Vidyavardhini's College of Engineering and Technology, Vasai

Department of Computer Science & Engineering (Data Science)

AY: 2025-26

Class:	**BE-CSE(DS)**	**Semester:**	**VII**
Course Code:	**CSDOL7011**	**Course Name:**	**NLP Lab**

Name of Student:	Hitesh Shetye
Roll No. :	49
Experiment No.:	**2**
Title of the Experiment:	**Text Preprocessing and Feature Engineering using Bag-of-Words and TF-IDF**
Date of Performance:	
Date of Submission:	

Evaluation

Performance Indicator	Max. Marks	Marks Obtained
Performance	5	
Understanding	5	
Journal work and timely submission	10	
Total	20	

Performance Indicator	Exceed Expectations (EE)	Meet Expectations (ME)	Below Expectations (BE)
Performance	4-5	2-3	1
Understanding	4-5	2-3	1
Journal work and timely submission	8-10	5-8	1-4

Checked by

Name of Faculty :

Signature :

Date :

Aim: To apply text preprocessing techniques and extract features from text using Bag-of-Words and TF-IDF methods.

Objective: To understand and apply basic text preprocessing and feature extraction techniques like Bag-of-Words and TF-IDF.

Tools Required:

1. Python (preferably via Jupyter Notebook or Google Colab)
2. NLTK (Natural Language Toolkit)
3. Scikit-learn (sklearn)
4. Pandas

Procedure:

1. Import necessary libraries:
    a. nltk, sklearn.feature_extraction.text, pandas, re
2. Load or define a small sample text dataset.
3. Perform the following text preprocessing:
    a. Convert text to lowercase
    b. Remove punctuation and special characters
    c. Tokenize the text
    d. Remove stop words
    e. Apply stemming or lemmatization
4. Feature Extraction:
    a. Apply Bag-of-Words (BoW) vectorization using CountVectorizer.
    b. Apply TF-IDF vectorization using TfidfVectorizer.
5. Display the resulting feature matrices.
6. Compare and interpret the outputs of BoW and TF-IDF.

Description of the Experiment:

This experiment demonstrates how raw text is cleaned, processed, and converted into structured numerical features using common feature engineering techniques. Students will understand the significance of preprocessing before feeding text data into machine learning models. Both BoW and TF-IDF representations help in quantifying the textual content into a usable format.

Detailed Description of the NLP Technique:

1. Text Preprocessing:

Text preprocessing is a vital step in NLP that transforms unstructured textual data into a clean, machine-readable format. Common steps include:

- **Tokenization**: Splitting text into words or tokens.
- **Stop Word Removal**: Removing common words that don't add significant meaning (e.g., "is", "the", "and").
- **Stemming**: Reducing words to their root form (e.g., "playing" → "play").
- **Lemmatization**: Reducing words to their dictionary form using context (e.g., "better" → "good").

2. Bag-of-Words (BoW):

BoW represents text by counting the frequency of each word in the document. It creates a vocabulary of known words and represents documents using word occurrence counts.

Pros: Simple and interpretable.

Cons: Ignores word order and semantic meaning.

3. TF-IDF (Term Frequency-Inverse Document Frequency):

TF-IDF improves on BoW by assigning weights to words based on their importance in a document relative to the entire corpus.

TF: How often a word appears in a document.

IDF: How rare the word is across all documents.

Formula:

$$\text{TF-IDF}(t,d) = \text{TF}(t,d) \times \log\left(\frac{N}{DF(t)}\right)$$

Where:

- t = term
- d = document
- N = total number of documents
- $DF(t)$ = number of documents containing term t

Code & Output:

Step 1: Import Necessary Libraries →

```
import nltk
import pandas as pd
import re
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt_tab')
```

Step 2: Define a Sample Dataset →

```
data = {
   'Text': [
      "Cats are playing in the garden.",
      "Dogs bark loudly at strangers.",
      "Birds are flying in the sky.",
      "Cats and dogs are friendly pets.",
      "The garden has many beautiful flowers."
   ]
}

df = pd.DataFrame(data)
print(df)
```

```
                                        Text
0         Cats are playing in the garden.
1          Dogs bark loudly at strangers.
2            Birds are flying in the sky.
3        Cats and dogs are friendly pets.
4  The garden has many beautiful flowers.
```

