categories that group words based on their roles in a sentence, such as nouns, verbs, adjectives, and prepositions.

- **Open word classes** can accept new words.
  Examples: Nouns (“blog”), Verbs (“google”), Adjectives.
- **Closed word classes** do not usually expand.
  Examples: Pronouns (“he”), Prepositions (“in”), Conjunctions (“and”).

They are essential for syntactic and semantic analysis in NLP.

🔸 Q2. (Apply, 5 marks)

Classify the words in the sentence: “AI is transforming the world rapidly” into their word classes.

✅ **Solution:**

- AI → Noun
- is → Verb
- transforming → Verb (present participle)
- the → Determiner
- world → Noun
- rapidly → Adverb

This classification is useful in tasks like POS tagging and dependency parsing.

🔸 Q3. (Evaluate, 6 marks)

Why are word classes important in NLP applications? Explain with examples.

✅ **Model Answer:**

Word classes help NLP systems understand the **syntactic and semantic role** of each word, enabling accurate language understanding.

- In **POS tagging**, tagging “run” as a **noun** or **verb** depends on its context.
- In **NER**, nouns often signal entities.
- In **machine translation**, verbs must match tense and voice.

Without correct word class identification, models may generate grammatically incorrect or ambiguous outputs.

✅ Topic 6: Part-of-Speech (POS) Tagging in NLP

🔹 What is POS Tagging?

POS tagging is the process of **assigning a part of speech** (noun, verb, adjective, etc.) to each word in a sentence based on:

- Its **definition**
- Its **context** in the sentence

🔸 Example:

Sentence:

"She will book a ticket."

Here, the word **"book"** is a **verb** (not a noun) — POS tagging helps determine this.

🔹 Common POS Tags (Penn Treebank Example):

Tag	Meaning	Example
NN	Noun	dog, AI
VB	Verb	run, go
JJ	Adjective	big
RB	Adverb	quickly
PRP	Pronoun	he, they
IN	Preposition	on, in
DT	Determiner	the, a

🔹 POS Tagging Techniques

Technique	Description
Rule-Based Tagging	Uses **manually defined grammar rules** based on patterns & context
Stochastic Tagging	Uses **probability models** (like Hidden Markov Models)
Transformation-Based	Uses **error-driven learning**, like **Brill's Tagger** (hybrid of rules + ML)

🔸 1. Rule-Based Tagging

- Applies **handcrafted rules**
- Uses suffixes, word position, and context

Example Rule:

"If a word ends in '-ly', tag as Adverb"

✅ Simple, but **limited and labor-intensive**

🔸 2. Stochastic Tagging (HMM)

- Uses **probabilities** from annotated corpora
- Selects the tag sequence with **highest probability**

$$P(\text{tag sequence} \mid \text{word sequence}) \propto P(\text{word} \mid \text{tag}) \cdot P(\text{tag} \mid \text{previous tag})$$

✅ More accurate than rule-based for large corpora

🔸 3. Transformation-Based Tagging (Brill's Tagger)

- Starts with a **baseline** (e.g., most frequent tag)
- Learns **correction rules** from training data

✅ Combines rule-based precision with statistical robustness

🔹 Applications of POS Tagging

Application	Purpose
Text summarization	Identifying nouns & key verbs
Question answering	Understanding sentence structure
Sentiment analysis	Detecting adjectives (positive/neg)
Named Entity Recognition	Tagging nouns as entities

📝 Exam-Oriented Questions & Model Answers

🔸 Q1. (Understand, 4 marks)

What is POS tagging? List any four POS tags with examples.

✅ **Model Answer:**

POS tagging is the process of assigning grammatical categories (like noun, verb, adjective) to words in a sentence using context and rules.

Examples:

- NN (Noun): "cat"
- VB (Verb): "run"
- JJ (Adjective): "smart"
- RB (Adverb): "quickly"

It helps in syntactic and semantic NLP tasks like parsing and question answering.

🔸 Q2. (Apply, 5 marks)

Tag the sentence: "AI changes everything rapidly."

✅ **Answer:**

- AI → NN
- changes → VB
- everything → NN
- rapidly → RB

🔸 Q3. (Evaluate, 6 marks)

Compare rule-based and stochastic POS tagging methods. Which is more effective and why?

