Welcome to GITAM

Dr. Justin Varghese

Department of CSE

GITAM
DEEMED TO BE UNIVERSITY

NATURAL LANGUAGE PROCESSING

GITAM
DEEMED TO BE UNIVERSITY

Course Description:

- Understand the leading trends and systems in natural language processing.
- Understand the concepts of morphology, syntax, semantics and pragmatics of the language
- and that they are able to give the appropriate examples that will illustrate the abovementioned concepts.
- Recognize the significance of pragmatics for natural language understanding.
- Describe the application based on natural language processing and to show the points of syntactic, semantic and pragmatic processing

GITAM
DEEMED TO BE UNIVERSITY

Course Outcomes:

At the end of the course, the student will be able to

- Understand approaches to syntax and semantics in NLP(L2)
- Apply approaches to discourse, generation, dialogue and summarization within NLP(L3)
- Analyze current methods for statistical approaches to machine translation(L4)
- Evaluate machine learning techniques used in NLP, including hidden Markov models and
- probabilistic context-free grammars as applied within NLP(L4)

GITAM
DEEMED TO BE UNIVERSITY

Course Content Unit 1:

Introduction – Models -and Algorithms - -Regular Expressions, Finite State Automata, Morphology,

Morphological Parsing, Stemming- Porter Stemmer Algorithm, Text Normalization, Edit Distance

Introduction to Natural Language Processing (NLP)

Natural Language Processing (NLP)

Natural Language Processing (NLP) is a branch of Artificial Intelligence (AI) that focuses on enabling computers to understand, interpret, and generate human language in a meaningful and intelligent manner.

GITAM
DEEMED TO BE UNIVERSITY

Natural Language Processing (NLP)

- Natural Language Processing (NLP) is a subfield of Artificial Intelligence (AI).
- It enables computers to understand, interpret, and generate human language.
- NLP combines concepts from computer science, linguistics, statistics, and machine learning.
- Human language is complex, ambiguous, and context-dependent, making NLP a challenging task.

Natural Language Processing (NLP)

- NLP converts unstructured text and speech data into structured representations.
- Common NLP processes include tokenization, syntactic analysis, semantic analysis, and language generation.
- NLP helps bridge the gap between human communication and machine understanding.

Tokenization(NLP)

Tokenization is a fundamental preprocessing step in Natural Language Processing (NLP) that involves breaking down text into smaller units called tokens.

- These tokens can be words, subwords, sentences, or characters, depending on the application.

Purpose of Tokenization

- Converts raw text into a form that machines can process
- Helps in analyzing the structure and meaning of text
- Serves as the first step for most NLP tasks

GITAM
DEEMED TO BE UNIVERSITY

Tokenization(NLP)

Types of Tokenization

Word Tokenization: Splits text into individual words

Example:“NLP is powerful” → [NLP, is, powerful]

Sentence Tokenization: Divides text into sentences

Example: “I love NLP.

It is useful.” →[I love NLP., It is useful.]

Subword Tokenization: Breaks words into smaller meaningful units

Example:“unhappiness” → [un, happiness]

(Used in models like BERT and GPT)

GITAM
DEEMED TO BE UNIVERSITY

Tokenization(NLP)

Types of Tokenization

Character Tokenization: Splits text into individual characters

Example:“AI” → [A, I]

Importance of Tokenization

- Preserves the structure of language
- Improves accuracy of NLP models
- Enables further processing such as POS tagging and parsing
- Handles large vocabularies efficiently (subword tokenization)

Syntactic analysis (NLP)

- Syntactic analysis is a stage in Natural Language Processing (NLP) that focuses on analyzing the grammatical structure of a sentence.
- It determines how words are arranged and how they relate to each other according to the rules of grammar.

Syntactic analysis (NLP)

Purpose of Syntactic Analysis

- Identifies the grammatical structure of sentences
- Determines relationships between words (subject, object, modifier, etc.)
- Ensures sentences follow correct grammatical rules

What Syntactic Analysis Does

- Analyzes word order
- Assigns Part-of-Speech (POS) tags
- Builds parse trees or dependency structures
- Checks grammatical correctness

GITAM
DEEMED TO BE UNIVERSITY

Syntactic analysis (NLP)

Types of Syntactic Analysis

- Constituency Parsing: Breaks sentences into phrases (noun phrase, verb phrase, etc.)

  Example: "The cat sleeps" → [NP The cat] [VP sleeps]
- Dependency Parsing: Identifies dependencies between words

  Example: sleeps → subject → cat

GITAM
DEEMED TO BE UNIVERSITY

Syntactic analysis (NLP)

Sentence: “The boy eats an apple”

- The → Determiner
- boy → Noun (subject)
- eats → Verb
- apple → Noun (object)

Syntactic analysis (NLP)

Importance of Syntactic Analysis

- Helps in understanding sentence meaning
- Essential for machine translation and question answering
- Improves accuracy of NLP applications
- Supports semantic analysis Applications
- Grammar checking
- Machine translation
- Information extraction
- Voice assistants
- Chatbots

Semantic analysis (NLP)

- Semantic analysis is a stage in Natural Language Processing (NLP) that focuses on understanding the meaning of words, phrases, and sentences rather than just their grammatical structure.
- It helps machines interpret what a sentence actually means.

Semantic analysis (NLP)

Purpose of Semantic Analysis

- Determines the meaning and intent of text
- Resolves ambiguity in language
- Enables machines to understand context and concepts

What Semantic Analysis Does

- Identifies relationships between words and concepts
- Interprets sentence meaning beyond grammar
- Handles polysemy (multiple meanings of a word)Supports context-based understanding

GITAM
DEEMED TO BE UNIVERSITY

Semantic analysis (NLP)

Key Techniques Used

- **Word Sense Disambiguation:**

  Determines the correct meaning of a word based on context

  Example: *"bank"* (river bank vs financial bank)

- **Named Entity Recognition (NER):**

  Identifies entities such as people, locations, organizations

  Example: *"India"* → *Location*

- **Semantic Role Labeling:**

  Identifies roles like who did what to whom

  Example: *"John ate an apple"*

  - John → Agent
  - apple → Object

- **Word Embeddings:**

  Represent words with semantic meaning in vector space

GITAM
DEEMED TO BE UNIVERSITY

Semantic analysis (NLP)

Sentence: *"The man opened the bank account"*

Semantic analysis identifies **bank** as a *financial institution*, not a riverbank.

Importance of Semantic Analysis

- Improves accuracy of NLP systems
- Essential for machine translation and question answering
- Enables sentiment and intent detection
- Supports intelligent chatbots and search engines

Applications

- Sentiment analysis
- Question answering systems
- Information retrieval
- Chatbots and virtual assistants
- Text summarization

GITAM
DEEMED TO BE UNIVERSITY

Language generation (NLP)

Language generation is a task in Natural Language Processing (NLP) that focuses on automatically producing meaningful, coherent, and grammatically correct human language from structured data or learned representations.

Language generation(NLP)

Purpose of Language Generation

- Converts machine-understood data into **human-readable text**
- Enables natural and interactive communication between humans and machines
- Produces fluent and context-aware responses

What Language Generation Does

- Selects appropriate words and sentence structures
- Maintains grammatical correctness and coherence
- Preserves meaning and context
- Adapts tone and style when required

Language generation(NLP)

Techniques Used in Language Generation

- **Rule-Based Generation:**

  Uses predefined templates and grammar rules
- **Statistical Methods:**

  N-gram language models generate text based on probability
- **Neural Network Models:**
  - Recurrent Neural Networks (RNN)
  - LSTM and GRU
  - Transformer-based models (GPT, T5, BART)

Example

Input data:

`{Temperature: 35°C}`

Generated text:

"The temperature today is 35 degrees Celsius."

GITAM
DEEMED TO BE UNIVERSITY

Language generation(NLP)

Applications

- Chatbots and virtual assistants
- Machine translation
- Text summarization
- Report generation
- Story and content generation

Importance of Language Generation

- Enhances user interaction
- Automates content creation
- Makes AI systems more natural and user-friendly

GITAM
DEEMED TO BE UNIVERSITY

Natural Language Processing (NLP)

It is widely used in applications such as:

- Search engines
- Chatbots and virtual assistants
- Machine translation
- Sentiment analysis
- Spam detection
- Text summarization
- Advances in deep learning and transformer-based models have significantly improved NLP performance.
- NLP enables more natural, efficient, and intelligent human–computer interaction.

GITAM
DEEMED TO BE UNIVERSITY

NLP Models

An NLP model is a mathematical or computational framework trained to perform language-related tasks such as text classification, sentiment analysis, machine translation, and question answering.

NLP models can be broadly classified into:

a) Rule-Based Models
b) Statistical Models
c) Machine Learning Models
d) Deep Learning Models

GITAM
DEEMED TO BE UNIVERSITY

Rule-Based NLP Models

Rule-based models rely on manually written linguistic rules.

Examples include regular expressions and grammar-based parsers.

Rule-Based NLP Models

Rule-based NLP models are systems that process natural language using manually written rules created by humans (linguists or domain experts).
These rules define how text should be analyzed, interpreted, or transformed based on fixed patterns.
They do not learn from data like machine learning models.
Instead, they rely on if–then logic, grammars, dictionaries, and regular expressions.

Rule-Based NLP Models

Key Components of Rule-Based NLP

- Rules – Hand-crafted instructions (if–else conditions)
- Lexicons / Dictionaries – Lists of words with meanings or tags
- Pattern Matching – Often using regular expressions
- Grammar Rules – Define sentence structure (syntax)

Rule-Based NLP Models

Simple Example 1: Rule-Based Sentiment Analysis

Task: Identify whether a sentence is *positive* or *negative*.

Rules

- If the sentence contains words like **good, excellent, happy** → Positive
- If the sentence contains words like **bad, terrible, sad** → Negative

Input

> "The movie was excellent and enjoyable"

Rule Execution

- Word *excellent* ∈ positive list

  ➡️ **Output: Positive sentiment**

GITAM
DEEMED TO BE UNIVERSITY

Rule-Based NLP Models

Simple Example 2: Rule-Based Named Entity Recognition (NER)

Task: Identify person names.

Rule

- If a word starts with a **capital letter** and follows titles like *Mr.*, *Dr.*, *Prof.* → Person

Input

> "Dr. Sharma is teaching AI"

Output

- **Dr. Sharma → PERSON**

GITAM
DEEMED TO BE UNIVERSITY

Rule-Based NLP Models

Simple Example 3: Rule-Based Chatbot

Task: Answer basic user queries.

Rules

```
text

IF user_input contains "hello"
   reply = "Hi! How can I help you?"
IF user_input contains "exam date"
   reply = "The exam is scheduled for next Monday."
```

Input

> "Hello, what is the exam date?"

Output

> "Hi! The exam is scheduled for next Monday."

GITAM
DEEMED TO BE UNIVERSITY

Rule-Based NLP Models

Advantages of Rule-Based NLP

- Easy to understand and explain
- No training data required
- High accuracy for small, well-defined tasks
- Deterministic (same input → same output)

Limitations of Rule-Based NLP

- Hard to scale for complex language
- Cannot handle ambiguity well
- Requires extensive manual effort
- Breaks when language patterns change

Rule-Based NLP Models

When Rule-Based Models Are Used

- Simple chatbots
- Grammar checkers
- Information extraction in fixed domains
- Form validation systems
- Early NLP systems (before deep learning)

Hence

Rule-based NLP models use human-written rules to process language. They are simple, transparent, and effective for narrow tasks but struggle with complex, real-world language.

GITAM
DEEMED TO BE UNIVERSITY

Statistical NLP Models

- Statistical NLP models analyze natural language using probability and statistics learned from data instead of hand-written rules.
- They rely on the idea that language patterns can be captured by observing word frequencies and probabilities in large text corpora.

