Vidyavardhini's College of Engineering & Technology

Department of Artificial Intelligence and Data Science

AY: 2025-26

Class:	BE	**Semester:**	VII
Course Code:	CSDOL7011	**Course Name:**	Natural Language Processing

Name of Student:	Konisha Jayesh Thakare
Roll No. :	71
Experiment No.:	2
Title of the Experiment:	Text Preprocessing and Feature Engineering using Bag-of-Words and TF-IDF
Date of Performance:	22.07.2025
Date of Submission:	29.07.2025

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 : Dr. Tatwadarshi P. Nagarhalli

Signature :

Date :

Aim: To implement text preprocessing techniques and extract features using Bag-of-Words (BoW) and Term Frequency–Inverse Document Frequency (TF-IDF) methods for Natural Language Processing tasks.

Theory:

Text data in its raw form cannot be directly used for machine learning models, as it contains noise such as punctuation, stopwords, and varying word forms. Text preprocessing involves steps such as lowercasing, tokenization, stopword removal, and stemming/lemmatization to convert raw text into a clean, structured format.

Once preprocessing is complete, text must be converted into a numerical representation. Two widely used feature engineering methods are:

Bag-of-Words (BoW): Represents text as a vector of word counts, ignoring grammar and order of words but capturing frequency. It is simple and effective for many tasks but leads to sparse, high-dimensional vectors.

TF-IDF (Term Frequency–Inverse Document Frequency): Improves upon BoW by weighting words based on their frequency across documents. Words that occur frequently in a specific document but rarely across the corpus get higher importance, helping to reduce the impact of common words.

These feature extraction techniques form the foundation of classical NLP pipelines for tasks like text classification, sentiment analysis, and information retrieval.

Procedure:

1. Collect or prepare a small text dataset.
2. Perform preprocessing steps:
3. Convert text to lowercase.
4. Remove punctuation and special characters.
5. Tokenize text into words.
6. Remove stopwords.
7. Apply stemming/lemmatization.
8. Apply Bag-of-Words to generate feature vectors.
9. Apply TF-IDF to generate feature vectors.
10. Compare the results of both methods.

