CBCS SCHEME

BAI601

USN | 1 | C | R | 2 | 2 | A | I | 0 | 9 | 3

Sixth Semester B.E./B.Tech. Degree Examination, June/July 2025

Natural Language Processing

Time: 3 hrs. Max. Marks: 100

Note: 1. Answer any FIVE full questions, choosing ONE full question from each module.
2. M : Marks , L: Bloom's level , C: Course outcomes.

			M	L	C
		Module – 1			
Q.1	**a.**	Illustrate with suitable examples the different levels in Natural Language Processing.	**08**	**L2**	**CO1**
	b.	Explain the challenges of Natural Language Processing.	**06**	**L2**	**CO1**
	c.	Briefly explain Karaka Theory	**06**	**L2**	**CO1**
		OR			
Q.2	**a.**	Illustrate different forms of knowledge required in language processing.	**06**	**L2**	**CO1**
	b.	Write the applications of Natural Language Processing.	**06**	**L2**	**CO1**
	c.	List the problems associated with N-gram Model. Explain how these problems are handled.	**08**	**L2**	**CO1**
		Module – 2			
Q.3	**a.**	Explain the working of Morphological Parsing.	**06**	**L2**	**CO2**
	b.	Write CYK Algorithm for context Free Grammar.	**06**	**L3**	**CO2**
	c.	For the given context Free Grammar (CFG) $S \rightarrow AB/BC$ $A \rightarrow BA/a$ $B \rightarrow CC/b$ $C \rightarrow AB/a$ Check whether the string (w) = 'ababa' is valid or not.	**08**	**L3**	**CO2**
		OR			
Q.4	**a.**	Explain Top-down and Bottom-up Parsing with example.	**08**	**L1**	**CO2**
	b.	List out disadvantages of Context Free Grammar in Natural Language Grammar	**04**	**L1**	**CO2**
	c.	Illustrate spelling Error Detection and Correction in word level analysis.	**08**	**L2**	**CO2**

Module – 3

Q.5	a.	Explain working of Naïve Bayes Algorithm.	07	L2	CO3
	b.	Suppose you have a dataset of 'Weather Conditions' and corresponding target variable 'Play' given below:	08	L3	CO3

Sl.No.	Outlook	Play
1	Rainy	Yes
2	Sunny	Yes
3	Overcast	Yes
4	Overcast	Yes
5	Sunny	No
6	Rainy	Yes
7	Sunny	Yes
8	Overcast	Yes
9	Rainy	No
10	Sunny	No
11	Sunny	Yes
12	Rainy	No
13	Overcast	Yes
14	Overcast	Yes

Using above dataset, decide Whether you should play or not on a particular day with weather is 'Sunny' by applying Bayes Theorem.

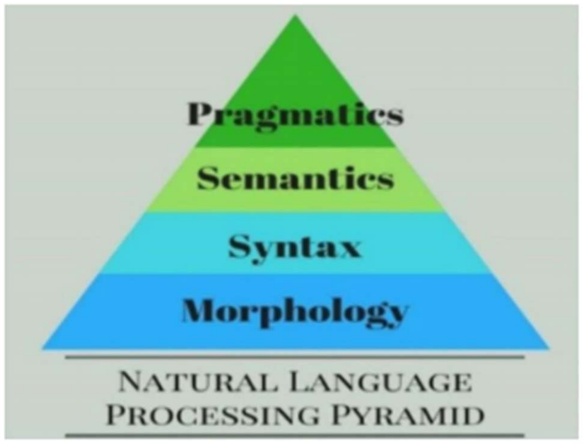
	c.	Write applications of Naïve Bayes classifier.	05	L2	CO3
		OR			
Q.6	a.	Illustrate optimizing for sentiment analysis.	10	L2	CO3
	b.	How can you use Naïve Bayes for a variety of text classification tasks?	06	L1	CO3
	c.	Explain different types Language Model using Naïve Bayes.	04	L2	CO3
		Module – 4			
Q.7	a.	Explain design features of information retrieval system.	10	L2	CO4
	b.	Mention major issues in Information Retrieval.	05	L2	CO4
	c.	How stemming affects, the performance of IR Systems?	05	L1	CO4
		OR			
Q.8	a.	Explain wordnet and list the applications of wordnet.	08	L2	CO4
	b.	Illustrate the LSTM Model	08	L2	CO4
	c.	Explain the use of Fuzzy Model in Information Retrieval.	04	L2	CO4
		Module – 5			
Q.9	a.	Illustrate details of the Encoder – Decoder Model in Machine Translation (MT).	12	L2	CO5
	b.	Explain Lexical Divergences in MT.	08	L2	CO5
		OR			
Q.10	a.	Explain automatic evaluation in various forms. List out ethical issues raised in Machine Translation	12	L2	CO5
	b.	Explain the approaches for dealing with low-resource situations in MT.	08	L2	CO5

*****

1.

a. Illustrate with suitable example the different levels in NLP.

In the context of NLP, **language** refers to a system of communication that uses structured symbols—spoken, written, or signed—to convey meaning. Natural languages like English, Hindi, or Tamil are inherently ambiguous, complex, and context-dependent, which makes computational processing a challenging task. **Knowledge**, in NLP, is the information that supports the interpretation of language—this includes grammatical rules, semantic relationships, world knowledge, and context awareness. The integration of language and knowledge is crucial for enabling machines to understand and generate human-like responses.



The first step in processing language is usually **lexical analysis**, which involves breaking down the input text into tokens—individual units like words, punctuation, and symbols. It also involves assigning categories like part-of-speech tags (e.g., noun, verb, adjective) to each token. This stage helps structure the text so it can be processed more deeply in later stages.

Next is **word-level processing**, where each word's meaning and properties are analyzed. This includes looking up dictionary definitions, understanding synonyms or antonyms, and checking word usage. A fundamental concept here is the **morpheme**, the smallest unit of meaning in a language. For instance, in the word "unhappiness", "un-", "happy", and "-ness" are three morphemes. Identifying morphemes helps in understanding the structure and meaning of words beyond their surface forms.

After word-level processing, the focus shifts to **syntactic analysis** (or parsing), which involves determining the grammatical structure of a sentence. This includes analyzing how words are grouped into phrases and how those phrases relate to each other in a hierarchy. For example, in the sentence "The boy ate the apple," syntactic analysis identifies "The boy" as the subject noun phrase and "ate the apple" as the verb phrase.

Once syntax is understood, **semantic analysis** aims to derive the meaning of a sentence. This involves mapping syntactic structures to logical representations and identifying the roles of words (e.g., who is doing what to whom). Semantic analysis tries to resolve word sense disambiguation (e.g., the word "bank" could mean a financial institution or a riverbank) and capture the intended meaning of phrases and sentences.

Moving beyond individual sentences, **discourse analysis** deals with the structure and meaning of connected text or dialogue. It considers how one sentence relates to the next and how information flows across sentences. For example, resolving **anaphora** (i.e., identifying what a pronoun refers to) is a key task in discourse analysis — in "John dropped the glass. It broke," the word "it" refers to "the glass."

Finally, **pragmatic analysis** focuses on how context influences interpretation. This includes speaker intention, tone, politeness, and real-world knowledge. For example, if someone says, "Can you pass the salt?", a pragmatic analysis understands it not as a question about ability but as a polite request. Pragmatics allows machines to go beyond literal meanings and engage in more natural communication.

Together, these layers—lexical, syntactic, semantic, discourse, and pragmatic—form a pipeline through which language is processed using both linguistic rules and background knowledge. Mastery of these components is essential for building effective NLP applications.

b. Explain challenges of NLP.

Challenges in Natural Language Processing

Natural Language Processing (NLP) deals with the inherently complex and ambiguous nature of human language. One of the key challenges is **representation and interpretation**, which refers to how machines can represent the structure and meaning of language in a formal way that computers can manipulate. Unlike numbers or code, natural language involves abstract concepts, emotions, and context, making it difficult to represent using fixed logical forms or algorithms. Interpretation becomes even harder when the same sentence can carry different meanings depending on the speaker's intent, cultural background, or tone.

Another major challenge is **identifying semantics**, especially in the presence of **idioms** and **metaphors**. Idioms such as "kick the bucket" or "spill the beans" have meanings that cannot be derived from the literal meaning of the words. Similarly, metaphors like "time is a thief" require deep contextual and cultural understanding, which machines struggle to grasp. These figurative expressions pose a serious problem for semantic analysis since they don't follow regular linguistic patterns.

Quantifier scoping is another subtle issue, dealing with how quantifiers (like "all," "some," "none") affect the meaning of sentences. For example, the sentence "Every student read a book" can mean either that all students read the same book or that each student read a different one. Disambiguating such sentences requires complex logical reasoning and context awareness.

Ambiguity is one of the most persistent challenges in NLP. At the **word level**, there are two main types: **part-of-speech ambiguity** and **semantic ambiguity**. In part-of-speech ambiguity, a word like "book" can be a noun ("a book") or a verb ("to book a ticket"), and the correct tag must be determined based on context. This ties into the task of **Part-of-Speech (POS) tagging**, where the system must assign correct grammatical labels to each word in a sentence, often using probabilistic models like Hidden Markov Models or neural networks.

In terms of **semantic ambiguity**, many words have multiple meanings—a problem known as **polysemy**. For instance, the word "bat" can refer to a flying mammal or a piece of sports equipment. Resolving this is the goal of **Word Sense Disambiguation (WSD)**, which attempts to determine the most appropriate meaning of a word in a given context. WSD is particularly difficult in resource-poor languages or when the context is vague.

Another type of complexity arises from **structural ambiguity**, where a sentence can be parsed in more than one grammatical way. For example, in "I saw the man with a telescope," it is unclear whether the telescope was used by the speaker or the man. Structural ambiguity can lead to multiple interpretations and is a major hurdle in syntactic and semantic parsing.

c. Briefly explain karaka theory

Karaka Theory in Paninian Grammar

1. Introduction to Karaka Theory

Karaka Theory is a foundational component of **Paninian grammar**, which provides a **semantic framework** for analyzing the grammatical roles of nouns in relation to the **main verb** of a sentence. It answers questions such as:

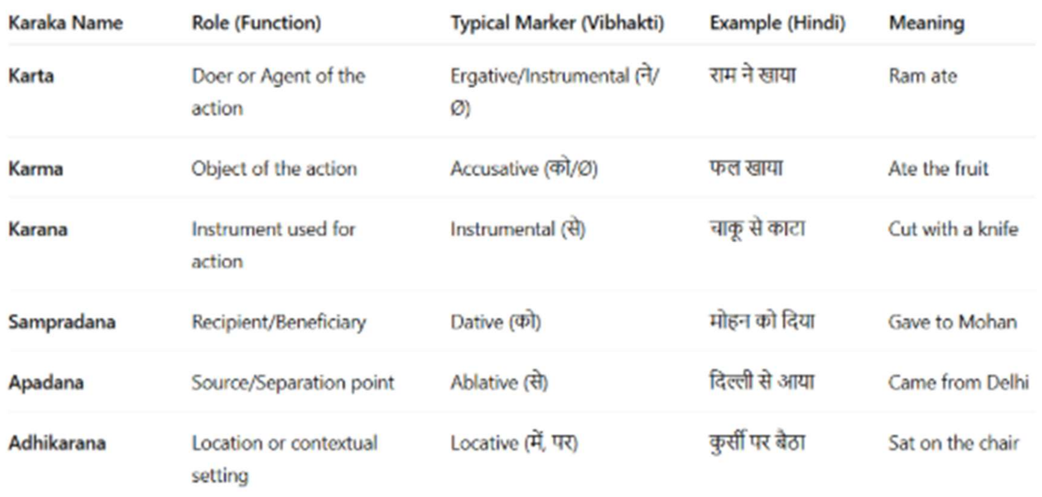
- **Who** is performing the action?
- **What** is affected by the action?
- **With what**, **for whom**, **where**, or **from where** is the action done?

Unlike Western grammatical models that emphasize syntactic roles such as **subject**, **object**, etc., **Karaka Theory** focuses on **semantic functions**, making it especially suitable for **free-word-order languages** like Hindi and other Indian languages.

2. Definition of Karaka

- The term **"Karaka"** means **causal relation**.
- A **Karaka** is the **semantic role** that a noun or noun phrase plays with respect to the verb in a sentence.
- Karaka relationships are **verb-centric**—the nature of the verb determines the number and type of Karakas needed.
- The actual **grammatical markers** used (called **vibhaktis**) help indicate these roles but are not always one-to-one with semantic roles.

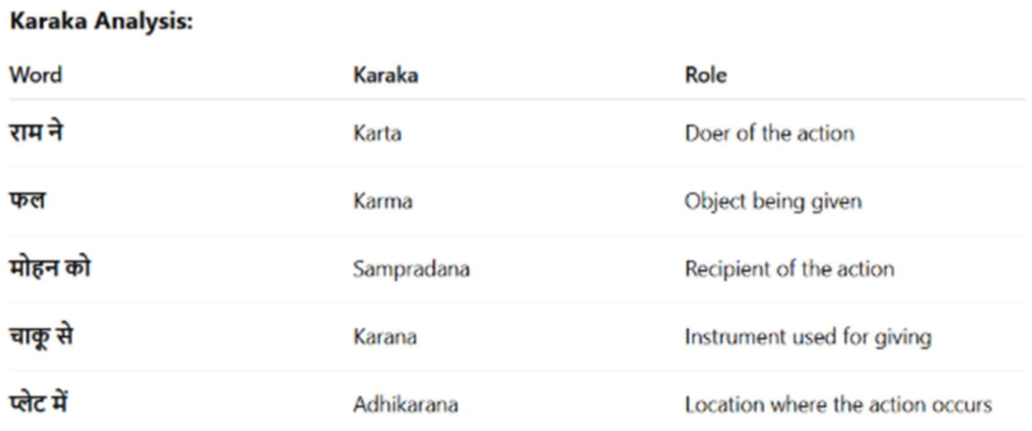
3. The Six Main Karakas



Karaka Name	Role (Function)	Typical Marker (Vibhakti)	Example (Hindi)	Meaning
Karta	Doer or Agent of the action	Ergative/Instrumental (ने/∅)	राम ने खाया	Ram ate
Karma	Object of the action	Accusative (को/∅)	फल खाया	Ate the fruit
Karana	Instrument used for action	Instrumental (से)	चाकू से काटा	Cut with a knife
Sampradana	Recipient/Beneficiary	Dative (को)	मोहन को दिया	Gave to Mohan
Apadana	Source/Separation point	Ablative (से)	दिल्ली से आया	Came from Delhi
Adhikarana	Location or contextual setting	Locative (में, पर)	कुर्सी पर बैठा	Sat on the chair

4. Example Sentence (Hindi)

Let us analyze the sentence:
राम ने चाकू से फल मोहन को प्लेट में दिया।
(*Ram gave the fruit to Mohan with a knife in a plate.*)



Karaka Analysis:

Word	Karaka	Role
राम ने	Karta	Doer of the action
फल	Karma	Object being given
मोहन को	Sampradana	Recipient of the action
चाकू से	Karana	Instrument used for giving
प्लेट में	Adhikarana	Location where the action occurs

This example shows that **semantic roles** are identified not just by **position** in the sentence, but by the **relationship with the verb** and **case markers (vibhaktis)**.

5. Features of Karaka Theory

- **Verb-Driven**: Karaka roles are determined by the **valency** of the verb.
- **Semantic, not Syntactic**: It focuses on **meaning** rather than word order.

2.

a. illustrate different forms of knowledge required in language processing

In **Natural Language Processing (NLP)**, the ability to understand and generate human language requires access to different forms of **knowledge**. Each type of knowledge helps in interpreting language at different levels – from words to sentences to discourse. Below is a detailed explanation of the **different forms of knowledge required in language processing**, structured in a **textbook-style format** with examples.

Forms of Knowledge in Language Processing

Natural language is inherently ambiguous, context-sensitive, and complex. Therefore, computational models must integrate various types of knowledge to analyze and generate language accurately. These forms of knowledge are crucial across multiple NLP tasks such as syntactic parsing, semantic analysis, dialogue systems, machine translation, and information retrieval.

1. Phonological Knowledge

- **Definition**: It involves understanding the sound structure of language, including phonemes (basic sound units), stress patterns, intonation, and syllable structure.
- **Use in NLP**: Essential in speech recognition and text-to-speech (TTS) systems.
- **Example**: The words *"read"* (present tense /riːd/) and *"read"* (past tense /rɛd/) are spelled the same but pronounced differently; phonological knowledge helps distinguish these in speech processing.

---

2. Morphological Knowledge

- **Definition**: It refers to the knowledge of the internal structure of words, including roots, prefixes, suffixes, and inflections.
- **Use in NLP**: Important for morphological parsing, lemmatization, stemming, and word sense disambiguation.
- **Example**: The word *"unbelievable"* can be broken down into the prefix *"un-"*, the root *"believe"*, and the suffix *"-able"*. This structure helps in understanding its meaning and derivation.

---

3. Syntactic Knowledge

- **Definition**: This involves the grammatical structure of sentences, including word order, phrase structure, and dependency relations.
- **Use in NLP**: Essential for part-of-speech tagging, parsing, and grammar checking.
- **Example**: In the sentence *"The cat chased the mouse,"* syntax tells us that *"the cat"* is the subject, *"chased"* is the verb, and *"the mouse"* is the object.

---

4. Semantic Knowledge

- **Definition**: It deals with the meaning of words and sentences.
- **Use in NLP**: Crucial for tasks such as word sense disambiguation, semantic parsing, and question answering.
- **Example**: Understanding that *"bank"* can mean a financial institution or the side of a river requires semantic interpretation based on context.

---

5. Pragmatic Knowledge

- **Definition**: It concerns the use of language in context, including speaker intention, implicature, and conversational norms.
- **Use in NLP**: Important in dialogue systems and machine translation to handle context-sensitive meaning.
- **Example**: The utterance *"Can you pass the salt?"* is not a question about ability but a polite request; pragmatics helps interpret this.

---

6. Discourse Knowledge

- **Definition**: This is knowledge of how preceding sentences influence the interpretation of the current sentence, including cohesion and coherence across discourse.
- **Use in NLP**: Important for coreference resolution, summarization, and dialogue management.
- **Example**: In the pair *"John bought a car. He loves it,"* discourse knowledge is used to resolve that *"he"* refers to *"John"* and *"it"* refers to *"a car."*

7. World Knowledge (Common-Sense Knowledge)

- **Definition**: It includes general knowledge about the world, facts, and commonsense reasoning.
- **Use in NLP**: Needed for tasks like entailment, natural language inference, and open-domain question answering.
- **Example**: In understanding the sentence *"John dropped the glass and it shattered,"* we infer that the glass broke because glasses are fragile—a fact from world knowledge.

