Natural Language Processing

UNIT-I

1. **Explain the Scope of Natural Language Processing (NLP) and Discuss Its Real-World Applications with Relevant Examples**

Introduction:
Natural Language Processing (NLP) is a field of Artificial Intelligence (AI) and Computer Science that focuses on the interaction between computers and human languages. Its main goal is to make machines understand, interpret, and respond to human language in a meaningful way.

Scope of NLP

The scope of NLP is very wide because language is used everywhere in communication. Some major scopes are:

1. **Understanding Human Language:**
   NLP helps computers understand written or spoken language like English, Hindi, etc.
   Example: Identifying the meaning of a sentence like *"I am feeling happy."*
2. **Language Generation:**
   NLP enables machines to generate responses in natural language.
   Example: Chatbots creating replies to user messages.
3. **Speech Processing:**
   NLP covers both speech-to-text and text-to-speech tasks.
   Example: Converting spoken audio into written text.
4. **Machine Translation:**
   NLP allows computers to translate text from one language to another.
   Example: Translating English to Hindi.
5. **Information Extraction:**
   NLP extracts important information like names, dates, or events from large documents.
   Example: Extracting the name of a person from a news article.
6. **Sentiment and Emotion Analysis:**
   NLP detects emotions or opinions in text.
   Example: Understanding if a customer review is positive or negative.
7. **Question Answering Systems:**
   NLP enables computers to answer questions asked in natural language.
   Example: AI systems answering queries about weather or general knowledge.

Real-World Applications of NLP with Examples

NLP is used in many areas of daily life. Some important applications are:

1. Machine Translation

- Converts text from one language to another.
- **Example:** Google Translate can translate English sentences to Hindi, Marathi, Tamil, etc.
- Useful for communication, tourism, and learning.

---

2. Sentiment Analysis

- Identifies emotions like positive, negative, or neutral in text.
- **Example:** Companies analyze Twitter or Facebook comments to know customer opinions about products.
- Used in marketing and customer service.

---

3. Chatbots and Virtual Assistants

- NLP powers chatbots and voice assistants that understand user queries.
- **Examples:**
  - Google Assistant answering "What is the weather today?"
  - Chatbots used by banks for answering customer questions.
- Saves time and reduces human workload.

---

4. Speech Recognition

- Converts spoken words into text.
- **Examples:**
  - Voice typing in smartphones
  - Voice command systems in cars
- Useful for people with disabilities.

---

5. Text Summarization

- Automatically creates short summaries of long documents.
- **Example:** Summarizing news articles for quick reading.
- Useful in media, education, and research fields.

---

6. Spam Detection in Emails

- NLP helps in identifying spam, promotional, or harmful emails.
- **Example:** Gmail filters spam messages using NLP algorithms.
- Protects users from phishing and fraud.

---

7. Information Retrieval

- Used in search engines to find relevant information from the internet.
- **Example:** Google search uses NLP to understand user queries like "Best hotels in Delhi".
- Used in knowledge retrieval and research.

---

8. Healthcare Applications

- NLP helps in analyzing patient data, clinical notes, and medical records.
- **Example:** Extracting symptoms and disease names from doctor reports.
- Supports better diagnosis and treatment.

---

2. Demonstrate Tokenization and Stemming on a Given Text Corpus

Given Example Text:

"The cats are playing in the garden."

(A) Tokenization

Definition:
Tokenization is the process of splitting a sentence or text into smaller units called **tokens** (words, phrases, etc.).

Step 1: Input Sentence

"The cats are playing in the garden."

Step 2: Remove Punctuation (optional step)

Sentence becomes:
"The cats are playing in the garden"

Step 3: Split into Tokens (words)

Tokens obtained:

["The", "cats", "are", "playing", "in", "the", "garden"]

Intermediate Result (After Tokenization):

Token No.	Token
1	The
2	cats
3	are
4	playing
5	in
6	the
7	garden

(B) Stemming

Definition:
Stemming reduces each word to its **root form** (not always a meaningful dictionary word).
Common stemmers: Porter, Snowball, Lancaster, etc.

Step 1: Take Tokens from Tokenization

["The", "cats", "are", "playing", "in", "the", "garden"]

Step 2: Apply Stemming Rules

- **The → the** (no change)
- **cats → cat** (remove plural 's')
- **are → are** (no change)

- **playing → play** (remove 'ing')
- **in → in** (no change)
- **the → the** (no change)
- **garden → garden** (no change)

(Using common Porter Stemmer logic)

Intermediate Result (After Stemming):

Original Word	Stemmed Word
cats	cat
playing	play
the	the
are	are
in	in
garden	garden

(C) Final Output

Final Token List after Tokenization + Stemming:

["the", "cat", "are", "play", "in", "the", "garden"]

Conclusion

- **Tokenization** splits text into separate meaningful units (words).
- **Stemming** reduces words to their base/root form.
- Both processes are useful in NLP tasks like text mining, search engines, and machine learning.

3.Compare and Contrast Stemming and Lemmatization

Stemming and Lemmatization are two text preprocessing methods used in NLP to reduce words to their base form. They help in improving text analysis, search, and machine learning tasks.

(A) Comparison with 10 Aspects

No.	Aspect	Stemming	Lemmatization
1	**Definition**	Removes suffixes to obtain root form	Converts word to base dictionary form (lemma)
2	**Approach**	Rule-based chopping of word endings	Uses vocabulary + grammar rules
3	**Output Form**	Usually not a valid word	Always a valid word
4	**Example**	“studies” → “studi”	“studies” → “study”
5	**Grammar Awareness**	Does not consider POS or meaning	Considers POS & linguistic meaning
6	**Accuracy**	Lower accuracy	Higher accuracy
7	**Complexity**	Simple and fast	Complex and slow
8	**Tools**	Porter, Snowball, Lancaster Stemmer	WordNet Lemmatizer, Spacy Lemmatizer
9	**Use Case**	Useful in large datasets & search engines	Useful in NLP tasks needing linguistic correctness
10	**Result Quality**	Rough and inconsistent	Clean, meaningful, standardized

(B) Examples for Clarity

Word	Stemming Output	Lemmatization Output
Playing	play	play
Studies	studi	study
Better	better	good
Cars	car	car

(C) Advantages & Limitations

1. Advantages of Stemming

1. **Fast Processing** – Suitable for large-scale text mining.
2. **Computationally Light** – Needs less memory and CPU.
3. **Improves Recall in Search** – Matches word variations (play, playing, played).
4. **Easy Implementation** – Rule-based, so simpler to apply.
5. **Useful in Informal Text** – Works for tweets, chats, blogs.

2. Limitations of Stemming

1. **Inaccurate Results** – Produces non-meaningful roots (e.g., "studies → studi").
2. **No Grammar Knowledge** – Does not consider part of speech.
3. **Over-stemming** – Different words cut to same root wrongly (e.g., "universe" & "university" → "univers").
4. **Under-stemming** – Similar words not reduced properly.
5. **Poor Readability** – Output may not be understandable.

---

3. Advantages of Lemmatization

1. **Higher Accuracy** – Produces meaningful dictionary words.
2. **POS Awareness** – Considers verbs, nouns, adjectives, etc.
3. **Improves NLP Quality** – Useful in translation, sentiment analysis.
4. **Semantic Understanding** – Connects words with similar meaning (better → good).
5. **Useful in Professional NLP Applications** – e.g., medical, legal documents.

