NLP:

## 1. What is NLP?

* **NLP (Natural Language Processing)** is a branch of **Artificial Intelligence** that enables computers to understand, interpret, and generate human language.
* It combines **Linguistics** (rules of language) + **Computer Science** + **Machine Learning**.

Example Applications:

* Google Translate
* Chatbots
* Sentiment Analysis
* Voice Assistants (Siri, Alexa, Google Assistant)

## 2. Why is NLP Needed?

* Human language is **unstructured** → computers understand **numbers**, not words.
* NLP helps convert **text/speech → machine-readable format**.
* Enables:
  1. **Communication** between humans & machines.
  2. **Automation** (customer support, summarization).
  3. **Information extraction** from large text (news, social media).

## 3. Challenges in NLP

* **Ambiguity**: "I saw a man with a telescope" (who has telescope?).
* **Context dependency**: "Apple" → fruit 🍎 or company
* **Sarcasm/Irony**: "Great! My phone just died ".
* **Multilinguality**: Thousands of languages worldwide.
* **Spelling/Grammar errors** in real-world text.

## 4. Approaches of NLP

### 1. Heuristic Approach :

* A **heuristic** is a *rule-of-thumb* or *shortcut* method that helps solve problems quickly, though not always perfectly.
* In **NLP**, the heuristic approach means solving language tasks using **handcrafted rules, patterns, or logic**, instead of machine learning or deep learning.

## Characteristics

* ✅ **Simple & Fast** → no training data required.
* ✅ **Good for small problems** or when domain knowledge is available.
* ❌ **Not scalable** → fails when data grows or language gets complex.
* ❌ **Low accuracy** compared to statistical or neural approaches.

## When to Use Heuristic Approach?

* When **data is scarce** (no large training dataset).
* When task is **well-defined** and rules are easy to write.
* For **prototyping** before moving to ML/DL models.

## 2. Machine Learning Approach :

* Instead of writing **manual rules (heuristics)**, the **machine learning approach** uses **data-driven models** to learn patterns from text.
* It converts text into **numerical features** (vectors) and then applies **ML algorithms** for prediction or classification.

## Why Use ML in NLP?

* Rule-based systems are rigid → they fail with complex/ambiguous language.
* ML models **adapt automatically** when trained on large datasets.
* They generalize better across unseen text.

## Workflow of ML-based NLP

1. **Text Preprocessing** (cleaning, tokenization, stopwords removal, stemming/lemmatization).
2. **Feature Extraction** (converting text into numbers):
   * Bag of Words (BoW)
   * TF-IDF (Term Frequency–Inverse Document Frequency)
   * N-grams
   * Word Embeddings (Word2Vec, GloVe)
3. **Model Training** with ML algorithms.
4. **Prediction / Classification** on new text.

## Machine Learning Algorithms Used in NLP

1. **Naive Bayes Classifier**
   * Based on probability (Bayes’ theorem).
   * Common in **spam filtering, sentiment analysis**.
2. **Logistic Regression**
   * Simple yet powerful classifier for text classification.
3. **Support Vector Machines (SVMs)**
   * Works well with high-dimensional data like text.
   * Used in **topic classification, intent detection**.
4. **Decision Trees & Random Forests**
   * Not as common, but used for **structured NLP tasks**.
5. **Hidden Markov Models (HMMs)**
   * Sequential models used in **POS tagging, speech recognition**.

## Example Use Cases :

* **Spam Email Classification** (Spam vs Not Spam)
* **Sentiment Analysis** (Positive / Negative reviews)
* **Topic Categorization** (Sports, Politics, Technology)
* **Named Entity Recognition (NER)**

# ****3. Deep Learning Approach in NLP :****

* Deep learning uses **artificial neural networks** (like a brain with layers of neurons) to understand text.
* Instead of rules (heuristic) or manual features (machine learning), deep learning **learns patterns automatically from big text data**.

### Key Features

* **Learns on its own** → No need to handcraft features.
* **Understands context** → Words have different meanings in different sentences, deep learning captures this.
* **Works best with large data** → The more text, the better.