Examples:

- N-gram language models
- Hidden Markov Models (HMM)

They handle uncertainty better than rule-based models but require large datasets.

Statistical NLP Models

- Statistical NLP models analyze natural language using probability and statistics learned from data instead of hand-written rules.
- They rely on the idea that language patterns can be captured by observing word frequencies and probabilities in large text corpora.

Examples:

- N-gram language models
- Hidden Markov Models (HMM)

They handle uncertainty better than rule-based models but require large datasets.

Statistical NLP Models

Examples of Statistical NLP Models

- N-gram Language Models
- Hidden Markov Models (HMM)
- Naïve Bayes Classifier
- Maximum Entropy (MaxEnt) Models
- Probabilistic Context-Free Grammar (PCFG)
- Latent Semantic Analysis (LSA)

Statistical NLP Models

What is an N-Gram Model?

An N-gram model is a statistical language model that predicts the next word based on the previous (N−1) words.

- Unigram (1-gram): considers 1 word at a time
- Bigram (2-gram): considers previous 1 word
- Trigram (3-gram): considers previous 2 words

1. N-gram Language Model

Explanation (2 sentences):

An N-gram model predicts a word based on the probabilities of the previous *n–1* words in a sequence. It captures local context using word frequency statistics from a corpus.

Task: Next-word prediction

Sentence:

"I love ___"

The model computes probabilities from word sequences and selects the most frequent continuation.

Output:

"NLP"

GITAM
DEEMED TO BE UNIVERSITY

Statistical NLP Models

What is an N-Gram Model?

An N-gram model is a statistical language model that predicts the next word based on the previous (N−1) words.

- Unigram (1-gram): considers 1 word at a time
- Bigram (2-gram): considers previous 1 word
- Trigram (3-gram): considers previous 2 words

1. N-gram Language Model

Explanation (2 sentences):

An N-gram model predicts a word based on the probabilities of the previous *n–1* words in a sequence. It captures local context using word frequency statistics from a corpus.

Task: Next-word prediction

Sentence:

"I love ___"

$$P(w_n \mid w_1, w_2, \ldots, w_{n-1}) \approx P(w_n \mid w_{n-(N-1)}, \ldots, w_{n-1})$$

The model computes probabilities from word sequences and selects the most frequent continuation.

Output:

"NLP"

GITAM
DEEMED TO BE UNIVERSITY

Statistical NLP Models

Numerical Example (Bigram)

Training data:

```
I like coffee
I like tea
```

Counts:

- (I, like) → 2
- (like, coffee) → 1
- (like, tea) → 1

Probability:

$$P(like|I) = 2/2 = 1$$

$$P(coffee|like) = 1/2$$

Statistical NLP Models

2. Hidden Markov Model (HMM)

Explanation (2 sentences):

Hidden Markov Models represent sequences using hidden states and observable outputs with transition and emission probabilities. They are widely used for sequence labeling tasks such as POS tagging.

Task: Part-of-Speech tagging

Sentence:

"Book a ticket"

HMM evaluates possible tag sequences and chooses the most probable one.

Output:

Book/VERB a/DET ticket/NOUN

Statistical NLP Models

Hidden Markov Models (HMM)

Given a sentence with words:

$$W = (w_1, w_2, w_3, \ldots, w_n)$$

The goal of a language model is to compute:

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

Using the **chain rule of probability**:

$$\boxed{P(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(w_i \mid w_1, w_2, \ldots, w_{i-1})}$$

🔍 Meaning

- Probability of a full sentence
- Is broken into **next-word prediction**
- Each word depends on all previous words

Statistical NLP Models

Hidden Markov Models (HMM)

Sentence:

```arduino
"I love NLP"
```

$$P(\text{I love NLP}) = P(\text{I}) \times P(\text{love} \mid \text{I}) \times P(\text{NLP} \mid \text{I love})$$

So the model keeps asking:

> "What is the probability of the **next word**, given what I've seen so far?"

Statistical NLP Models

3. Naïve Bayes Classifier

Explanation (2 sentences):

Naïve Bayes is a probabilistic classifier based on Bayes' theorem with an assumption of feature independence. It calculates the probability of each class and selects the most likely one.

Task: Sentiment Analysis

Sentence:

"This movie is good"

The model computes class probabilities using word occurrences.

Output:

Positive sentiment

2. Bayes' Theorem (Core Formula)

$$P(C \mid X) = \frac{P(X \mid C)\,P(C)}{P(X)}$$

Where:

- C = class (Spam / Not Spam, Positive / Negative)
- X = document (set of words)

GITAM
DEEMED TO BE UNIVERSITY

Statistical NLP Models

A **Maximum Entropy (MaxEnt) model** is a **probabilistic classification model** used in NLP that chooses the probability distribution with the **highest entropy** (i.e., most uniform / least biased) **subject to known constraints from data**.

4. Maximum Entropy (MaxEnt) Model

Explanation (2 sentences):

The Maximum Entropy model selects the probability distribution that satisfies feature constraints while remaining as uniform as possible. It can use multiple overlapping features without assuming independence.

Task: Text Classification

Sentence:

"You have won a free prize"

The model assigns weights to features and predicts the most probable class.

Output:

Spam email

3. Core Probability Formula

$$P(y \mid x) = \frac{1}{Z(x)} \exp\left(\sum_i \lambda_i f_i(x, y)\right)$$

Where:

- y → class/label (e.g., POS tag, sentiment)
- x → input (sentence or features)
- $f_i(x, y)$ → feature functions (binary or real-valued)
- λ_i → learned weights
- $Z(x)$ → normalization constant

GITAM
DEEMED TO BE UNIVERSITY

Statistical NLP Models

A Probabilistic Context-Free Grammar (PCFG) is an extension of a Context-Free Grammar (CFG) in which each grammar rule is assigned a probability.

5. Probabilistic Context-Free Grammar (PCFG)

Explanation (2 sentences):

PCFG extends context-free grammars by assigning probabilities to grammar rules. It resolves syntactic ambiguity by selecting the most probable parse tree.

Task: Syntactic Parsing

Sentence:

"The cat sat on the mat"

The model evaluates grammar rule probabilities to determine sentence structure.

Output:

Most probable syntactic structure

A PCFG consists of:

$$G = (N, \Sigma, R, S, P)$$

Where:

- N → Non-terminals (S, NP, VP, ...)
- Σ → Terminals (words)
- R → Grammar rules
- S → Start symbol
- P → Probabilities for each rule

GITAM
DEEMED TO BE UNIVERSITY

Statistical NLP Models

6. Latent Semantic Analysis (LSA)

Explanation (2 sentences):

Latent Semantic Analysis identifies hidden semantic relationships using statistical patterns in word co-occurrence. It reduces dimensionality to capture underlying meaning in text.

Task: Document Similarity

Sentence:

"Artificial intelligence improves machine learning systems"

LSA compares semantic vectors to find related documents.

Output:

Document grouped with AI-related content

Statistical NLP models use probability and statistical methods to model language structure, resolve ambiguity, and make predictions based on data.
Latent Semantic Analysis (LSA) is an unsupervised statistical technique in NLP used to discover hidden (latent) semantic relationships between words and documents by analyzing word co-occurrence patterns.

GITAM
DEEMED TO BE UNIVERSITY

Statistical NLP Models

Example: Statistical Part-of-Speech (POS) Tagging

Task: Decide whether a word is a *noun* or a *verb* based on probability.

Sentence:

> "Book a ticket"

The word "**Book**" is ambiguous.

From a training corpus:

- *Book* used as **verb** before a noun → 80%
- *Book* used as **noun** → 20%

The model computes probabilities and chooses the **most likely tag**.

Output:

> Book/VERB a/DET ticket/NOUN

Key Idea

Instead of fixed rules, the model selects the **most probable interpretation** based on past data.
Statistical NLP models use probability distributions learned from text data to handle ambiguity and make language decisions more flexibly than rule-based systems.

GITAM
DEEMED TO BE UNIVERSITY

Machine Learning Models in NLP

Machine Learning (ML) models in NLP learn language patterns automatically from labeled data instead of using hand-written rules or simple probability counts.
They use features extracted from text and a learning algorithm to make predictions.
These models learn patterns automatically from data.
Common algorithms:
- Naïve Bayes
- Support Vector Machines (SVM)
- Logistic Regression

Feature engineering is crucial for good performance.

Machine Learning Models in NLP

Example: Machine Learning–Based Sentiment Analysis

Task: Classify a sentence as *Positive* or *Negative*.

Training Data (Labeled):

- "This movie is great" → Positive
- "The movie is boring" → Negative

The model learns that words like **great** are associated with *positive* sentiment and **boring** with *negative* sentiment.

Input

> "The movie is great"

Model Decision

Using learned weights and features, the model predicts the most likely class.

Output

> **Positive sentiment**

Key Idea

- The model **learns from examples** and generalizes to new, unseen text.
- Machine Learning models in NLP use labeled data and learning algorithms to automatically capture language patterns and perform tasks such as classification and prediction.

GITAM
DEEMED TO BE UNIVERSITY

Machine Learning Models in NLP

1. Support Vector Machine (SVM)

Explanation (2 sentences):

Support Vector Machines are supervised learning models that find an optimal hyperplane to separate different classes. They are effective for text classification using high-dimensional feature spaces like TF-IDF.

Task: Text Classification

Sentence:

"This email contains a free offer"

The SVM evaluates feature vectors and class boundaries to determine the category.

Output:

Spam email

For linear SVM:

$$f(x) = w \cdot x + b$$

Classification rule:

$$\text{Class} = \begin{cases} +1 & \text{if } f(x) \geq 0 \\ -1 & \text{if } f(x) < 0 \end{cases}$$

GITAM
DEEMED TO BE UNIVERSITY

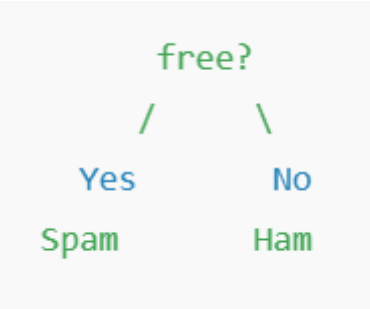
Machine Learning Models in NLP

2. Decision Tree

Explanation (2 sentences):

Decision Trees classify text by applying a series of rule-based decisions on extracted features. They provide interpretable models by splitting data based on feature importance.

Task: Sentiment Classification



Sentence:

"The service was excellent"

The tree follows decision rules learned from training data.

Output:

Positive sentiment

GITAM
DEEMED TO BE UNIVERSITY

Machine Learning Models in NLP

3. Random Forest

Explanation (2 sentences):

Random Forest is an ensemble learning method that combines multiple decision trees to improve accuracy.

It reduces overfitting by averaging predictions from different trees.

Task: Review Classification

Sentence:

"The product quality is very poor"

Multiple trees vote to determine the final prediction.

Output:

Negative sentiment

A Random Forest is an ensemble learning algorithm that builds multiple decision trees and combines their predictions.

GITAM
DEEMED TO BE UNIVERSITY

Machine Learning Models in NLP

4. Logistic Regression

Explanation (2 sentences):

Logistic Regression is a probabilistic linear classifier used for binary and multi-class text classification. It estimates class probabilities using weighted input features.

Task: Sentiment Analysis

Sentence:

"This movie was boring"

The model computes probabilities and selects the most likely class.

Output:

Negative sentiment

Linear Score:

$$z = w \cdot x + b$$

Sigmoid Function:

$$P(y = 1 \mid x) = \sigma(z) = \frac{1}{1 + e^{-z}}$$

GITAM
DEEMED TO BE UNIVERSITY

Deep Learning Models in NLP

Deep Learning models in NLP use multi-layer neural networks to automatically learn complex language representations directly from large amounts of text data. Unlike traditional ML models, they do not rely heavily on manual feature extraction.