4. Limitations of Lemmatization

1. **Slower Processing** – Needs linguistic libraries and rules.
2. **Depends on Vocabulary** – Requires lexical resources (e.g., WordNet).
3. **More Complex to Implement** – Not rule-based like stemming.
4. **Higher Memory Usage** – Needs dictionary + morphological analysis.
5. **Difficult with Ambiguous Words** – Requires correct POS tagging.

4.Evaluate the Significance of Stop Word Removal in Text Preprocessing and Discuss Situations Where Retaining Stop Words Might Be Beneficial

Introduction

In Natural Language Processing (NLP), text preprocessing is an important step to clean and prepare raw text for analysis or machine learning. One common preprocessing task is **stop word removal**.

Stop words are commonly used words in a language that carry very little meaning on their own. Examples in English include: *“is, am, are, the, a, an, and, of, to, in,”* etc.

Significance of Stop Word Removal

Stop word removal is significant for various reasons:

1. Reduces Noise

Stop words do not add meaningful information in many NLP tasks. Removing them helps in keeping only important words.

- Example: In a sentence **“The movie was very good”**, the word “good” gives more meaning than “the”, “was”, “very”.

---

2. Improves Processing Efficiency

When unnecessary words are removed, the size of data becomes smaller. This helps in:

- Faster training of models
- Less storage usage
- Lower computational cost

---

3. Enhances Accuracy in Text Mining

Removing stop words can make patterns clearer for tasks like clustering, information retrieval, and topic modeling.

- Example: In topic modeling, removing stop words reveals the main topic words clearly such as *“sports, politics, economy,”* etc.

---

4. Improves Search Engine Performance

Search engines ignore stop words to focus on important keywords.

- Example: Searching **“Hotels in Mumbai”**, the word “in” is ignored and only “Hotels” and “Mumbai” matter.

---

5. Helps in Document Similarity Measurement

Stop word removal gives accurate similarity scores between documents by removing common, meaningless words.

---

6. Useful in Feature Extraction

In machine learning, features are important words. Removing stop words helps extract better features for classification tasks such as spam detection.

Summary of Benefits

- Reduces dimensionality
- Eliminates unimportant words
- Improves model performance
- Saves memory and computation time
- Makes semantic meaning clearer

When Retaining Stop Words Might Be Beneficial

Although stop words are often removed, in some situations they are **useful** and should be kept. These include:

1. Sentiment Analysis

Words like *“not”, “isn’t”, “never”* can completely change the meaning.

- Example: “I am **not** happy” vs “I am happy”.
  Removing “not” gives the opposite meaning, so stop words must be kept.

---

2. Text Summarization

Stop words help maintain sentence structure and readability.
Removing them may cause the summary to become unclear.

---

3. Machine Translation

In translation tasks, every word matters for correct grammar and meaning.

- Example: Translating **“He is in the room”** into other languages needs all words.

---

4. Speech Recognition

Stop words help maintain natural grammar and sentence flow when converting speech to text.

---

5. Linguistic Analysis

Stop words are important in analyzing language patterns, grammar, and writing style.

---

6. Question Answering Systems

Some stop words such as “who”, “what”, “when” are essential to understand the type of question.

- Example: Removing “who” or “what” makes the question meaningless.

---

7. Conversational AI and Chatbots

To generate natural replies, chatbots need stop words for sentence structure.

- Without stop words, answers would sound robotic and incomplete.

---

(5) Design a Regular Expression to Identify All Email Addresses in a Text Document

Regular Expression (Regex)

To identify email addresses, a commonly used regex pattern is:

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

Meaning of the Pattern

- **[a-zA-Z0-9._%+-]+** → matches the username part (letters, digits, dots, underscore, etc.)
- **@** → mandatory symbol separating username and domain
- **[a-zA-Z0-9.-]+** → matches the domain name

- **.** → matches a literal dot .
- **[a-zA-Z]{2,}** → matches top-level domain (like com, org, edu)

Example Text

Consider the text:

“For queries contact us at support123@gmail.com or admin.office@university.edu. You may also visit our website.”

Applying the Regex

Using the regex, we extract:

- support123@gmail.com
- admin.office@university.edu

Explanation of Results

The regex successfully found two email addresses in the text because they follow the required structure:

- username
- @ symbol
- domain
- top-level domain (TLD)

Words without email structure (like “website”) are ignored as expected.

---

(6) Define Minimum Edit Distance in NLP and Calculate the Distance between “kitten” and “sitting”

Definition

The **Minimum Edit Distance** is the minimum number of operations required to convert one word into another. The operations usually allowed are:

1. **Insertion**
2. **Deletion**

3. **Substitution**

This concept is widely used in:

- Spell checking
- Machine translation
- Speech recognition
- DNA sequence analysis

Example: kitten → sitting

We compute using **dynamic programming**.

Words:

- Source word = **kitten** (length = 6)
- Target word = **sitting** (length = 7)

Step 1: Initialize Matrix

Create a (7 × 8) matrix because we include an empty string row and column.

		s	**i**	**t**	**t**	**i**	**n**	**g**
	0	1	2	3	4	5	6	7
k	1							
i	2							
t	3							
t	4							
e	5							
n	6							

Step 2: Fill Matrix Using Rules

If characters match → take diagonal value
If not → take 1 + min(insert, delete, substitute)

Final completed matrix (distances filled):

		s	**i**	**t**	**t**	**i**	**n**	**g**
	0	1	2	3	4	5	6	7
k	1	1	2	3	4	5	6	7
i	2	2	1	2	3	4	5	6
t	3	3	2	1	2	3	4	5
t	4	4	3	2	1	2	3	4
e	5	5	4	3	2	2	3	4
n	6	6	5	4	3	3	2	3

Minimum Edit Distance

The final value is found at **bottom-right corner = 3**

So, **Minimum Edit Distance (kitten → sitting) = 3**

Operations Used

The sequence of changes:

1. **kitten → sitten** (substitution of k → s)
2. **sitten → sittin** (substitution of e → i)
3. **sittin → sitting** (insertion of g)

So total operations = **3**

Conclusion

- Minimum Edit Distance helps measure similarity between words.
- For “kitten” and “sitting”, the minimum edit distance is **3**, meaning only three operations can transform one to the other.

UNIT-II

(2) Calculate the TF-IDF Score for the Word "data"

Given Details

- Total documents (N) = **5**
- Documents containing "data" (df) = **2**
- Term frequency in document (tf) = **4**
- Total terms in the document = **20**

Step 1: Compute Term Frequency (TF)

$$TF = \frac{\text{Number of times the word appears}}{\text{Total terms in the document}}$$

$$TF = \frac{4}{20} = 0.2$$

Step 2: Compute Inverse Document Frequency (IDF)

$$IDF = \log\left(\frac{N}{df}\right)$$

$$IDF = \log\left(\frac{5}{2}\right)$$

$$IDF = \log(2.5)$$

Using natural log:

$$\log(2.5) \approx 0.916$$

Step 3: Compute TF-IDF Score

$$TF\text{-}IDF = TF \times IDF$$

$$TF\text{-}IDF = 0.2 \times 0.916 = 0.1832$$

Final Result

TF-IDF score for “data” ≈ 0.183

Interpretation

- TF-IDF is **low** if a term is frequent across many documents.
- TF-IDF is **high** if a term is frequent in one document but **rare in the corpus**.
- In this case, the word “data” appears in multiple documents, so its TF-IDF is moderate.