### Main Models Used in NLP

1. **Word Embeddings** → Convert words into numbers with meaning.
   * Examples: Word2Vec, GloVe, FastText.
2. **RNN (Recurrent Neural Networks)** → Good for sequences like sentences.
   * But slow and forgets long text.
3. **LSTM/GRU** → Improved RNNs that remember long-term dependencies.
4. **CNN (Convolutional Neural Networks)** → Also used for text (not only images).
5. **Transformers (Modern Era)** → Powerful models that handle long text better.
   * Example: BERT, GPT, T5.

### Advantages

1. Learns automatically from data
2. Captures meaning and context of words
3. Very accurate for translation, chatbots, sentiment analysis

### Disadvantages

1. Needs a lot of data
2. Requires high computing power (GPUs)
3. Works like a "black box" (hard to explain how it makes decisions)

💡 **Memory Tip:**  
 Think of 3 stages in NLP evolution:

* Heuristic → Rules (teacher gives strict rules)
* Machine Learning → Features (student learns with notes + examples)
* Deep Learning → Neural Networks (student learns directly from reading books without extra help)

### 🌐 ****NLP Pipeline (Step-by-Step Process)****

1. **Text Collection**
   * Collect data (from websites, books, chats, etc.).
   * Example: Tweets, reviews, articles.
2. **Text Preprocessing**
   * Clean and prepare the text for analysis.
   * Includes:
     + Lowercasing
     + Removing punctuation/stopwords
     + Tokenization (splitting into words)
     + Lemmatization/Stemming
3. **Text Representation**
   * Convert text into numbers (since ML models only understand numbers).
   * Methods:
     + **Bag of Words (BoW)**
     + **TF-IDF (Term Frequency–Inverse Document Frequency)**
     + **Word Embeddings** (Word2Vec, GloVe, BERT).
4. **Feature Engineering**
   * Extract useful features from text.
   * Example: word counts, sentiment score, part of speech tags.
5. **Model Building**
   * Train ML/DL models on the processed data.
   * Examples:
     + ML: Naive Bayes, SVM, Logistic Regression
     + DL: RNN, LSTM, Transformers
6. **Evaluation**
   * Check model performance.
   * Metrics: Accuracy, Precision, Recall, F1-score.
7. **Deployment**
   * Use the trained model in real-world apps (chatbots, translators, recommendation systems).

**Easy way to remember NLP pipeline** : **C → P → R → F → M → E → D**

* **C**ollection
* **P**reprocessing
* **R**epresentation
* **F**eature engineering
* **M**odel building
* **E**valuation
* **D**eployment

| **Aspect** | **Feature Engineering in ML** | **Feature Engineering in DL** |
| --- | --- | --- |
| **Definition** | Process of manually creating features from raw data to improve model performance. | Process is mostly automated, as neural networks learn features directly from raw data. |
| **Role of Human** | High involvement – humans design features based on domain knowledge (e.g., TF-IDF, n-grams). | Low involvement – deep models extract features automatically (e.g., word embeddings, CNN filters). |
| **Complexity** | Requires strong understanding of the problem domain and data patterns. | The model itself handles complexity, requiring less manual feature crafting. |
| **Examples in NLP** | Bag-of-Words, TF-IDF, POS tagging, handcrafted rules. | Word2Vec, GloVe, BERT embeddings, sequence representations. |
| **Dependency on Data** | Works well with smaller datasets if features are engineered properly. | Requires large datasets to learn meaningful features automatically. |
| **Flexibility** | Less flexible – features are fixed once designed. | Highly flexible – model can learn new feature representations during training. |
| **Computation** | Generally lower computation, but higher manual effort. | Higher computation, but less manual effort. |

# 📌 ****Data Acquisition in NLP :****

### What is Data Acquisition?

* **Data Acquisition** means **collecting text data** that we will use for Natural Language Processing tasks like sentiment analysis, chatbot training, translation, summarization, etc.
* Without **good and enough data**, NLP models cannot learn patterns or understand language.

