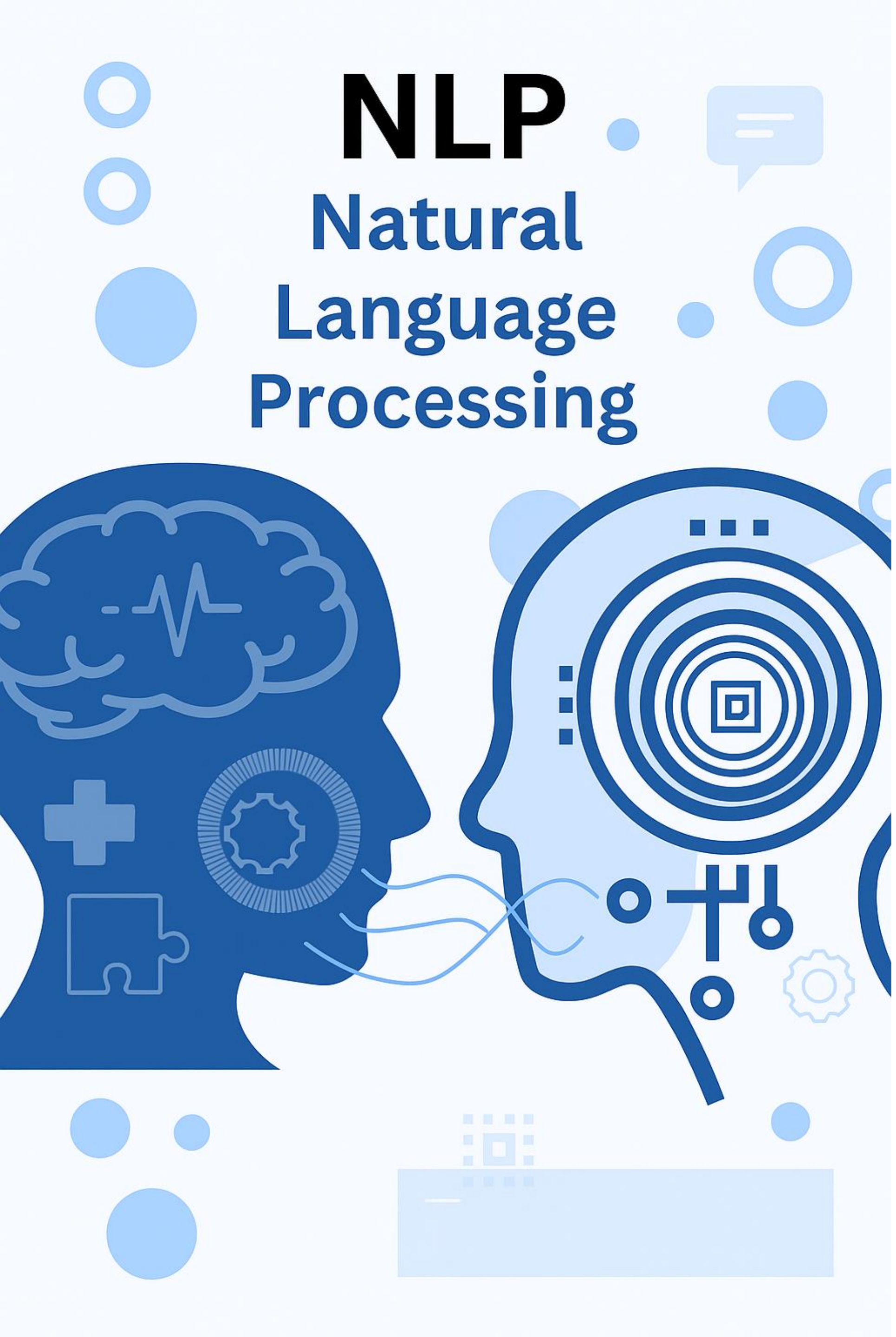

NATURAL LANGUAGE PROCESSING (NLP)