---

3. Compare and Contrast TF-IDF and Pointwise Mutual Information (PPMI)

Introduction

TF-IDF and PPMI are both weighting methods used in NLP to represent words in a vector space model. They help give importance to meaningful terms and reduce noise.

Definition of TF-IDF

TF-IDF = **Term Frequency × Inverse Document Frequency**

- Gives high weight to words that occur frequently in a document but not frequently across all documents.
- Formula:
  TF-IDF = TF × log (N/df)

Definition of PPMI

PPMI = **Positive Pointwise Mutual Information**

- Measures how strongly two words (or a word and context) are associated relative to random chance.
- Formula:

$$PMI(w,c) = \log \frac{P(w,c)}{P(w)P(c)}$$

- **PPMI** = max(PMI, 0) → negative values are replaced by 0.

Comparison Table

Aspect	TF-IDF	PPMI
Meaning	Measures importance of terms in documents	Measures association between word and context
Input	Document-term matrix	Co-occurrence matrix
Focus	Document relevance	Semantic association
Values	Always positive	Can be negative (but PPMI removes negatives)
Normalization	Uses log on IDF	Uses log on probability ratio
Best suited for	IR, search engines	Word embeddings, semantic similarity
Example Use	Ranking documents for search queries	Word2Vec, context vectors
Handles stop words	Low weights for stop words	PMI may still give high weight if co-occurs strongly

---

Strengths & Weaknesses

✓ Strengths of TF-IDF

1. Simple & easy to compute
2. Good for information retrieval
3. Reduces weight of common words (stop words)
4. Improves document ranking performance

✕ Weaknesses of TF-IDF

1. Does not capture word meaning or semantics
2. Ignores word context and order
3. Sparse representation
4. Not suitable for capturing similarity between words

✓ Strengths of PPMI

1. Captures semantic relationships

2. Good for context-based models (e.g., embeddings)
3. Highlights rare-but-meaningful co-occurrences
4. Widely used in distributional semantics

✕ Weaknesses of PPMI

1. Sensitive to rare events (sparsity problem)
2. Requires large corpora for stability
3. High dimensionality and memory usage
4. Negative values removed (information loss)

---

4. Effectiveness of Smoothing Techniques in Addressing Data Sparsity in N-gram Language Models

Introduction

N-gram models suffer from **data sparsity**, meaning many valid word combinations never appear in training data. Smoothing techniques adjust probabilities to avoid zero values.

Common Smoothing Methods

(A) Laplace (Add-One) Smoothing

Idea

- Add 1 to every count:

$$P(w_i|w_{i-1}) = \frac{Count(w_{i-1}, w_i) + 1}{Count(w_{i-1}) + V}$$

where **V = vocabulary size**

Effectiveness

✓ Eliminates zero probabilities
✕ Overestimates rare words

Example

If “cat eats” never occurs:

- Without smoothing: Count = 0 → Probability = 0

- With Laplace: Count = 0+1 = 1 → Non-zero probability

Use Cases

- Good for small datasets
- Useful in basic language models and teaching examples

(B) Good-Turing Smoothing

Idea

Re-estimates probability of unseen events based on frequency of rare events.

- If **Nc** = number of n-grams that occur **c** times:

$$c^* = \frac{(c+1)N_{c+1}}{N_c}$$

Effectiveness

✓ Better handling of rare events than Laplace
✓ Good for large corpora
✕ Complex to compute for large c values

Example

If 120 bigrams occur once (N1 = 120) and 30 bigrams occur twice (N2 = 30):

$$c^* = \frac{(1+1)\cdot 30}{120} = \frac{60}{120} = 0.5$$

So count “1” replaced by “0.5”.

Use Cases

- Speech recognition
- Handwriting recognition
- Natural language modeling in large corpora

(C) Backoff and Interpolation Methods

Idea

Use lower-order N-grams when higher-order N-grams are missing.

Example:

If trigram P(w3 | w1 w2) missing → use bigram P(w3 | w2)
If bigram missing → use unigram P(w3)

Effectiveness

✓ Very effective for large vocabularies
✓ Keeps useful context when available
✗ Requires tuning of weights

Examples of Application

- **Katz Backoff**: Uses Good-Turing discounting + backoff
  Used in:
  ✓ Speech recognition (e.g., early voice assistants)
- **Kneser-Ney Smoothing**: Advanced interpolation method
  Used in:
  ✓ Machine translation
  ✓ Statistical language modeling

Overall Evaluation

Method	**Strengths**	**Weaknesses**
Laplace	Simple, removes zeros	Over-smooths, low accuracy
Good-Turing	Accurate for rare events	Complex math for large corpora
Backoff/Interpolation	Maintains context, scales well	Needs parameter tuning

---

(5) Design a Vector Space Model for Retrieving Documents Based on a Query

Introduction

A Vector Space Model (VSM) represents documents and queries as vectors in a multi-dimensional space. It allows ranking of documents based on similarity to a user's query.

Steps Involved

Step 1: Preprocessing

Before vector generation, text must be cleaned using NLP techniques such as:

(a) Tokenization

- Split text into words.
- Example: "AI improves healthcare" → [AI, improves, healthcare]

(b) Lowercasing

- Convert words to lowercase to avoid duplicates.
- "AI" and "ai" become "ai".

(c) Stop Word Removal

- Remove common words like: the, is, are, etc.

(d) Stemming/Lemmatization

- Convert words to base form.
- Example: improves → improve

After preprocessing, each document becomes a bag of meaningful terms.

Step 2: Term Weighting Using TF-IDF

(a) Term Frequency (TF)

$$TF(t,d) = \frac{\text{Number of times term appears in document}}{\text{Total terms in document}}$$

(b) Inverse Document Frequency (IDF)

$$IDF(t) = \log\left(\frac{N}{df(t)}\right)$$

Where:

- N = total number of documents
- df(t) = number of documents containing term t

(c) TF-IDF Score

$$\boldsymbol{TF\text{-}IDF(t,d) = TF(t,d) \times IDF(t)}$$

This gives high weight to important and rare terms.

Step 3: Represent Documents and Query as Vectors

After TF-IDF calculation, each document becomes a vector such as:

$$\boldsymbol{d_1 = (w_{1,1}, w_{1,2}, \ldots, w_{1,n})}$$

Where w values are TF-IDF scores.

Example:

- **Query: "machine learning"**
- **Document D1 vector: (0.2, 0.6, 0, 0.1,...)**

Step 4: Similarity Computation Using Cosine Similarity

We measure similarity between document vector d and query vector q:

$$\boldsymbol{\cos(\theta) = \frac{d \cdot q}{\| d \| \cdot \| q \|}}$$

Where:

- **d · q = dot product of vectors**
- **||d|| and ||q|| = magnitudes (lengths) of vectors**

Cosine Value Range

- **1 → documents are very similar**
- **0 → no similarity**
- **−1 → opposite meaning (rare in TF-IDF space)**

Step 5: Ranking Documents

- **Compute cosine similarity for all documents**
- **Sort them in descending order**

- **Return top-ranked results to user**

(6) Define Word2Vec and Explain Skip-gram vs CBOW

Definition of Word2Vec

Word2Vec is a neural network-based technique used to generate dense vector representations (embeddings) for words. These embeddings capture **semantic meaning**, such that similar words have similar vector representations.