### Sources of Data Acquisition

1. **APIs (Application Programming Interfaces)**
   * Many websites provide APIs to access text data (e.g., Twitter API for tweets, NewsAPI for news articles).
   * Example: Getting tweets about "AI" using Twitter API.
2. **Web Scraping**
   * If a website does not provide an API, we can extract data directly from its pages using libraries like BeautifulSoup or Scrapy.
   * Example: Scraping Amazon reviews.
3. **Open-Source Datasets**
   * Kaggle, Hugging Face Datasets, UCI Machine Learning Repository, etc.
   * Example: IMDB Movie Reviews dataset for sentiment analysis.
4. **Databases & Logs**
   * Text data can be collected from company databases, chat logs, customer feedback, emails, etc.
   * Example: Collecting customer support chats to train a chatbot.
5. **Manual Collection / Crowdsourcing**
   * Creating your own dataset by surveys, forms, or platforms like Amazon Mechanical Turk.

# 📌 ****Text Preprocessing :****

* Text preprocessing means **cleaning and preparing raw text** so a machine can understand it.
* Raw text (from books, websites, chats, etc.) is **messy** – it has punctuation, special characters, different cases (UPPER/lower), stopwords, and so on.So, preprocessing makes the text **clean, consistent, and machine-readable**.

## ****Steps of Text Preprocessing (with Simple Examples)****

1. **Lowercasing**
   * Convert everything into lowercase.
   * Example: "I Love NLP" → "i love nlp"
   * Why? So machine doesn’t treat "NLP" and "nlp" as different.

text = "NLP is FUN"

text = text.lower()

1. **Remove html tags**

* When we scrape data from websites, text often contains **HTML tags** (<p>, <div>, <br> etc.), so we need to remove them.

**✅ Method 1: Using BeautifulSoup**

from bs4 import BeautifulSoup

def remove\_html\_tags(text):

    """

    Remove HTML tags from a given text using BeautifulSoup

    """

    soup = BeautifulSoup(text, "html.parser")

    return soup.get\_text()

# Example

sample\_text = "<p>Hello <b>World!</b> This is <a href='#'>NLP</a>.</p>"

clean\_text = remove\_html\_tags(sample\_text)

print(clean\_text)  # Output: Hello World! This is NLP.

**✅ Method 2: Using Regex**

import re

def remove\_html\_tags\_regex(text):

    """

    Remove HTML tags using regex

    """

    clean = re.compile('<.\*?>')  # Pattern for HTML tags

    return re.sub(clean, '', text)

# Example

sample\_text = "<p>Hello <b>World!</b> This is <a href='#'>NLP</a>.</p>"

clean\_text = remove\_html\_tags\_regex(sample\_text)

print(clean\_text)  # Output: Hello World! This is NLP.

1. **Tokenization**

## 1. ****Split by Spaces (Basic Python)****

text = "Hello world, this is AI."

tokens = text.split()

print(tokens)

## o/p : ['Hello', 'world,', 'this', 'is', 'AI.']

## 2. ****Regex Tokenization****

import re

text = "Hello world, this is AI."

tokens = re.findall(r"\w+", text)   # only words

print(tokens)

## ****o/p : ['Hello', 'world', 'this', 'is', 'AI']****

## ****3.NLTK Word Tokenizer****

import nltk

from nltk.tokenize import word\_tokenize

nltk.download('punkt')

text = "Hello world, this is AI."

tokens = word\_tokenize(text)

## ****o/p : ['Hello', 'world', ',', 'this', 'is', 'AI', '.']****

## ****4.**** ****NLTK Sentence Tokenizer:****

from nltk.tokenize import sent\_tokenize

text = "Hello world. This is AI. Tokenization is fun!"

sentences = sent\_tokenize(text)

print(sentences)

## ****o/p : ['Hello world.', 'This is AI.', 'Tokenization is fun!']****

## ****5.**** ****TextBlob Tokenization****

from textblob import TextBlob

text = "Hello world, this is AI."

blob = TextBlob(text)

print(blob.words)      # word tokenization

print(blob.sentences)  # sentence tokenization

## ****o/p : ['Hello', 'world', 'this', 'is', 'AI']****

## ****[Sentence("Hello world,"), Sentence("this is AI.")]****

## ****6.**** ****spaCy Tokenization****

import spacy

nlp = spacy.load("en\_core\_web\_sm")

text = "Hello world, this is AI."

doc = nlp(text)

tokens = [token.text for token in doc]

print(tokens)