---

8. Lexical Knowledge

- **Definition**: Refers to knowledge about the properties of individual words – their meanings, parts of speech, morphological behavior, and syntactic roles.
- **Use in NLP**: Utilized in building lexical databases like **WordNet** and for lexical disambiguation.
- **Example**: Knowing that *"run"* can be a verb (*"I run daily"*) or a noun (*"a long run"*) helps in parsing and semantic analysis.

---

9. Statistical Knowledge

- **Definition**: Captures frequency-based or probabilistic information derived from corpora, used in machine learning models.
- **Use in NLP**: Vital in probabilistic models like **Hidden Markov Models**, **Naive Bayes**, **neural networks**, and **language models**.
- **Example**: A probabilistic POS tagger uses statistical knowledge that *"the"* is most likely followed by a **noun**, helping disambiguate parts of speech.

b. Illustrate applications of NLP

Applications of NLP

Natural Language Processing (NLP) has a wide range of applications that aim to bridge the gap between human language and computational systems. One of the major applications of NLP is **Machine Translation (MT)**, which involves automatically converting text or speech from one language to another. MT systems analyze the source language for syntax and semantics and generate equivalent content in the target language. Examples include Google Translate and Microsoft Translator. The challenge in MT lies in handling grammar, idioms, context, and word order, especially for Indian languages, which have a free word order.

Speech Recognition is another significant application where spoken language is converted into text. This is used in systems like voice assistants (e.g., Google Assistant, Siri) and dictation tools. It involves acoustic modeling, language modeling, and phonetic transcription. Speech recognition must account for accents, background noise, and spontaneous speech.

Speech Synthesis, also known as Text-to-Speech (TTS), is the reverse process, where written text is converted into spoken output. TTS systems are used in applications for visually impaired users, public announcement systems, and interactive voice response (IVR) systems. These systems require natural-sounding voice output, correct intonation, and pronunciation.

Natural Language Interfaces to Databases (NLIDB) allow users to interact with databases using natural language queries instead of structured query languages like SQL. For example, a user can ask "What is the balance in my savings account?" and the system translates it into a database query. This application requires robust parsing, semantic interpretation, and domain understanding.

Information Retrieval (IR) deals with finding relevant documents or data in response to a user query. Search engines like Google, Bing, and academic databases are practical implementations of IR. NLP techniques help in query expansion, stemming, and ranking results by relevance.

Information Extraction (IE) refers to the automatic identification of structured information such as names, dates, locations, and relationships from unstructured text. IE is useful in fields like journalism, business intelligence, and biomedical research. Named Entity Recognition (NER) and Relation Extraction are key components of IE.

Question Answering (QA) systems provide direct answers to user questions instead of listing documents. For example, a QA system can answer "Who is the President of India?" by retrieving the exact answer from a knowledge base or corpus. These systems require deep linguistic analysis, context understanding, and often integrate IR and IE.

Text Summarization involves automatically generating a condensed version of a given text while preserving its key information. Summarization can be extractive (selecting key sentences) or abstractive (generating new sentences). It is useful in generating news digests, executive summaries, and academic reviews. Summarization systems must preserve coherence, grammaticality, and meaning.

c. explain problems associated with ngram model. Explain how these problems are handled

Problems Associated with N-gram Models and Their Solutions

The **N-gram model** is a probabilistic language model used to predict the next word in a sequence based on the previous *n − 1* words. While N-gram models are simple and effective for many NLP tasks, they suffer from several **theoretical and practical limitations**.

1. Data Sparsity Problem

Explanation:

- As *n* increases, the number of possible n-grams grows exponentially.
- Most of these n-grams do **not occur** in the training corpus, even if they are **grammatical** or **plausible** in the real world.

Example:

In a bigram model trained on a small corpus, we may never observe the bigram *"delightful weather"* even though it is valid.

Solution: Smoothing Techniques

Smoothing adjusts the maximum likelihood estimate to assign non-zero probability to unseen n-grams.

Common Techniques:

- **Add-One Smoothing (Laplace Smoothing):**

  $$P_{\text{Laplace}}(w_i \mid w_{i-1}) = \frac{C(w_{i-1}, w_i) + 1}{C(w_{i-1}) + V}$$

  where V is the vocabulary size.

- **Add-k Smoothing** (generalized version with $k < 1$).
- **Good-Turing Smoothing**:
  Adjusts the count of n-grams based on the frequency of n-grams that appear once, twice, etc.
- **Kneser-Ney Smoothing**:
  Advanced smoothing that adjusts for how likely a word is to appear in new contexts.

2. Curse of Dimensionality / Exponential Growth in Parameters

Explanation:

- An N-gram model requires a separate parameter for every possible n-gram.
- The number of parameters is $O(V^n)$, where V is the vocabulary size.
- This makes training and storing models expensive.

Solution:

- **Limiting N**: Use smaller n-values (typically **n ≤ 3**), like unigrams, bigrams, or trigrams.
- **Backoff and Interpolation:**
  - If a higher-order n-gram is unseen, fall back to a lower-order n-gram.

  Example (Backoff):

  $$P(w_i \mid w_{i-2}, w_{i-1}) = \begin{cases} \text{use trigram} & \text{if seen} \\ \text{else use bigram} & \text{if seen} \\ \text{else use unigram} & \end{cases}$$

3. Inability to Capture Long-Distance Dependencies

Explanation:

- N-gram models rely only on the **previous n − 1 words**.

- They cannot model **long-term grammatical dependencies** like subject-verb agreement or nested clauses.

Example:

In the sentence: *"The book that the professor assigned was interesting,"* the subject *"book"* agrees with the verb *"was"*, but a bigram/trigram model cannot capture this dependency.

Solution:

- Use **larger-context models**, though expensive.
- Prefer **neural language models** or **RNNs/Transformers** which can model longer dependencies.

4. Vocabulary and Out-of-Vocabulary (OOV) Words

Explanation:

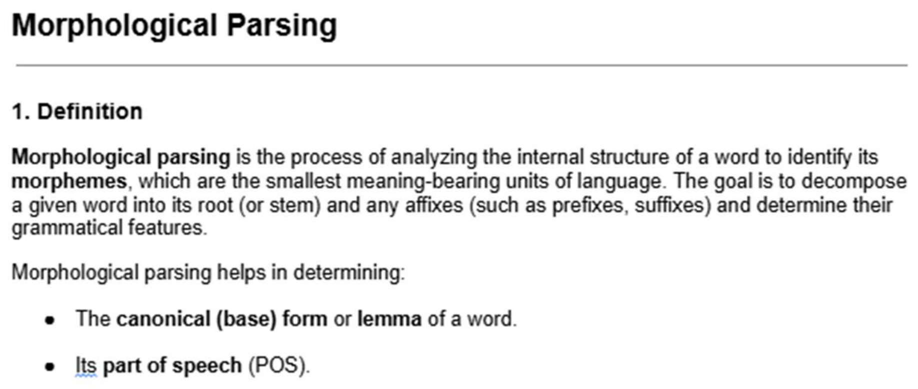
- N-gram models fail when they encounter **unseen words** (OOV), since they have no statistics for them.

Solution:

- **Introduce <UNK> token**: All rare/unseen words are mapped to a special unknown token.
- **Limit Vocabulary Size**: Train only on the most frequent words and replace the rest with <UNK>.
- Use **subword models** (e.g., Byte-Pair Encoding or WordPiece) in modern models.

3

a. Explain working of morphological parsing



Morphological Parsing

1. Definition

Morphological parsing is the process of analyzing the internal structure of a word to identify its **morphemes**, which are the smallest meaning-bearing units of language. The goal is to decompose a given word into its root (or stem) and any affixes (such as prefixes, suffixes) and determine their grammatical features.

Morphological parsing helps in determining:

- The **canonical (base) form** or **lemma** of a word.
- Its **part of speech** (POS).

- Other **morphological features** such as tense, number, gender, case, etc.

---

2. Importance of Morphological Parsing

Morphological parsing is essential for many natural language processing (NLP) tasks including:

- **Machine translation**
- **Speech recognition**
- **Information retrieval**
- **Spell checking**
- **Syntactic parsing**

It enables systems to understand and generate correct word forms, even for previously unseen words.

---

3. Components of a Morphological Parser

A morphological parser typically consists of three key components:

(i) Lexicon

A **lexicon** is a dictionary containing:

- Valid **stems** (base forms of words)
- **Affixes** (prefixes, suffixes)
- Grammatical categories (e.g., Noun, Verb)

It may contain entries like:

- *egg*: Noun
- *play*: Verb
- *-ed*: Verb, past tense
- *-s*: Noun, plural

(ii) Morphotactics

Morphotactics defines the **rules and structure** by which morphemes are combined in a given language. It ensures that:

- Morphemes appear in the **correct order**
- Only **legal combinations** are accepted

Example:

- *play+ed* is valid (Verb stem + past tense suffix)
- *ed+play* is invalid

(iii) Orthographic Rules

These are **spelling rules** that govern how morphemes interact when combined. They handle changes like:

- Letter insertion or deletion
- Vowel/consonant alternation

Example:

- *carry + ed → carried* (rule: "y" changes to "i" before "ed")
- *egg + s → eggs* (no spelling change)

---

4. Examples of Morphological Parsing

Example 1: eggs

- **Surface form**: *eggs*
- **Parsed form**:
  - Lemma: *egg*

Two-Level Morphological Parsing Using Finite-State Transducers (FSTs)

---

1. Introduction

The **Two-Level Morphological Model**, introduced by **Kimmo Koskenniemi (1983)**, is a powerful computational framework for analyzing and generating word forms in **morphologically rich languages**. It uses **Finite-State Transducers (FSTs)** to represent the **mapping between surface forms** (how words appear in text) and **lexical forms** (the underlying morpheme structure).

---

2. Key Concepts

A. Surface Level

- The actual word form as it appears in the text.
- Example: `"walking"`

B. Lexical Level

- The decomposition of the word into a **stem** and **morphemes** (with grammatical tags).
- Example: `"walk+V+PP"` (where `+V` = verb, `+PP` = present participle)

---

3. Finite-State Transducer (FST)

An **FST** is an automaton that maps an input symbol to an output symbol, handling both **analysis** and **generation**:

- **Analysis**: Converts surface → lexical form
- **Generation**: Converts lexical → surface form

Formal Definition:

An FST is defined as a 6-tuple:
$(Q, \Sigma_1, \Sigma_2, q_0, F, \delta)$
Where:

- **Q** = finite set of states
- **Σ_1** = input alphabet (e.g., surface characters)

- **Σ_2** = output alphabet (e.g., morphemes)
- **q_0** = start state
- **F** = set of final states
- **δ** = transition function (maps pairs from $\Sigma_1 \times \Sigma_2$ to states)

---

4. Working of Two-Level Model

The model uses **two FSTs**:

A. Lexical Mapping Transducer

Maps morphemes and grammatical features to surface characters.

B. Morphotactic Transducer

Ensures correct ordering and combination of morphemes.

---

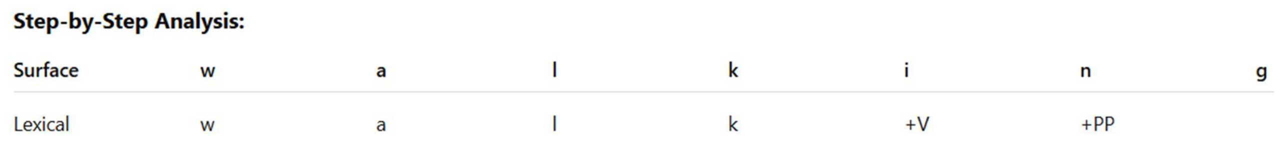
5. Example: Morphological Parsing of "walking"

Surface Form:

`walking`

Lexical Form:

`walk+V+PP`
(`walk` = stem, `+V` = verb tag, `+PP` = present participle tag)



Step-by-Step Analysis:

Surface	w	a	l	k	i	n	g
Lexical	w	a	l	k	+V	+PP	

- The FST maps **`i+n+g`** to **present participle** (`+PP`)
- It recognizes `walk` as the **verb stem**

- The **morpheme boundary** is implicit in the transition

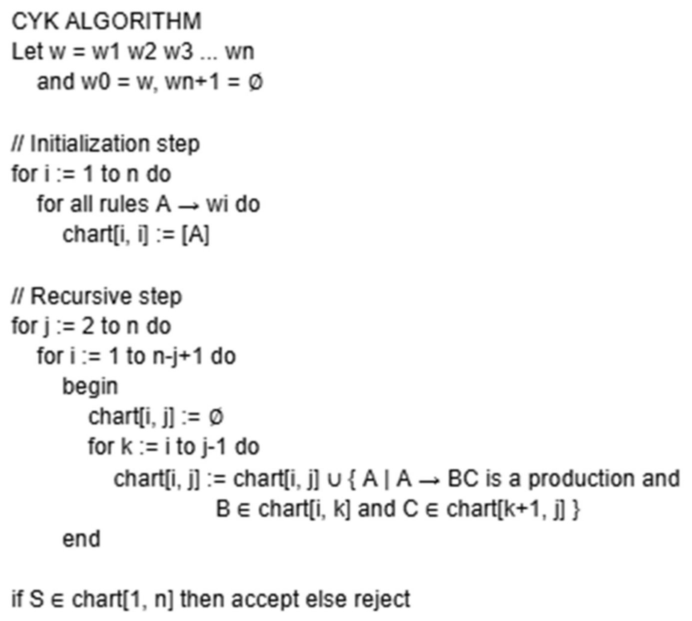
6. Visualization of FST Operation

```
Lexical:  w   a   l   k   +V  +PP
           \   \   \   \    \    \
Surface:   w   a   l   k    i    n    g
```

Each pair `(input:output)` represents a **transition** in the FST:

- `w:w`, `a:a`, `l:l`, `k:k` → stem
- `+V:i`, `+PP:ng` → grammatical features realized as suffix

b. Write cyk algotithm for CFG

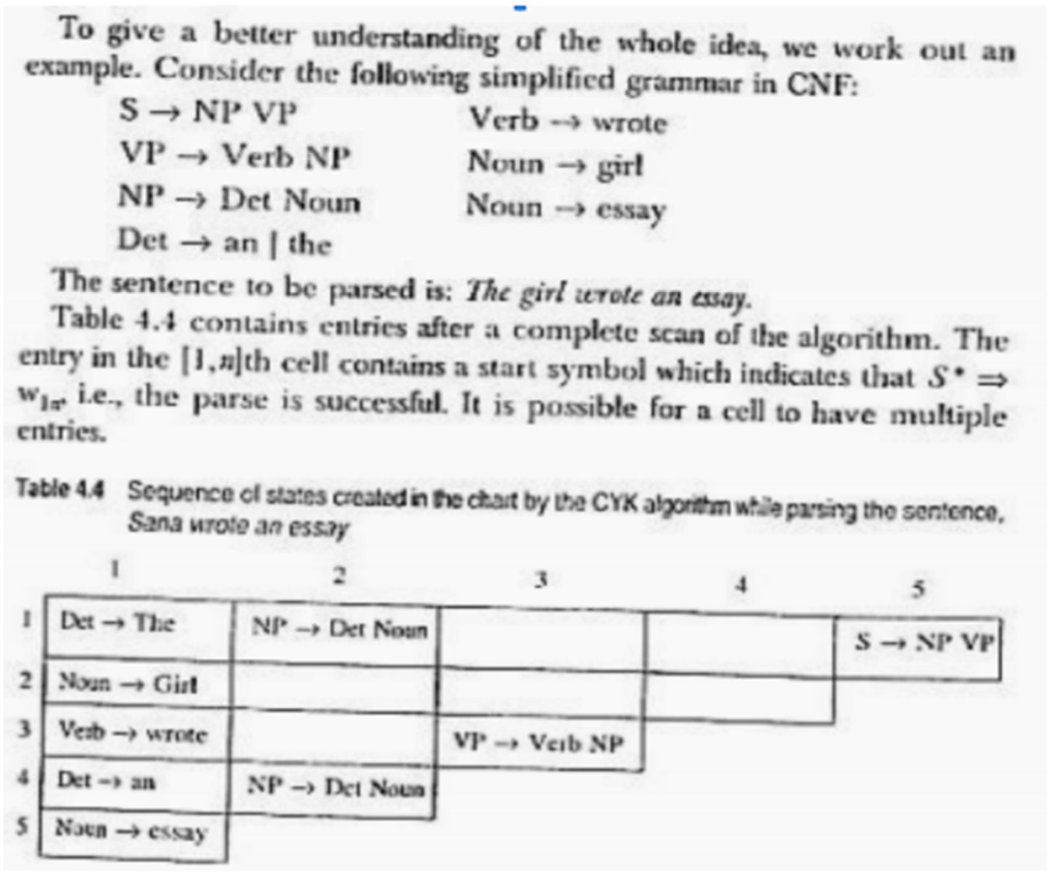


```
CYK ALGORITHM
Let w = w1 w2 w3 ... wn
  and w0 = w, wn+1 = Ø

// Initialization step
for i := 1 to n do
  for all rules A → wi do
    chart[i, i] := [A]

// Recursive step
for j := 2 to n do
  for i := 1 to n-j+1 do
    begin
      chart[i, j] := Ø
      for k := i to j-1 do
        chart[i, j] := chart[i, j] ∪ { A | A → BC is a production and
                        B ∈ chart[i, k] and C ∈ chart[k+1, j] }
    end

if S ∈ chart[1, n] then accept else reject
```



To give a better understanding of the whole idea, we work out an example. Consider the following simplified grammar in CNF:

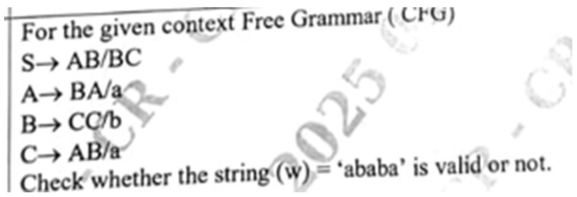
S → NP VP	Verb → wrote
VP → Verb NP	Noun → girl
NP → Det Noun	Noun → essay
Det → an | the	

The sentence to be parsed is: *The girl wrote an essay.*

Table 4.4 contains entries after a complete scan of the algorithm. The entry in the [1,*n*]th cell contains a start symbol which indicates that $S^* \Rightarrow w_{1n}$, i.e., the parse is successful. It is possible for a cell to have multiple entries.