Step 3: Text Preprocessing →

```
stop_words = set(stopwords.words('english'))
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()

def preprocess(text):
   # Lowercase
   text = text.lower()

   # Remove punctuation/special characters
   text = re.sub(r'[^a-z\s]', '', text)

   # Tokenize
   tokens = nltk.word_tokenize(text)

   # Remove stopwords
   tokens = [word for word in tokens if word not in stop_words]

   # Apply stemming
   # tokens = [stemmer.stem(word) for word in tokens]

   # OR Apply lemmatization
```

```
    tokens = [lemmatizer.lemmatize(word) for word in tokens]

    return ' '.join(tokens)

df['Clean_Text'] = df['Text'].apply(preprocess)
print(df)
```

```
                                    Text                  Clean_Text
0         Cats are playing in the garden.           cat playing garden
1          Dogs bark loudly at strangers.     dog bark loudly stranger
2            Birds are flying in the sky.             bird flying sky
3        Cats and dogs are friendly pets.         cat dog friendly pet
4  The garden has many beautiful flowers.  garden many beautiful flower
```

Step 4a: Bag-of-Words Vectorization →

```
bow_vectorizer = CountVectorizer()
bow_matrix = bow_vectorizer.fit_transform(df['Clean_Text'])

print("Bag of Words Feature Matrix:")
print(pd.DataFrame(bow_matrix.toarray(),  columns=bow_vectorizer.get_feature_names_out()))
```

```
Bag of Words Feature Matrix:
   bark  beautiful  bird  cat  dog  flower  flying  friendly  garden  loudly  \
0     0          0     0    1    0       0       0         0       1       0
1     1          0     0    0    1       0       0         0       0       1
2     0          0     1    0    0       0       1         0       0       0
3     0          0     0    1    1       0       0         1       0       0
4     0          1     0    0    0       1       0         0       1       0

   many  pet  playing  sky  stranger
0     0    0        1    0         0
1     0    0        0    0         1
2     0    0        0    1         0
3     0    1        0    0         0
4     1    0        0    0         0
```

Step 4b: TF-IDF Vectorization →

```
tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix = tfidf_vectorizer.fit_transform(df['Clean_Text'])

print("TF-IDF Feature Matrix:")
print(pd.DataFrame(tfidf_matrix.toarray(),  columns=tfidf_vectorizer.get_feature_names_out()))
```

```
TF-IDF Feature Matrix:
       bark  beautiful     bird       cat       dog    flower   flying  \
0  0.000000   0.000000  0.00000  0.531772  0.000000  0.000000  0.00000
1  0.523358   0.000000  0.00000  0.000000  0.422242  0.000000  0.00000
2  0.000000   0.000000  0.57735  0.000000  0.000000  0.000000  0.57735
3  0.000000   0.000000  0.00000  0.444002  0.444002  0.000000  0.00000
4  0.000000   0.523358  0.00000  0.000000  0.000000  0.523358  0.00000

   friendly    garden    loudly      many       pet   playing      sky  \
0  0.000000  0.531772  0.000000  0.000000  0.000000  0.659118  0.00000
1  0.000000  0.000000  0.523358  0.000000  0.000000  0.000000  0.00000
2  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.57735
3  0.550329  0.000000  0.000000  0.000000  0.550329  0.000000  0.00000
4  0.000000  0.422242  0.000000  0.523358  0.000000  0.000000  0.00000

   stranger
0  0.000000
1  0.523358
2  0.000000
3  0.000000
4  0.000000
```

Step 5: Compare and Interpret Outputs →

BoW Matrix:
Each cell indicates the number of times a word appears in a sentence.
Simple frequency count; does not account for word importance across the corpus.
TF-IDF Matrix:
Each cell contains a weight (not count).
Words common across all documents get lower scores.
Helps emphasize unique, informative words.

Conclusion:

In this experiment, I learned how to clean and prepare text data by converting it to lowercase, removing stopwords and special characters, and breaking sentences into individual words. I then used Bag-of-Words and TF-IDF techniques to transform the text into numerical format. While Bag-of-Words counted how often each word appeared, TF-IDF highlighted the importance of words across the dataset. These steps helped me understand the importance of text preprocessing and how it makes data suitable for machine learning models.