✅ **Model Answer:**

Rule-based tagging uses handcrafted linguistic rules, while **stochastic tagging** uses statistical models (e.g., HMM) trained on annotated corpora.

Feature	Rule-Based	Stochastic
Based on	Grammar rules	Probabilities
Accuracy	Medium	Higher (on large data)
Flexibility	Low (manual effort)	High (auto learning)

Stochastic methods are generally more effective in real-world NLP due to better accuracy and adaptability.

🔍 Quick Concepts You Asked

✅ What is a Word Cloud?

A **word cloud** is a **visual representation** of the most **frequent words** in a document or dataset.

- **Larger font** = higher frequency
- Used in: EDA, content summarization, dashboards

📌 **Example Use:**

Creating a word cloud of job descriptions shows which skills (like "Python", "SQL") appear most.

✅ What is Bag of Words (BoW)?

The **Bag of Words** model represents text as a **vector of word frequencies**, **ignoring grammar and order**.

🔸 **Steps:**

1. Create vocabulary from the corpus

2. Count occurrences of each word
3. Represent each document as a vector of word counts

📌 **Example:**

Corpus:

- Doc1: "AI is smart"
- Doc2: "AI is the future"

Vocabulary: {AI, is, smart, the, future}

Vectors:

Document	AI	is	smart	the	future
Doc1	1	1	1	0	0
Doc2	1	1	0	1	1

✅ Widely used in **text classification**, **spam detection**, and **document similarity**.

✅ Topic 7: Issues in POS Tagging

While **POS tagging** is a foundational NLP task, it's not always straightforward. There are **challenges and edge cases** that can significantly affect tagging accuracy.

🔹 Major Issues in POS Tagging:

---

1. Ambiguity (Lexical or Syntactic)

Many English words can have **multiple parts of speech**, depending on the context.
This is the **biggest challenge** in POS tagging.

📌 **Examples:**

Word	POS (Case 1)	POS (Case 2)
Book	Verb → “Book a seat”	Noun → “Read a book”
Play	Noun → “a play by Shakespeare”	Verb → “kids play outside”

✅ Context is essential to decide the correct tag.

---

2. Unknown Words (Out-of-Vocabulary / OOV)

When a word **does not exist in the training data**, the tagger cannot find its tag easily.

This happens with:

- **Proper nouns**: e.g., "Raheeltronics"
- **New slang/technical words**: e.g., "Finfluencer", "Deepfake"

🧠 Solution: Use **morphological cues**, suffix patterns (like "-tion" → noun), or context to guess.

---

3. Multiple Valid Tag Sequences

Sometimes, a sentence can be **tagged in more than one correct way**, especially if punctuation is missing or structure is vague.

Example:

"Visiting relatives can be annoying."

- "Visiting" = Verb? or Adjective?
- Ambiguity in structure makes tagging harder.

---

4. Words with Changing Roles (Dynamic Tags)

Some words **change role based on sentence**.

Example:

"Google it" → Verb
"Use Google" → Noun

These dynamic uses require **contextual understanding**, which is hard for traditional taggers.

---

5. Idioms, Phrasal Verbs, Multiword Expressions

Phrases like:

- "give up"
- "take over"
- "kick the bucket"

These don't behave like isolated words. Taggers often tag each word separately, leading to **incorrect analysis**.

✅ Modern taggers (like spaCy or BERT-based ones) handle such expressions better.

---

6. Non-standard Text / Informal Language

In texts like:

- Tweets
- Chat messages
- Comments

You find:

- Misspellings: “luv” instead of “love”
- Emoticons, emojis
- Abbreviations like “gr8”, “u”, “r”

These confuse rule-based taggers and even some ML models.

📝 Exam-Oriented Questions & Model Answers

🔸 Q1. (Understand, 4 marks)

Mention any four challenges faced during POS tagging.

✅ **Model Answer:**

1. **Ambiguity** – Same word can have multiple tags based on context (e.g., “book” as noun/verb).
2. **Unknown Words** – New or rare words not seen in training data cause tagging errors.
3. **Multiword Expressions** – Idioms and phrasal verbs may be wrongly tagged word-by-word.
4. **Informal Texts** – Misspellings and slang in social media confuse standard taggers.

🔸 Q2. (Evaluate, 6 marks)

What is lexical ambiguity in POS tagging? How can it be resolved?