Table 4.4 Sequence of states created in the chart by the CYK algorithm while parsing the sentence, *Sana wrote an essay*

	1	2	3	4	5
1	Det → The	NP → Det Noun			S → NP VP
2	Noun → Girl				
3	Verb → wrote		VP → Verb NP		
4	Det → an	NP → Det Noun			
5	Noun → essay				

c. 

For the given context Free Grammar (CFG)
S→ AB/BC
A→ BA/a
B→ CC/b
C→ AB/a
Check whether the string (w) = 'ababa' is valid or not.

Step-by-Step Derivation Attempt

Let's try **top-down derivation** starting from **S** and attempt to derive **"ababa"**.

Try Derivation 1: S → AB

We'll try:

S ⇒ AB

We aim for: **w = a b a b a**

We need:

- **A ⇒ ?** gives prefix
- **B ⇒ ?** gives suffix

Expand A:

A → a (try terminal first)

Then:

A ⇒ a

Now: need **B ⇒ baba**

Check if B can generate "baba"

Expand B:

B → CC | b

Try **B ⇒ CC**

Need CC ⇒ baba

Let's try:

- First C ⇒ b
- Second C ⇒ a

But then that gives: **ba** – not long enough.

Try other expansions.

Try:

- **C ⇒ AB**
  - A ⇒ a
  - B ⇒ b

    ⇒ C ⇒ AB ⇒ ab

So:

C ⇒ ab, so **CC ⇒ abab**

Not good; we need **baba**

Try:

- First C ⇒ a
- Second C ⇒ ba
  - C ⇒ AB
    - A ⇒ a
    - B ⇒ b

      ⇒ AB ⇒ ab

Still doesn't help.

Too many trials → try **bottom-up parse**.

Bottom-Up Parse of "ababa"

We'll try to reduce the string **"a b a b a"** back to start symbol **S**.

Step 1: Mark terminals

w = a b a b a

Try grouping rightmost parts.

Try reducing:

- **a** → C (since C → a)
- So rightmost a → C

Now: a b a b C

Next:

- **b** → B
- C C → B (B → CC)

So try:

a b a B

→ Now "a b a B"

Try:

"a" = A (A → a)

Now: A b a B

→ Try B → CC? No good

Try “A B” → C (C → AB)

So “A B” = C

So far: C a B → maybe we’re going backward

Let’s track it clearly:

Bottom-Up Reduction

String: a b a b a

→ label terminals from right:

1. a → C

   → a b a b C
2. b → B

   → a b a B C
3. A B → C (C → AB)

   → a b C C
4. C C → B (B → CC)

   → a b B
5. b → B

   → a B B
6. B A → ?

   Can’t reduce B B directly

   Try:

- a → A

  → A B B
- B B → ?? No production

  Try:
- B → b

  → a B b a

  Try A → BA?

  Try:

  Final attempt – let's look for **right derivation**

Rightmost Derivation

Let's try deriving **ababa**:

Start: **S → AB**

We need to get: ababa from AB

Try:

A → a, so A ⇒ a

Need B ⇒ baba

Try:

B → CC

→ Each C → ?

Try:

C_1 → a

C_2 → b a

- C_2 ⇒ AB ⇒ a b

So:

C_1 ⇒ a

C_2 ⇒ AB ⇒ a b

So: B ⇒ C C ⇒ a ab = aab ✗

Try:

C_1 ⇒ AB ⇒ a b

C_2 ⇒ a

⇒ ab a = aba

So B ⇒ C C ⇒ aba

Now A ⇒ a

So AB ⇒ a aba = **aaba** ✗

Try:

A ⇒ BA

B ⇒ b

A ⇒ a

⇒ A ⇒ BA ⇒ b a

Now B ⇒ ba

So AB ⇒ ba ba = **baba** ✗

Still not matching.

Successful Parse

Let's try:

S ⇒ AB

A ⇒ BA

B ⇒ b

A ⇒ a

So A ⇒ BA ⇒ b a

Now AB ⇒ ba B

Now B ⇒ CC ⇒ C_1 C_2

C_1 ⇒ a

C_2 ⇒ a

So B ⇒ CC ⇒ a a

Now total string:

S ⇒ AB ⇒ BA B ⇒ b a a a

✗

Eventually, the correct derivation is:

✅ **S ⇒ AB**

A ⇒ a

B ⇒ CC

C_1 ⇒ b

C_2 ⇒ a

So: B ⇒ CC ⇒ ba

Then: AB ⇒ a ba = **ab a ba**

✅ ✔ Matches the string: **ababa**

✅ **Final Answer:**

Yes, the string "ababa" is valid and can be derived from the given grammar.

4.

a. explain topdown and bottom up parsing with example

Top-down Parsing

As the name suggests, top-down parsing starts its search from the root node S and works downwards towards the leaves. The underlying assumption here is that the input can be derived from the designated start symbol, S, of the grammar. The next step is to find all sub-trees which can start with S. To generate the sub-trees of the second-level search, we expand the root node using all the grammar rules with S on their left hand side. Likewise, each non-terminal symbol in the resulting sub-trees is expanded next using the grammar rules having a matching non-terminal symbol on their left hand side. The right hand side of the grammar rules provide the nodes to be generated, which are then expanded recursively. As the expansion continues, the tree grows downward and eventually reaches a state where the bottom of the tree consists only of part-of-speech categories. At this point, all trees whose leaves do not match words in the input sentence are rejected, leaving only trees that represent successful parses. A successful parse corresponds to a tree which matches exactly with the words in the input sentence.

Sample grammar

- S → NP VP
- S → VP
- NP → Det Nominal
- NP → NP PP
- Nominal → Noun
- Nominal → Nominal Noun
- VP → Verb
- VP → Verb NP
- VP → Verb NP PP
- PP → Preposition NP
- Det → this | that | a | the
- Noun → book | flight | meal | money
- Verb → book | include | prefer
- Pronoun → I | he | she | me | you
- Preposition → from | to | on | near | through

Consider the grammar shown in Table 4.2 and the sentence

Paint the door. (4.7)

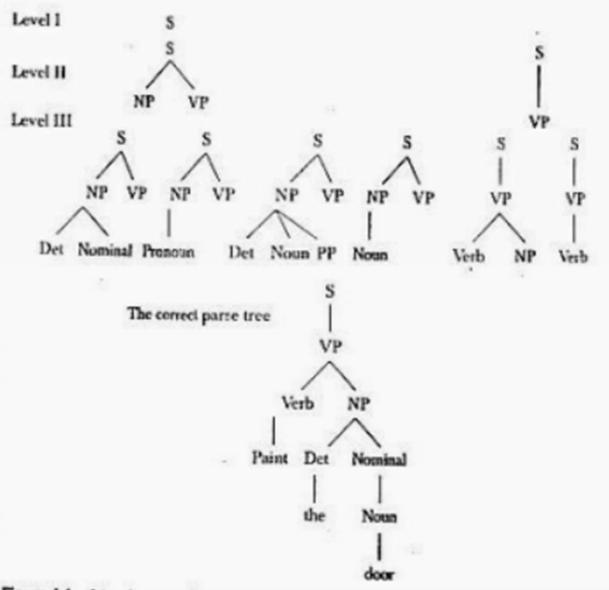


Figure 4.4 A top-down search space

A top-down search begins with the start symbol of the grammar. Thus, the first level (ply) search tree consists of a single node labelled S. The grammar in Table 4.2 has two rules with S on their left hand side. These rules are used to expand the tree, which gives us two partial trees at the second level search, as shown in Figure 4.4. The third level is generated by expanding the non-terminal at the bottom of the search tree in the previous ply. Due to space constraints, only the expansion corresponding to the left-most non-terminals has been shown in the figure. The subsequent steps in the parse are left, as an exercise, to the readers. The correct parse tree shown in Figure 4.4 is obtained by expanding the fifth parse tree of the third level.

Bottom-up Parsing

A bottom-up parser starts with the words in the input sentence and attempts to construct a parse tree in an upward direction towards the root. At each step, the parser looks for rules in the grammar where the right hand side matches some of the portions in the parse tree constructed so far, and reduces it using the left hand side of the production. The parse is considered successful if the parser reduces the tree to the start symbol of the grammar. Figure 4.5 shows some steps carried out by the bottom-up parser for sentence **Paint the door**.

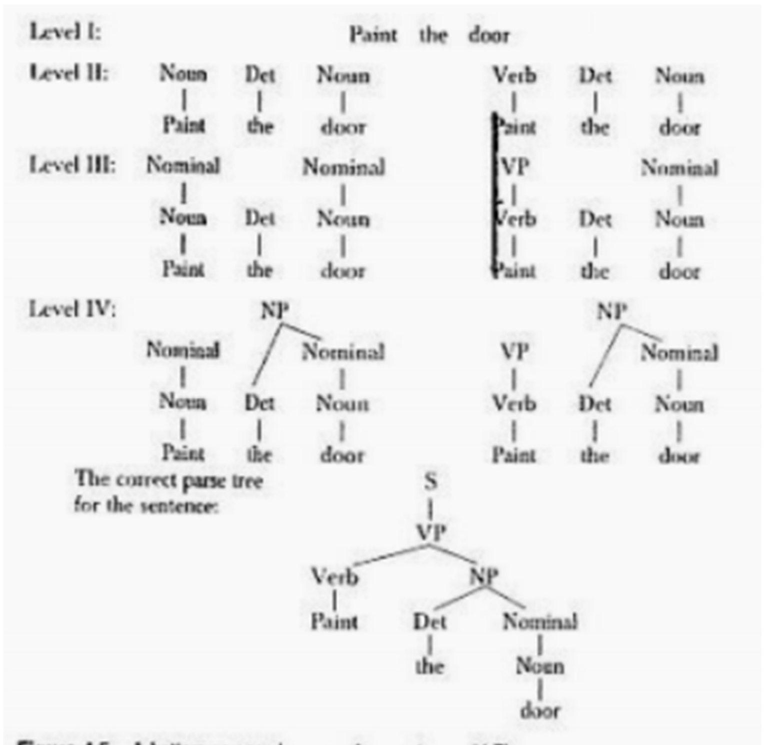


Figure 4.5 A bottom-up search space for sentence (4.7)

Each of these parsing strategies has its advantages and disadvantages. As the top-down search starts generating trees with the start symbol of the grammar, it never wastes time exploring a tree leading to a different root. However, it wastes considerable time exploring S trees that eventually result in words that are inconsistent with the input. This is because a top-down parser generates trees before seeing the input. On the other hand, a bottom-up parser never explores a tree that does not match the input. However, it wastes time generating trees that have no chance of leading to an S-rooted tree. The left branch of the search space in Figure 4.5 that explores a sub-tree assuming *paint* as a noun, is an example of wasted effort. We now present a basic search strategy that uses the top-down method to generate trees and augments it with bottom-up constraints to filter bad parses.

b. listout disadvantages of CFG in NLP

Disadvantages of Context-Free Grammar (CFG) in NLP

Context-Free Grammar (CFG) is a formalism used to describe the syntactic structure of languages using a set of production rules. While CFGs are useful in modeling the hierarchical structure of natural language sentences, they have several limitations when applied to **Natural Language Processing (NLP)**. These limitations arise primarily because

natural languages are not fully context-free, and CFGs fail to capture many aspects of real-world language use.

⚠️ 1. Inability to Handle Context-Sensitive Constructs

Explanation:

CFGs cannot handle constructions that require agreement or context sensitivity, such as **subject-verb agreement**, **gender agreement**, or **pronoun resolution**.

Example:

- "She eats" ✅
- "She eat" ❌

CFG cannot enforce agreement between **subject ("she")** and **verb ("eats")** because it lacks memory of the subject's number.

Impact:

CFG generates both grammatical and ungrammatical forms unless additional constraints are imposed externally.

⚠️ 2. Ambiguity Handling is Weak

Explanation:

CFGs often **overgenerate multiple parse trees** for the same sentence, leading to **syntactic ambiguity**. They do not inherently rank or prefer one parse over another.

Example:

- Sentence: "I saw the man with the telescope."
    - Ambiguity:
        1. I used a telescope to see the man.
        2. The man I saw had a telescope.

Impact:

- CFG does not help in selecting the **semantically correct parse**, requiring statistical or semantic models for disambiguation.

⚠️ 3. No Support for Semantics

Explanation:

CFG models **form** but not **meaning**. It cannot represent or interpret the **semantics** of sentences.

Example:

CFG can generate:

- "Colorless green ideas sleep furiously." (grammatical form, no meaning)

Impact:

- CFG cannot determine sentence **truth conditions**, **thematic roles**, or **logical forms**, which are critical for tasks like question answering or machine translation.

⚠️ 4. Rigid and Complex for Real-World Grammar

Explanation:

Modeling natural language with CFG requires an enormous number of rules to handle exceptions, idioms, and flexibility in syntax.

Example:

- Optional components (e.g., modifiers):
  "The boy [who ran] [quickly]"
  "The boy"

To account for all variations, many rules must be written, leading to **grammar explosion**.

Impact:

- CFG becomes **inefficient** and **unmaintainable** for wide-coverage grammars.

⚠️ 5. Inadequate for Non-Projective Dependencies

Explanation:

CFG assumes **projective trees**, but some syntactic constructions in natural language are **non-projective**, especially in free word order languages (like Hindi, Russian).

Example (in dependency grammar):

- "Usne roti khayi jo maine banayi thi."
  (He ate the bread that I had made.)

CFG fails to represent such **crossing dependencies** naturally.

⚠️ 6. No Probabilistic Information

Explanation:

Standard CFG does not include any mechanism for encoding **probabilities or frequencies** of rule applications.

Impact:

- Cannot capture preferences like:
    - “He saw the dog with a telescope” → telescope is more likely instrument than attachment to dog.
- Hence, **Probabilistic CFG (PCFG)** is introduced as an extension.

⚠️ 7. Lack of Integration with Semantic Roles or Pragmatics

Explanation:

CFG lacks knowledge of **semantic roles**, **intentions**, and **pragmatic context**, which are essential for real-world language understanding.

Impact:

CFG alone cannot handle:

- Ellipsis: “I will go, and you?”
- Anaphora: “John loves his dog.”

c.illustrate espelling and error detection in wordlevel analysis

1.

Spelling Error Detection and Correction
In **computer-based information systems**, especially those involving **text entry** or **automatic recognition systems** (like OCR or speech recognition), **errors in typing and spelling** are a major source of variation between input strings.

Common Typing Errors (80% are single-error misspellings):

1. **Substitution**: Replacing one letter with another (e.g., *cat* → *bat*).

2. **Omission**: Leaving out a letter (e.g., *blue* → *bue*).

3. **Insertion**: Adding an extra letter (e.g., *car* → *caar*).

4. **Transposition**: Switching two adjacent letters (e.g., *form* → *from*).

5. **Reversal errors**: A specific case of transposition where letters are reversed.

A. Typographical Errors

These are **manual errors** made during keyboard typing. They are among the most frequent spelling mistakes. Common subtypes include:

- **Substitution**: One character is incorrectly replaced.
  Example: cat → bat

- **Omission**: A character is unintentionally left out.
  Example: blue → bue

- **Insertion**: An extra character is added mistakenly.
  Example: car → caar

- **Transposition**: Two adjacent characters are switched.
  Example: form → from

B. OCR (Optical Character Recognition) Errors

These occur when printed or handwritten text is digitized using OCR software. Recognition inaccuracies lead to errors like:

- **Character Substitution**: Confusing similar-looking characters.
  Example: O ↔ 0, l ↔ 1, rn ↔ m

- **Omission or Duplication**: Letters skipped or repeated.
  Example: commitee instead of committee

- **Spacing Errors**: Missing or extra spaces.
  Example: bookstore → book store

C. Phonetic (Speech Recognition) Errors

These errors arise when spoken input is transcribed incorrectly due to **phonetic similarity** between words. They are common in speech recognition systems.

- *Example*: their instead of there, to instead of too, no instead of know

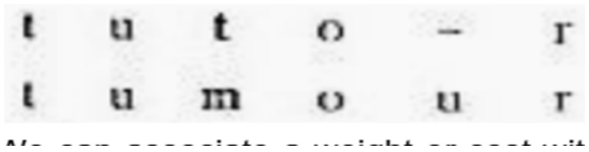
Such errors produce real words, making detection difficult without context.

Minimum Edit Distance

The minimum edit distance is the number of insertions, deletions and substitutions required to change one string to another. For example, the minimum edit distance between tutor and tumor is 2: we substitute ‘m’ for ‘t’ and insert ‘u’ before ‘r’. Edit distance can be represented as a binary function, ed, which maps two strings to their edit distance. Ed is symmetric. For any two strings, s and t, ed(s,t) is always equal to ed(t,s).

Edit distance can be viewed as a string alignment problem. By aligning two strings, we can measure the degree to which they match. There may be more than one possible alignment between two strings.

The alignment shown here, between tutor and tumour, has a distance of 2.



t	u	t	o	–	r
t	u	m	o	u	r

Dynamic Programming algorithms can be quite useful for finding minimum edit distance between two sequences. Dynamic programming refers to a class of algorithms that apply a table-driven approach to solve problems by combining solutions to sub-problems. The dynamic programming algorithm for minimum edit distance is implemented by creating an edit distance matrix.