## ****o/p : ['Hello', 'world', ',', 'this', 'is', 'AI', '.']****

## ****7.**** ****Keras / TensorFlow Tokenizer (Deep Learning)****

from tensorflow.keras.preprocessing.text import Tokenizer

text = ["Hello world", "This is AI"]

tokenizer = Tokenizer()

tokenizer.fit\_on\_texts(text)

print(tokenizer.word\_index)  # word → index mapping

print(tokenizer.texts\_to\_sequences(text))

## ****o/p : {'hello': 1, 'world': 2, 'this': 3, 'is': 4, 'ai': 5}****

## ****[[1, 2], [3, 4, 5]]****

## ****8.**** ****WordPunct Tokenizer (NLTK)****

from nltk.tokenize import WordPunctTokenizer

text = "Hello world, this is AI."

tokens = WordPunctTokenizer().tokenize(text)

print(tokens)

## ****o/p : ['Hello', 'world', ',', 'this', 'is', 'AI', '.']****

1. **Removing Punctuation**
   * Delete symbols like .,!?;@#.
   * Example: "Hello!!! NLP??" → "Hello NLP"
   * Because punctuation usually doesn’t add meaning.

import string

text = "Hello!!! How are you?"

text = text.translate(str.maketrans('', '', string.punctuation))

1. **Removing Numbers** (optional)
   * If numbers are not important, remove them.
   * Example: "I have 2 apples" → "I have apples"

import re

text = "I have 2 apples and 3 mangoes"

text = re.sub(r'\d+', '', text)

1. **Removing Stopwords**
   * Stopwords = very common words that don’t add much meaning.
   * Example: "I love playing with the dog" → "love playing dog"
   * Words like "is", "the", "and", "with" are removed.

from nltk.corpus import stopwords

nltk.download('stopwords')

stop\_words = set(stopwords.words('english'))

text = "I am learning NLP"

tokens = word\_tokenize(text)

filtered = [w for w in tokens if w.lower() not in stop\_words]

print(filtered)

1. **Remove Duplicate/repeated word**

text = "this is a test this is only a test"

words = text.split()

seen = set()

unique\_words = []

for word in words:

    if word.lower() not in seen:   # case insensitive

        unique\_words.append(word)

        seen.add(word.lower())

1. **Stemming**
   * Cut words to their root form.
   * Example: "playing" → "play", "studies" → "studi"
   * Fast but sometimes not perfect.

from nltk.stem import PorterStemmer

ps = PorterStemmer()

words = ["playing", "played", "plays"]

stemmed = [ps.stem(w) for w in words]

o/p : ['play', 'play', 'play']

1. **Lemmatization**
   * Smarter than stemming → gives real dictionary word.
   * Example: "playing" → "play", "studies" → "study"

* Example: "better" → "good"
  + More accurate but slower.

from nltk.stem import WordNetLemmatizer

nltk.download('wordnet')

lemmatizer = WordNetLemmatizer()

words = ["playing", "better", "rocks"]

lemmas = [lemmatizer.lemmatize(w, pos="v") for w in words]

o/p : ['play', 'better', 'rock']

1. **Removing Extra Whitespaces**
   * Clean up multiple spaces.
   * Example: "I love NLP" → "I love NLP"

text = "   NLP    is   fun   "

text = " ".join(text.split())

1. **Handling Special Characters / Emojis**
   * Remove or replace them depending on task.
   * Example: "I love NLP 😊" → "I love NLP"

import emoji

text = "I am happy 😊"

emoji = emoji.demojize(text)

1. **Spelling Correction (optional)**

* Fix wrong spellings.
* Example: "I lvoe NLP" → "I love NLP"

from textblob import TextBlob

text = " I lvoe NLP"

blob = TextBlob(text)

corrected = str(blob.correct())

1. **Text Normalization**

Make words consistent.

* Example: "U.S.A" → "USA", "won’t → will not"

slang\_dict = {"u": "you", "r": "are", "gr8": "great"}

text = "u r gr8 in NLP"

tokens = text.split()

normalized = [slang\_dict.get(w, w) for w in tokens]

print(" ".join(normalized))

o/p : "you are great in NLP”

**Regular Expression(Regex) :**

## ****Definition****

* Regular Expressions (**regex**) are a sequence of characters used to define a search pattern.
* They are mainly used for **string searching, pattern matching, and text manipulation**.
* In Python, regex operations are handled by the re module.

import re

# Search for a pattern in a string

pattern = r"hello"

text = "hello world"

result = re.search(pattern, text)

if result:

    print("Pattern found!")

else:

    print("Pattern not found.")