Examples:

- Recurrent Neural Networks (RNN)
- Long Short-Term Memory (LSTM)
- Transformers (BERT, GPT)

They achieve high accuracy but require significant computational power.

Deep Learning Models in NLP

Example: Deep Learning–Based Sentiment Analysis

Task: Classify a sentence as *Positive* or *Negative*.

Model Used: Recurrent Neural Network (RNN) / LSTM

Training Data:

- "I love this product" → Positive
- "I hate this product" → Negative

The network learns **word order, context, and meaning** through hidden layers.

Input

> "I love this product"

Model Decision

The neural network processes the sentence word by word and captures contextual meaning.

Output

> **Positive sentiment**

Key Idea

- Deep learning models **understand context and relationships between words** by learning hierarchical representations.
- Deep Learning models in NLP automatically learn rich contextual features from data and achieve high performance on complex language tasks.

GITAM
DEEMED TO BE UNIVERSITY

Deep Learning Models in NLP

1. Convolutional Neural Network (CNN)

Explanation (2 sentences):

Convolutional Neural Networks capture local patterns such as phrases or n-grams using convolution filters.

In NLP, they are effective for text classification tasks where key patterns determine meaning.

Task: Sentiment Analysis

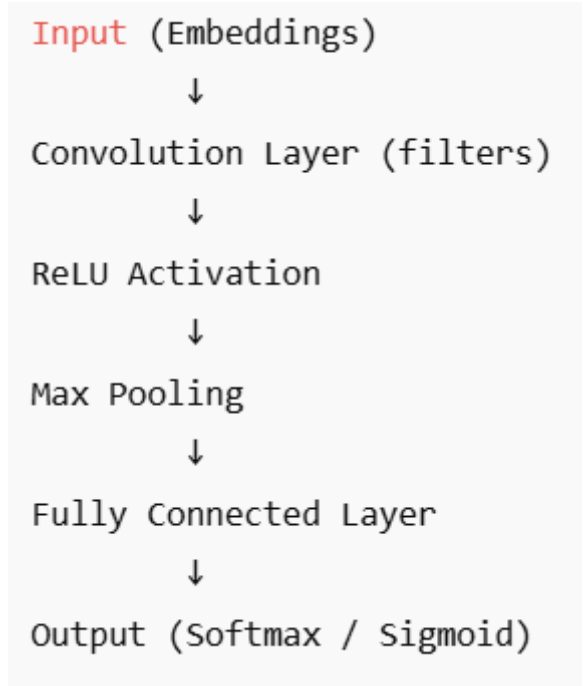
Sentence:

"This movie was extremely good"

The CNN detects important word patterns contributing to sentiment.

Output:

Positive sentiment



GITAM
DEEMED TO BE UNIVERSITY

Deep Learning Models in NLP

2. Recurrent Neural Network (RNN)

Explanation (2 sentences):

Recurrent Neural Networks process text sequentially, maintaining a hidden state that captures previous context. They are suitable for modeling word order and short-term dependencies.

Task: Sentence Classification

Sentence:

"I like this product"

The RNN processes each word in sequence to understand context.

Output:

Positive sentiment

At time step t:

$$h_t = f(W_h h_{t-1} + W_x x_t + b)$$

$$y_t = g(W_y h_t)$$

Where:

- x_t → input word vector
- h_t → hidden state (memory)
- y_t → output
- f → activation (tanh/ReLU)

GITAM
DEEMED TO BE UNIVERSITY

Deep Learning Models in NLP

3. Long Short-Term Memory (LSTM)

Explanation (2 sentences):

LSTM is a specialized RNN designed to overcome the vanishing gradient problem by using memory cells and gates. It effectively captures long-range dependencies in text.

Task: Next-word Prediction

Sentence:

"I am going to the"

The LSTM uses long-term context to predict the next word.

Output:

"park"

Each LSTM cell has **three main gates**:

1. **Forget Gate** (f_t) – decides what to forget
2. **Input Gate** (i_t) – decides what new information to add
3. **Output Gate** (o_t) – decides what to output

GITAM
DEEMED TO BE UNIVERSITY

Deep Learning Models in NLP

4. Gated Recurrent Unit (GRU)

Explanation (2 sentences):

GRU is a simplified version of LSTM with fewer gates, making it computationally efficient. It balances performance and speed while modeling sequential data.

Task: Machine Translation

Sentence:

"I love NLP"

GRU has **two gates**:

- **"Update gate – decides what information to keep from the past"**
- **"Reset gate – decides what information to forget"**

The GRU encodes the sentence and generates the translated output.

Output:

"मैं NLP से प्यार करता हूँ"

GITAM
DEEMED TO BE UNIVERSITY

Deep Learning Models in NLP

5. Transformer (Attention-Based Model)

Explanation (2 sentences):

Transformers use self-attention mechanisms to model relationships between all words in a sentence simultaneously. They remove recurrence, enabling parallel processing and better context understanding.

Task: Sentence Understanding

Sentence:

"He went to the bank to deposit money"

The transformer analyzes contextual relationships across the sentence.

Output:

Correct contextual meaning of *bank* (financial institution)

A **Transformer** is a **deep learning architecture** designed to process **sequential data** (like text) without relying on recurrence or convolution.

Self-Attention (or intra-attention) is a mechanism in Transformers that **helps the model focus on important words in a sequence** when encoding a specific word.

- Each word looks at **all other words in the sentence** and decides how much attention to give them.
- Enables **context-aware word representations** without sequential processing.

A Transformer is a deep learning model architecture used in NLP that processes text using attention mechanisms instead of sequential processing. It allows the model to understand context and relationships between all words in a sentence simultaneously, making it faster and more effective than RNN-based models.

GITAM
DEEMED TO BE UNIVERSITY

NLP Algorithms

Algorithms in NLP define how language data is processed and learned. Key NLP algorithms include:

a) Text Preprocessing Algorithms
b) Feature Extraction Algorithms
c) Sequence Modeling Algorithms
d) Attention-Based Algorithms

Text Preprocessing Algorithms

Text preprocessing algorithms prepare raw text into a clean, structured, and machine-readable form before applying NLP models.

They reduce noise and improve the accuracy and efficiency of NLP systems.

Preprocessing prepares raw text for analysis.

Steps include:

- Tokenization
- Stop-word removal
- Stemming and Lemmatization

These steps reduce noise and improve model performance.

Text Preprocessing Algorithms

Example: Tokenization and Stopword Removal

Task: Clean a sentence for analysis.

Input:

> "This is a very simple example sentence."

Preprocessing Steps

1. **Tokenization** – Split text into words
   → `["This", "is", "a", "very", "simple", "example", "sentence"]`
2. **Stopword Removal** – Remove common words (is, a, very)
   → `["simple", "example", "sentence"]`

Output
Cleaned text: **simple example sentence**

Key Idea
Preprocessing algorithms **transform raw text into meaningful tokens** that NLP models can effectively process.
Text preprocessing algorithms clean and normalize text to improve the performance of NLP models.

GITAM
DEEMED TO BE UNIVERSITY

Text Preprocessing Algorithms

Stemming and Lemmatization in NLP

Text preprocessing algorithms prepare raw text into a clean, structured, and machine-readable form before applying NLP models.

Why Do We Need Stemming and Lemmatization?

In NLP, the same word can appear in different forms:

- play, playing, played, plays
- run, running, ran

Stemming and Lemmatization reduce words to a common base form, helping models treat them as the same word.

GITAM
DEEMED TO BE UNIVERSITY

Text Preprocessing Algorithms

Stemming and Lemmatization in NLP

Stemming is a **rule-based process** that **chops off prefixes or suffixes** to get a root form (stem).

The stem **may not be a valid dictionary word**.

studi and happi are not real words.

Popular Stemming Algorithms

- Porter Stemmer (most common)
- Snowball Stemmer
- Lancaster Stemmer

Advantages of Stemming

- Fast and simple
- Reduces vocabulary size
- Useful in search engines

Example of Stemming

Original Word	Stem
playing	play
played	play
player	play
studies	studi
happiness	happi

Disadvantages of Stemming

- Produces non-meaningful words
- Less accurate for language understanding

GITAM
DEEMED TO BE UNIVERSITY

Text Preprocessing Algorithms

Lemmatization in NLP

Lemmatization reduces a word to its base or dictionary form (lemma) using vocabulary and grammar rules. Output is always a valid word.

Lemmatization Uses POS Tags

running (verb) → run

running (noun) → running

More accurate than stemming.

Popular Stemming Algorithms

- Porter Stemmer (most common)
- Snowball Stemmer
- Lancaster Stemmer

Advantages of Stemming

- Produces meaningful words
- Higher linguistic accuracy
- Better for NLP tasks requiring understanding

Disadvantages of Lemmatization

- Slower than stemming
  Requires POS tagging
  Computationally expensive

Example of Lemmatization

Original Word	Lemma
playing	play
played	play
studies	study
better	good
running	run

Example Sentence (Lemmatization)

Sentence:

> "The students are studying studies"

After lemmatization:

> "the student be study study"

GITAM
DEEMED TO BE UNIVERSITY

Feature Extraction Algorithms

Feature extraction algorithms convert processed text into informative attributes (features) that can be used by machine learning or statistical models.
They capture syntactic, semantic, or statistical properties of text.
Feature Extraction Algorithms

- Bag of Words (BoW)
- TF-IDFN-grams
- Part-of-Speech (POS) Tagging
- Named Entity Recognition (NER)
- Word Embeddings (Word2Vec, GloVe)

Feature extraction algorithms

Feature extraction algorithms are techniques used in machine learning and Natural Language Processing (NLP) to convert raw data (text, images, signals, etc.) into numerical features that a machine learning model can understand and process.

- Raw text cannot be directly used by algorithms.
- Feature extraction transforms text into structured numerical representations.
- These features capture important information such as word frequency, importance, and meaning.

Feature Extraction Algorithms

1. Bag of Words (BoW)

Idea: Represent text by word frequency.

Sentence:

> "I like NLP"

Feature Vector:

- I: 1
- like: 1
- NLP: 1

GITAM
DEEMED TO BE UNIVERSITY

Bag of Words (BoW) Model

BoW represents text as word frequency vectors.

Example:
"I love NLP" → {I:1, love:1, NLP:1}

Advantages:
- Simple to implement

Limitations:
- Ignores word order and context

Feature Extraction Algorithms

2. TF-IDF (Term Frequency–Inverse Document Frequency)

Idea: Assign higher weight to important words and lower weight to common words.

Example:

The word "**NLP**" gets a higher weight than "**I**" because it appears less frequently across documents.

3. N-grams

Idea: Capture word sequences.

Sentence:

> "I love NLP"

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

GITAM
DEEMED TO BE UNIVERSITY

TF-IDF Feature Extraction

TF-IDF assigns importance to words based on frequency.

TF: Term Frequency
IDF: Inverse Document Frequency

It reduces the weight of common words and highlights important terms.

Feature Extraction Algorithms

4. Part-of-Speech (POS) Tagging

Idea: Extract grammatical roles.

Sentence:

> "Book a ticket"

Features:

- Book/VERB
- a/DET
- ticket/NOUN

5. Named Entity Recognition (NER)

Idea: Identify entities like names and places.

Sentence:

> "Dr. Sharma works at Google"

Features:

- Dr. Sharma → PERSON
- Google → ORGANIZATION

Key Idea

These algorithms **convert text into numerical or symbolic features** that models can process.

Feature extraction algorithms represent text using statistical and linguistic features such as BoW, TF-IDF, n-grams, POS tags, and named entities.

GITAM
DEEMED TO BE UNIVERSITY

Word Embeddings

Word embeddings represent words as dense vectors.

