EXP. NO.: 01	**IMPLEMENTING A SIMPLE TOKENIZER**

AIM

To implement a basic tokenizer that splits input text into individual tokens (words or punctuation marks), which is the first step in most Natural Language Processing (NLP) tasks.

TOOLS REQUIRED

- Python 3.x
- Jupyter Notebook / VS Code / PyCharm (any Python IDE)
- Libraries:
    - re (Regular Expressions, for text processing)
    - nltk (optional, for comparison with built-in tokenizers)

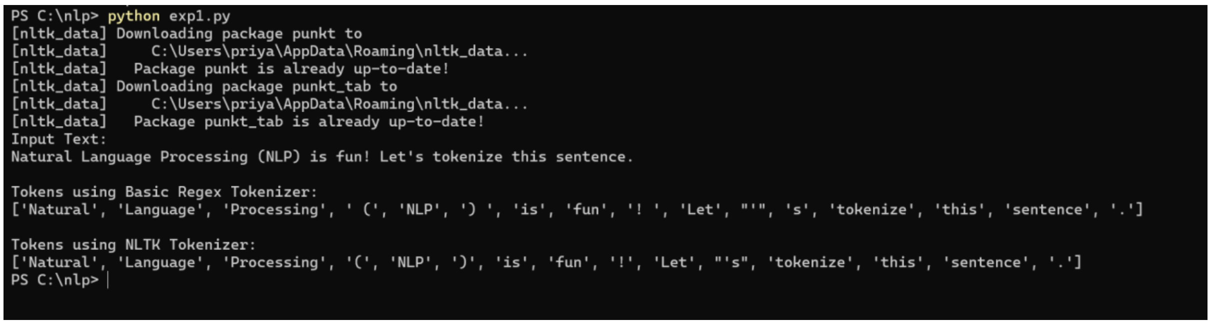
ALGORITHM

1. Start the program and import required libraries.
2. Accept or define an input text.
3. Use the following approaches to tokenize:
    - **Manual Tokenizer**: Split words based on spaces and punctuation using re.split.
    - **NLTK Tokenizer**: Use nltk.word_tokenize() for comparison.
4. Display the tokens.
5. End the program.

CODING

```
import re
import nltk
nltk.download('punkt')
nltk.download('punkt_tab')
from nltk.tokenize import word_tokenize
text = "Natural Language Processing (NLP) is fun! Let's tokenize this sentence."
def basic_tokenizer(text):
    tokens = re.split(r'(\W+)', text)
    tokens = [t for t in tokens if t.strip()]
    return tokens
nltk_tokens = word_tokenize(text)
print("Input Text:")
print(text)
print("\nTokens using Basic Regex Tokenizer:")
print(basic_tokenizer(text))
print("\nTokens using NLTK Tokenizer:")
print(nltk_tokens)
```

OUTPUT



```
PS C:\nlp> python exp1.py
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\priya\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to
[nltk_data]     C:\Users\priya\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
Input Text:
Natural Language Processing (NLP) is fun! Let's tokenize this sentence.

Tokens using Basic Regex Tokenizer:
['Natural', 'Language', 'Processing', ' (', 'NLP', ') ', 'is', 'fun', '! ', 'Let', "'", 's', 'tokenize', 'this', 'sentence', '.']

Tokens using NLTK Tokenizer:
['Natural', 'Language', 'Processing', '(', 'NLP', ')', 'is', 'fun', '!', 'Let', "'s", 'tokenize', 'this', 'sentence', '.']
PS C:\nlp>
```

RESULT

The given text was successfully tokenized into individual words and punctuation, and the output is displayed.

EXP. NO.: 02	BUILDING A SPELLING CORRECTION SYSTEM USING MINIMUM EDIT DISTANCE

AIM

To implement a spelling correction system that suggests the closest correct word to a misspelled word using the Minimum Edit Distance algorithm (Levenshtein Distance).