## ****Examples****

### 1. Match at the beginning :

import re

text = "Python is great"

result = re.match(r"Python", text)

print(result.group())  # Output: Python

**2. Search anywhere :**

text = "I love Python programming"

result = re.search(r"Python", text)

print(result.group())  # Output: Python

**3. Find all matches**

text = "cat mat bat rat"

result = re.findall(r"\b\w{3}\b", text)

print(result)  # Output: ['cat', 'mat', 'bat', 'rat']

**4. Replace text/Remove text :**

text = "Hello 123, this is 456"

result = re.sub(r"\d+", "#", text)

print(result)  # Output: Hello #, this is #

**5. Split text :**

text = "apple,banana;grape orange"

result = re.split(r"[,; ]+", text)

print(result)  # Output: ['apple', 'banana', 'grape', 'orange']

1. **Remove links :**

patt = r"http[s]?://\S+"

text = "playing https://chatgpt.com/c/68b8502d-eac8-832e-b167-88139ff600ce

parents & http://chatgpt.com/c/68b8502d-eac8-832e sfcsdvsd"

data = re.sub(patt,"",text)

**Ex :** Find this all possible mobile number from text.

Text = “My mobile number is 9309313335 and also (999)-(333)-4444”

Pattern = r“\d{10}|\(\d{3}\)-\(\d{3}\)-\d{4}”

all\_phone\_num = re.findall(Pattern,Text)

**Some Important Pattern :**

| **Pattern** | **Meaning** | **Example** | **Output** |
| --- | --- | --- | --- |
| . | Any character except newline | re.findall(r"h.t", "hat hot hit") | ['hat', 'hot', 'hit'] |
| ^ | Start of string | re.findall(r"^Hello", "Hello World") | ['Hello'] |
| $ | End of string | re.findall(r"world$", "Hello world") | ['world'] |
| \* | 0 or more repetitions | re.findall(r"ab\*", "ab abb abbb a") | ['ab', 'abb', 'abbb', 'a'] |
| + | 1 or more repetitions | re.findall(r"ab+", "ab abb abbb a") | ['ab', 'abb', 'abbb'] |
| ? | 0 or 1 repetition | re.findall(r"ab?", "ab abb a") | ['ab', 'ab', 'a'] |
| {n} | Exactly n times | re.findall(r"\d{3}", "123 4567 89") | ['123', '456'] |
| {n,} | n or more times | re.findall(r"\d{2,}", "1 12 123 1234") | ['12', '123', '1234'] |
| {n,m} | Between n and m times | re.findall(r"\d{2,3}", "1 12 123 1234") | ['12', '123', '123'] |
| [] | Character set | re.findall(r"[aeiou]", "apple") | ['a', 'e'] |
| [^] | Negation (not in set) | re.findall(r"[^aeiou]", "apple") | ['p', 'p', 'l'] |
| \d | Digit (0–9) | re.findall(r"\d", "a1b2c3") | ['1', '2', '3'] |
| \D | Non-digit | re.findall(r"\D", "a1b2c3") | ['a','b','c'] |
| \s | Whitespace | re.findall(r"\s", "hi there") | [' '] |
| \S | Non-whitespace | re.findall(r"\S", "hi there") | ['h','i','t','h','e','r','e'] |
| \w | Word character (letters, digits, \_) | re.findall(r"\w", "hi\_123!") | ['h','i','\_','1','2','3'] |
| \W | Non-word character | re.findall(r"\W", "hi\_123!") | ['!'] |
| `a | b` | re.findall(r"cat|dog", "cat dog rat") | ['cat', 'dog'] |
| ( ) | Group | re.findall(r"(ab)+", "ababab xyz") | ['ab', 'ab', 'ab'] |
| \b | Word boundary | re.findall(r"\bcat\b", "cat scatter") | ['cat'] |
| (?i) | Case insensitive | re.findall(r"(?i)hello", "Hello hELLo") | ['Hello', 'hELLo'] |