Examples:
- Word2Vec
- GloVe
- FastText

They capture semantic relationships such as similarity and analogy.

Sequence Modeling Algorithms

Sequence Modeling Algorithms are techniques used in machine learning and Natural Language Processing (NLP) to model and learn patterns from sequential data, where the order of elements is important.

- Language is naturally sequential (words appear in a specific order).
- The meaning of a word often depends on previous and future words.
- Sequence modeling algorithms capture dependencies and context within text.

Sequence Modeling Algorithms

Sequence modeling algorithms process text as an ordered sequence, where the position of each word matters.
They model dependencies between words to understand context and meaning.
Why Sequence Modeling is Important

- Preserves word order and context
- Captures temporal and grammatical relationships
- Essential for understanding sentence structure and meaning

Sequence Modeling Algorithms

Common Sequence Modeling Algorithms in NLP

Hidden Markov Models (HMM):Probabilistic models that represent sequences with hidden states.Used in speech recognition and POS tagging.

Conditional Random Fields (CRF):Discriminative models used for labeling sequence data.Commonly used in Named Entity Recognition (NER).

Recurrent Neural Networks (RNN):Neural networks designed to handle sequential data by maintaining memory of past inputs.

Long Short-Term Memory (LSTM):An improved version of RNN that overcomes the vanishing gradient problem and captures long-term dependencies.

Gated Recurrent Units (GRU):A simplified variant of LSTM with fewer parameters.

Sequence Modeling Algorithms

Example: Hidden Markov Model (HMM) for POS Tagging

Task: Assign part-of-speech tags to a sentence.

Sentence:

> "Book a ticket"

The model considers:

- **Transition probability** (probability of a tag following another tag)
- **Emission probability** (probability of a word given a tag)

Using these probabilities, the algorithm finds the **most likely sequence of tags**.

Sequence Modeling Algorithms

1. Conditional Random Fields (CRF)

Task: Named Entity Recognition (NER)

Sentence:

> "Dr. Sharma works at Google"

CRF considers **neighboring labels** and word features to assign entity tags.

Output:

- Dr. Sharma → PERSON
- Google → ORGANIZATION

Sequence Modeling Algorithms

2. Recurrent Neural Networks (RNN)

Task: Sentiment Analysis

Sentence:

> "I like this movie"

The RNN processes words **one by one**, carrying context through hidden states.

Output:

> Positive sentiment

GITAM
DEEMED TO BE UNIVERSITY

Sequence Modeling Algorithms

3. Long Short-Term Memory (LSTM)

Task: Language Modeling

Sentence:

> "I am going to the"

LSTM remembers long-term dependencies and predicts the next word.

Output:

> "park"

GITAM
DEEMED TO BE UNIVERSITY

Sequence Modeling Algorithms

4. Gated Recurrent Units (GRU)

Task: Machine Translation

Sentence:

> "I love NLP" (English)

GRU captures context efficiently to generate translated output.

Output:

> "मैं NLP से प्यार करता हूँ" (Hindi)

HMM, CRF, RNN, LSTM, and GRU are sequence modeling algorithms that handle ordered data by learning dependencies across words.

GITAM
DEEMED TO BE UNIVERSITY

Transformer and Attention Models

Transformer and attention-based models process text by focusing on the most important words in a sentence, regardless of their position. They use self-attention instead of recurrence, allowing better understanding of long-range context.
Transformers use self-attention to understand context.
Advantages:
- Parallel processing
- Better long-range dependency handling

Examples:
- BERT
- GPT
- RoBERTa

GITAM
DEEMED TO BE UNIVERSITY

Transformer and Attention Models

Example: BERT (Bidirectional Encoder Representations from Transformers)

Task: Sentence Understanding

Sentence:

> "He went to the bank to deposit money"

BERT looks at **both left and right context** to understand that *bank* refers to a **financial institution**.

Output:

> Correct contextual meaning of *bank*

GITAM
DEEMED TO BE UNIVERSITY

Transformer and Attention Models

Example: GPT (Generative Pre-trained Transformer)

Task: Text Generation

Prompt:

> "Artificial Intelligence is"

GPT uses **left-to-right attention** to predict the next most likely words.

Output:

> "Artificial Intelligence is transforming modern industries."

GITAM
DEEMED TO BE UNIVERSITY

Transformer and Attention Models

Example: RoBERTa (Robustly Optimized BERT Approach)

Task: Text Classification

Sentence:

> "This course is very useful"

RoBERTa applies optimized transformer training to capture strong sentence representations.

Output:

> Positive sentiment

Key Idea

Transformer models use **attention mechanisms** to capture **contextual relationships between all words simultaneously**.

BERT, GPT, and RoBERTa are transformer-based attention models that excel at understanding and generating human language.

GITAM
DEEMED TO BE UNIVERSITY

Importance of Models and Algorithms in NLP

Models and algorithms form the foundation of NLP systems by:

- Enabling machines to understand semantic meaning
- Supporting automation of language-based tasks
- Improving human-computer interaction
- Powering applications like chatbots, search engines, and voice assistants

Regular Expression

A Regular Expression is a sequence of characters defining a search pattern.

- Used for pattern matching and text manipulation.
- Widely used in NLP for rule-based text processing.

Regular Expression

Basic Regex Rules

- Literals match exact characters.
- Metacharacters define patterns.
- Quantifiers control repetition.
- Anchors define position.

Four Core Categories:

- Literals
- Metacharacters
- Quantifiers
- Anchors

GITAM
DEEMED TO BE UNIVERSITY

Regular Expression

Literals

Literals match exact characters as they appear in text.
They do not have special meanings.

Literals – Examples

Pattern: cat
Text: The cat is sleeping.
Match: cat

Pattern: dog
Text: A dog barked.
Match: dog

Regular Expression

Metacharacters

- Metacharacters define patterns and have special meanings.
- They allow flexible matching

Metacharacters – Examples

Symbol: . → Any character

Pattern: c.t

Matches: cat, cut, cot

Symbol: [] → Character set

Pattern: [aeiou]

Matches any vowel

GITAM
DEEMED TO BE UNIVERSITY

Regular Expression

Quantifiers

Quantifiers control how many times a character or pattern repeats.

Metacharacters – Examples

Symbol: * → 0 or more

Pattern: ab*

Matches: a, ab, abb

Symbol: + → 1 or more

Pattern: go+gle

Matches: google, gooogle

GITAM
DEEMED TO BE UNIVERSITY

Regular Expression

Anchors

Anchors define the position of a pattern in a string.

Anchors – Examples

Symbol: ^ → Start of string

Pattern: ^Hello

Matches: Hello world

Symbol: $ → End of string

Pattern: world$

Matches: Hello world

GITAM
DEEMED TO BE UNIVERSITY

Regular Expression

Literals → Exact matching
Metacharacters → Pattern flexibility
Quantifiers → Repetition control
Anchors → Position control

Used in NLP preprocessing and rule-based systems.

Regular Expression

Common Regex Symbols

. → Any character
* → Zero or more
+ → One or more
? → Optional
^ → Start of string
$ → End of string

GITAM
DEEMED TO BE UNIVERSITY

Regular Expression

Character Classes

- [abc] → Matches a, b, or c
- [a-z] → Matches lowercase letters
- \d → Digits
- \w → Word characters
- \s → Whitespaces