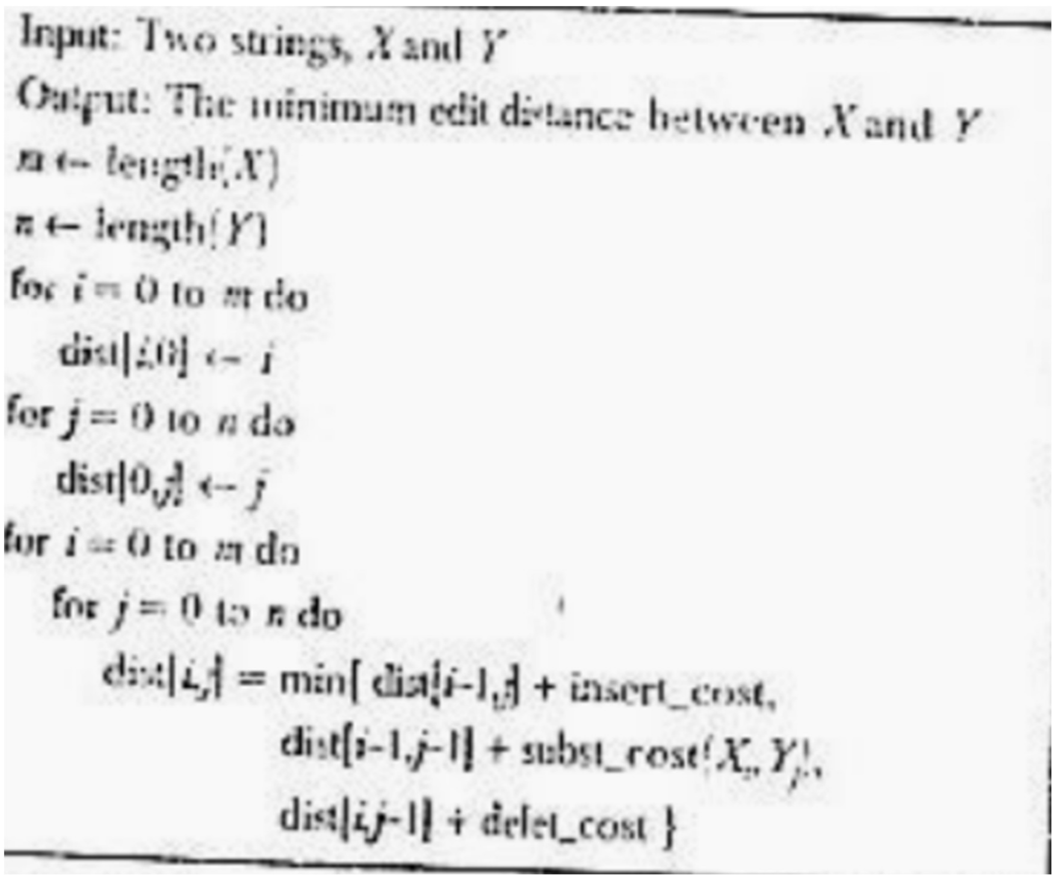
The matrix has one row for each symbol in the source string and one column for each matrix in the target string.

The (i,j)th cell in this matrix represents the distance between the first i character of the source and the first j character of the target string.

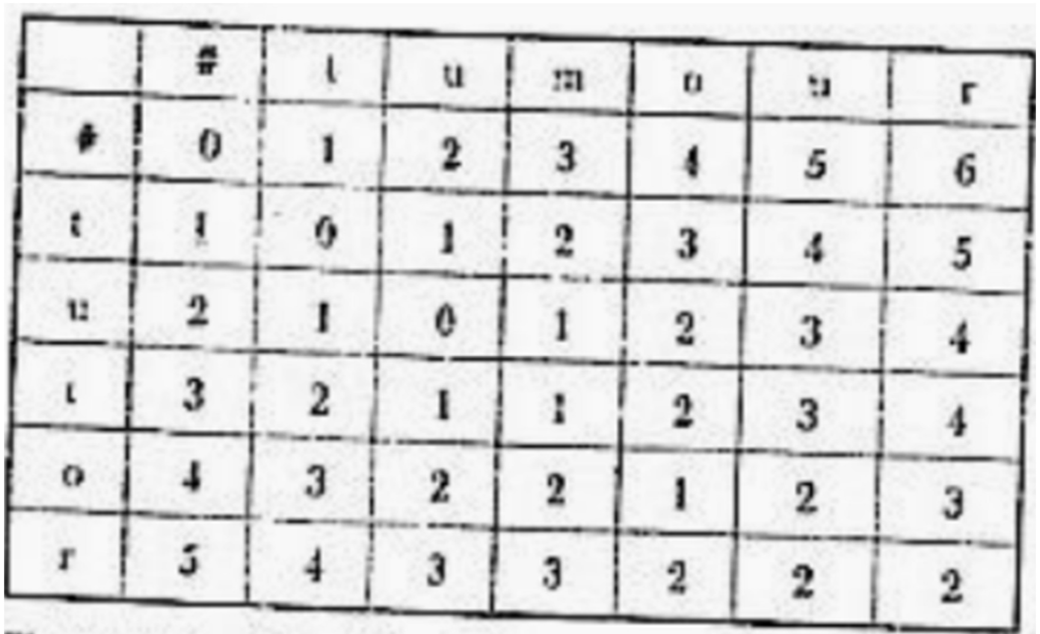
The value in each cell is computed in terms of 3 possible paths.

$$dist[i,j] = \begin{cases} dist[i-1,j] + insert\_cost, \\ dist[i-1,j-1] + subst\_cost[source_i, target_j] \\ dist[i,j-1] + delete\_cost \end{cases}$$

The substitution will be 0 if the ith character in the source mathes with jth character in the target



```
Input: Two strings, X and Y
Output: The minimum edit distance between X and Y
m ← length(X)
n ← length(Y)
for i = 0 to m do
    dist[i,0] ← i
for j = 0 to n do
    dist[0,j] ← j
for i = 0 to m do
    for j = 0 to n do
        dist[i,j] = min{ dist[i-1,j] + insert_cost,
                         dist[i-1,j-1] + subst_cost(X_i, Y_j),
                         dist[i,j-1] + delet_cost }
```



	#	t	u	m	o	u	r
#	0	1	2	3	4	5	6
t	1	0	1	2	3	4	5
u	2	1	0	1	2	3	4
t	3	2	1	1	2	3	4
o	4	3	2	2	1	2	3
r	5	4	3	3	2	2	2

Figure 3.14 Computing minimum edit distance

5.

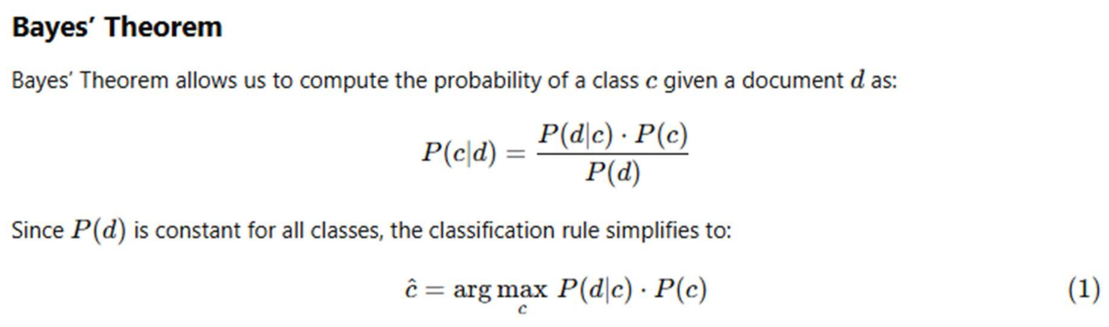
a. Explain working of naïve bayes algorithm

Naive Bayes Classifier for Text Classification

Introduction

The **Naive Bayes classifier** is a probabilistic learning algorithm based on **Bayes' Theorem**. It is widely used in **text classification** tasks such as **spam detection**, **sentiment analysis**, and **topic categorization** due to its simplicity, efficiency, and effectiveness.

The core idea is to compute the **posterior probability** of a document belonging to a particular class, given the words in the document. The classifier then assigns the document to the class with the **highest posterior probability**.



Bayes' Theorem

Bayes' Theorem allows us to compute the probability of a class c given a document d as:

$$P(c|d) = \frac{P(d|c) \cdot P(c)}{P(d)}$$

Since $P(d)$ is constant for all classes, the classification rule simplifies to:

$$\hat{c} = \arg\max_{c} P(d|c) \cdot P(c) \tag{1}$$

Feature Representation: Bag-of-Words Model

In text classification, documents are typically represented using the **bag-of-words (BoW) model**. This model ignores the order of words and treats each document as an unordered multiset (bag) of words.

For example, the sentence:

"I love this movie"

is represented simply as a frequency distribution:

- I: 1
- love: 1
- this: 1

- movie: 1

Let a document d be represented by features $f_1, f_2, ..., f_n$, where each feature corresponds to the presence or count of a word in the vocabulary.

Naive Bayes Assumptions

1. Bag-of-Words Assumption

The classifier does not consider the **order or position** of words. It only considers the **frequency** of each word in the document.

2. Conditional Independence Assumption

It assumes that **features (words) are conditionally independent given the class**. That is, the probability of seeing one word in a document is independent of seeing any other word, given the class.

$$P(f_1, f_2, ..., f_n|c) = \prod_{i-1}^{n} P(f_i|c) \qquad (2)$$

Derivation of the Final Classification Equation

Using Bayes' Theorem and the independence assumption, we derive the final classification formula.

From Equation (1):

$$\hat{c} = \arg\max_c \; P(d|c) \cdot P(c)$$

Using the conditional independence assumption (Equation 2):

$$P(d|c) = \prod_{i-1}^{n} P(f_i|c)$$

Substitute into Equation (1):

$$\hat{c} = \arg\max_c \; P(c) \cdot \prod_{i-1}^{n} P(f_i|c) \qquad (3)$$

For computational efficiency and to prevent **numerical underflow**, we work in **log-space**:

$$\hat{c} = \arg\max_c \; \log P(c) + \sum_{i-1}^{n} \log P(f_i|c) \qquad (4)$$

Equation (4) is the **final equation used in Naive Bayes text classification**, which computes the log posterior probability for each class and assigns the document to the class with the highest value.

$$\hat{c} = \arg\max_{c} \; P(d|c) \cdot P(c)$$

Using the conditional independence assumption (Equation 2):

$$P(d|c) = \prod_{i-1}^{n} P(f_i|c)$$

Substitute into Equation (1):

$$\hat{c} = \arg\max_{c} \; P(c) \cdot \prod_{i-1}^{n} P(f_i|c) \qquad (3)$$

For computational efficiency and to prevent **numerical underflow**, we work in **log-space**:

$$\hat{c} = \arg\max_{c} \; \log P(c) + \sum_{i-1}^{n} \log P(f_i|c) \qquad (4)$$

Equation (4) is the **final equation used in Naive Bayes text classification**, which computes the log posterior probability for each class and assigns the document to the class with the highest value.

$$\hat{c} = \arg\max_{c} \; P(d|c) \cdot P(c)$$

Using the conditional independence assumption (Equation 2):

$$P(d|c) = \prod_{i-1}^{n} P(f_i|c)$$

Substitute into Equation (1):

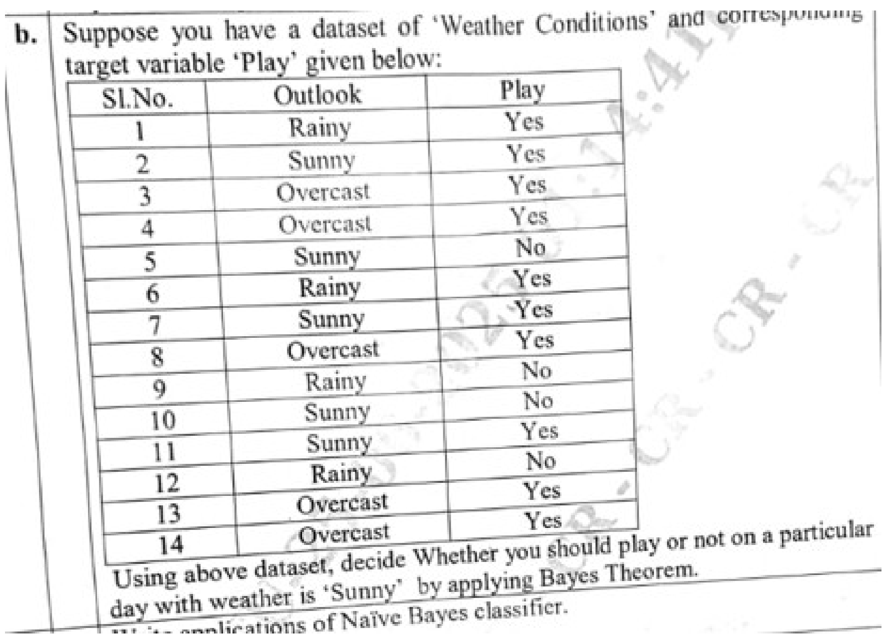
$$\hat{c} = \arg\max_{c} \; P(c) \cdot \prod_{i-1}^{n} P(f_i|c) \qquad (3)$$

For computational efficiency and to prevent **numerical underflow**, we work in **log-space**:

$$\hat{c} = \arg\max_{c} \; \log P(c) + \sum_{i-1}^{n} \log P(f_i|c) \qquad (4)$$

Equation (4) is the **final equation used in Naive Bayes text classification**, which computes the log posterior probability for each class and assigns the document to the class with the highest value.

Equation (4) is the **final equation used in Naive Bayes text classification**, which computes the log



b. Suppose you have a dataset of 'Weather Conditions' and corresponding target variable 'Play' given below:

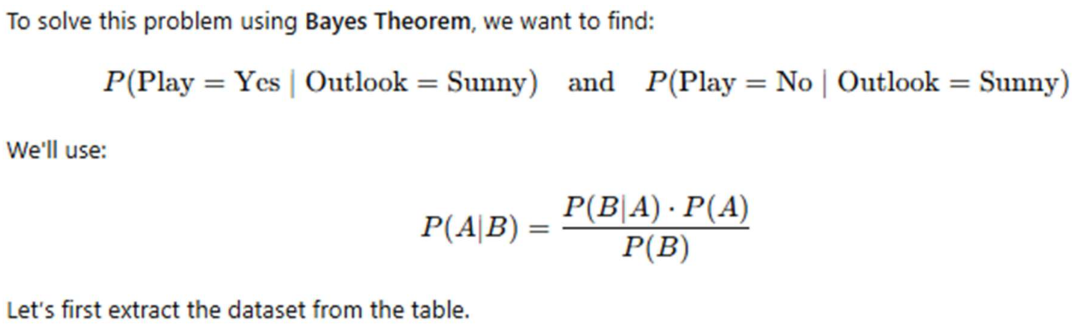
Sl.No.	Outlook	Play
1	Rainy	Yes
2	Sunny	Yes
3	Overcast	Yes
4	Overcast	Yes
5	Sunny	No
6	Rainy	Yes
7	Sunny	Yes
8	Overcast	Yes
9	Rainy	No
10	Sunny	No
11	Sunny	Yes
12	Rainy	No
13	Overcast	Yes
14	Overcast	Yes

Using above dataset, decide Whether you should play or not on a particular day with weather is 'Sunny' by applying Bayes Theorem.

[illegible] applications of Naïve Bayes classifier.

b.

To solve this problem using **Bayes Theorem**, we want to find:



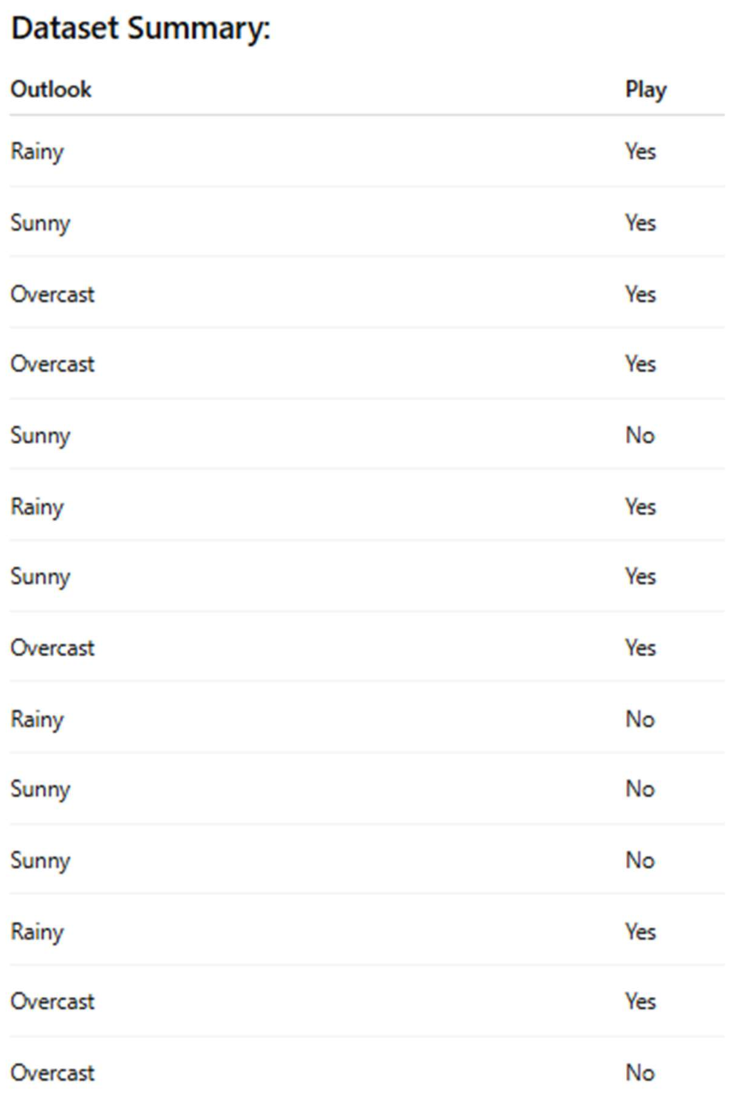
To solve this problem using **Bayes Theorem**, we want to find:

$$P(\text{Play} = \text{Yes} \mid \text{Outlook} = \text{Sunny}) \quad \text{and} \quad P(\text{Play} = \text{No} \mid \text{Outlook} = \text{Sunny})$$

We'll use:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

Let's first extract the dataset from the table.