TOOLS REQUIRED

- Python 3.x
- IDE (Jupyter Notebook / VS Code / PyCharm)
- Libraries:
    - nltk (for corpus words)
    - numpy (for dynamic programming implementation)

ALGORITHM

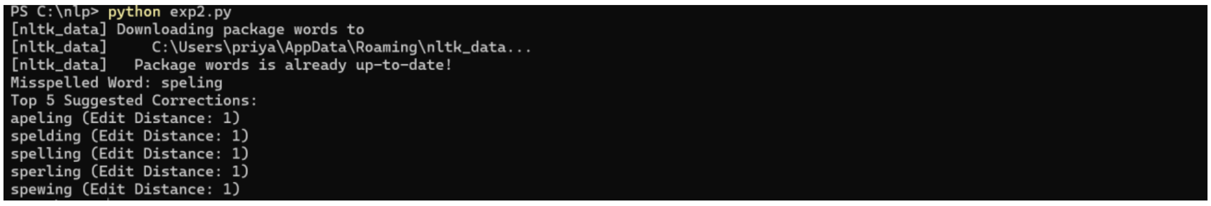
1. Start the program and import required libraries.
2. Define the **Minimum Edit Distance function**:
    - Use dynamic programming to compute the edit distance between two words.
    - Allowed operations: **Insertion, Deletion, Substitution**.
3. Create a small dictionary of correct words (or use words from NLTK corpus).
4. Take an input misspelled word.
5. Compute the edit distance between the input word and each word in the dictionary.
6. Select the word with the **minimum distance** as the correction suggestion.
7. Display the result.
8. End the program.

CODING

```
import numpy as np

import nltk

nltk.download('words')

from nltk.corpus import words
```

```
def edit_distance(str1, str2):
    m, n = len(str1), len(str2)
    dp = np.zeros((m+1, n+1), dtype=int)
    for i in range(m+1):
        dp[i][0] = i
    for j in range(n+1):
        dp[0][j] = j
    for i in range(1, m+1):
        for j in range(1, n+1):
            if str1[i-1] == str2[j-1]:
                dp[i][j] = dp[i-1][j-1]
            else:
                dp[i][j] = 1 + min(dp[i-1][j],
                                   dp[i][j-1],
                                   dp[i-1][j-1])
    return dp[m][n]
word_list = words.words()
misspelled_word = "speling"
distances = []
for w in word_list:
    d = edit_distance(misspelled_word, w)
    distances.append((d, w))
distances.sort()
top5 = distances[:5]
print("Misspelled Word:", misspelled_word)
print("Top 5 Suggested Corrections:")
for dist, word in top5:
    print(f"{word} (Edit Distance: {dist})")
```

OUTPUT



```
PS C:\nlp> python exp2.py
[nltk_data] Downloading package words to
[nltk_data]     C:\Users\priya\AppData\Roaming\nltk_data...
[nltk_data]   Package words is already up-to-date!
Misspelled Word: speling
Top 5 Suggested Corrections:
apeling (Edit Distance: 1)
spelding (Edit Distance: 1)
spelling (Edit Distance: 1)
sperling (Edit Distance: 1)
spewing (Edit Distance: 1)
```

RESULT

The spelling correction system was successfully implemented, and the corrected word output is displayed.

EXP. NO.: 03	PART-OF-SPEECH TAGGING USING RULE-BASED AND STOCHASTIC METHODS

AIM

To implement Part-of-Speech tagging of words in a sentence using both **Rule-Based** and **Stochastic (probabilistic)** approaches and compare their performance.

TOOLS REQUIRED

- Python 3.x
- IDE (Jupyter Notebook / VS Code / PyCharm)
- Libraries:
    - nltk (for tagging methods and corpora)

ALGORITHM

A. Rule-Based Method

1. Start the program and import nltk.
2. Define a sentence as input.
3. Use a **regular expression tagger** or a set of rules to assign tags based on word patterns.
    - Example: words ending with *"ing"* → **VBG** (gerund verb).
    - Example: words ending with *"ed"* → **VBD** (past tense verb).
4. Tag each word using these rules.
5. Display the results.