Regular Expression

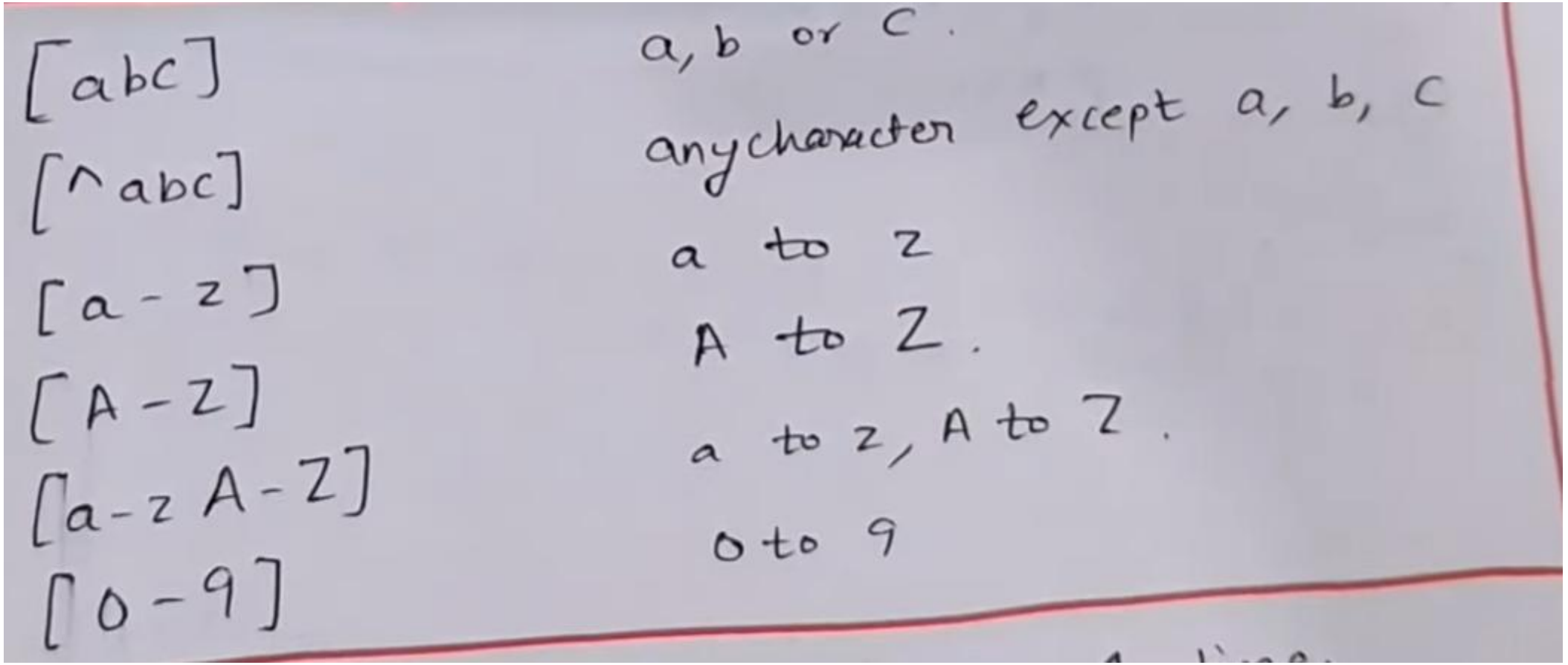
Regex Examples in NLP

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

- Date extraction
  \d{2}/\d{2}/\d{4}

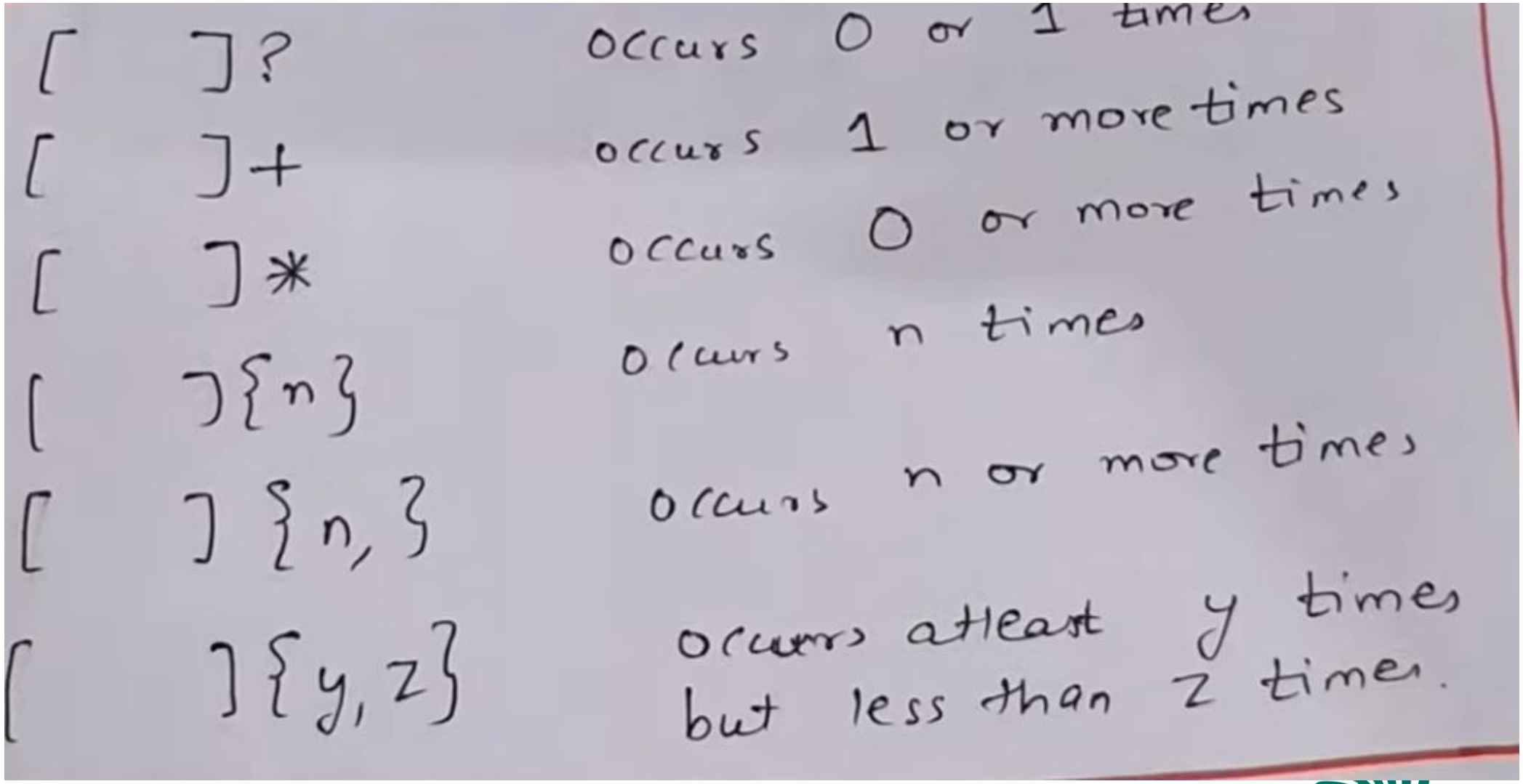
GITAM
DEEMED TO BE UNIVERSITY

Regular Expression



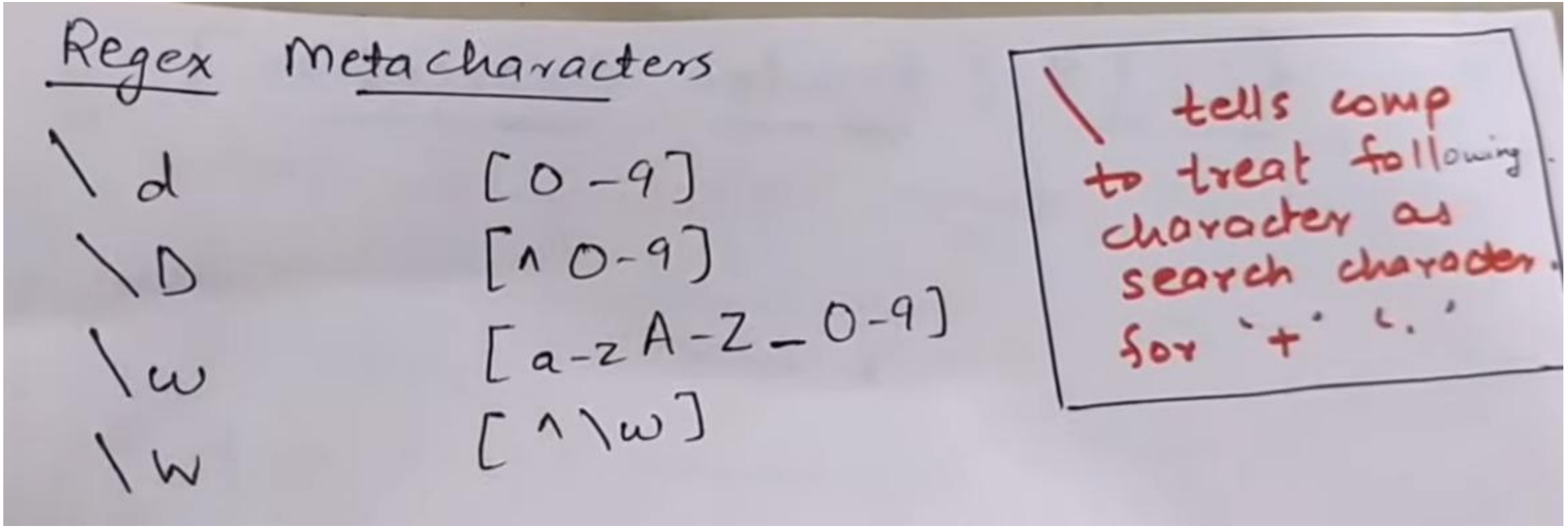
GITAM
DEEMED TO BE UNIVERSITY

Regular Expression



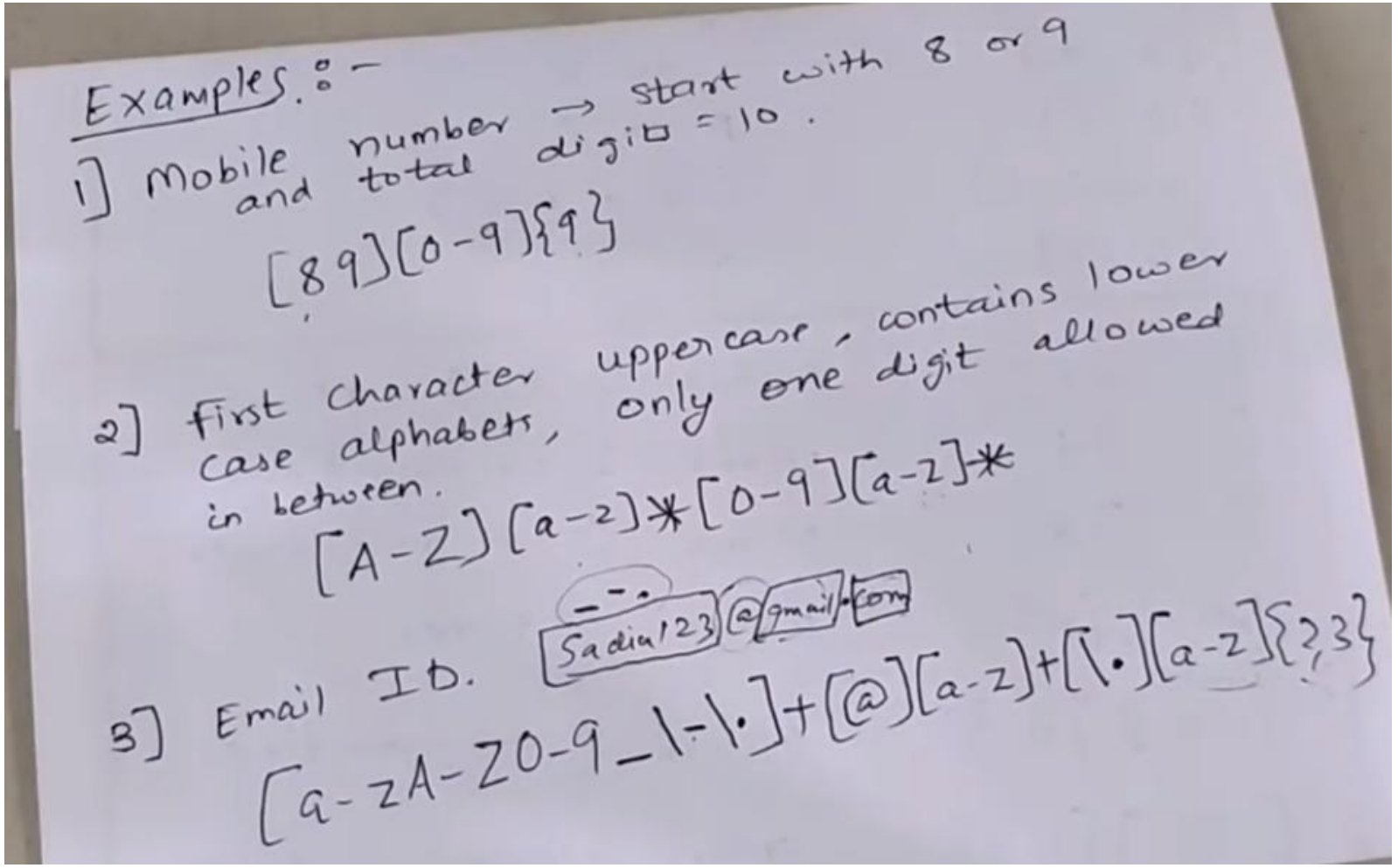
GITAM
DEEMED TO BE UNIVERSITY

Regular Expression



GITAM
DEEMED TO BE UNIVERSITY

Regular Expression



GITAM
DEEMED TO BE UNIVERSITY

Regular Expression

Advantages:

- Fast and efficient
- Easy to implement
- Good for fixed patterns

Limitations:

- No contextual understanding
- Not suitable for complex language tasks

Automata Theory – Overview

- Automata theory deals with abstract machines and the problems they can solve.
- It provides a mathematical foundation for computer science.
- Finite State Automata (FSA) are the simplest computational models.

What is a Finite State Automaton?

- A Finite State Automaton is a model of computation with a finite number of states.
- It reads input symbols one at a time and changes states accordingly.
- Used to recognize patterns in strings.

Formal Definition of FSA

- An FSA is formally defined as a 5-tuple $(Q, \Sigma, \delta, q_o, F)$.
- Q – Finite set of states.
- Σ – Input alphabet.
- δ – Transition function.
- q_o – Start state, F – Set of accepting states.

States and Alphabet

- States represent different stages of computation.
- Alphabet is the set of valid input symbols.
- Example: $\Sigma = \{a, b\}$

Transition Function

- Defines how the automaton moves between states.
- Based on current state and input symbol.
- $\delta(q, a) \rightarrow q'$

Deterministic Finite Automata (DFA)

- Exactly one transition for each input symbol from a state.
- No ambiguity in state transitions.
- No ε (epsilon) transitions allowed.

DFA – Example Language

- Language: Strings ending with 'ab'.
- Examples accepted: ab, aab, bab.
- Examples rejected: a, b, ba.

DFA – Hand-Drawn Style Diagram (Exam)

- q0 --a--> q1
- q1 --b--> q2 (Accepting State)
- q1 --a--> q1 (Loop)
- q2 is the final accepting state.

DFA – Explanation

- Start at q0.
- Move to q1 on reading 'a'.
- Move to q2 on reading 'b'.
- Accept if input ends in q2.

Construct a DFA that accepts all strings ending with 'a'.

Given:

Σ = {a, b},

Q = {q0, q1},

F = {q1}

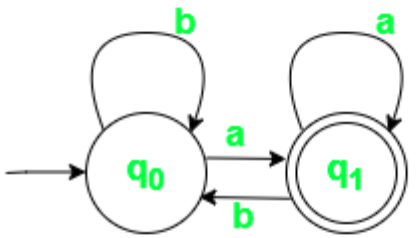


Fig 1. State Transition Diagram for DFA with Σ = {a, b}

State\Symbol	a	b
q0	q1	q0
q1	q1	q0

Non-Deterministic Finite Automata (NFA)

- Multiple transitions may exist for same input symbol.
- ε-transitions are allowed.
- Computation can follow multiple paths simultaneously.