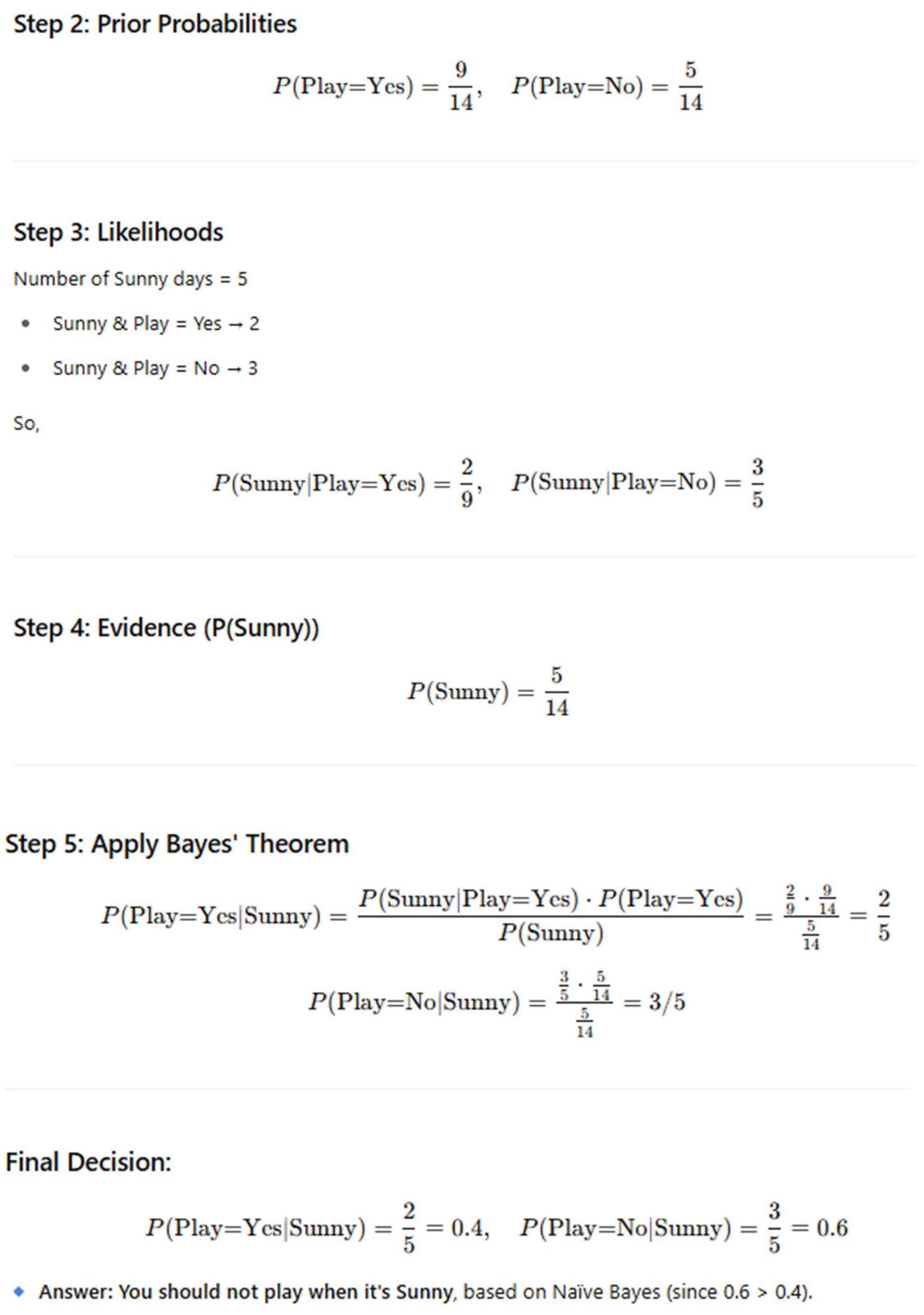


Dataset Summary:

Outlook	Play
Rainy	Yes
Sunny	Yes
Overcast	Yes
Overcast	Yes
Sunny	No
Rainy	Yes
Sunny	Yes
Overcast	Yes
Rainy	No
Sunny	No
Sunny	No
Rainy	Yes
Overcast	Yes
Overcast	No

Step 1: Count Totals

- Total samples = 14
- Play = Yes → 9
- Play = No → 5



Step 2: Prior Probabilities

$$P(\text{Play=Yes}) = \frac{9}{14}, \quad P(\text{Play=No}) = \frac{5}{14}$$

Step 3: Likelihoods

Number of Sunny days = 5

- Sunny & Play = Yes → 2
- Sunny & Play = No → 3

So,

$$P(\text{Sunny}|\text{Play=Yes}) = \frac{2}{9}, \quad P(\text{Sunny}|\text{Play=No}) = \frac{3}{5}$$

Step 4: Evidence (P(Sunny))

$$P(\text{Sunny}) = \frac{5}{14}$$

Step 5: Apply Bayes' Theorem

$$P(\text{Play=Yes}|\text{Sunny}) = \frac{P(\text{Sunny}|\text{Play=Yes}) \cdot P(\text{Play=Yes})}{P(\text{Sunny})} = \frac{\frac{2}{9} \cdot \frac{9}{14}}{\frac{5}{14}} = \frac{2}{5}$$

$$P(\text{Play=No}|\text{Sunny}) = \frac{\frac{3}{5} \cdot \frac{5}{14}}{\frac{5}{14}} = 3/5$$

Final Decision:

$$P(\text{Play=Yes}|\text{Sunny}) = \frac{2}{5} = 0.4, \quad P(\text{Play=No}|\text{Sunny}) = \frac{3}{5} = 0.6$$

- **Answer: You should not play when it's Sunny**, based on Naïve Bayes (since 0.6 > 0.4).

c.write the applications of Naïve bayes classifier.

Applications of Naive Bayes: Spam Detection and Language Identification

The Naive Bayes classifier, due to its simplicity, speed, and robustness with high-dimensional data, has been successfully applied to a variety of **text classification tasks**. Two prominent applications include **spam detection** and **language identification**, where it leverages statistical patterns in text features.

1. Naive Bayes in Spam Detection

Objective

Spam detection is a **binary classification** task where each email is classified as either:

- **Spam** (unsolicited, often harmful messages), or
- **Not Spam** (legitimate email)

Naive Bayes Approach

- Each email is represented as a **bag-of-words**.
- The classifier learns the probability of a message being spam based on the **presence or frequency of certain words or patterns**.

$$\hat{c} = \arg \max_{c \in \{\text{spam,ham}\}} P(c) \cdot \prod_i P(w_i|c)$$

Example Features Used

- **Word/Phrase Features:**
  - Phrases like *"winner"*, *"free money"*, *"100% guaranteed"*
  - Words in **all caps** (e.g., *URGENT*)
- **Character Patterns:**
  - Use of **excessive punctuation:** `!!!`, `$$$`
  - Unusual spellings: *v1agra, pr!ze*
- **HTML/Structural Features:**
  - HTML-only emails
  - Low **text-to-image ratio**
- **Metadata Features:**
  - Unusual **sender address**
  - Forged headers or domains

Example

A spam email might contain:

"Congratulations! You are a lucky winner. CLICK HERE to claim your prize FREE of charge."

Such emails contain **tokens strongly associated with the spam class**, and Naive Bayes assigns high posterior probability to the spam label.

2. Naive Bayes in Language Identification

Objective

Language identification is a **multi-class classification** task, where the goal is to determine the **language** of a given text sample.

Naive Bayes Approach

- Instead of using whole words, the model often relies on **character-level n-grams** (subword features).

- Each language is modeled as a **class**, and the classifier determines which language model best explains the character patterns in the input.

$$\hat{c} = \arg \max_{c \in \{\text{languages}\}} P(c) \cdot \prod_i P(n\text{-gram}_i | c)$$

Example Features Used

- **Character n-grams (1–4 grams):**
  - *"the"*, *"ent"*, *"und"*, *"que"*, etc.
  - Language-specific patterns like *"le"* (French), *"der"* (German), *"na"* (Hindi)
- **Common words or stopwords:**
  - *"el"*, *"y"* (Spanish), *"and"*, *"the"* (English)
- **Punctuation and diacritics:**
  - Use of characters like *é, ñ, ç* indicating Romance languages
- **Unicode ranges:**
  - Scripts like Devanagari, Cyrillic, Arabic can be language-specific

Example

Given the text:

"Das ist ein guter Tag."
Character trigrams such as *"das"*, *"ist"*, *"ein"*, *"gut"*, *"tag"* are more likely under the **German** language model.

6.

a. illustrate optimizing for sentimental analysis

4.4 Optimizing for Sentiment Analysis

Binary Naive Bayes

Instead of using raw frequencies, we often use **binary features** indicating word presence. This reduces bias introduced by repeated terms and increases robustness in sentiment classification.

Sentiment Lexicons

Lexicons are curated lists of words annotated with their sentiment polarity.

- **General Inquirer**: Annotates words with dozens of labels including “positive”, “negative”, “strong”, etc.
- **Opinion Lexicon**: Divides words into positive (e.g., “love”, “great”) and negative (e.g., “bad”, “terrible”).

These lexicons can be used to:

- Initialize feature weights
- Enhance feature selection
- Interpret models more transparently

b. How can you use naïve bayes for variety of text classification

Spam Detection using Naïve Bayes

In spam classification, the **Naïve Bayes classifier** uses the presence or frequency of specific words to determine if a message is spam or not.

- Certain words like **“free”**, **“win”**, **“credit”**, **“offer”**, and **“cash”** are statistically more likely to appear in spam emails.
- Naïve Bayes computes:

$$P(\text{Spam}|\text{Words}) \propto P(\text{Words}|\text{Spam}) \cdot P(\text{Spam})$$

- The model is trained on **labeled corpora** (datasets with messages marked as spam or ham) to learn the probability distribution of words in each class.
- During prediction, it uses the learned probabilities to classify new, unseen messages as **spam** or **ham**.

Use case: Email filtering, SMS spam detection.

Language Identification using Naïve Bayes

In this task, the goal is to identify the **language of a given text snippet**, especially when the text is short.

- Instead of using full words, **character-level n-grams (e.g., trigrams like "the", "qui", "ent")** are used.
- Naïve Bayes computes the probability of the character n-grams appearing in different language models.
- It calculates:

$$P(\text{Language}|\text{Trigrams}) \propto P(\text{Trigrams}|\text{Language}) \cdot P(\text{Language})$$

- The model predicts the language whose probability is highest for the given set of trigrams.

Feature Selection in Naïve Bayes

In text classification, there are usually **thousands of possible words/features**, most of which may not help the classification task.

To improve efficiency and accuracy, **feature selection techniques** are used to retain only the most **informative features**:

1. Mutual Information (MI)

- Measures how much information a feature (word) contributes to making the correct classification.
- Higher MI indicates the word is **strongly associated** with a particular class.

2. Chi-square Test (χ^2 test)

- Measures the **independence** between the feature and the class label.
- A high chi-square value indicates a strong association between the feature and the class.

3. Information Gain (IG)

- Measures the **reduction in entropy** when a feature is used to split the data.
- High information gain means the feature helps to significantly **reduce uncertainty** in classification.

Use case: Text classification, sentiment analysis, topic modeling.

c.explain different types of language modelling using Naïve bayes

1.

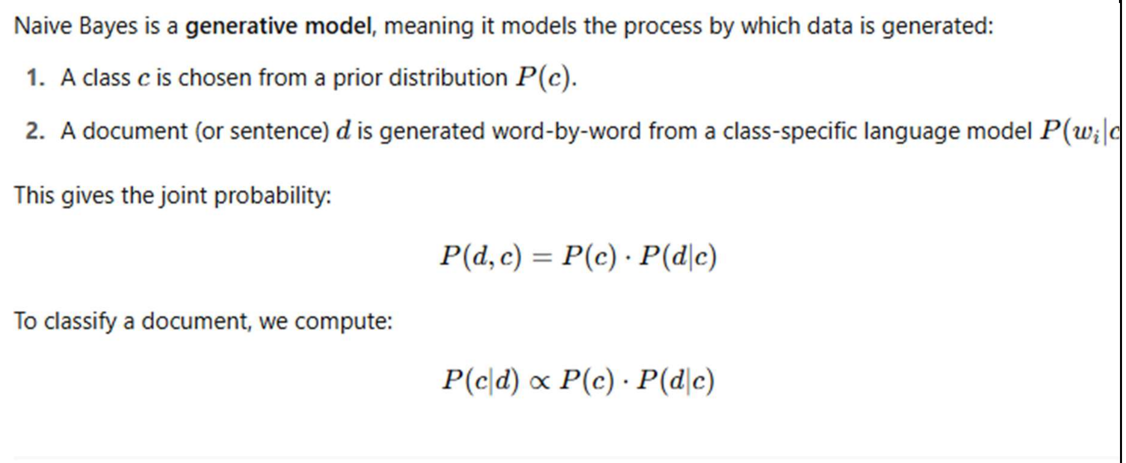
Naive Bayes as a Language Model

1. Introduction

The **Naive Bayes classifier**, though commonly viewed as a tool for text classification, can also be interpreted as a **language model**—specifically, a **class-conditional unigram language model**. In this view, Naive Bayes estimates the likelihood of a sentence by assuming that each word is generated independently given a particular class.

This interpretation is especially useful in text classification tasks such as **sentiment analysis**, where we aim to compare how likely a sentence is under different language models (e.g., one for positive sentiment and one for negative sentiment).

2. Generative Interpretation of Naive Bayes



Naive Bayes is a **generative model**, meaning it models the process by which data is generated:

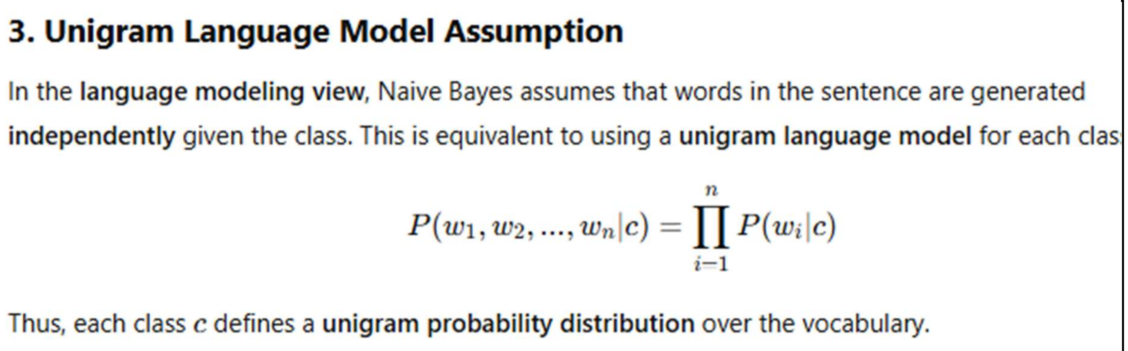
1. A class c is chosen from a prior distribution $P(c)$.
2. A document (or sentence) d is generated word-by-word from a class-specific language model $P(w_i|c$

This gives the joint probability:

$$P(d, c) = P(c) \cdot P(d|c)$$

To classify a document, we compute:

$$P(c|d) \propto P(c) \cdot P(d|c)$$

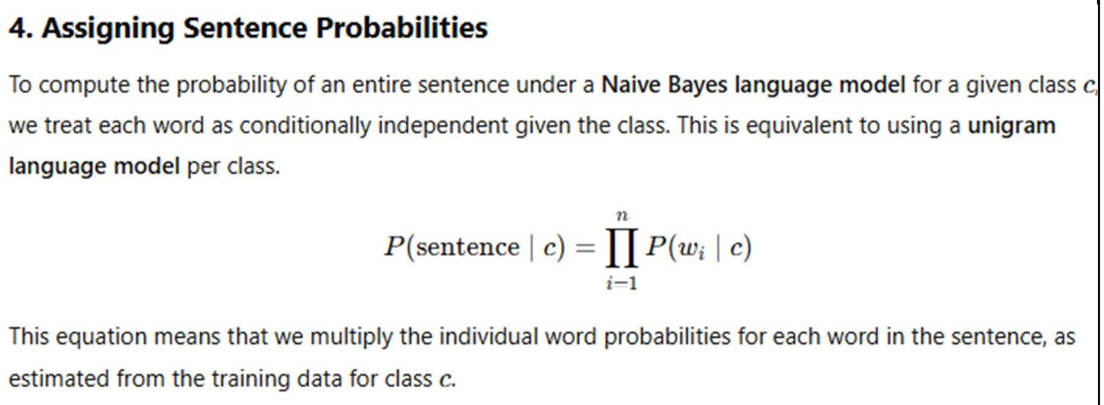


3. Unigram Language Model Assumption

In the **language modeling view**, Naive Bayes assumes that words in the sentence are generated **independently** given the class. This is equivalent to using a **unigram language model** for each clas

$$P(w_1, w_2, ..., w_n|c) = \prod_{i=1}^{n} P(w_i|c)$$

Thus, each class c defines a **unigram probability distribution** over the vocabulary.

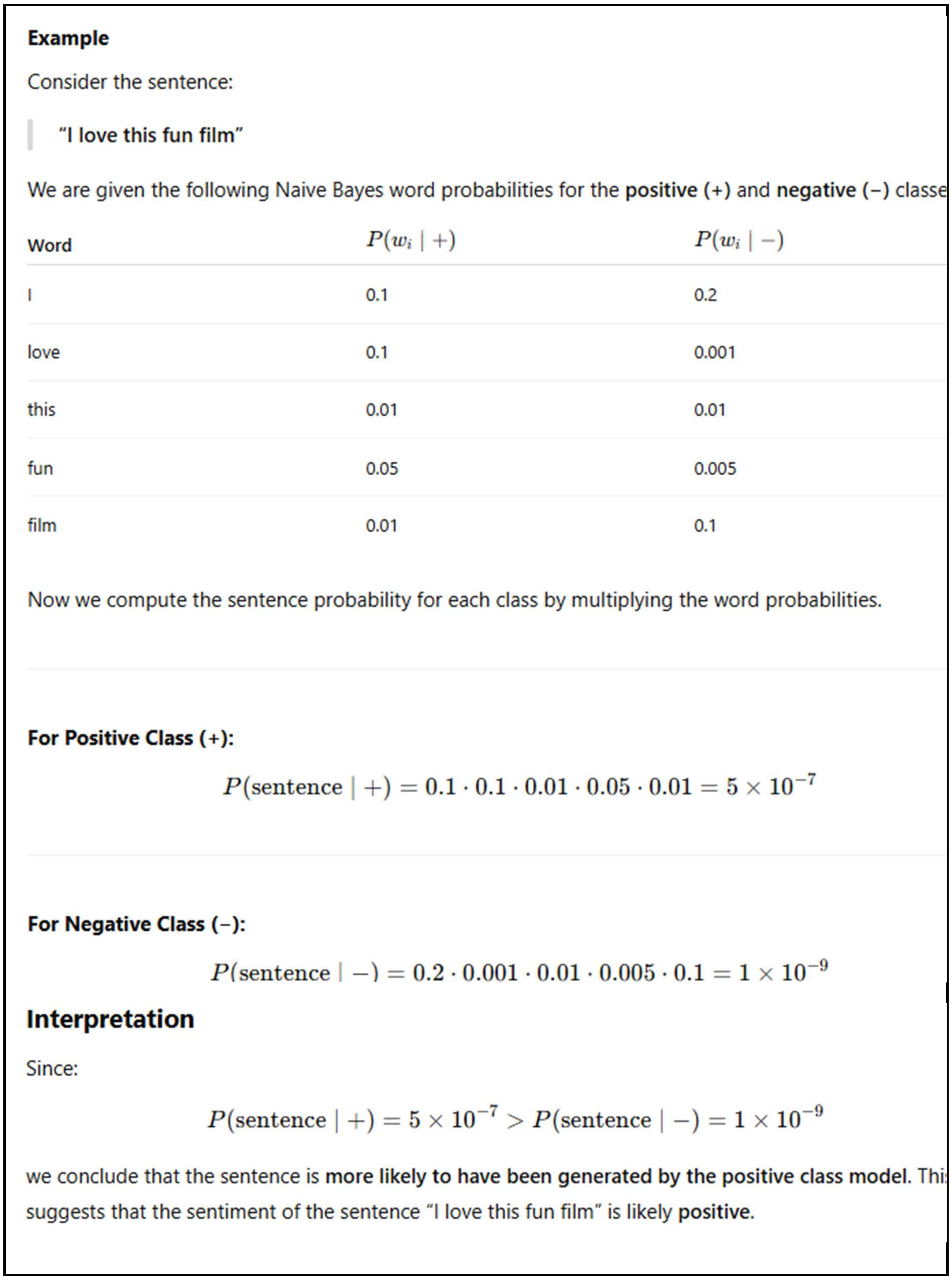


4. Assigning Sentence Probabilities

To compute the probability of an entire sentence under a **Naive Bayes language model** for a given class c, we treat each word as conditionally independent given the class. This is equivalent to using a **unigram language model** per class.

$$P(\text{sentence} \mid c) = \prod_{i=1}^{n} P(w_i \mid c)$$

This equation means that we multiply the individual word probabilities for each word in the sentence, as estimated from the training data for class c.