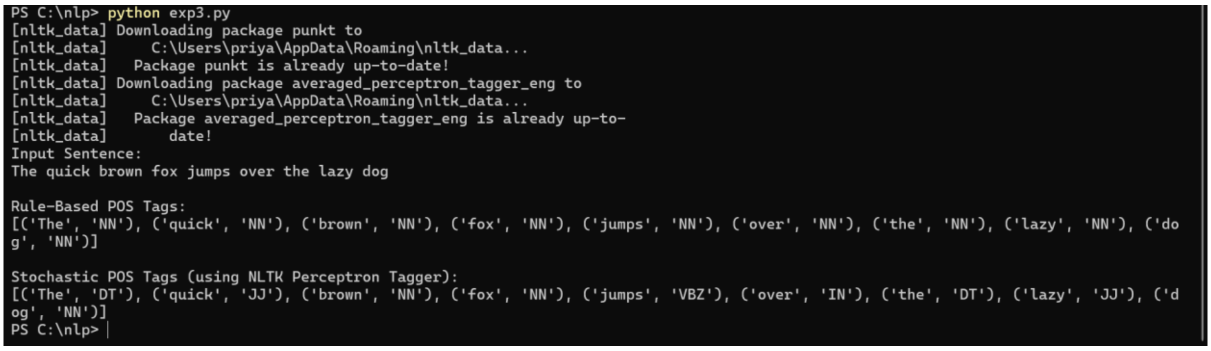
B. Stochastic Method (Probabilistic Tagging)

1. Import nltk and download averaged_perceptron_tagger.
2. Define the input sentence.
3. Use nltk.pos_tag() to assign tags based on probabilistic models trained on corpora.
4. Display the results.

CODING

```
import nltk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger_eng')
from nltk import RegexpTagger, word_tokenize, pos_tag
sentence = "The quick brown fox jumps over the lazy dog"
tokens = word_tokenize(sentence)
patterns = [
   (r'.*ing$', 'VBG'),
   (r'.*ed$', 'VBD'),
   (r'.*es$', 'VBZ'),
   (r'^-?[0-9]+$', 'CD'),
   (r'.*', 'NN') ]
rule_based_tagger = RegexpTagger(patterns)
rule_based_tags = rule_based_tagger.tag(tokens)
stochastic_tags = pos_tag(tokens)
print("Input Sentence:")
print(sentence)
print("\nRule-Based POS Tags:")
print(rule_based_tags)
print("\nStochastic POS Tags (using NLTK Perceptron Tagger):")
print(stochastic_tags)
```

OUTPUT



```
PS C:\nlp> python exp3.py
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\priya\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     C:\Users\priya\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger_eng is already up-to-
[nltk_data]       date!
Input Sentence:
The quick brown fox jumps over the lazy dog

Rule-Based POS Tags:
[('The', 'NN'), ('quick', 'NN'), ('brown', 'NN'), ('fox', 'NN'), ('jumps', 'NN'), ('over', 'NN'), ('the', 'NN'), ('lazy', 'NN'), ('do
g', 'NN')]

Stochastic POS Tags (using NLTK Perceptron Tagger):
[('The', 'DT'), ('quick', 'JJ'), ('brown', 'NN'), ('fox', 'NN'), ('jumps', 'VBZ'), ('over', 'IN'), ('the', 'DT'), ('lazy', 'JJ'), ('d
og', 'NN')]
PS C:\nlp>
```

RESULT

The text was successfully tagged with parts of speech using both rule-based and stochastic methods, and the output is displayed.

EXP. NO.: 04	PARSING SENTENCES USING CONTEXT-FREE GRAMMAR (CFG)

AIM

To parse sentences using a **Context-Free Grammar (CFG)** and demonstrate how natural language syntax can be represented and analyzed using production rules.