NFA – Example Language

- Language: Strings containing substring 'aba'.
- Examples: aba, ababa, xaba.
- NFA simplifies design for such patterns.

NFA – Hand-Drawn Style Diagram (Exam)

- q0 --a--> q1
- q1 --b--> q2
- q2 --a--> q3 (Accepting State)
- q0 has self-loop for other symbols.

Construct an NFA that accepts strings ending in 'a'.

Given:

Σ = {a, b},

Q = {q0, q1},

F = {q1}

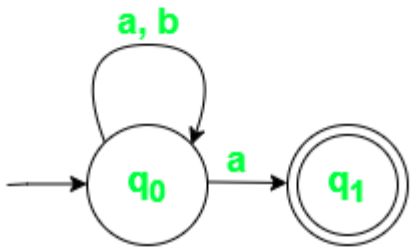


Fig 2. State Transition Diagram for NFA with Σ = {a, b}

State Transition Table for above Automaton,

State\Symbol	a	b
q_0	$\{q_0, q_1\}$	q_0
q_1	φ	φ

DFA vs NFA

- Both recognize regular languages.
- NFA easier to design, DFA easier to implement.
- NFA can be converted to DFA.

Tokenization in NLP

- Process of breaking text into smaller units called tokens.
- Tokens can be words, numbers, punctuation marks.
- Example: 'I love NLP' → I | love | NLP

Lexical Analysis

- Converts a stream of characters into tokens.
- Used as the first stage in NLP and compilers.
- FSAs are ideal due to efficiency.

FSA for Identifiers

- Identifier starts with a letter.
- Followed by letters or digits.
- Example: var123, count1

FSA for Numbers

- Digits with optional decimal point.
- Example: 123, 45.67
- Modeled using DFA.

Regular Expressions and FSA

- Regex patterns are converted into FSAs.
- Used for fast text matching.
- Example: [A-Za-z][A-Za-z0-9]*

Morphological Analysis – Definition

- Study of internal structure of words.
- Deals with roots, prefixes, and suffixes.
- Important for linguistically rich languages.

Inflectional Morphology

- Changes grammatical form without changing meaning.
- Examples: play → played, run → running

Derivational Morphology

- Creates new words.
- Examples: teach → teacher, happy → happiness

FSA in Morphological Analysis

- FSAs model valid word formations.
- Transitions represent affix addition.
- Efficient and rule-based.

Finite State Transducers (FST)

- Map surface word to base form.
- Example: walked → walk + past
- Used in lemmatization.

Advantages of FSA in NLP

- Fast and memory efficient.
- Easy to implement and debug.
- Highly interpretable.

Limitations of FSA

- Cannot handle deep syntax.
- No long-term memory.
- Limited to regular languages.

Summary

- FSA is foundational to NLP.
- Used in tokenization, morphology, lexical analysis.
- Still relevant alongside modern NLP models.

Morphology in NLP

- Morphology deals with the internal structure of words.
- It explains how words are formed, modified, and related to meaning.
- Essential for understanding linguistic patterns in NLP.

Introduction to Morphology

- Morphology is a branch of linguistics.
- It focuses on word formation rules in a language.
- In NLP, morphology helps machines process word variations.
- Example: play, plays, played, playing

What is Morphology in NLP?

- Morphology in NLP studies how words change form.
- It analyzes grammatical features like tense, number, and gender.
- Helps systems normalize and interpret words correctly.

What is a Morpheme?

- A morpheme is the smallest meaningful unit of language.
- Words may contain one or more morphemes.
- Example: unhappy = un (negation) + happy (root)

Characteristics of Morphemes

- Smallest unit of meaning
- Cannot be further divided meaningfully
- May be free or bound
- Used to express grammatical or semantic information

Types of Morphemes

- Morphemes are broadly classified into:
- • Free Morphemes
- • Bound Morphemes
- This classification is fundamental in morphological analysis.

Free Morphemes

- Free morphemes can stand alone as independent words.
- They carry complete meaning.
- Examples: book, run, happy, computer

Bound Morphemes

- Bound morphemes cannot appear alone.
- They must be attached to another morpheme.
- Examples: un-, re-, -ed, -ing

Prefixes

- Prefixes are bound morphemes added before the root word.
- They modify meaning.
- Examples:
- • unhappy (un + happy)
- • rewrite (re + write)

Suffixes

- Suffixes are bound morphemes added after the root word.
- They may change tense or word class.
- Examples:
- • playing (play + ing)
- • happiness (happy + ness)

Inflectional Morphology

- Inflectional morphology modifies grammatical form.
- Does not change word class or core meaning.
- Used for tense, number, aspect, comparison.

Inflectional Examples

- Verb tense: walk → walked
- Plurality: cat → cats
- Comparison: big → bigger
- Meaning remains the same

Derivational Morphology

- Derivational morphology creates new words.
- Often changes part of speech.
- Adds new meaning to the root word.

Derivational Examples

- Verb to noun: teach → teacher
- Adjective to noun: happy → happiness
- Noun to adjective: nation → national

Root and Stem

- Root: basic core meaning of a word.
- Stem: root plus derivational morphemes.
- Example: playing → stem = play

Morphological Analysis

- Process of breaking words into morphemes.
- Helps identify structure and meaning.
- Example: unrealizable = un + real + ize + able

Importance of Morphology in NLP

- Reduces vocabulary size
- Improves POS tagging accuracy
- Helps machine translation
- Supports speech recognition

Morphology vs Syntax

- Morphology studies word structure.
- Syntax studies sentence structure.
- Example:
- Morphology: dogs = dog + s
- Syntax: Dogs bark loudly

Morphology vs Semantics

- Morphology focuses on word form.
- Semantics focuses on meaning.
- Example:
- Morphology: plural marker '-s'
- Semantics: meaning of 'bank'

Comparison Summary

- Morphology: word formation
- Syntax: sentence formation
- Semantics: meaning interpretation
- All three work together in NLP systems

Conclusion

- Morphology is a foundation of NLP.
- It helps machines understand word variations.
- Essential for accurate language understanding.

Morphological Parsing in NLP

- Morphological parsing analyzes the internal structure of words.
- It identifies roots, affixes, and grammatical features.
- It is essential for understanding word formation and meaning in NLP.

Introduction to Morphological Parsing

- Morphological parsing is a key preprocessing step in NLP.
- It breaks words into morphemes.
- It assigns grammatical and linguistic information to each morpheme.
- Used heavily in linguistically rich languages.

Definition of Morphological Parsing

- Morphological parsing is the process of decomposing words into roots and affixes.
- It also identifies morphological features such as tense, number, and gender.
- It provides a structured linguistic representation of words.

Why Morphological Parsing is Needed

- Words appear in many forms in natural language.
- Parsing helps normalize these forms.
- Improves accuracy of NLP tasks like POS tagging and translation.
- Reduces ambiguity in word interpretation.

Morphemes in Morphological Parsing

- Morphemes are the smallest meaning-bearing units.
- Morphological parsing identifies each morpheme.
- Example: reorganization = re + organize + ation

Root in Morphological Parsing

- The root is the core part of a word.
- It carries the primary lexical meaning.
- Roots cannot usually be further analyzed.
- Example: write in rewriting

Stem in Morphological Parsing

- A stem is the root plus derivational affixes.
- It is the base to which inflectional affixes attach.
- Example: national → stem for nationalized

Affixes in Morphological Parsing

- Affixes are bound morphemes attached to roots or stems.
- They modify meaning or grammar.
- Main types: prefixes, infixes, suffixes.

Prefixes

- Prefixes are attached before the root.
- They usually modify meaning.
- Examples:
- unhappy = un + happy
- rewrite = re + write

Infixes

- Infixes are inserted inside the root word.
- They are rare in English.
- Common in languages like Tagalog.
- Example: sulat → sumulat

Suffixes

- Suffixes are attached after the root.
- They can be derivational or inflectional.
- Examples:
- teacher = teach + er
- played = play + ed

Types of Morphology

- Morphological parsing identifies two main types:
- Inflectional morphology
- Derivational morphology

Inflectional Morphology

- Inflectional morphology expresses grammatical features.
- Does not change word class or meaning.
- Examples:
- walk → walked
- cat → cats

Derivational Morphology

- Derivational morphology creates new words.
- Often changes part of speech.
- Examples:
- happy → happiness
- teach → teacher

Grammatical Features

- Morphological parsing identifies grammatical features.
- These features add syntactic and semantic information.

Grammatical Features – Examples

- Tense: play → played
- Number: dog → dogs
- Gender: actor → actress
- Case: boy → boy’s

Morphological Parsing in NLP

- These are core rule-based components used in morphological parsing.
- Other aspects like stemming, lemmatization, FSTs, or learning-based models are intentionally excluded.

Lexicons in Morphological Parsing

- A lexicon is a structured dictionary of morphemes.

- It contains:
- • Roots / stems
- • Affixes (prefixes, suffixes)
- • Grammatical features (POS, tense, number)
- Lexicons act as the knowledge base for morphological analysis.

Lexicon Structure and Features

- Each lexicon entry stores:
- • Surface form (word/morpheme)
- • Base form
- • Morphological category
- • Feature tags

- Example:
- play → Verb, base
- played → Verb + Past Tense

Lexicon Examples

- Sample Lexicon Entries:

- Root Lexicon:
- run → Verb
- child → Noun

- Affix Lexicon:
- -s → Plural
- -ed → Past Tense

- These entries are used to validate word formations.

Morphotactics: Definition

- Morphotactics defines the legal order of morphemes.

- It specifies:
- • Which affixes can attach to which roots
- • Valid morpheme sequences

- Morphotactics prevents invalid word formations.

Morphotactic Rules Explained

- Rules describe morpheme sequencing patterns.

- Examples:
- • Noun + Plural (-s)
- • Verb + Past (-ed)
- • Verb + Progressive (-ing)

- Invalid:
- • plural before root
- • tense after plural

Morphotactic Examples

- Valid formations:
- play + ed → played
- book + s → books

- Invalid formations:
- -ed + play
- play + s + ed

- Morphotactics ensures grammatical correctness.

Orthographic Rules: Definition

- Orthographic rules handle spelling changes during morpheme combination.
- They explain how letters change when affixes are added.
- These rules bridge morphology and spelling.

Orthographic Rule Examples

- Common Orthographic Rules:

- 1. Drop final 'e':
- make + ing → making

- 2. Double consonant:
- run + ing → running

- 3. Change 'y' to 'i':
- city + es → cities

Combined Example (Lexicon + Rules)

- Word: running

- Lexicon:
- run → Verb
- -ing → Progressive

- Morphotactics:
- Verb + ing → valid

- Orthographic Rule:
- Double consonant (n → nn)

- Final parsed output:
- run + ing → running

Applications of Morphological Parsing

- Machine Translation
- Speech Recognition
- POS Tagging
- Information Retrieval

Summary of of Morphological Parsing

- Morphological parsing provides deep word-level analysis.
- It enhances linguistic understanding in NLP.
- It is crucial for accurate language processing.

What is a Stem?

- In linguistics (study of language and its structure), a **stem** is part of a word, that is common to all of its inflected variants.
- CONNECT
- CONNECTED
- CONNECTION
- CONNECTING

The process of reducing such inflected (or sometimes derived) words to their word stem is known as **Stemming**.

For example, CONNECTED, CONNECTION and CONNECTING can be reduced to the stem CONNECT.