Example:
Vectors of "king" and "queen" become mathematically related.

Primary Models of Word2Vec

Word2Vec supports two main architectures:

(A) Continuous Bag of Words (CBOW)

Concept

- Predicts the **target word** based on surrounding **context words**.
- Input: context → Output: target word
- Example sentence: "the cat sits on the mat"

To predict "sits", CBOW uses context: ["the", "cat", "on", "the", "mat"]

Characteristics

✓ Faster training
✓ Works well with small datasets
✓ Averages context, so handles noise fairly well

(B) Skip-gram Model

Concept

- Predicts the **context words** given a **target word**
- Input: target word → Output: context words

Using same example, Skip-gram takes "sits" and predicts:
["the", "cat", "on", "the", "mat"]

Characteristics

✓ Good for large datasets
✓ Works well for rare words
✓ Better representations for infrequent terms

Major Differences Between CBOW and Skip-gram

Aspect	**CBOW**	**Skip-gram**
Input	Context words	Target word
Output	Target word	Context words
Speed	Faster	Slower
Accuracy for rare words	Lower	Higher
Training data requirement	Small corpora	Large corpora
Noise Handling	Good	Medium
Meaning preservation	Slightly lower	Higher semantic accuracy

Example of Use Cases

CBOW used in:

- Real-time predictions
- Small or medium datasets
- Applications requiring fast training

Skip-gram used in:

- Semantic similarity tasks
- Machine translation
- Large text corpora (e.g., Wikipedia, Google News)

UNIT-III

---

1. Explain the concept of sequence labeling in NLP. How is it used in part-of-speech tagging, and what role does it play in named entity recognition (NER)?

Concept of Sequence Labeling

- **Sequence labeling** means assigning a **label to each element** in an input sequence (usually words in a sentence).
- The output is also a **sequence** of the same length.
- Useful for understanding linguistic structure.

How it Works

- Input: sequence of words
- Output: sequence of labels
  Example:
  Sentence: *"I love NLP"*
  Labels: PRON VERB NOUN

Use in Part-of-Speech (POS) Tagging

- POS tagging assigns **grammatical labels** (noun, verb, adjective, etc.) to words.
- Example:
  Sentence: "John loves music"
  Sequence Labels: NNP VBZ NN

Why sequence labeling suits POS?

- POS of each word depends on:
  - Surrounding words
  - Sentence structure
    Example: **“book”**
- As noun: “read a book”
- As verb: “book a ticket”

So sequence labeling models context dependencies.

Role in Named Entity Recognition (NER)

- NER identifies **entities** such as:
    - Person (PER)
    - Organization (ORG)
    - Location (LOC)
    - Date, Money, etc.

Example sentence:
"John works at Google in California"
NER labels:

- John → PER
- Google → ORG
- California → LOC

NER also uses **sequence labeling** because:

- Entities span multiple words (e.g., “New York City”)
- Labels depend on context

Example labeling format (BIO scheme):

- B = Beginning of entity
- I = Inside entity
- O = Outside entity

Example:
Sentence: "New York City is big"
Labels: B-LOC I-LOC I-LOC O O

2.Given the sentence "John bought a car yesterday", apply a basic Hidden Markov Model (HMM) for POS tagging. Define the states, observations, and transitions, and perform the tagging for each word

Given Sentence:

"John bought a car yesterday"

We apply **Hidden Markov Model (HMM)** for POS tagging.

Step 1: Define HMM Components

(a) Observations

These are the **words** in the sentence:

- John
- bought
- a
- car
- yesterday

(b) States

Possible POS tags (for simplicity):

- NNP = Proper Noun
- VBD = Verb (Past tense)
- DT = Determiner
- NN = Noun
- RB = Adverb

(c) Transition Probabilities

Define likely POS transitions (conceptually):

- NNP → VBD (Person then action)
- VBD → DT (Verb then determiner)
- DT → NN (Determiner then noun)
- NN → RB (Noun then adverb/time)

(d) Emission Probabilities

Probability that a word is generated by a tag, e.g.:

- P(John | NNP) is high
- P(bought | VBD) is high
- P(a | DT) is high
- P(car | NN) is high

- P(yesterday | RB) is high

Step 2: Apply HMM to Perform Tagging

We tag each word based on **best transition + emission** probability.

Word	**Likely POS Tag**	**Reason**
John	NNP	Proper noun (person name)
bought	VBD	Past tense verb
a	DT	Determiner
car	NN	Common noun
yesterday	RB	Time adverb

Step 3: Final POS Tagged Output

Sentence with POS tags:

John/NNP bought/VBD a/DT car/NN yesterday/RB

Why HMM Works Here?

- Uses **transition probabilities** to model grammar
- Uses **emission probabilities** to model word-tag association
- Selects the best tag sequence using algorithms like **Viterbi**

Conclusion

- **Sequence labeling** assigns meaningful tags to sequences of words.
- It is used for **POS tagging** to assign grammatical categories.
- It plays a major role in **NER** for identifying entities.
- **HMM-based POS tagging** uses probabilities to choose the best tag sequence.

3.Compare and contrast Conditional Random Fields (CRFs) with Hidden Markov Models (HMMs) for sequence labeling tasks. Highlight their strengths and limitations.

(A) Hidden Markov Models (HMMs)

Concept

- HMMs are generative probabilistic models.
- They model the joint probability of sequences:
$$P(X, Y) = P(Y) \cdot P(X \mid Y)$$
where X = observed words, Y = hidden labels.

Key Characteristics

- Uses transition probabilities: $P(y_t \mid y_{t-1})$
- Uses emission probabilities: $P(x_t \mid y_t)$
- Assumes Markov property:
    - Current state depends only on previous state.
- Assumes observations are independent given labels.

(B) Conditional Random Fields (CRFs)

Concept

- CRFs are discriminative probabilistic models.
- They model conditional probability:
$P(Y \mid X)$, directly mapping observations to labels.

Key Characteristics

- Use features from whole input sequence, such as:
    - word identity
    - word prefixes/suffixes
    - capitalization, POS, etc.
- No independence assumptions on observations.

(c) Comparison Table

Aspect	HMM	CRF
1. Model Type	Generative model	Discriminative model
2. Probability Modeled	Models joint probability $P(X, Y)$	Models conditional probability (P(Y
3. Feature Usage	Limited features	Can use rich and many features
4. Observation Independence	Assumes independence between observations	No independence assumption
5. Context Handling	Limited local context	Global context of full sequence
6. Output Dependency	Only depends on previous state	Depends on entire label sequence
7. Training Data	Can use partially labeled data	Needs fully labeled data
8. Performance	Lower accuracy for complex NLP	Higher accuracy in NER & POS
9. Implementation Difficulty	Easier to implement	More complex and computationally heavy
10. Typical Applications	Basic POS tagging	POS, NER, chunking, entity extraction

(D) Strengths

HMM Strengths

- Simple and fast.
- Easy to train using labeled or partially labeled data.
- Works well for simpler tasks.

CRF Strengths

- Better performance on complex tasks like NER.
- Uses multiple features without independence assumptions.
- Reduces tagging errors by considering **global sequence**.

