Practical Assignment No. 4	
Title:	**Text preprocessing**
Problem Statement:	Pre-process text data by tokenization, stemming, and removing stop-words.
Objective:	To develop applications for text processing using natural language processing techniques.
Outcome:	CO606.2: Develop text processing application using natural language processing techniques.
Software or Hardware Requirements:	Anaconda/Java/GCC
Theory:	**Text preprocessing:** Text preprocessing is a fundamental step in Natural Language Processing (NLP) and Text Mining. Raw text data collected from sources such as web pages, social media, or documents is often unstructured, inconsistent, and noisy. Therefore, before feeding it into an analytical model or classifier, it must be cleaned and converted into a structured format. The main goal of text preprocessing is to: ● Remove irrelevant and redundant information. ● Standardize text representation. ● Improve the efficiency and performance of NLP models. **Text Preprocessing Pipeline** Text preprocessing generally includes the following key steps: 1. Text Cleaning 2. Tokenization 3. Stop-word Removal

4. Stemming
5. Lemmatization
6. Lowercasing / Case Normalization
7. Punctuation and Special Character Removal
8. Removing Numbers
9. Handling Emojis, URLs, and Hashtags (for social media text)

Each step plays a specific role in improving text quality.

Techniques of Text Preprocessing

(a) Text Cleaning

Definition:
Text cleaning removes unnecessary characters, symbols, HTML tags, URLs, or other noise from the text.

Purpose:
To ensure that the text contains only relevant linguistic information.

Example:
Input:

"Hello!!! Visit us at https://example.com 😊"
After Cleaning:
"Hello Visit us"

Python Example:

```
import re
cleaned_text = re.sub(r"http\S+|www\S+|[^A-Za-z\s]", "", text)
```

(b) Tokenization

Definition:
Tokenization is the process of breaking text into smaller units called tokens (words, phrases, or sentences).
Tokens act as input for feature extraction.

	Example: Input: “Machine learning improves automation.” Tokens: ['Machine', 'learning', 'improves', 'automation'] Python Example: from nltk.tokenize import word_tokenize tokens = word_tokenize(text) **(c) Lowercasing (Case Normalization)** Definition: Converting all text into lowercase to ensure uniformity and avoid treating the same word differently (e.g., *Machine* and *machine*). Python Example: text = text.lower() **(d) Stop-word Removal** Definition: Stop-words are common words such as *the, is, in, at, on, for* that do not carry significant meaning. Removing them reduces noise and dimensionality. Example: Input: ['machine', 'learning', 'is', 'important'] Output: ['machine', 'learning', 'important'] Python Example: from nltk.corpus import stopwords stop_words = set(stopwords.words('english')) filtered = [w for w in tokens if w.lower() not in stop_words] **(e) Stemming** **Definition:** Stemming reduces words to their **root form** by removing suffixes and prefixes using rule-based algorithms. However, the stem may not always be a valid English word. **Example:**

Original	Stemmed
studying	studi
studies	studi
student	student

Python Example:

```
from nltk.stem import PorterStemmer
ps = PorterStemmer()
stemmed = [ps.stem(w) for w in filtered]
```

(f) Lemmatization

Definition:
Lemmatization is similar to stemming but returns a **valid base (dictionary) word**, using linguistic analysis and part-of-speech tagging.

Example:

Original	Lemma
studies	study
better	good
running	run

Python Example:

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
lemmas = [lemmatizer.lemmatize(w) for w in filtered]
```

(g) Punctuation Removal

Definition:
Punctuations (like commas, periods, and exclamation marks) are removed as they usually do not contribute to the meaning in analysis.

Example:

Input: “Hello, how are you?”
Output: “Hello how are you”

Python Example:

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

(h) Removing Numbers

In many text analytics tasks, numbers do not add semantic value and can be removed.

Python Example:

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

(i) Handling URLs, Emojis, and Hashtags

For social media or web text, additional cleaning steps include:

- Removing URLs: `re.sub(r'http\S+', '', text)`
- Removing Emojis: `emoji.replace_emoji(text, replace='')`
- Removing Hashtags/Symbols: `re.sub(r'#\w+', '', text)`

Applications of Text Preprocessing

- Sentiment Analysis (e.g., positive/negative review classification)
- Spam Filtering (e.g., email classification)
- Chatbots and Virtual Assistants
- Machine Translation
- Document Categorization
- Text Summarization and Search Engines

Advantages

- Removes noise and irrelevant data

	• Reduces computational complexity • Improves model accuracy and interpretability • Ensures consistent text representation • Enhances efficiency of NLP pipelines
Input/Datasets/Test Cases:	Dataset- Name of the Dataset: Description of the Dataset: Dataset Characteristics: Subject Area: Associated Tasks: Feature Type: # Instances: # Features:
Results:	Execute code to use any one dataset and all text preprocessing techniques. Take a print of this code with output for submission as part of results.
Analysis and conclusion:	Write your own analysis of output and conclusion(Minimum 2 statements of Analysis for two types of NBimplementation each, Minimum 1 Statement Conclusion)
References:	Reference /Links(min Any 2, include dataset ref.). Write references in IEEE format.