Introduction to Porter Stemmer

- Proposed by Martin F. Porter in 1980.
- The Porter Stemming algorithm (or Porter Stemmer) is used to remove the suffixes from an English word and obtain its stem which becomes very useful in the field of Information Retrieval (IR).
- This process reduces the number of terms kept by an IR system which will be advantageous both in terms of space and time complexity.
- Most widely used English stemmer.
- Rule-based suffix stripping algorithm.

Key Idea of Porter Algorithm

- Words are reduced step by step.
- Each step applies specific suffix rules.
- Rules depend on word structure.

Basic Definitions

- Consonants (C) and Vowels (V).
- Vowels: a, e, i, o, u.
- ‘y’ can be vowel or consonant.

Word Structure

- General form: [C](VC)^m[V]
- m = measure (count of VC sequences).
- Rules depend on value of m.

Measure (m) Explained

- m indicates word complexity.
- Higher m → longer morphological structure.
- Example: trouble → m = 1.

Measure (m) in Porter Stemmer

- • m (measure) = number of VC (vowel–consonant) sequences
- • Word form: [C](VC)^m[V]
- • V = vowel (a, e, i, o, u, sometimes y)
- • C = consonant
- • Used in rules like (m > 0), (m > 1)
- • Prevents over-stemming

How to Calculate m

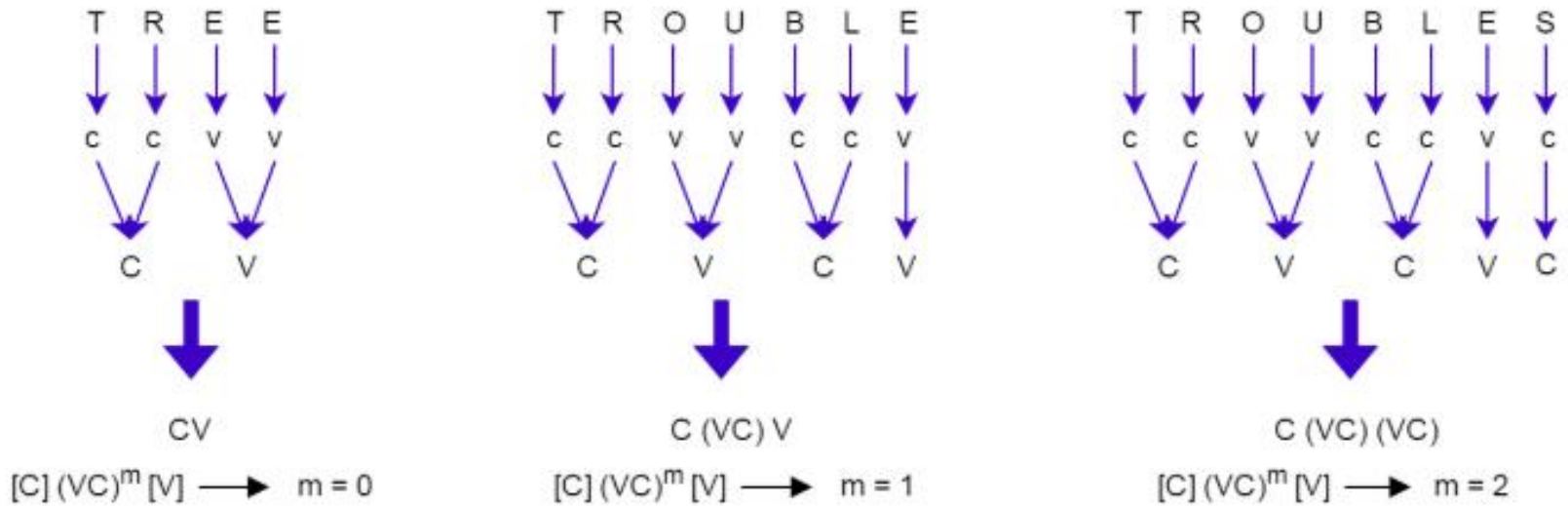
- Step 1: Mark each letter as V or C
- Step 2: Group consecutive letters into C or V
- Step 3: Count VC patterns only
- Step 4: Total VC count = m

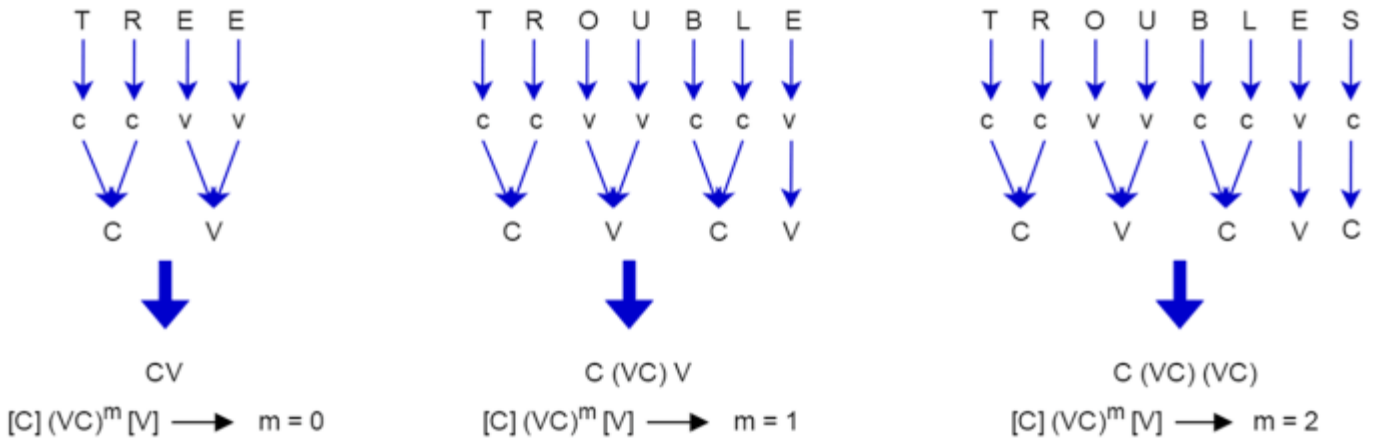
- Note:
- • Count VC, not CV
- • Ending vowel does not increase m

What is m?

The value **m** found in the above expression is called the **measure** of any word or word part when represented in the form **[C](VC)^{m}[V]**. Here are some examples for different values of m:

- m=0 → TREE, TR, EE, Y, BY
- m=1 → TROUBLE, OATS, TREES, IVY
- m=2 → TROUBLES, PRIVATE, OATEN, ROBBERY





Porter Stemming Algorithm

SSES	→	SS	(m>0) ATIONAL	→	ATE
IES	→	I	(m>0) TIONAL	→	TION
SS	→	SS	(m>0) ENCI	→	ENCE
S	→		(m>0) ANCI	→	ANCE

Examples of m Calculation

- TREE → C V → m = 0
- TROUBLE → C V C V → m = 1
- PRIVATE → C V C V C V → m = 2
- ROBBERY → C V C V C V → m = 2
- BY → C V → m = 0

Rules

The rules for replacing (or removing) a suffix will be given in the form as shown below.

> **(condition) S1 → S2**

This means that if a word ends with the suffix S1, and the stem before S1 satisfies the given condition, S1 is replaced by S2. The condition is usually given in terms of m in regard to the stem before S1.

> (m > 1) EMENT →

Here S1 is 'EMENT' and S2 is null. This would map REPLACEMENT to REPLAC, since REPLAC is a word part for which m = 2.

Conditions

The conditions may contain the following:

- *S – the stem ends with S (and similarly for the other letters)
- *v* – the stem contains a vowel
- *d – the stem ends with a double consonant (e.g. -TT, -SS)
- *o – the stem ends cvc, where the second c is not W, X or Y (e.g. -WIL, -HOP)

And the condition part may also contain expressions with and, or and not.

> (m>1 and (*S or *T)) tests for a stem with m>1 ending in S or T.
> (*d and not (*L or *S or *Z)) tests for a stem ending with a double consonant and does not end with letters L, S or Z.

How rules are obeyed?

In a set of rules written beneath each other, only one is obeyed, and this will be the one with the longest matching S1 for the given word. For example, with the following rules,

> 1. SSES → SS
> 2. IES → I
> 3. SS → SS
> 4. S →

(Here the conditions are all null) CARESSES maps to CARESS since SSES is the longest match for S1. Equally CARESS maps to CARESS (since S1="SS") and CARES to CARE (since S1="S").

The Algorithm

Step 1a

1. SSES → SS
2. IES → I
3. SS → SS
4. S →

Step 1b

1. (m>0) EED → EE
2. (*v*) ED →
3. (*v*) ING →

If the second or third of the rules in Step 1b is successful, the following is performed.

1. AT → ATE
2. BL → BLE
3. IZ → IZE
4. (*d and not (*L or *S or *Z)) → single letter
5. (m=1 and *o) → E

Step 1c

1. (*v*) Y → I

Step 2

1. (m>0) ATIONAL → ATE
2. (m>0) TIONAL → TION
3. (m>0) ENCI → ENCE
4. (m>0) ANCI → ANCE
5. (m>0) IZER → IZE
6. (m>0) ABLI → ABLE
7. (m>0) ALLI → AL
8. (m>0) ENTLI → ENT
9. (m>0) ELI → E
10. (m>0) OUSLI → OUS
11. (m>0) IZATION → IZE
12. (m>0) ATION → ATE
13. (m>0) ATOR → ATE
14. (m>0) ALISM → AL
15. (m>0) IVENESS → IVE
16. (m>0) FULNESS → FUL
17. (m>0) OUSNESS → OUS
18. (m>0) ALITI → AL
19. (m>0) IVITI → IVE
20. (m>0) BILITI → BLE

Step 3

1. (m>0) ICATE → IC
2. (m>0) ATIVE →
3. (m>0) ALIZE → AL
4. (m>0) ICITI → IC
5. (m>0) ICAL → IC
6. (m>0) FUL →
7. (m>0) NESS →

Step 4

1. (m>1) AL →
2. (m>1) ANCE →
3. (m>1) ENCE →
4. (m>1) ER →
5. (m>1) IC →
6. (m>1) ABLE →
7. (m>1) IBLE →
8. (m>1) ANT →
9. (m>1) EMENT →
10. (m>1) MENT →
11. (m>1) ENT →
12. (m>1 and (*S or *T)) ION →
13. (m>1) OU →
14. (m>1) ISM →