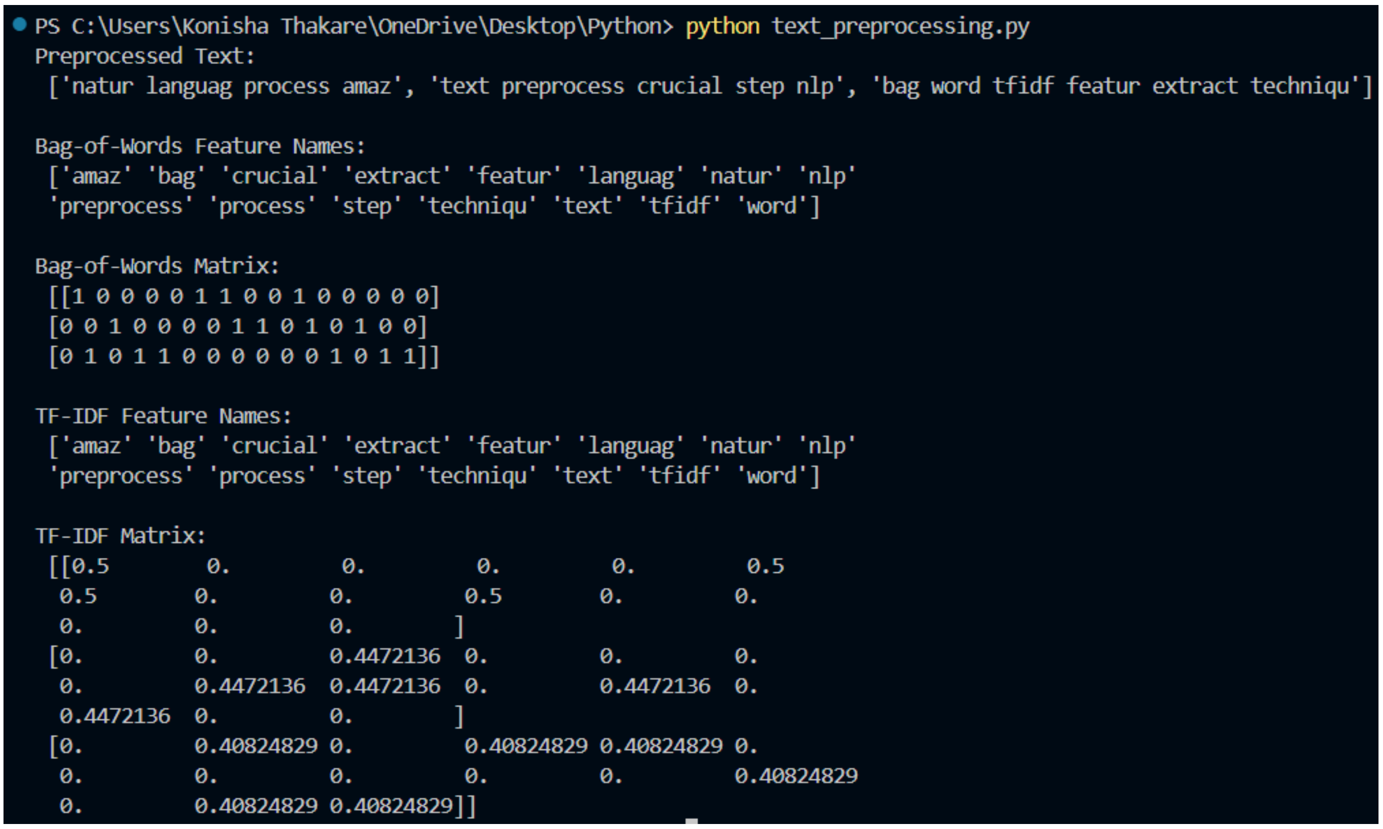
Algorithm

1. Input raw text data.
2. Apply preprocessing (lowercase, tokenization, stopword removal, stemming/lemmatization).
3. Initialize CountVectorizer for Bag-of-Words.
4. Initialize TfidfVectorizer for TF-IDF.
5. Fit and transform the text data into numerical feature vectors.
6. Display and compare the generated feature representations.

Implementation:

```
import nltk, string
from sklearn.feature_extraction.text import CountVectorizer,
TfidfVectorizer
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
def safe_download(pkg):
    try: nltk.data.find(pkg)
    except LookupError: nltk.download(pkg.split("/")[-1])
for pkg in
["tokenizers/punkt","tokenizers/punkt_tab/english","corpora/stopwords"]:
    safe_download(pkg)
corpus=["Natural Language Processing is amazing!","Text preprocessing is a
crucial step in NLP.","Bag of Words and TF-IDF are feature extraction
techniques."]
ps=PorterStemmer()
stop_words=set(stopwords.words('english'))
def preprocess(text):
    text=text.lower()
    text=text.translate(str.maketrans("","",string.punctuation))
    tokens=nltk.word_tokenize(text)
    tokens=[ps.stem(word) for word in tokens if word not in stop_words]
    return " ".join(tokens)
processed_corpus=[preprocess(doc) for doc in corpus]
print("Preprocessed Text:\n",processed_corpus)
bow=CountVectorizer()
bow_features=bow.fit_transform(processed_corpus)
print("\nBag-of-Words Feature Names:\n",bow.get_feature_names_out())
print("\nBag-of-Words Matrix:\n",bow_features.toarray())
tfidf=TfidfVectorizer()
tfidf_features=tfidf.fit_transform(processed_corpus)
print("\nTF-IDF Feature Names:\n",tfidf.get_feature_names_out())
print("\nTF-IDF Matrix:\n",tfidf_features.toarray())
```

Output:



```
● PS C:\Users\Konisha Thakare\OneDrive\Desktop\Python> python text_preprocessing.py
  Preprocessed Text:
   ['natur languag process amaz', 'text preprocess crucial step nlp', 'bag word tfidf featur extract techniqu']

  Bag-of-Words Feature Names:
   ['amaz' 'bag' 'crucial' 'extract' 'featur' 'languag' 'natur' 'nlp'
   'preprocess' 'process' 'step' 'techniqu' 'text' 'tfidf' 'word']

  Bag-of-Words Matrix:
   [[1 0 0 0 0 1 1 0 0 1 0 0 0 0 0]
   [0 0 1 0 0 0 0 1 1 0 1 0 1 0 0]
   [0 1 0 1 1 0 0 0 0 0 0 1 0 1 1]]

  TF-IDF Feature Names:
   ['amaz' 'bag' 'crucial' 'extract' 'featur' 'languag' 'natur' 'nlp'
   'preprocess' 'process' 'step' 'techniqu' 'text' 'tfidf' 'word']

  TF-IDF Matrix:
   [[0.5        0.         0.         0.         0.         0.5
    0.5        0.         0.         0.5        0.         0.
    0.         0.         0.        ]
   [0.         0.         0.4472136  0.         0.         0.
    0.         0.4472136  0.4472136  0.         0.4472136  0.
    0.4472136  0.         0.        ]
   [0.         0.40824829 0.         0.40824829 0.40824829 0.
    0.         0.         0.         0.         0.         0.40824829
    0.         0.40824829 0.40824829]]
```

Conclusion:

This experiment demonstrated how raw text can be preprocessed and transformed into numerical feature representations for NLP tasks. Bag-of-Words provides a simple frequency-based representation but ignores the importance of words across documents, leading to sparsity and limited context understanding. TF-IDF improves upon this by assigning higher weights to words that are unique and informative within a document while reducing the influence of commonly occurring words. Thus, TF-IDF is often more effective in real-world applications, although both methods remain fundamental to many NLP pipelines.