(E) Limitations

HMM Limitations

- Needs strong independence assumptions.
- Struggles with rich contextual features.
- Lower accuracy in real-world NLP.

CRF Limitations

- Training is **computationally expensive**.
- Requires labeled data for supervised training.
- Complexity increases with feature size.

4.Evaluate the effectiveness of named entity recognition (NER) in extracting structured information from unstructured text. Discuss how the performance of an NER system can be assessed.

(A) Introduction to NER

Named Entity Recognition (NER) is an NLP task that identifies and classifies entities in text into predefined categories such as:

- Person (PER)
- Organization (ORG)
- Location (LOC)
- Date, Time, Money, etc.

Example:
Sentence: **"Elon Musk founded SpaceX in 2002 in California."**
NER Output:

- Elon Musk → PER
- SpaceX → ORG
- 2002 → DATE
- California → LOC

(B) Effectiveness of NER in Extracting Structured Information

NER helps convert **unstructured text into structured data**.
It is very useful in:

1. Information Extraction

Extracting key entities from news, web pages, research papers.

2. Search Engines

Entities improve search relevance (e.g., Google Knowledge Graph).

3. Business Intelligence

NER helps extract:

- company names
- product names
- financial entities

4. Healthcare

Extracting diseases, symptoms, drugs from medical text.

5. Social Media Analysis

NER identifies:

- trending people
- events
- places

6. Chatbots & Virtual Assistants

NER helps chatbots understand:

- names
- dates
- locations

7. Document Summarization

NER highlights important entities for summarizing content.

(C) Performance Evaluation of NER Systems

To measure NER performance, we compare model output with **manually annotated (gold) data**.

Common Evaluation Metrics

1. **Precision**

$$Precision = \frac{Correct\ Entities\ Found}{Total\ Entities\ Found}$$

   - Measures accuracy of extracted entities.

2. **Recall**

$$Recall = \frac{Correct\ Entities\ Found}{Total\ True\ Entities}$$

   - Measures coverage (how many true entities were found).

3. **F1-Score**

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

   - Balances precision and recall.

Example Evaluation

Suppose:

- System found 20 entities
- 15 were correct
- Actual correct entities = 25

Then:

- Precision = 15/20 = 0.75
- Recall = 15/25 = 0.60
- F1 = 2*(0.75*0.60)/(0.75+0.60) = 0.666

(D) Challenges & Limitations

- Ambiguous names (e.g., “Amazon” = company or river?)
- Domain-specific entities (medical, legal)
- Spelling variations (social media text)
- Multilingual text

- Nested entities (e.g., Bank of America Tower → ORG + LOC)

Conclusion

- NER is highly effective for converting raw text into structured information.
- It is widely used in search engines, finance, healthcare, and social media analysis.
- Performance is measured using **Precision, Recall, and F1-score**.
- Despite challenges, NER plays a critical role in modern NLP applications.

5.Design a dependency parsing algorithm to analyze the syntactic structure of a sentence. Discuss how you would handle complex sentences and what factors would affect parsing accuracy.

(A) Introduction

Dependency parsing is a method in NLP that analyzes the **grammatical relations between words** in a sentence.
It represents sentences as **directed graphs**, where:

- **Nodes = words**
- **Edges = grammatical dependencies**

Example:
Sentence: *"John eats apples"*
Dependencies:

- *eats* → root
- *John* → subject of *eats*
- *apples* → object of *eats*

(B) Designing a Dependency Parsing Algorithm

Step 1: Input Processing

- Take raw sentence
- Perform tokenization and POS tagging

Example:
Sentence: *"John eats apples"*

Tokens: ["John", "eats", "apples"]
POS tags: NNP, VBZ, NNS

Step 2: Initialize Data Structures

Modern parsers use:

- **Stack** (holds partially processed words)
- **Buffer** (holds remaining input words)
- **Dependency List** (stores head–dependent relations)

Initial state:

- Stack: [ROOT]
- Buffer: [John, eats, apples]
- Dependencies: []

Step 3: Apply Parsing Actions

Typical actions:

1. **SHIFT** → move first buffer word to stack
2. **REDUCE** → remove top of stack
3. **LEFT-ARC** → assign dependency from buffer → stack
4. **RIGHT-ARC** → assign dependency from stack → buffer

Example operations:

1. SHIFT → Stack: [ROOT, John]
2. SHIFT → Stack: [ROOT, John, eats]
3. LEFT-ARC(subject) → eats ← John
4. SHIFT → Stack: [ROOT, eats, apples]
5. RIGHT-ARC(object) → eats → apples

Step 4: Output Dependency Tree

Final dependencies:

- eats (ROOT verb)
- eats ← John (nsubj)
- eats → apples (obj)

(C) Handling Complex Sentences

Complex sentences may include:

- Subordinate clauses
- Embeddings
- Coordination
- Long-distance dependencies

Handling Strategies:

1. **POS tags and morphological features**
    - Helps identify clause boundaries
2. **Clause segmentation**
    - Break long sentences into smaller syntactic units
3. **Handling conjunctions**
   Example: "John and Mary ate apples"
    - Both "John" and "Mary" are conjuncts linked to "ate"
4. **Using statistical or neural models**
    - Neural dependency parsers (e.g., BERT-based) handle ambiguity
5. **Use of pre-trained models**
    - Stanford CoreNLP
    - SpaCy
    - UDPipe

(D) Factors Affecting Parsing Accuracy

1. **Ambiguity in natural language**
    - Example: "Flying planes can be dangerous" (who flies?)

2. **Quality of POS tagging**
    - Incorrect POS tags lead to incorrect dependencies
3. **Domain mismatch**
    - Medical vs legal vs social media text
4. **Sentence length**
    - Longer sentences → more errors
5. **Punctuation**
    - Missing punctuation confuses clause boundaries
6. **Training data size**
    - More annotated data improves accuracy
7. **Morphologically rich languages**
    - Languages like Hindi, Turkish need special handling
8. **Spelling or grammar errors**
    - Common in social media text

6.Define the term "context-free grammar" (CFG) in the context of syntax and parsing. How does it differ from dependency parsing, and what are the primary components of a CFG?

(A) Definition of Context-Free Grammar (CFG)

A **Context-Free Grammar (CFG)** is a set of production rules used to describe the **phrase structure of sentences** in natural language.

CFG generates sentences by applying **rewrite rules**.

Example rule:

S → NP VP

NP → Det N

VP → V NP

Where:

- **S = sentence**
- **NP = noun phrase**

- **VP = verb phrase**

(B) Components of a CFG

A CFG consists of **four main components**:

1. **V (Non-terminals)**
    - Variables representing syntactic categories
    - Examples: S, NP, VP, PP, Det, N
2. **Σ (Terminals)**
    - Actual words in a language
    - Example: “cat”, “saw”, “the”
3. **R (Production Rules)**
    - Rewrite rules like:
      VP → V NP
      NP → Det N
4. **S (Start Symbol)**
    - Starting non-terminal
    - Usually: S = Sentence

(C) Example of CFG Parsing

Sentence: *“the cat sleeps”*

Step-by-step derivation:

S → NP VP

NP → Det N

VP → V

Det → the

N → cat

V → sleeps

Final parse tree shows phrase structure hierarchy.