15. (m>1) ATE →
16. (m>1) ITI →
17. (m>1) OUS →
18. (m>1) IVE →
19. (m>1) IZE →

Step 5a

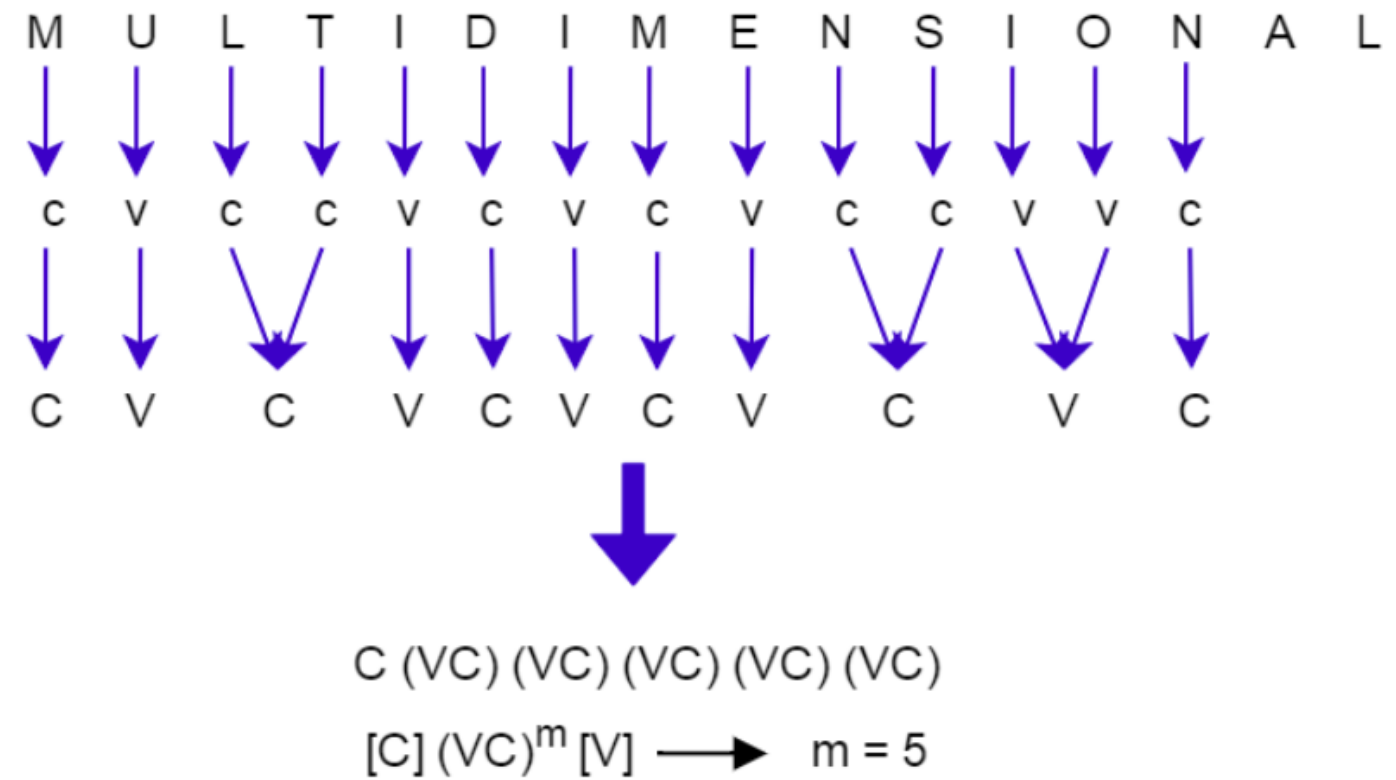
1. (m>1) E →
2. (m=1 and not *o) E →

Step 5b

1. (m > 1 and *d and *L) → single letter

For each word you input to the algorithm, all the steps from 1 to 5 will be executed and the output will be produced at the end.

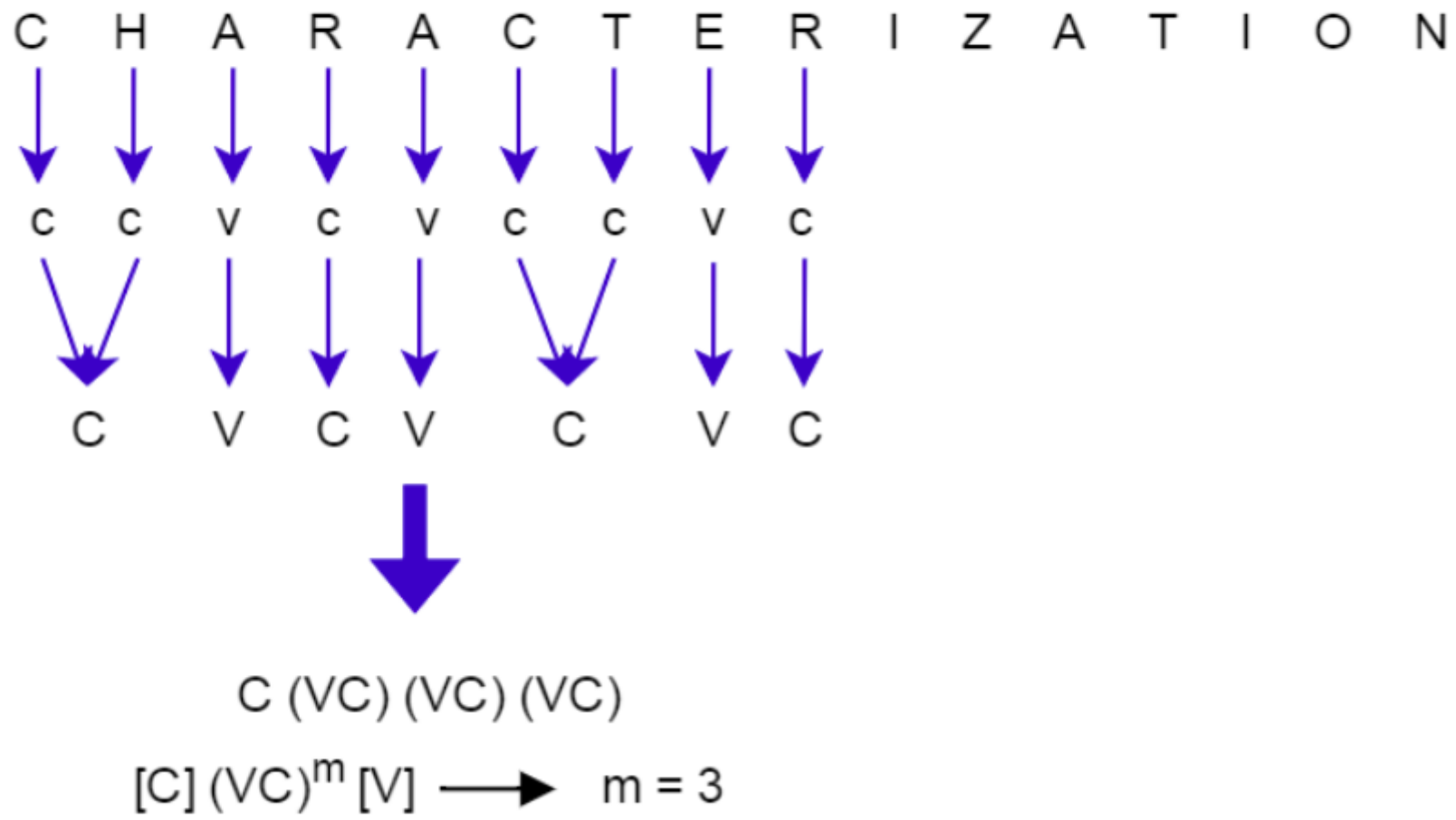
In the first example, we input the word **MULTIDIMENSIONAL** to the Porter Stemming algorithm. Let's see what happens as the word goes through steps 1 to 5.



- The suffix will not match any of the cases found in steps 1, 2 and 3.
- Then it comes to step 4.
- The stem of the word has m > 1 (since m = 5) and ends with "**AL**".
- Hence in step 4, "**AL**" is deleted (replaced with null).
- Calling step 5 will not change the stem further.
- Finally the output will be **MULTIDIMENSION**.

> MULTIDIMENSIONAL → **MULTIDIMENSION**

In the second example, we input the word **CHARACTERIZATION** to the Porter Stemming algorithm. Let's see what happens as the word goes through steps 1 to 5.



- The suffix will not match any of the cases found in step 1.
- So it will move to step 2.
- The stem of the word has m > 0 (since m = 3) and ends with "**IZATION**".
- Hence in step 2, "**IZATION**" will be replaced with "**IZE**".
- Then the new stem will be **CHARACTERIZE**.
- Step 3 will not match any of the suffixes and hence will move to step 4.
- Now m > 1 (since m = 3) and the stem ends with **"IZE"**.
- So in step 4, **"IZE"** will be deleted (replaced with null).
- No change will happen to the stem in other steps.
- Finally the output will be **CHARACTER**.

> CHARACTERIZATION → CHARACTERIZE → **CHARACTER**

Use in Search Engines

- Matches variations of query terms.
- Improves recall.
- Used in indexing.

Advantages

- Fast and simple.
- No dictionary required.
- Widely adopted.

Limitations

- Language-specific.
- No semantic understanding.
- Errors in stemming.

What is Text Normalization?

- Process of converting raw text into standard format
- Removes noise like spelling errors, slang, emojis
- Improves machine understanding
- Example: 'Hiiii!!! I luv NLP 😍' → 'hi i love nlp'

Why Text Normalization is Important

- Improves performance of NLP tasks
- Reduces vocabulary size
- Avoids duplicate words (Hello, hello, HELLO)
- Increases model accuracy

Technique 1: Lowercasing

- Converts all text into lowercase
- Reduces complexity in processing
- Example: 'Machine Learning Is AMAZING'
- Becomes: 'machine learning is amazing'

Technique 2: Removing Punctuation

- Removes symbols like ! @ # $ % , . ?
- Helps remove noise from text
- Example: 'Hello!!! How are you???'
- Becomes: 'hello how are you'

Technique 3: Tokenization

- Splits text into words or sentences
- Basic unit for NLP processing
- Example sentence: 'I love NLP'
- Tokens: ['I', 'love', 'NLP']

Technique 4: Stopword Removal

- Removes common words like is, the, and, on
- Improves efficiency of processing
- Example: 'This is a book on NLP'
- Becomes: 'book NLP'

Technique 5: Stemming

- Reduces words to root form
- Fast but may not be meaningful
- Examples: running → run, studies → studi
- Used in search engines

Technique 6: Lemmatization

- Converts words to meaningful base form
- More accurate than stemming
- Examples: better → good, went → go
- Uses grammar rules and vocabulary

Conclusion & Advanced Techniques

- Expanding contractions: don't → do not
- Spelling correction improves data quality
- Handling emojis: 😊 → happy
- Text normalization improves overall NLP performance

Why Edit Distance is Important

- Helps systems tolerate spelling mistakes
- Improves search engine query matching
- Used in plagiarism detection and OCR correction
- Improves user experience in NLP applications

Basic Edit Operations

- Insertion: add a character (cat → cart)
- Deletion: remove a character (cart → cat)
- Substitution: replace a character (cat → cut)
- Some models also allow transposition (form → from)

Levenshtein Distance (Core Technique)

- Most commonly used edit distance algorithm
- Allows insertion, deletion, substitution
- Each operation usually has cost = 1
- Uses Dynamic Programming for efficient computation

Dynamic Programming Concept

- Problem is broken into smaller subproblems
- A matrix (table) is constructed for two strings
- Each cell stores minimum edits required so far
- Final answer is found in bottom-right cell

Levenshtein Formula

- D[i][j] = min of three possibilities:
- Deletion: D[i-1][j] + 1
- Insertion: D[i][j-1] + 1
- Substitution: D[i-1][j-1] + cost (0 if same, 1 if different)

Worked Example: kitten → sitting

- kitten → sitten (substitute k → s)
- sitten → sittin (substitute e → i)
- sittin → sitting (insert g)
- Total Edit Distance = 3

DP Table Example: cat → cut

- A matrix is built comparing each character
- Only one substitution needed: a → u
- Final distance value = 1
- Demonstrates power of DP approach

Damerau–Levenshtein Distance

- Extension of Levenshtein distance
- Adds transposition as an operation
- Better models human typing errors
- Example: form → from (distance = 1 instead of 2)

Other Distance Techniques

- Hamming Distance: works only for equal length strings
- Jaro Distance: used in name matching
- Jaro-Winkler: improves Jaro by giving weight to prefixes
- Each technique useful in specific NLP tasks

Real-world Use Case: Spell Checker

- User types: accomodation
- System compares with dictionary words
- Closest match found: accommodation
- Edit distance helps system choose best suggestion

Advantages and Limitations

- Advantages: simple, effective, language independent
- Works well for spelling correction and fuzzy matching
- Limitations: ignores meaning (car vs automobile)
- Computationally expensive for long documents

Conclusion

- Edit distance is a foundational NLP technique
- Levenshtein is most widely used algorithm
- Supports many real-world intelligent systems
- Essential concept for NLP and Information Retrieval

END OF UNIT -1