PMDS606L

MODULE 2

LECTURE 1

Dr. Kamanasish Bhattacharjee

Assistant Professor

Dept. of Analytics, SCOPE, VIT

OBJECTIVES OF TEXT PRE-PROCESSING

- Enhance data quality
- Remove unnecessary data which does not contribute to task
- Enable accurate feature extraction
- Improve model performance and efficiency

LOWERCASING

Input: “Natural Language Processing is FUN”

Output: “natural language processing is fun”

REMOVE NUMBERS

Application-specific: in sentiment analysis, numbers may not add value

Example:
Input: “The price is 3000 INR”

Output: “The price is INR”

REMOVE REPEATED CHARACTERS

Input: “I am sooooo happy”

Output: “I am so happy”

REMOVE STOPWORDS

Stopwords: common words with little semantic content

Input: “This is a simple example to demonstrate how stopword removal works in natural language processing.”

Output: “simple example demonstrate stopword removal works natural language processing”

HANDLING CONTRACTIONS

```
import contractions
contractions.fix("I'm learning NLP")

# Output: "I am learning NLP"
```

HANDLING EMOJIS

Replace emojis with descriptive tags

😊 → happy_face

Libraries: emoji, demoji

TOKENIZATION

- Computers process numerical data, not text.
- Tokenization enables text to be:

  Indexed

  Analyzed

  Fed into machine learning or deep learning models
- It reduces complexity by decomposing sentences into meaningful parts.

WORD TOKENIZATION

- Splitting sentences into words using whitespace and punctuation
- Input: "Natural Language Processing is fun."
- Output: ['Natural', 'Language', 'Processing', 'is', 'fun', '.']

SUBWORD TOKENIZATION

- Breaks rare or unknown words into meaningful subword units (morphemes).
- Improves handling of out-of-vocabulary words in neural models.
- **Byte Pair Encoding (BPE)** (used in GPT models), **WordPiece** (used in BERT), **Unigram Language Model** (used in SentencePiece)
- Input: "unhappiness”
- Output: ['un', '##happi', '##ness']

CHARACTER TOKENIZATION

- Splits text into individual characters.
- Useful in languages like Chinese, or for tasks like language modelling and spelling correction.
- Input: "NLP”
- Output: ['N', 'L', 'P']

SENTENCE TOKENIZATION

- Splits a paragraph into sentences.
- Based on punctuation and capitalization cues.
- Input: "Dr. Smith is a linguist. He lives in the U.S.”
- Output: ["Dr. Smith is a linguist.", "He lives in the U.S."]

WHITESPACE TOKENIZATION

- Split the text based on spaces.

- text = "Tokenization is essential in NLP."
- tokens = text.split()
- print(tokens)
- # Output: ['Tokenization', 'is', 'essential', 'in', 'NLP.’]

- Punctuation remains attached to words (e.g., 'NLP.').

RULE-BASED / REGEX-BASED TOKENIZATION

- Uses **regular expressions** to define patterns for splitting.
- Can handle punctuation better than whitespace tokenization.
- import re
- text = "Tokenization: essential in NLP, isn't it?"
- tokens = re.findall(r'\b\w+\b', text)
- print(tokens)
- # Output: ['Tokenization', 'essential', 'in', 'NLP', 'isn', 't', 'it’]
- Cannot handle contractions or special characters perfectly.

NLTK TOKENIZATION

- Uses the **Punkt tokenizer** model (unsupervised machine learning-based).
- Handles punctuation and contractions better than regex.

- from nltk.tokenize import word_tokenize
- text = "Tokenization is essential in NLP, isn't it?"
- tokens = word_tokenize(text)
- print(tokens)
- # Output: ['Tokenization', 'is', 'essential', 'in', 'NLP', ',', 'is', "n't", 'it', '?']

spaCy TOKENIZATION

- Uses an advanced rule-based tokenizer with dependency parsing.

- import spacy
- nlp = spacy.load("en_core_web_sm")
- doc = nlp("Tokenization is essential in NLP, isn't it?")
- tokens = [token.text for token in doc]
- print(tokens)
- # Output: ['Tokenization', 'is', 'essential', 'in', 'NLP', ',', 'is', "n't", 'it', '?']