(D) How CFG Differs from Dependency Parsing

Aspect	Context-Free Grammar (CFG)	Dependency Parsing
Structure Focus	Phrase structure	Word-to-word relations
Representation	Tree with hierarchical constituents	Graph with dependencies
Main Units	NP, VP, PP (phrases)	Head & dependents
Grammar Type	Rule-based	Syntactic + statistical
Output	Constituency tree	Dependency tree
Example Use	Syntax teaching, grammar checking	Information extraction, MT
Handles Free Word Order	Difficult	Better suited
Uses POS Tags	Not directly required	Required
Linguistic Tradition	Chomskyan	Tesnière’s theory
Example Relation	NP contains Det + N	Verb governs subject/object

(E) Applications of CFG

CFGs are used in:

- Language teaching & linguistic studies
- Compiler design (programming languages)
- Grammar checking tools
- Syntax-based machine translation

(F) Strengths & Limitations of CFG

Strengths

- Simple rule-based framework
- Good for hierarchical structure
- Useful for formal languages

Limitations

- Cannot easily handle:
  - Ambiguity
  - Free word order languages
  - Long-distance dependencies

UNIT-IV

1. **Explain the concept of sentiment analysis in text classification. Discuss the role of classifiers in sentiment analysis and provide examples of typical sentiment labels used in the analysis.**

1. Sentiment Analysis in Text Classification (10 Marks)

(A) Concept of Sentiment Analysis

- **Sentiment Analysis** is a Natural Language Processing (NLP) task that identifies and categorizes opinions or emotions expressed in text.
- It helps determine whether a piece of text is **positive, negative, or neutral**.
- It is widely used in areas like:
  - Social media monitoring
  - Customer feedback analysis
  - Product reviews
  - Public opinion mining

Example:

Text: *"The camera quality of this phone is amazing!"*
Sentiment: **Positive**

Text: *"The food was cold and tasteless."*
Sentiment: **Negative**

So, sentiment analysis converts unstructured text into **meaningful emotional insights**.

(B) Sentiment Analysis as a Text Classification Task

- Sentiment analysis is performed as a **text classification** task where text samples (sentences, reviews, tweets, etc.) are assigned sentiment labels.
- Steps include:
  1. **Text Preprocessing**
     - Tokenization, stop word removal, stemming, etc.
  2. **Feature Extraction**
     - Bag-of-Words, TF-IDF, or word embeddings (Word2Vec, GloVe)
  3. **Model Training**
     - Using machine learning or deep learning models
  4. **Classification**
     - Assigning sentiment category to new, unseen text

(C) Role of Classifiers in Sentiment Analysis

Classifiers are algorithms that **learn patterns** from labeled training data and **predict sentiment labels** for new text.

Common Classifiers Used:

1. **Naive Bayes Classifier**
   - Uses probability to classify text.
   - Works well for short messages and reviews.
2. **Support Vector Machines (SVM)**
   - Handles high-dimensional text data.
   - Good accuracy for sentiment tasks.
3. **Logistic Regression**
   - Simple and effective for binary sentiment classification.
4. **Random Forest / Decision Trees**
   - Combine multiple decision trees to improve accuracy.
5. **Deep Learning Models**
   - LSTM, GRU, CNN, Transformers (BERT)

    - Handle context and sequence relationships better.

Role Summary:

- Classifiers **recognize patterns** in text (positive/negative words, polarity, intensity).
- They **learn from training data** and **predict sentiment** for new inputs.
- They convert qualitative human opinions into **quantifiable outputs**.

(D) Typical Sentiment Labels Used

Sentiment labels vary depending on the task. Some common label sets are:

(1) Binary Sentiment Labels

- **Positive**
- **Negative**

Example:

- “I love this movie!” → Positive
- “This product is terrible.” → Negative

(2) Ternary Sentiment Labels

- **Positive**
- **Negative**
- **Neutral**

Example:

- “The phone is okay.” → Neutral

(3) Fine-Grained Sentiment Labels

- **Very positive**
- **Positive**
- **Neutral**
- **Negative**
- **Very negative**

Used in detailed review systems (e.g., movie reviews).

(4) Emotion Categorization (Advanced)

- **Happy, Angry, Sad, Disgust, Fear, Surprise**
  Example:
- “I am so angry about this delay!” → Angry

Conclusion

- Sentiment analysis is a key NLP task that classifies text into emotional categories.
- It is widely used in real-world applications like customer feedback analysis, social media monitoring, and marketing.
- Classifiers play an important role by **learning patterns** from data and **predicting** sentiment for unseen text.
- Typical labels range from simple binary labels (Positive/Negative) to multi-class emotion categories.

---

2. **Given a set of text data labeled with sentiment (positive/negative), demonstrate how you would train a logistic regression model for sentiment classification. Outline the steps, including preprocessing and model evaluation**.

(A) Problem Description

We are given text data labeled as:

- **Positive**
- **Negative**

Goal: Train a **Logistic Regression model** to predict sentiment of new text.

(B) Step-by-Step Approach

Step 1: Collect and Prepare Dataset

Example dataset structure:

Text	Sentiment
"I love this product"	Positive

Text	Sentiment
"This is the worst service"	Negative

We divide the dataset into:

- **Training set** (e.g., 80%)
- **Test set** (e.g., 20%)

Step 2: Text Preprocessing

Preprocessing ensures text becomes clean and uniform.

Common preprocessing operations:

1. **Lowercasing**
   "This phone is Good" → "this phone is good"
2. **Tokenization**
   ["this", "phone", "is", "good"]
3. **Stop-word Removal**
   Remove words like: *the, is, are*
4. **Stemming/Lemmatization**
   "playing" → "play"
5. **Punctuation Removal**
   "," "." "!" etc. removed

Resulting cleaned text becomes easier for model to use.

Step 3: Feature Extraction

Machine learning models **cannot understand raw text**.
So we convert text into numerical vectors.

Two common methods:

(A) Bag of Words (BoW)

- Counts word occurrences

(B) TF-IDF (Term Frequency–Inverse Document Frequency)

- Gives importance to less frequent words

Example TF-IDF vector:
“I love this product” → [0.5, 0.3, 0.7, ...]

Step 4: Model Selection

We choose **Logistic Regression**, a common linear classifier for binary tasks.

Why Logistic Regression?

- Suitable for **binary sentiment** (positive/negative)
- Fast and interpretable
- Works well with text feature vectors

Step 5: Training the Model

- Input: TF-IDF vectors from training set
- Output: Learned parameters (weights)

Model learns patterns such as:

- Words like **“love”, “great”, “excellent”** → Positive
- Words like **“bad”, “worst”, “hate”** → Negative

Mathematically, Logistic Regression computes:

$$P(y=1|x) = \frac{1}{1+e^{-(w^T x+b)}}$$

Where:

- (x) = feature vector for text
- (w), (b) = model parameters
- Output ≈ probability of **positive sentiment**

Step 6: Model Evaluation

We evaluate on test dataset after training.