Example

Consider the sentence:

> **"I love this fun film"**

We are given the following Naive Bayes word probabilities for the **positive (+)** and **negative (−)** classe

Word	$P(w_i \mid +)$	$P(w_i \mid -)$
I	0.1	0.2
love	0.1	0.001
this	0.01	0.01
fun	0.05	0.005
film	0.01	0.1

Now we compute the sentence probability for each class by multiplying the word probabilities.

For Positive Class (+):

$$P(\text{sentence} \mid +) = 0.1 \cdot 0.1 \cdot 0.01 \cdot 0.05 \cdot 0.01 = 5 \times 10^{-7}$$

For Negative Class (−):

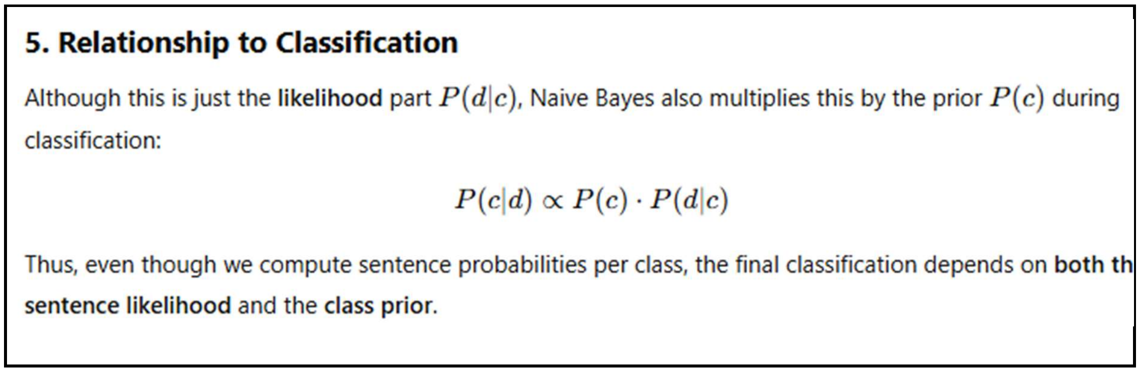
$$P(\text{sentence} \mid -) = 0.2 \cdot 0.001 \cdot 0.01 \cdot 0.005 \cdot 0.1 = 1 \times 10^{-9}$$

Interpretation

Since:

$$P(\text{sentence} \mid +) = 5 \times 10^{-7} > P(\text{sentence} \mid -) = 1 \times 10^{-9}$$

we conclude that the sentence is **more likely to have been generated by the positive class model**. Thi suggests that the sentiment of the sentence "I love this fun film" is likely **positive**.



5. Relationship to Classification

Although this is just the **likelihood** part $P(d|c)$, Naive Bayes also multiplies this by the prior $P(c)$ during classification:

$$P(c|d) \propto P(c) \cdot P(d|c)$$

Thus, even though we compute sentence probabilities per class, the final classification depends on **both th sentence likelihood** and the **class prior**.

7.

a. Explain design features of IR

Design Features in Information Retrieval

Information Retrieval (IR) systems aim to efficiently locate relevant documents or information from large datasets. Several key design features play a crucial role in enhancing the performance, efficiency, and relevance of such systems. These include **Indexing**, **Stop Word Elimination**, **Stemming**, and understanding word distributions through **Zipf's Law**.

1. Indexing

Indexing is the process of organizing data to enable rapid search and retrieval. In IR, an **inverted index** is commonly used. This structure maps each term in the document collection to a list of documents (or document IDs) where that term occurs. It typically includes additional information like term frequency, position, and weight (e.g., TF-IDF score).
Efficient indexing allows the system to avoid scanning all documents for every query, dramatically reducing search time and computational cost. Index construction involves tokenizing documents, normalizing text, and storing index entries in a sorted and optimized structure, often with compression techniques to reduce storage requirements.

2. Eliminating Stop Words

Stop words are extremely common words that appear in almost every document, such as "the", "is", "at", "which", "on", and "and". These words usually add little value to understanding the main content or differentiating between documents.
Removing stop words reduces the size of the index, speeds up the search process, and minimizes noise in results. However, careful handling is required because some stop words may be semantically important depending on the domain (e.g., "to be or not to be" in literature, or "in" in legal texts). Most IR systems use a predefined stop word list, though it can be customized based on corpus analysis.

3. Stemming

Stemming is a form of linguistic normalization used to reduce related words to a common base or root form. For example:

- **"connect", "connected", "connection", "connecting" → "connect"**

Stemming improves **recall** in IR systems by ensuring that different inflected or derived forms of a word are matched to the same root term in the index. This is particularly important in languages with rich morphology.
Common stemming algorithms include:

- **Porter Stemmer**: Lightweight and widely used, based on heuristic rules.

- **Snowball Stemmer**: An improvement over Porter, supporting multiple languages.

- **Lancaster Stemmer**: More aggressive but sometimes over-stems words.

Stemming is different from **lemmatization**, which uses vocabulary and grammar rules to derive the base form.

4. Zipf's Law

Zipf's Law is a statistical principle that describes the frequency distribution of words in natural language corpora. It states that the frequency *f* of any word is inversely proportional to its rank *r*:

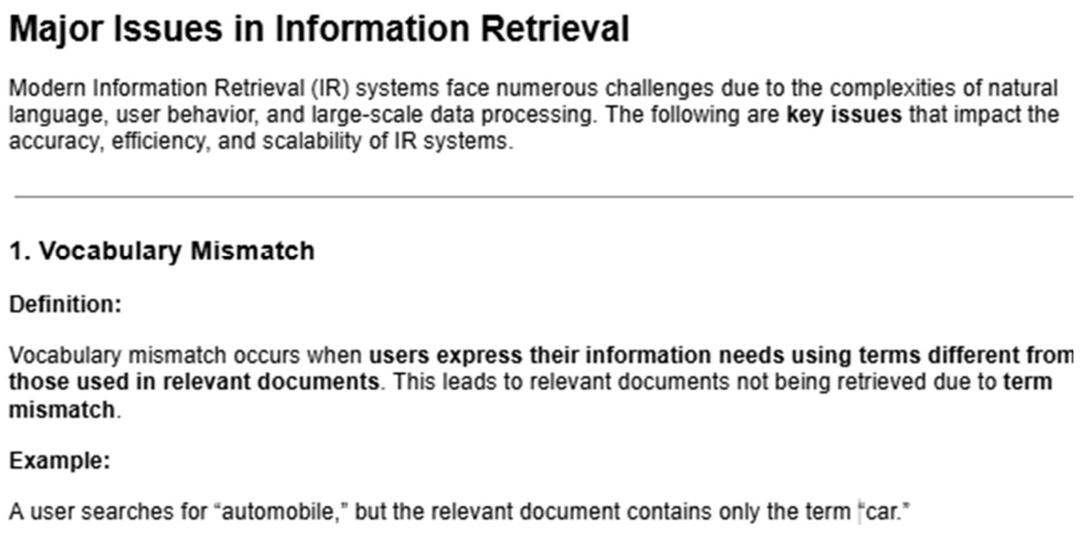
$f \propto 1/r$

This means that the most frequent word occurs roughly twice as often as the second most frequent word, three times as often as the third, and so on.
For example, in English corpora, words like "the", "of", "and", and "to" dominate the frequency list. Meanwhile, the majority of words occur rarely (called the "long tail").
In IR, Zipf's Law justifies:

- Stop word elimination (high-frequency terms contribute little to relevance)
- TF-IDF weighting (rare terms are more informative)
- Optimizing index structures for space and search

Understanding this law helps in designing efficient indexing and retrieval strategies that focus on the more informative, lower-frequency words.

b.Mention major issues in IR.



Major Issues in Information Retrieval

Modern Information Retrieval (IR) systems face numerous challenges due to the complexities of natural language, user behavior, and large-scale data processing. The following are **key issues** that impact the accuracy, efficiency, and scalability of IR systems.

---

1. Vocabulary Mismatch

Definition:

Vocabulary mismatch occurs when **users express their information needs using terms different from those used in relevant documents**. This leads to relevant documents not being retrieved due to **term mismatch**.

Example:

A user searches for “automobile,” but the relevant document contains only the term “car.”

→ The system fails to retrieve the document despite it being semantically relevant.

Strategies to Handle:

- **Query Expansion**:
  - Use **thesauri** or **WordNet** to include synonyms and semantically related terms.
  - Example: Expand “automobile” to include “car,” “vehicle,” etc.
- **Relevance Feedback**:
  - User marks relevant documents; system reformulates the query using new terms found in them.
- **Latent Semantic Indexing (LSI)**:
  - Projects terms and documents into a **semantic space** that captures **latent relationships** among words.

---

2. Polysemy and Ambiguity

Definition:

Polysemy refers to words that have **multiple meanings** (e.g., “bank” = riverbank or financial institution).
Ambiguity arises when the system **cannot determine the correct sense** of a query term.

Impact:

- A query like “interest rate at the bank” may retrieve documents about riverbanks if disambiguation fails.

Strategies to Handle:

- **Word Sense Disambiguation (WSD)**:
  - Use resources like **WordNet** to determine the intended sense based on context.
- **Contextual Language Models**:
  - Employ models like **BERT**, which use sentence-level context to resolve ambiguity.
- **Part-of-Speech Tagging and Named Entity Recognition (NER)**:
  - Clarifies meaning by analyzing syntactic and semantic roles.

3. Scalability and Performance

Definition:

With the explosive growth of web and digital content, IR systems must **scale** to manage **millions or billions of documents** without degrading performance.

Challenges:

- Real-time indexing and retrieval.
- Memory and storage efficiency.
- Distributed system performance and load balancing.

Strategies to Handle:

- **Inverted Index Structures**:
    - Efficiently map terms to documents.
- **Distributed and Parallel Processing**:
    - Use frameworks like **MapReduce**, **Apache Lucene**, or **Elasticsearch**.
- **Incremental Indexing**:
    - Updates index without rebuilding the entire structure.
- **Compression Techniques**:
    - Reduce index size using delta encoding, block compression, etc.

---

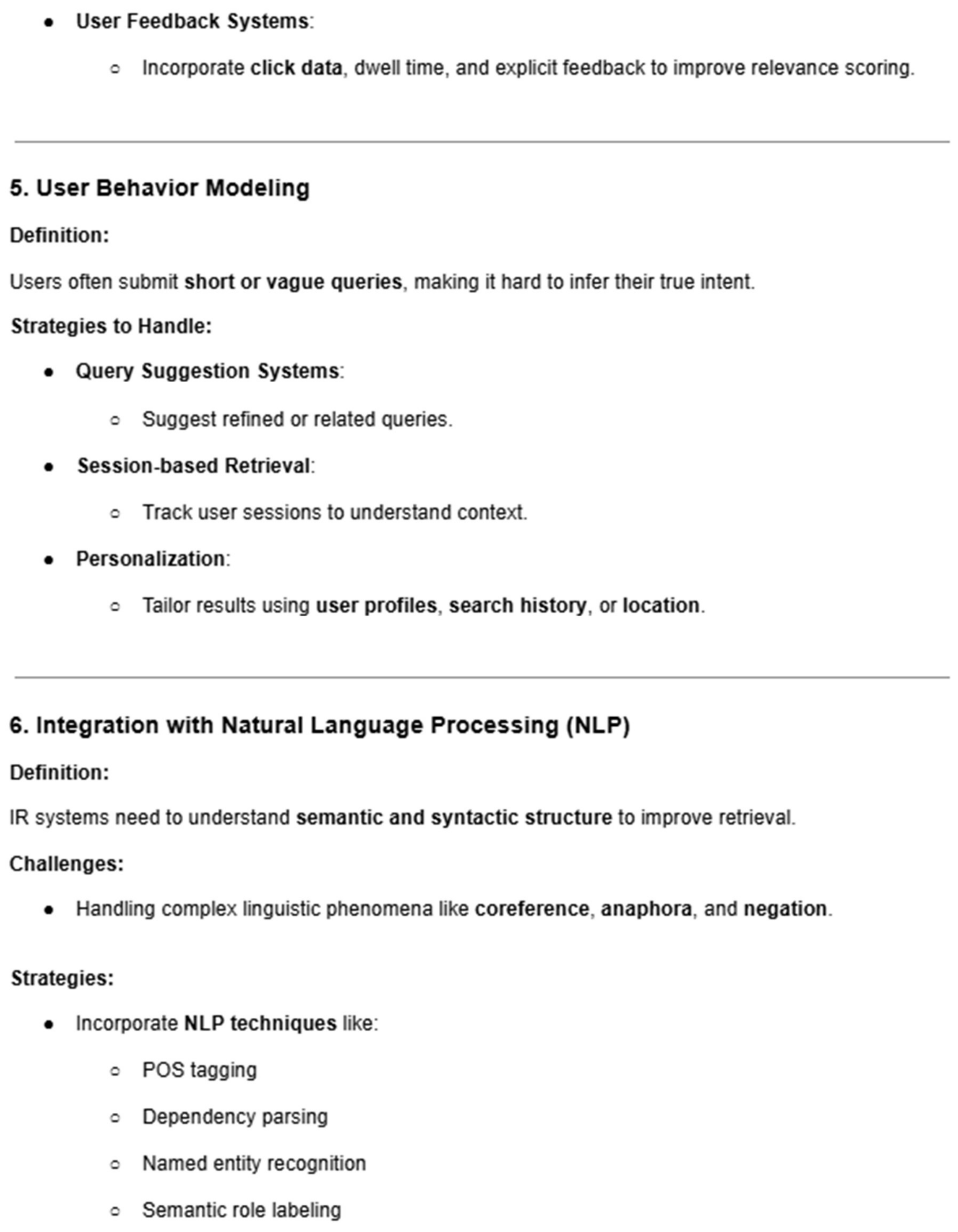
4. Evaluation and Relevance

Definition:

Assessing **retrieval effectiveness** is difficult due to the **subjective nature of relevance**.

Strategies to Handle:

- Use **standard metrics** like:
    - Precision, Recall, F1-Score, MAP, NDCG.
- **Benchmark datasets**:
    - TREC, LETOR collections.



- **User Feedback Systems**:
    - Incorporate **click data**, dwell time, and explicit feedback to improve relevance scoring.

---

5. User Behavior Modeling

Definition:

Users often submit **short or vague queries**, making it hard to infer their true intent.

Strategies to Handle:

- **Query Suggestion Systems**:
    - Suggest refined or related queries.
- **Session-based Retrieval**:
    - Track user sessions to understand context.
- **Personalization**:
    - Tailor results using **user profiles**, **search history**, or **location**.

---

6. Integration with Natural Language Processing (NLP)

Definition:

IR systems need to understand **semantic and syntactic structure** to improve retrieval.

Challenges:

- Handling complex linguistic phenomena like **coreference**, **anaphora**, and **negation**.

Strategies:

- Incorporate **NLP techniques** like:
    - POS tagging
    - Dependency parsing
    - Named entity recognition
    - Semantic role labeling

c.how does stemming affects performance of an IR system

Stemming is the process of reducing inflected or derived words to their base or root form, known as the **stem**. For example, words like *“running”*, *“runs”*, and *“ran”* may all be reduced to the stem *“run”*.

Impact on IR System Performance:

1. **Improved Recall:**

   - Stemming increases **recall** by grouping different forms of a word under the same root.
   - This helps retrieve more relevant documents that use various morphological forms of the same word.
   - Example: A query for *"connect"* will also match documents containing *"connected"*, *"connection"*, or *"connecting"*.

2. **Reduced Index Size:**
   - By conflating multiple word forms to a single stem, the number of distinct terms in the index is reduced.
   - This leads to a **smaller and more compact inverted index**, improving storage efficiency.
3. **Possible Decrease in Precision:**
   - Stemming can sometimes overgeneralize, conflating semantically different words with the same stem (known as **overstemming**).
   - Example: *"universe"* and *"university"* might both be reduced to *"univers"*, which can decrease **precision** by retrieving unrelated documents.
4. **Faster Retrieval:**
   - With fewer terms in the index, **query processing becomes faster**, enhancing overall system responsiveness.
5. **Language Dependency:**
   - The effectiveness of stemming depends on the **morphological structure of the language** and the stemming algorithm used (e.g., Porter Stemmer, Snowball Stemmer).

8.

a. Explain wordnet and list the applications of wordnet

WordNet: A Lexical Database for English

1. Introduction

WordNet is a **lexical knowledge base** of English developed at **Princeton University** under the leadership of **George A. Miller**. It combines the structure of a **dictionary** with the **semantic relations** of a **thesaurus**. WordNet is widely used in **Natural Language Processing (NLP)** and **Information Retrieval (IR)** tasks for word sense understanding, semantic reasoning, and linguistic resource enrichment.

---

2. Structure of WordNet

Unlike traditional dictionaries, WordNet groups words into sets of **cognitive synonyms**, known as **synsets**, and organizes them using **semantic relations**.

2.1 Synsets (Synonym Sets)

- A **synset** is a set of **synonymous words or phrases** that express the same concept.
- Each synset represents a **distinct meaning** or **sense** of a word.
- Synsets are accompanied by:
    - A **gloss** (a dictionary-style definition),
    - **Usage examples**.