✅ **Model Answer:**

Lexical ambiguity occurs when a word can belong to **more than one word class**.
Example: “run” (noun) vs “run” (verb). The tag depends on the sentence structure.

Resolution strategies:

- Use of **contextual rules** (rule-based taggers)
- **Probabilistic models** like HMM
- **Neural models** (e.g., BERT) that understand sentence-level context

Lexical ambiguity is the **most common and critical** challenge in POS tagging.

✅ Topic 8: Hidden Markov Models (HMM) for POS Tagging

🔹 What is a Hidden Markov Model?

A **Hidden Markov Model (HMM)** is a **statistical model** used when:

- You observe a sequence (e.g., words in a sentence)
- But the actual states (e.g., POS tags) are **hidden** (not directly observed)

🔑 In NLP, HMM is used to predict the **sequence of tags** for a given sequence of words.

🔸 HMM POS Tagging Overview:

Layer	Example
Observed	The → cat → eats → fish
Hidden	DT → NN → VBZ → NN

🔹 HMM Components in POS Tagging

1. **States**: The possible POS tags (e.g., Noun, Verb, etc.)
2. **Observations**: The words in the sentence
3. **Transition Probabilities**:

$$P(t_i|t_{i-1})$$

Probability of current tag given the previous tag

4. **Emission Probabilities**:

$$P(w_i|t_i)$$

Probability of word given its tag

5. **Initial Probabilities**:

$$P(t_1)$$

Probability that a sentence starts with a given tag

🔹 Goal of HMM in POS Tagging:

Given a sentence (word sequence), find the **most probable tag sequence**:

$$\hat{T} = \arg\max_{T} P(T|W)$$

Using Bayes' Theorem, this is approximated by:

$$P(T|W) \propto P(W|T) \cdot P(T)$$

Where:

- $P(T)$ = Transition probabilities
- $P(W|T)$ = Emission probabilities

✅ We use the **Viterbi Algorithm** to efficiently find the best tag sequence.

🔹 Viterbi Algorithm (in brief)

- Dynamic programming method
- Stores the **most probable tag path** for each word position
- Runs in **O(n × |tags|²)** time

🔹 Example (Simplified)

Sentence: “Book a flight”

Possible tags:

- Book: NN or VB
- a: DT
- flight: NN

HMM considers:

- Which tag sequence is more **probable** given tag transitions
- Which tags are more **likely to emit** these words

If:

- P(VB | START) = 0.6
- P(NN | START) = 0.4
- P("Book" | VB) = 0.8
- P("Book" | NN) = 0.2

Then HMM picks:

- VB for "Book", not NN, because the probability path is higher

---

🔹 Advantages of HMM for POS Tagging:

Strength	Details

Handles sequence context	Learns dependencies between tags
Probabilistic	More flexible than strict grammar rules
Can deal with ambiguity	Chooses most likely tag sequence

🔹 Limitations

- Struggles with **unknown words**
- Assumes that the current tag depends only on the **previous tag** (Markov assumption)
- Newer deep learning models (like BERT) outperform HMMs in accuracy

---

📝 Exam-Oriented Questions & Model Answers

🔸 Q1. (Understand, 4 marks)

What is a Hidden Markov Model? How is it used in POS tagging?

✅ **Model Answer:**

A Hidden Markov Model (HMM) is a statistical model where the **observed output** depends on **hidden internal states**. In POS tagging:

- Words are observed
- POS tags are hidden
- HMM estimates the **most likely tag sequence** for a sentence using:
  - Transition probabilities between tags
  - Emission probabilities from tags to words

🔸 Q2. (Apply, 5 marks)

Given:

- P(NN | DT) = 0.8
- P(VB | DT) = 0.2
- P("book" | NN) = 0.1
- P("book" | VB) = 0.7

Which tag is more probable for the sentence: “The book”?

✅ **Solution:**

Calculate:

- For NN: 0.8 × 0.1 = 0.08
- For VB: 0.2 × 0.7 = 0.14

So, **“book” is more likely a verb (VB)** in this context.

🔸 Q3. (Evaluate, 6 marks)

Explain how HMM solves ambiguity in POS tagging with an example.

✅ Model Answer:

HMM resolves ambiguity by using **transition and emission probabilities** to determine the most probable tag sequence.