Common evaluation metrics:

(1) Accuracy

$$Accuracy = \frac{Correct\ Predictions}{Total\ Samples}$$

(2) Precision

- Measures correctness for **positive** predictions

(3) Recall

- Measures how many actual positives were detected

(4) F1-Score

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

(5) Confusion Matrix

Shows:

- True Positive (TP)
- False Positive (FP)
- True Negative (TN)
- False Negative (FN)

Step 7: Final Prediction

After evaluation, the model can classify new text:

Example inputs:

- “The movie was fantastic!” → **Positive**
- “The product stopped working in 2 days.” → **Negative**

(C) Summary

To train a Logistic Regression model for sentiment classification:

1. **Prepare labeled dataset**

2. **Preprocess text** (cleaning, tokenizing, stop-word removal)
3. **Convert text to numeric vectors** (TF-IDF/BoW)
4. **Train Logistic Regression on training data**
5. **Evaluate using accuracy, precision, recall, F1-score**
6. **Predict sentiment for new text**

---

3. **Compare the use of logistic regression with multinomial logistic regression for text classification. Discuss the advantages and disadvantages of using multinomial logistic regression over binary logistic regression for multi-class classification problems.**

Introduction

Text classification is the task of assigning categories (like spam/ham, positive/negative, topic labels, etc.) to text documents. Logistic Regression is a popular linear classifier used for text classification due to its simplicity, speed, and good performance with high-dimensional sparse data (like bag-of-words or TF-IDF).

There are two forms of logistic regression commonly used:

1. **Binary Logistic Regression** → Used when there are only **two classes** (e.g., positive vs negative).
2. **Multinomial Logistic Regression** → Used when there are **more than two classes** (e.g., classifying news into politics, sports, entertainment, business).

Binary vs Multinomial Logistic Regression (Comparison Table)

Aspect	**Binary Logistic Regression**	**Multinomial Logistic Regression**
Number of Classes	2 classes only	3 or more classes
Output Types	Probability of class 1 vs class 0	Probability distribution over multiple classes
Decision Function	Sigmoid function	Softmax function
Goal	Separate data into two groups	Assign data to one among several groups
Loss Function Used	Binary Cross-Entropy Loss	Multiclass Cross-Entropy Loss

Aspect	Binary Logistic Regression	Multinomial Logistic Regression
Model Complexity	Relatively simple	More complex
Training Strategy	Direct training for 2 classes	One-vs-Rest or Softmax based
Use in NLP	Sentiment analysis (pos/neg), spam detection	Topic labeling, emotion classification
Performance	Very good for binary tasks	Better for true multi-class tasks
Interpretability	Very easy to interpret	Slightly harder to interpret due to multiple dimensions

---

Explanation (Theory)

1. Logistic Regression for Text Classification

- Text documents are converted to numeric features using **Bag of Words, TF-IDF, or word embeddings**.
- Logistic regression learns weights for features indicating how strongly a word pushes the classification toward a certain label.

---

2. Binary Logistic Regression

- Used when the target variable has **only two classes**.
- Uses the **sigmoid function**:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

- Produces probability between 0 and 1.
- Example tasks:
  - Spam vs Not spam
  - Positive vs Negative sentiment

---

3. Multinomial Logistic Regression

- Used when target variable has **three or more classes**.
- Uses **softmax function** to compute probabilities:

$$P(y = i|x) = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$$

- Assigns text to the class with the highest probability.

Advantages of Multinomial Logistic Regression over Binary Logistic Regression

1. **Handles Multi-Class Problems Naturally**
   No need to convert multi-class problems into multiple binary problems.
2. **Better Probability Distribution**
   Gives probability for each class (useful for ranking outputs).
3. **More Efficient When Classes are Many**
   Reduces need for multiple binary classifiers like One-vs-Rest.
4. **Better Accuracy in True Multi-Class Settings**
   Captures relationships among multiple classes.

Disadvantages of Multinomial Logistic Regression

1. **More Compute-Intensive**
   Requires more processing for softmax and multiple output weights.
2. **More Complex to Implement and Interpret**
   Weight interpretation becomes harder when many classes exist.
3. **Needs More Training Data**
   Each class must be represented sufficiently for good accuracy.

Use Cases

Task	**Example**	**Type**
Sentiment (2-class)	positive/negative	Binary Logistic Regression
Topic Classification	politics/sports/business	Multinomial Logistic Regression

Task	Example	Type
Emotion Classification	happy/sad/angry	Multinomial Logistic Regression
Spam Detection	spam/ham	Binary Logistic Regression

Q4. Evaluate the impact of regularization techniques (L1 and L2) on text classification models. Discuss how regularization helps prevent overfitting and the trade-offs involved in using different regularization methods.

Introduction

Text classification models such as Logistic Regression, SVM, Naive Bayes, and Neural Networks often work with **high-dimensional feature spaces**, especially when using Bag-of-Words or TF-IDF representations. Such models are at high risk of **overfitting**, meaning they perform very well on training data but poorly on unseen test data.

Regularization is a technique used to **reduce overfitting by penalizing large weights** in the model. The two most common types are:

- **L1 regularization (Lasso)**
- **L2 regularization (Ridge)**

Overfitting in Text Classification

Overfitting occurs when:

- The model learns **noise**, **rare words**, or **irrelevant patterns**
- It performs well on training data but poorly on test data

Example scenario:

- Features include thousands of rare words → model memorizes them → poor generalization.

Regularization helps the model learn **general patterns**, not just memorize data.

L1 (Lasso) Regularization

Definition

L1 regularization adds the **absolute value of weights** to the loss function.

$$L = Loss + \lambda \sum |w_i|$$

Impact on Text Classification

- Encourages **sparsity** → makes many weights **exactly zero**
- Performs **feature selection** → useful in high-dimensional text data
- Reduces model complexity

Advantages

- Helps in **feature selection**
- Makes models simpler and faster
- Works well when many features are irrelevant

Disadvantages

- Can be unstable when features are correlated
- Optimization is slower than L2

L2 (Ridge) Regularization

Definition

L2 regularization adds the **square of the weights** to the loss function.

$$L = Loss + \lambda \sum w_i^2$$

Impact on Text Classification

- Shrinks large weights but **does not make them zero**
- Distributes weights across correlated features
- Helps stabilize the model

Advantages

- Works well with correlated features
- Improves generalization
- Faster optimization (convex and smooth)

Disadvantages

- Does not perform explicit feature selection
- Model may remain large with many small weights

How Regularization Prevents Overfitting

Regularization reduces overfitting by:

1. **Penalizing large weights** → discourages the model from relying on rare or noisy features
2. **Reducing model complexity** → simpler models generalize better
3. **Improving stability** → model behaves well on unseen data
4. **Controlling variance** → reduces fluctuations in predictions

When λ (regularization strength) is chosen properly:

- Training error increases slightly
- Test error decreases significantly

This is called the **bias–variance tradeoff**.

Trade-offs Involved

1. Choosing Between L1 and L2

Aspect	**L1 Regularization**	**L2 Regularization**
Weight effect	Forces weights to zero	Shrinks weights smoothly
Feature Selection	Yes	No
Sparse models	Yes	No
Correlated features	Keeps one, discards others	Splits weights across them
Computation	Slower (non-smooth)	Faster (smooth)

---

2. Choosing Regularization Strength (λ)

- If **λ is too small** → weak regularization → overfitting persists
- If **λ is too large** → too much regularization → underfitting occurs

So, correct tuning is required using:

- **Cross-validation**
- **Validation sets**

Usage in Real Text Classification Models

- **L1 Regularization**
  - Used in sparse linear models like **L1-Logistic Regression**
  - Selects important words/features
- **L2 Regularization**
  - Used in **SVM, Logistic Regression, Neural Networks**
  - Common in deep learning and embeddings

Conclusion

Regularization plays a very important role in text classification, especially when dealing with high-dimensional data. Both **L1 and L2 regularization help prevent overfitting** but impact the model in different ways:

- **L1** encourages **sparse models** and **feature selection**.
- **L2** gives **stable solutions** and works better with **correlated features**.