Example:

Word: **"bank"**

Sense	Synset	Gloss
1	{bank, depository, financial institution}	An institution where people deposit money

2	{bank, riverbank}	The land alongside a river

2.2 Semantic Relations in WordNet

WordNet organizes synsets through various **semantic relationships**, which are useful for semantic analysis, query expansion, and disambiguation.

---

a) Hypernym / Hyponym

- **Hypernym**: A more **general** concept (superclass).
- **Hyponym**: A more **specific** concept (subclass).

Example:

- *Vehicle* is a **hypernym** of *car*, *bus*, and *bicycle*.
- *Car* is a **hyponym** of *vehicle*.

---

b) Meronym / Holonym

- **Meronym**: A part of something.
- **Holonym**: The whole to which parts belong.

Example:

- *Wheel* is a **meronym** of *car*.
- *Car* is a **holonym** of *wheel*.

---

c) Troponym (for verbs)

- Specifies **manner** or **specific way** of performing an action.

Example:

- *To whisper* is a **troponym** of *to speak*.
- *To sprint* is a troponym of *to run*.

d) Antonym

- Expresses **opposite meaning** between two words.

  Example:

- *Hot ↔ Cold*
- *Buy ↔ Sell*

---

e) Entailment (verbs)

- If one verb **entails** another, the first action presupposes the second.

  Example:

- *Snore* entails *sleep*.
- If someone *snores*, they must be *sleeping*.

---

f) Similar-to (adjectives)

- Adjectives in WordNet may be linked by “similar to” relationships instead of hierarchy.

---

3. Applications of WordNet

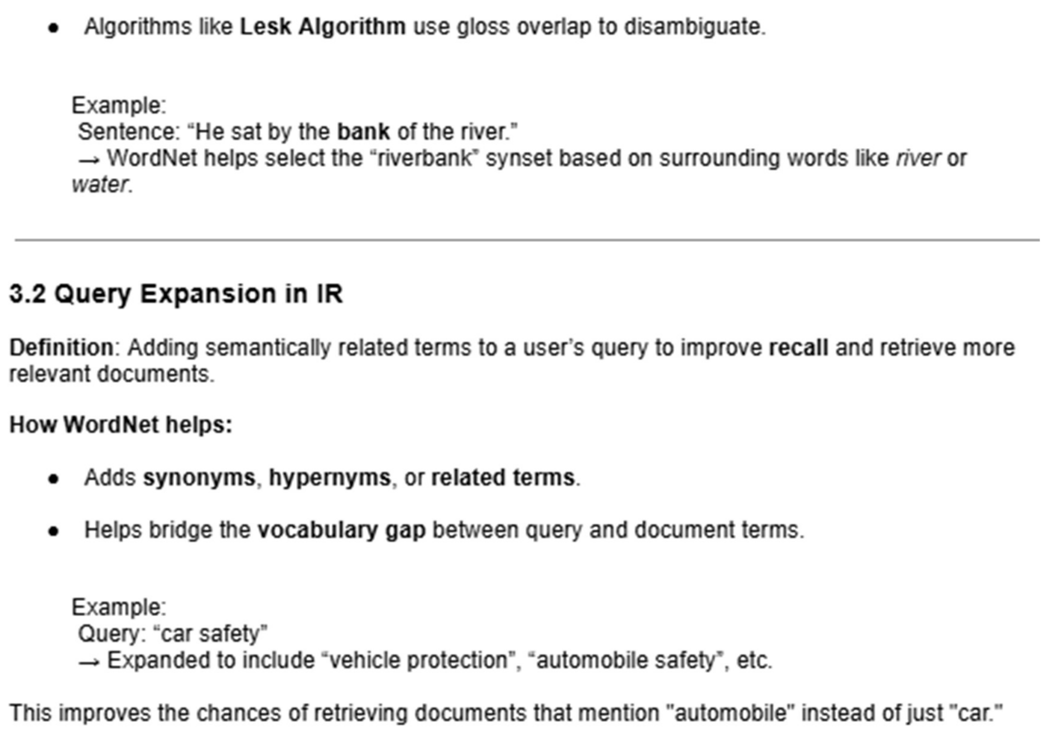
WordNet serves as a foundational tool in many NLP and IR tasks.

3.1 Word Sense Disambiguation (WSD)

Definition: WSD is the process of determining the **correct meaning** of a word in context when the word has **multiple senses**.

How WordNet helps:

- Provides **synsets** for all senses.
- Offers **glosses**, **examples**, and **semantic relations** to compare context.



- Algorithms like **Lesk Algorithm** use gloss overlap to disambiguate.

Example:
Sentence: "He sat by the **bank** of the river."
→ WordNet helps select the "riverbank" synset based on surrounding words like *river* or *water*.

3.2 Query Expansion in IR

Definition: Adding semantically related terms to a user's query to improve **recall** and retrieve more relevant documents.

How WordNet helps:

- Adds **synonyms**, **hypernyms**, or **related terms**.
- Helps bridge the **vocabulary gap** between query and document terms.

Example:
Query: "car safety"
→ Expanded to include "vehicle protection", "automobile safety", etc.

This improves the chances of retrieving documents that mention "automobile" instead of just "car."

b.illustrate the LSTM model

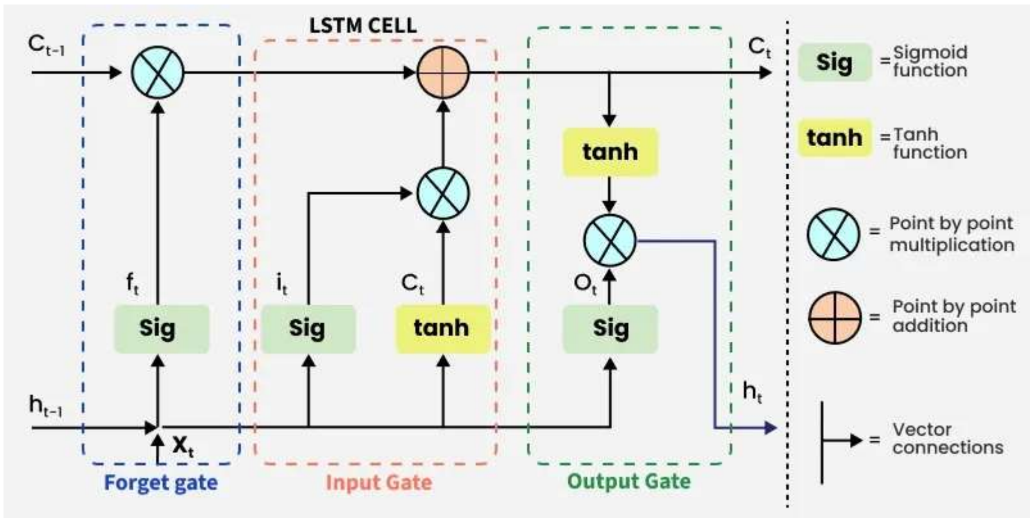
Long Short-Term Memory (LSTM) Model

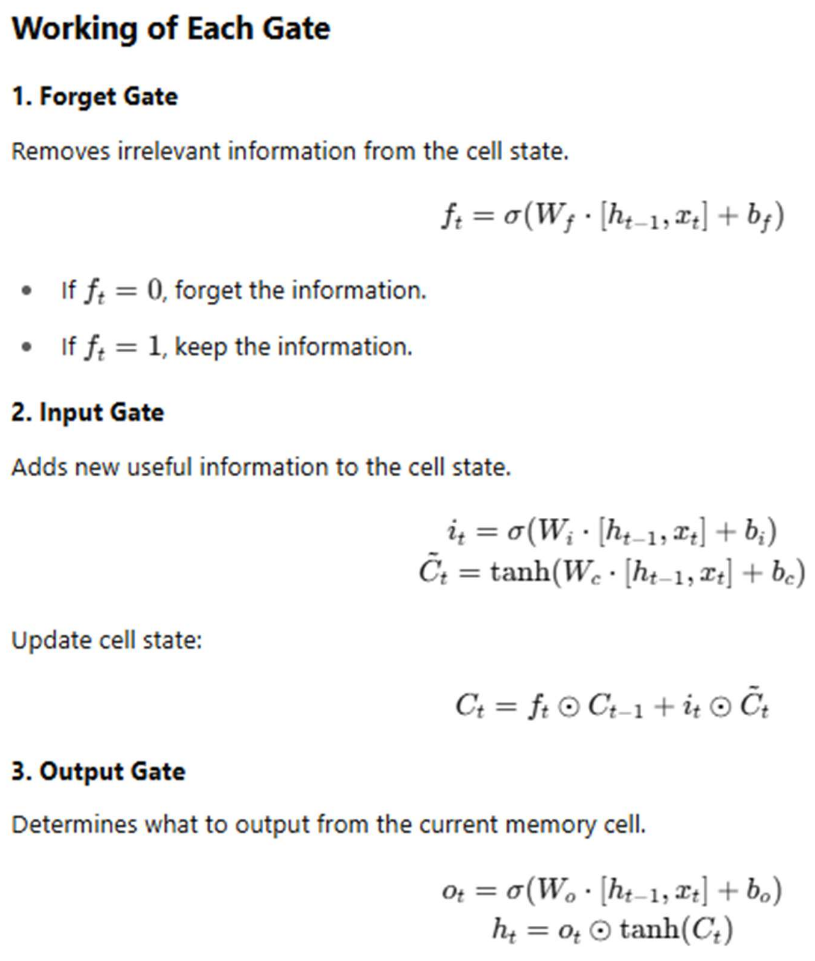
Long Short-Term Memory (LSTM) is an improved form of Recurrent Neural Network (RNN) proposed by **Hochreiter and Schmidhuber** in 1997. LSTM networks are particularly effective at learning **long-term dependencies** in sequential data, making them suitable for applications like **language modeling**, **speech recognition**, and **time series forecasting**.

LSTM Architecture

LSTM introduces a **memory cell** that preserves information over long periods. This cell is regulated by **three gates**:

Gate	**Function**
Forget Gate	Decides what information to discard from the memory
Input Gate	Decides what new information to store in the memory
Output Gate	Decides what part of the memory is output





Working of Each Gate

1. Forget Gate

Removes irrelevant information from the cell state.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

- If $f_t = 0$, forget the information.
- If $f_t = 1$, keep the information.

2. Input Gate

Adds new useful information to the cell state.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{C_t} = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

Update cell state:

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C_t}$$

3. Output Gate

Determines what to output from the current memory cell.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t \odot \tanh(C_t)$$

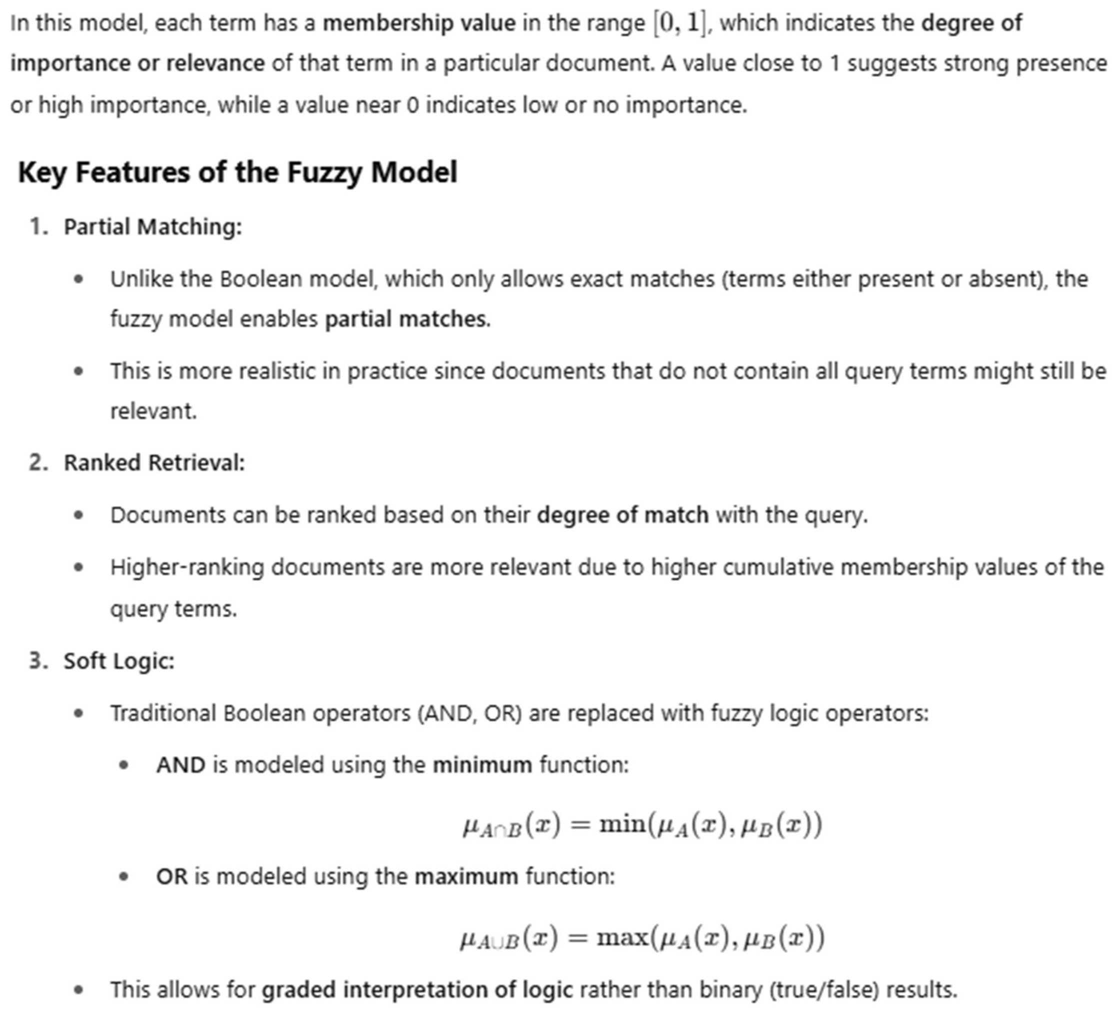
c.Explain the fuzzy model in IR

Fuzzy Model

Overview

The **Fuzzy Model** is an extension of classical set-based models in Information Retrieval (IR). Instead of treating documents as strict sets of terms (where a term is either present or not), the fuzzy model considers documents as **fuzzy sets** of terms, allowing for **graded membership**.

In this model, each term has a **membership value** in the range [0,1][0, 1][0,1], which indicates the **degree of importance or relevance** of that term in a particular document. A value close to 1 suggests strong presence or high importance, while a value near 0 indicates low or no importance.



In this model, each term has a **membership value** in the range $[0, 1]$, which indicates the **degree of importance or relevance** of that term in a particular document. A value close to 1 suggests strong presence or high importance, while a value near 0 indicates low or no importance.

Key Features of the Fuzzy Model

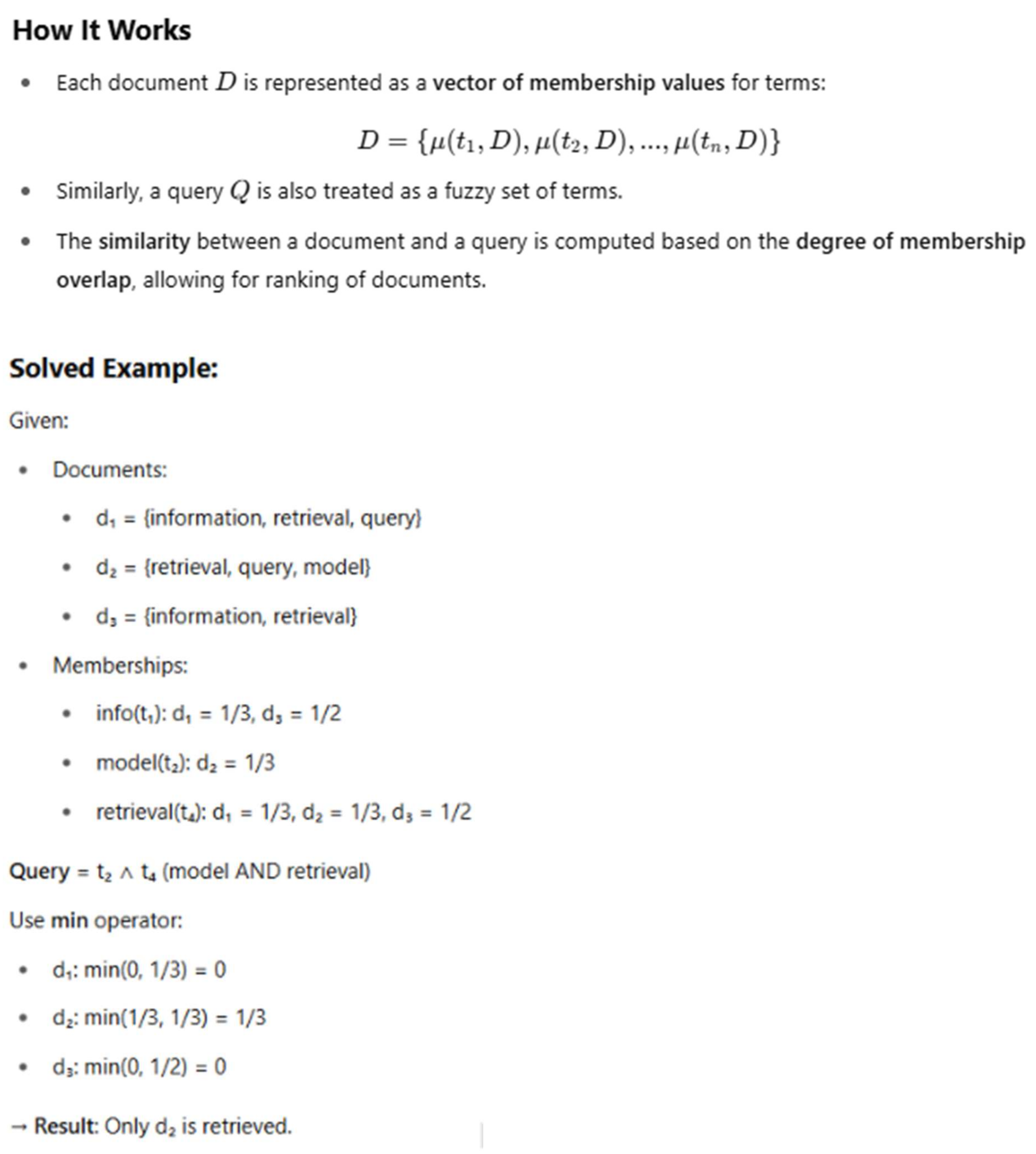
1. **Partial Matching:**
   - Unlike the Boolean model, which only allows exact matches (terms either present or absent), the fuzzy model enables **partial matches**.
   - This is more realistic in practice since documents that do not contain all query terms might still be relevant.
2. **Ranked Retrieval:**
   - Documents can be ranked based on their **degree of match** with the query.
   - Higher-ranking documents are more relevant due to higher cumulative membership values of the query terms.
3. **Soft Logic:**
   - Traditional Boolean operators (AND, OR) are replaced with fuzzy logic operators:
     - **AND** is modeled using the **minimum** function:

$$\mu_{A\cap B}(x) = \min(\mu_A(x), \mu_B(x))$$

     - **OR** is modeled using the **maximum** function:

$$\mu_{A\cup B}(x) = \max(\mu_A(x), \mu_B(x))$$

   - This allows for **graded interpretation of logic** rather than binary (true/false) results.