Example:
In the sentence “They can fish”:

- “can” could be **modal verb** or **noun**
- HMM uses probabilities like:
  - P(MD | PRP) and P("can" | MD)
  - P(NN | PRP) and P("can" | NN)

Whichever path gives a **higher total probability**, HMM selects that tag sequence. Thus, it balances **context and word likelihood**.

✅ Topic 9: Maximum Entropy Models (MaxEnt) for POS Tagging

🔹 What is a Maximum Entropy Model?

A **Maximum Entropy (MaxEnt)** model is a type of **probabilistic classifier** based on the principle:

"Among all models that fit the known data, choose the one with the maximum entropy."

This means it **makes the least assumptions** beyond what's observed — it doesn’t prefer any outcome unless the data supports it.

🔹 Why Use MaxEnt for POS Tagging?

MaxEnt models are:

- **Flexible**: Can use any feature (word, suffix, previous tags, etc.)
- **Discriminative**: Directly model the probability of tags **given** the input
- Better suited than HMMs for **non-sequential, feature-rich tagging**

🔹 MaxEnt in POS Tagging: The Basics

Goal:

Given a word and its **contextual features**, predict the most likely **POS tag**.

$$P(t|w, f_1, f_2, ..., f_n) = \frac{1}{Z(w)} \cdot \exp\left(\sum_i \lambda_i f_i(t, w)\right)$$

Where:

- t: POS tag
- w: word
- f_i: feature functions
- λ_i: model weights
- $Z(w)$: normalization constant

🔹 Example Features in MaxEnt POS Tagger:

Feature	Example Value
Current word	"play"
Previous word	"they"
Next word	"cricket"
Word suffix	"-ing", "-ed"
Is capitalized?	Yes / No
Previous predicted tag	NN / VB

✅ These features help handle ambiguity (e.g., "play" as noun or verb).

🔹 Advantages of MaxEnt for POS Tagging

Strength	Description
Feature-rich	Can use any number/type of contextual features
No independence assumption	Unlike HMM, doesn't assume Markov property
More accurate	Especially for complex or ambiguous sentences

🔹 Limitations

Limitation	Description
Slow training	Due to iterative optimization
Needs labeled data	Requires a large annotated corpus
Overfitting risk	If too many features are used without regularization

---

🔹 MaxEnt vs HMM (Quick Comparison)

Criteria	HMM	MaxEnt
Model Type	Generative	Discriminative

Context Used	Only previous tag (Markov)	Any features (word + context)
Assumptions	Needs independence	No strong assumptions
Feature Flexibility	Limited	Very high

📝 Exam-Oriented Questions & Model Answers

🔸 Q1. (Understand, 4 marks)

What is a Maximum Entropy model in POS tagging?

✅ **Model Answer:**

A Maximum Entropy model is a **probabilistic classifier** that predicts the most likely POS tag for a word using **contextual features**, while assuming no additional information (maximum entropy).

It uses feature functions (e.g., current word, previous word/tag, suffixes) to compute:

$$P(t|w) = \frac{1}{Z} \cdot \exp\left(\sum \lambda_i f_i(t, w)\right)$$

MaxEnt models are flexible and more accurate than HMMs in feature-rich environments.

🔸 Q2. (Apply, 5 marks)

What features would you extract from the sentence "They are playing cricket" to perform MaxEnt POS tagging on "playing"?

✅ **Answer:**

Possible features:

- Current word: "playing"
- Previous word: "are"
- Next word: "cricket"
- Suffix: "-ing"
- Is capitalized: No
- Previous tag: (e.g., PRP for "They", VBP for "are")

These features help the model predict "playing" as a **verb (VBG)** in this context.

🔸 Q3. (Evaluate, 6 marks)

Compare Maximum Entropy and Hidden Markov Models for POS tagging. Which is more flexible?

✅ **Model Answer:**

Feature	HMM	MaxEnt
Type	Generative	Discriminative
Feature Support	Limited (tags/words only)	Supports many contextual features
Assumptions	Markov & independence	Minimal assumptions

MaxEnt is more flexible as it:

- Uses richer features (prefixes, suffixes, previous tags)
- Doesn't assume tag independence or sequence order
- Handles real-world data with complex structures more effectively

Thus, MaxEnt outperforms HMM in complex tagging tasks.