Choosing the right method and strength (λ) gives a better balance between **bias and variance**, improving the **generalization performance** of text classification models.

Q5 Design a text classification model for detecting spam emails. Describe the preprocessing steps, feature extraction techniques, classifier selection, and evaluation metrics you would use.

Introduction

Spam email detection is a common **binary text classification task** where emails are classified as:

- **Spam** → unwanted/malicious emails
- **Ham** → legitimate emails

The goal is to automatically filter spam emails using **NLP and machine learning**.

Step 1: Preprocessing

Before feeding text to a model, emails need to be cleaned and standardized. Common preprocessing steps include:

1. **Lowercasing**
   - Convert all words to lowercase: "FREE Offer!" → "free offer"
2. **Tokenization**
   - Split text into words or tokens:
     "Get free money now" → ["get", "free", "money", "now"]
3. **Removing Stop Words**
   - Remove common words with little meaning: "the, is, at, and"
4. **Removing Punctuation and Numbers**
   - Clean text of symbols, digits, and email-specific artifacts
5. **Stemming / Lemmatization**
   - Reduce words to base form: "offering" → "offer"
6. **Handling URLs and Email Addresses**
   - Replace links with a token like <URL>
   - Replace email addresses with <EMAIL>

Step 2: Feature Extraction

Emails must be converted into **numerical features** for the model.

(A) Bag-of-Words (BoW)

- Represents text as a vector of word counts.
- Example: "free money" → [0, 1, 1, 0, …]

(B) TF-IDF (Term Frequency–Inverse Document Frequency)

- Gives **importance to rare words** like "lottery" or "win"

- Example formula:

$$TF - IDF(t,d) = TF(t,d) \times \log(\frac{N}{df(t)})$$

(C) Optional Features

- Word n-grams: bigrams/trigrams to capture phrases like "free money"
- Special symbols or patterns: exclamation marks, all-caps words
- Email metadata: sender domain, subject length, etc.

Step 3: Classifier Selection

For spam detection (binary classification), common classifiers include:

1. **Naive Bayes (Multinomial)**
   - Works well with word counts
   - Assumes word independence
   - Fast and simple
2. **Logistic Regression**
   - Effective with TF-IDF features
   - Outputs probability of spam
3. **Support Vector Machine (SVM)**
   - Good for high-dimensional sparse text features
4. **Random Forest / Gradient Boosting**
   - Ensemble methods for better accuracy
5. **Deep Learning (Optional)**
   - LSTM, CNN, or Transformers for large datasets

Step 4: Model Training

- Split dataset into **training (80%)** and **test (20%)** sets

- Train classifier on training data using features extracted
- Tune hyperparameters (e.g., regularization strength, n-gram range) using **cross-validation**

Step 5: Model Evaluation

To measure the performance of the spam classifier, use:

1. **Accuracy**

$$Accuracy = \frac{CorrectPredictions}{TotalEmails}$$

2. **Precision**
    - How many predicted spam emails are actually spam

$$Precision = \frac{TP}{TP + FP}$$

3. **Recall** (Sensitivity)

- How many actual spam emails were correctly detected

$$Recall = \frac{TP}{TP + FN}$$

4. **F1-Score**

- Harmonic mean of precision and recall

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

5. **Confusion Matrix**

	Predicted Spam	Predicted Ham
Actual Spam	TP	FN
Actual Ham	FP	TN

- Metrics like **precision and recall** are more important than accuracy due to class imbalance (spam is usually less than ham).

Step 6: Deployment

- Once trained and evaluated, the model can be used to **filter incoming emails**.
- Probabilistic classifiers (Logistic Regression, Naive Bayes) allow threshold adjustment to control sensitivity.
- Continuous updates improve performance as spam evolves.

Conclusion

A complete text classification pipeline for spam detection includes:

1. **Preprocessing:** cleaning, tokenizing, stop word removal, stemming
2. **Feature Extraction:** TF-IDF or Bag-of-Words, n-grams, metadata
3. **Classifier Selection:** Naive Bayes, Logistic Regression, SVM
4. **Model Training:** split data, tune hyperparameters
5. **Evaluation:** accuracy, precision, recall, F1-score
6. **Deployment:** real-time email filtering

Result: A reliable spam detection system that can automatically filter unwanted emails, improve efficiency, and reduce user exposure to malicious content.

Q6 Define the concept of cross-entropy loss in the context of text classification. Explain how gradient descent is used to minimize the cross-entropy loss function during the training of a sentiment analysis model.

(A) Concept of Cross-Entropy Loss

- In **text classification**, such as sentiment analysis, the goal is to predict the **correct class** for a given text.
  Example:
  Text: "I love this movie!" → Sentiment: **Positive**
- **Cross-entropy loss** measures the difference between:
  - The **predicted probability distribution** from the model, and
  - The **true distribution** (actual label).
- Formula for binary classification:

$$L = -\frac{1}{N}\sum_{i=1}^{N}[\,y_i\log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)]$$

Where:

- N= number of training examples
- y_i= true label (0 or 1)
- $\hat{y}_i$= predicted probability of class 1
- For **multi-class classification** (softmax output):

$$L = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log(\hat{y}_{i,c})$$

Where:

- C= number of classes
- $y_{i,c}$= 1 if example $i$belongs to class c, 0 otherwise
- $\hat{y}_{i,c}$= predicted probability of class c

Intuition:

- If the model predicts probability close to 1 for the correct class → **loss is small**
- If it predicts probability close to 0 for the correct class → **loss is large**
- Minimizing cross-entropy ensures the model predicts **accurate probabilities**.

(B) Using Gradient Descent to Minimize Cross-Entropy Loss

- **Gradient Descent** is an optimization algorithm used to **update model parameters** (weights) to minimize the loss function.

Step 1: Compute Gradient

- Calculate the derivative of the cross-entropy loss with respect to each model parameter w:

$$\frac{\partial L}{\partial w} = \frac{1}{N}\sum_{i=1}^{N}(\,\hat{y}_i - y_i)x_i$$

Where $x_i$are the features of example i.

- The gradient tells **how much and in which direction to adjust weights** to reduce the loss.

Step 2: Update Weights

- Update the weights iteratively:

$$w \leftarrow w - \eta \frac{\partial L}{\partial w}$$

Where:

- η= learning rate (step size)
- Repeat until convergence (loss stops decreasing significantly).

Step 3: Training Loop for Sentiment Analysis

1. Convert text into numerical features (e.g., TF-IDF, word embeddings)
2. Initialize weights randomly
3. For each epoch:
   a. Forward pass → compute predicted probabilities
   b. Compute **cross-entropy loss**
   c. Backward pass → compute **gradients**
   d. Update weights using **gradient descent**
4. Repeat until the model achieves satisfactory accuracy on training/validation data

---

(C) Summary

- **Cross-entropy loss** measures how well the model's predicted probabilities match the actual labels.
- **Gradient descent** is used to iteratively **adjust model weights** to minimize this loss.
- In sentiment analysis:
  - Correctly predicts "positive" for positive reviews and "negative" for negative reviews
  - Ensures the model **generalizes well** to new, unseen texts.