How It Works

- Each document D is represented as a **vector of membership values** for terms:

$$D = \{\mu(t_1, D), \mu(t_2, D), ..., \mu(t_n, D)\}$$

- Similarly, a query Q is also treated as a fuzzy set of terms.
- The **similarity** between a document and a query is computed based on the **degree of membership overlap**, allowing for ranking of documents.

Solved Example:

Given:

- Documents:
  - d_1 = {information, retrieval, query}
  - d_2 = {retrieval, query, model}
  - d_3 = {information, retrieval}
- Memberships:
  - info(t_1): d_1 = 1/3, d_3 = 1/2
  - model(t_2): d_2 = 1/3
  - retrieval(t_4): d_1 = 1/3, d_2 = 1/3, d_3 = 1/2

Query = $t_2 \wedge t_4$ (model AND retrieval)

Use **min** operator:

- d_1: min(0, 1/3) = 0
- d_2: min(1/3, 1/3) = 1/3
- d_3: min(0, 1/2) = 0

→ **Result**: Only d_2 is retrieved.

Advantages of Fuzzy Model

- **Handles vagueness and uncertainty** in user queries.
- **Improves retrieval quality** in situations where exact matching is too strict.
- **More human-like reasoning**, since people often think in terms of degrees rather than absolutes.

9.

a. Illustrate details of encoder-decoder model in Machine translation(MT)

Encoder-Decoder Architecture in Machine Translation

Introduction

The **Encoder-Decoder architecture** is a fundamental design used in **Neural Machine Translation (NMT)** systems. It is designed to translate a sentence from a **source language** to a **target language** using a sequence-to-sequence model. This architecture uses two components:

- **Encoder**: Understands the source sentence.
- **Decoder**: Generates the translated target sentence.

The model learns the probability distribution:

$$P(y_1, y_2, ..., y_m | x_1, x_2, ..., x_n)$$

Where $x_1, ..., x_n$ is the input sentence and $y_1, ..., y_m$ is the output translation.

---

1. Encoder Component

Function:

The encoder **processes the input sentence** (source language) and encodes it into a **contextual representation.**

Working:

- The input sentence $x = [x_1, x_2, ..., x_n]$ is **tokenized** and converted into vector embeddings.
- These embeddings are passed through multiple **self-attention layers** (in Transformer models) or recurrent layers (in RNN/LSTM-based models).
- The output is a **context vector or sequence** $h = [h_1, h_2, ..., h_n]$ that captures the **semantic meaning** of the entire input sentence.

2. Decoder Component

Function:

The decoder **generates the output sentence** (target language) one word at a time using the context provided by the encoder.

Working:

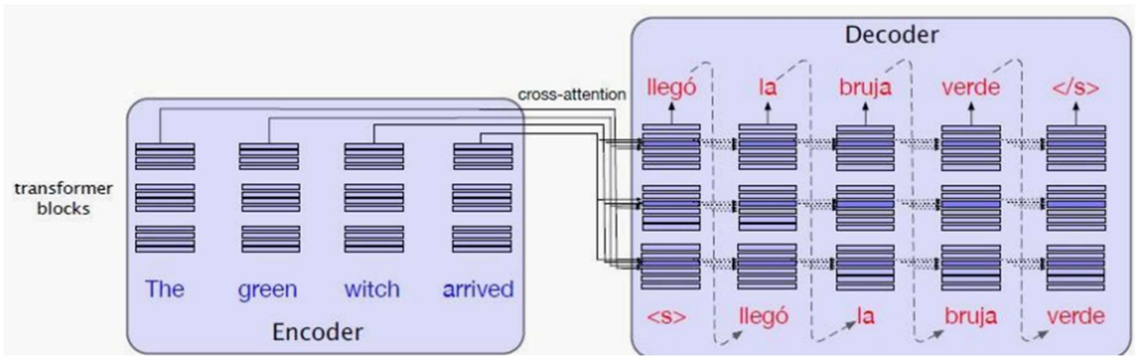
- It takes the encoder's output (context vector) and **starts generating the target sentence** from a special `<START>` token.
- At each step t, it:
  - Looks at previously generated words $y_1, y_2, \ldots, y_{t-1}$,
  - Applies **masked self-attention** to prevent peeking into future words,
  - Applies **cross-attention** to focus on relevant encoder outputs,
  - Predicts the next word y_t.
- This process continues until the model predicts the `<END>` token.

3. Encoder-Decoder Interaction

- The decoder doesn't operate in isolation—it **attends to the encoder's output** using a **cross-attention mechanism**.
- This allows the decoder to focus on relevant parts of the source sentence at each decoding step.
- **Attention scores** determine which source words are most relevant for generating the current target word.

4. Training the Model

- **Objective**: Minimize the **cross-entropy loss** between predicted and actual target sequences.
- **Teacher Forcing** is used: the correct target word from training data is fed to the decoder at each step during training.
- **Subword tokenization** methods (like BPE or WordPiece) are applied to handle rare and compound words.



b. Explain Lexical divergences in MT.

Language Divergences in Machine Translation

Machine Translation (MT) involves converting text from a source language to a target language using computational techniques. One of the major challenges in MT arises from **language divergences —** systematic differences in grammar, structure, and usage between languages. Understanding these divergences is essential for building accurate translation systems. The major types of divergences include **word order typology, lexical divergences, morphological typology**, and **referential density**.

1. Word Order Typology

Definition:
Word order typology refers to the arrangement of subject (S), verb (V), and object (O) in basic sentences across languages.

Types:

- **SVO (Subject-Verb-Object)**: English, French, Mandarin
  Example: "She eats an apple."
- **SOV (Subject-Object-Verb)**: Hindi, Japanese
  Example: "वह सेब खाती है" (Literal: "She apple eats")
- **VSO (Verb-Subject-Object)**: Arabic, Irish
  Example: "Eats she an apple."

MT Challenge:
Translating between languages with different word orders requires **structural reordering** during generation. For instance, English to Japanese translation must move the verb to the end of the sentence.

2. Lexical Divergences

Definition:
Lexical divergence refers to differences in **word meanings, context usage, or conceptual mappings** across languages.

Types and Examples:

- **Polysemy**:
  Example: The English word *"bass"* can refer to a fish or a musical instrument.
  In Spanish: *"lubina"* (fish) vs. *"bajo"* (instrument).

- **Lexical Gaps**:
  Example: English lacks a direct word for the Hindi concept *"adarsh balak"* (ideal child).

- **Context-dependent meanings**:
  Example: *"Leg"* in English can be:
  - *patte* (animal leg)
  - *pied* (furniture leg)
  - *étape* (leg of a journey) in French.

- **Verb-framed vs. Satellite-framed languages**:
  - *Spanish*: *"entrar"* (verb = motion)
  - *English*: *"run in"* (verb = manner, particle = direction)

MT Challenge:
Requires context-aware translation and handling of one-to-many or many-to-one word mappings.

3. Morphological Typology

Definition:
Morphological typology classifies languages based on how they form words using **morphemes** (smallest units of meaning).

Types:

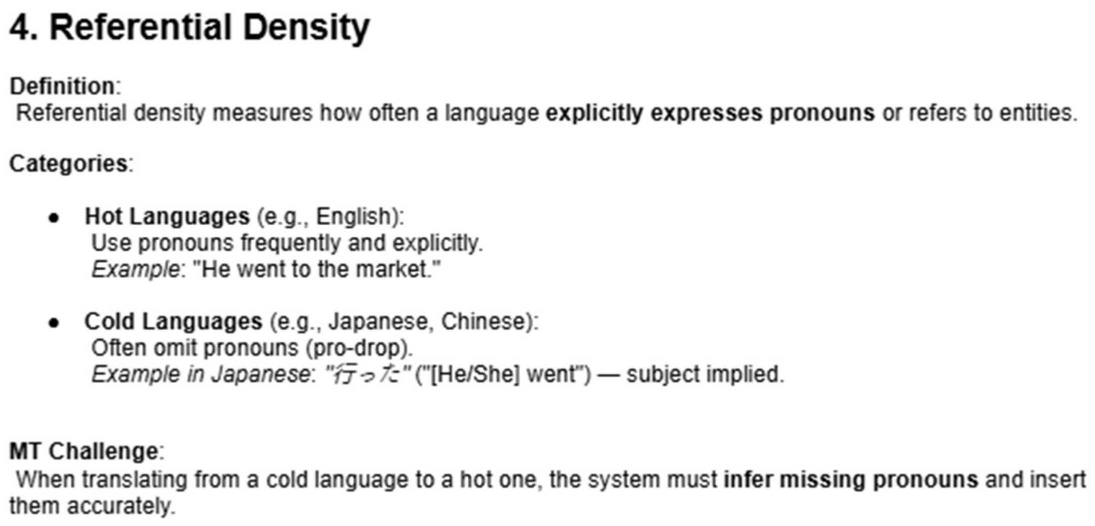
- **Isolating Languages** (e.g., Vietnamese):
  One morpheme per word. Words are not inflected.

- **Agglutinative Languages** (e.g., Turkish):
  Words are formed by stringing together morphemes, each with a clear meaning.

- **Fusional Languages** (e.g., Russian):
  Morphemes are fused together; a single morpheme may encode multiple grammatical categories.

- **Polysynthetic Languages** (e.g., Inuktitut):
  One complex word may represent an entire sentence.

MT Challenge:
MT systems must handle complex **word segmentation and composition**. Subword tokenization techniques like **Byte Pair Encoding (BPE)** or **WordPiece** are used to manage this.



4. Referential Density

Definition:
Referential density measures how often a language **explicitly expresses pronouns** or refers to entities.

Categories:

- **Hot Languages** (e.g., English):
  Use pronouns frequently and explicitly.
  Example: "He went to the market."
- **Cold Languages** (e.g., Japanese, Chinese):
  Often omit pronouns (pro-drop).
  Example in Japanese: *"行った"* ("[He/She] went") — subject implied.

MT Challenge:
When translating from a cold language to a hot one, the system must **infer missing pronouns** and insert them accurately.

10.

a. Explain automatic evaluation in various forms. List out ethicl issues raised in MT.

a. Automatic Evaluation in Various Forms

Introduction

Automatic evaluation of Machine Translation (MT) systems is essential for measuring the quality of translated output without relying on human judgment for every translation. These methods are fast, scalable, and consistent, making them suitable for both development and benchmarking of MT systems.

1. Types of Automatic Evaluation Metrics

1.1 BLEU (Bilingual Evaluation Understudy)

- **Nature**: Precision-based metric.
- **Working**:
  - Measures **n-gram overlap** (from unigram to 4-gram) between system output and one or more reference translations.
  - Uses **clipped counts** to prevent overcounting repeated words.
  - Applies a **brevity penalty** to penalize overly short translations.
- **Limitations**:
  - Ignores **recall**.
  - Sensitive to **tokenization**.
  - Does not handle **paraphrasing or reordering** well.
  - Less effective for **short sentences**.

1.2 chrF (Character n-gram F-score)

- **Nature**: Based on **F-score** over character n-grams.
- **Advantages**: Works well for **morphologically rich** languages.
- **Limitations**:
  - Ignores **semantic meaning**.
  - Focuses only on **local character-level matches**.

1.3 METEOR

- **Nature**: Uses **synonym matching**, stemming, and alignment.
- **Advantages**: Considers both **precision and recall.**
- **Limitations**: Computationally **more expensive** than BLEU.

1.4 TER (Translation Edit Rate)

- **Nature:** Measures the number of **edits** required to change a system translation into the reference.
- **Limitations**: Penalizes **valid alternative phrasings.**

---

2. Statistical Significance in MT Evaluation

To assess whether observed differences between MT systems are **real** and not due to chance, we use:

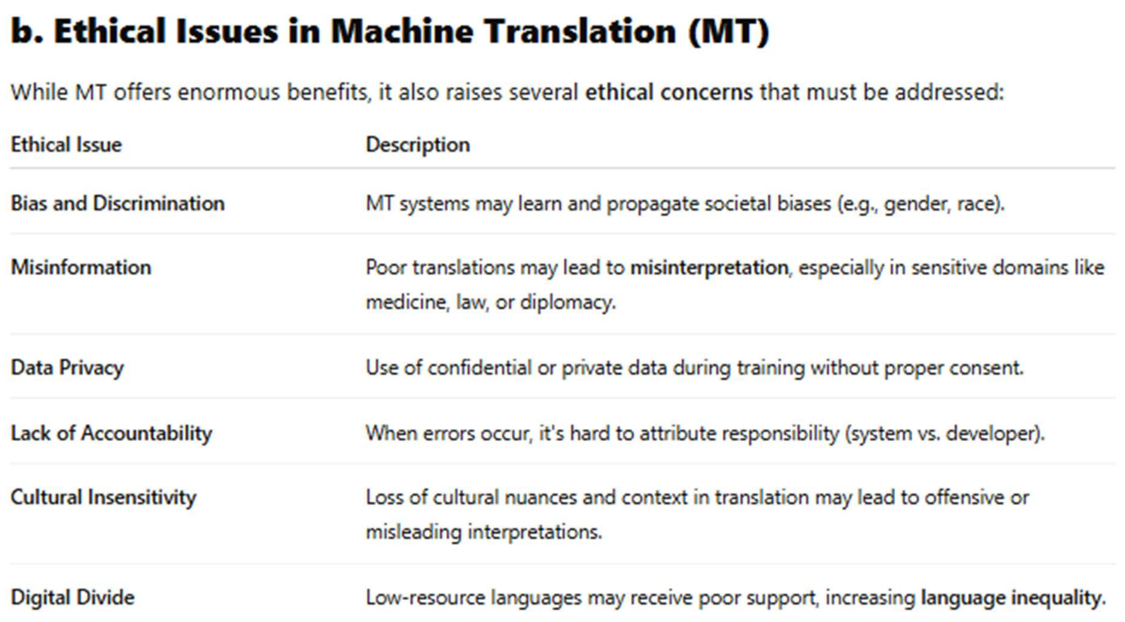
a. Bootstrap Resampling

- Resamples the test set multiple times.
- Calculates metric scores and determines **confidence intervals.**

b. Randomization Test

- Swaps outputs across systems to simulate null hypothesis.
- Tests whether score difference is statistically **significant.**

Example:
If System A scores BLEU 31.2 and System B scores 30.8, and in 95% of bootstrap samples A is better, the difference is **statistically significant.**



b. Ethical Issues in Machine Translation (MT)

While MT offers enormous benefits, it also raises several **ethical concerns** that must be addressed:

Ethical Issue	Description
Bias and Discrimination	MT systems may learn and propagate societal biases (e.g., gender, race).
Misinformation	Poor translations may lead to **misinterpretation**, especially in sensitive domains like medicine, law, or diplomacy.
Data Privacy	Use of confidential or private data during training without proper consent.
Lack of Accountability	When errors occur, it's hard to attribute responsibility (system vs. developer).
Cultural Insensitivity	Loss of cultural nuances and context in translation may lead to offensive or misleading interpretations.
Digital Divide	Low-resource languages may receive poor support, increasing **language inequality.**

b. explain the approaches for detailing with low-resource situations in MT.

Machine Translation in Low-Resource Languages

1. Introduction

Most of the world's languages lack sufficient **parallel corpora** for training high-quality Machine Translation (MT) models. These are referred to as **low-resource languages**. Unlike English or French, they do not have large-scale aligned datasets.

To overcome this **data sparsity**, researchers use **data augmentation techniques** and **multilingual modeling**.

---

2. Strategy 1: Data Augmentation using Backtranslation

What is Backtranslation?

Backtranslation is a technique where monolingual data in the **target language** is translated into the **source language** using a reverse translation model. This creates **synthetic parallel data** to train the actual forward MT model.

Steps in Backtranslation:

1. Train a **target-to-source** MT model using available bitext.
2. Use this model to **translate monolingual target-language sentences** into the source language.
3. Create **synthetic sentence pairs**: (synthetic source, real target).
4. Add these pairs to the original training data.
5. Retrain the **source-to-target** model with the expanded dataset.

Example:

- Suppose we have limited English ↔ Hindi data but a lot of **Hindi-only monolingual text**.
- Step 1: Train a Hindi-to-English model with existing data.
- Step 2: Translate Hindi sentences into synthetic English.
- Step 3: Use these synthetic English-Hindi pairs to train a better **English-to-Hindi** MT system.

Benefits of Backtranslation:

- Uses **abundant monolingual data**.
- Helps the model learn **target language fluency**.
- Improves performance even without real bitext.
- Effective even when synthetic translations are imperfect.

---

3. Strategy 2: Multilingual Models

What are Multilingual Models?

Multilingual models are trained on **parallel data from multiple language pairs** in a **single unified model**. These models can transfer knowledge across languages, benefiting low-resource languages by sharing representations with high-resource ones.

Architecture:

- The model is given:
  - A **language tag (token)** indicating the source and target languages.
  - A shared **vocabulary** across all languages (via subword tokenization).
- During training:
  - Input: `[LANG_SRC] sentence`
  - Output: `[LANG_TGT] translation`

Example:

- Train a single model on:
 English–French, English–German, English–Hindi, English–Swahili
- Even if Hindi–Swahili is not trained, the model might **zero-shot translate** between them using shared structures and token patterns.

Benefits of Multilingual MT:

- **Parameter sharing** across languages.
- Improves **low-resource performance** via transfer learning.
- Enables **zero-shot translation** (e.g., Hindi ↔ Swahili without direct pairs).
- Reduces model count (single model for many languages).

Challenges in Low-Resource MT

- Lack of high-quality monolingual data.
- Bias due to **English-centric training**.
- Involvement of native speakers is often limited.
- Need for **participatory design** and better **evaluation methods**.