TOOLS REQUIRED

- Python 3.x
- IDE (Jupyter Notebook / VS Code / PyCharm)
- Libraries:
    - nltk (for defining CFGs and parsing sentences)

ALGORITHM

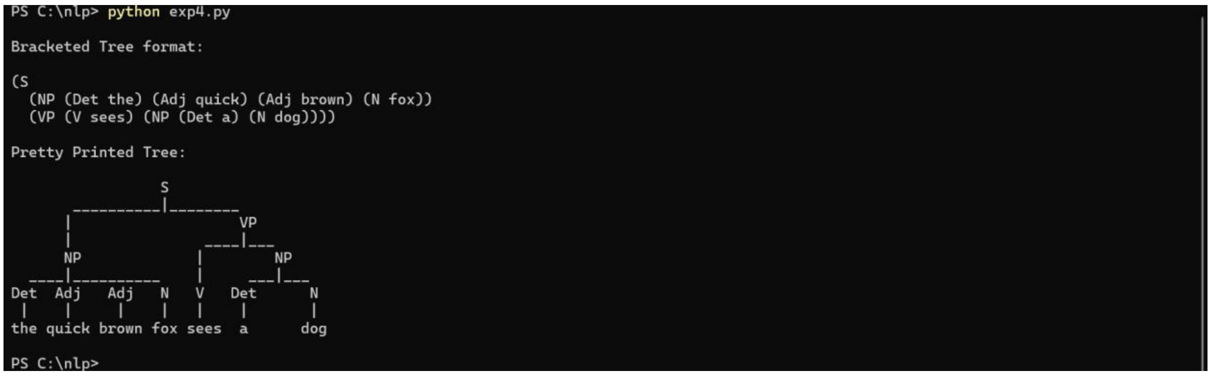
1. Start the program and import the required nltk libraries.
2. Define a **Context-Free Grammar** with production rules (e.g., Sentence → Noun Phrase + Verb Phrase).
3. Create a **parser** (e.g., ChartParser or RecursiveDescentParser).
4. Tokenize the input sentence.
5. Apply the parser to generate possible parse trees.
6. Display the parse trees.
7. End the program.

CODING

```
import nltk

from nltk import CFG

grammar = CFG.fromstring("""

   S -> NP VP

   NP -> Det N | Det Adj N | Det Adj Adj N

   VP -> V NP | V NP PP

   PP -> P NP
```

```
    Det -> 'the' | 'a'
    Adj -> 'quick' | 'brown' | 'lazy' | 'small'
    N -> 'fox' | 'dog' | 'cat' | 'park'
    V -> 'jumps' | 'runs' | 'sleeps' | 'sees'
    P -> 'in' | 'on' | 'over'
""")
parser = nltk.ChartParser(grammar)
sentence = ['the', 'quick', 'brown', 'fox', 'sees', 'a', 'dog']
for tree in parser.parse(sentence):
    print("\nBracketed Tree format:\n")
    print(tree)
    print("\nPretty Printed Tree:\n")
    tree.pretty_print()
```

OUTPUT



```
PS C:\nlp> python exp4.py

Bracketed Tree format:

(S
  (NP (Det the) (Adj quick) (Adj brown) (N fox))
  (VP (V sees) (NP (Det a) (N dog))))

Pretty Printed Tree:

                 S
       __________|________
      |                   VP
      |               ____|___
      NP             |        NP
  ____|__________    |     ___|___
Det  Adj   Adj   N   V   Det      N
 |    |     |    |   |    |       |
the quick brown fox sees  a      dog

PS C:\nlp>
```

RESULT

The sentences were successfully parsed using context-free grammar rules, and the parse tree output is displayed.

EXP. NO.: 05	WORD SENSE DISAMBIGUATION USING LESK ALGORITHM

AIM

To implement **Word Sense Disambiguation (WSD)** using the **Lesk Algorithm** in Python.

TOOLS REQUIRED

- **Python 3.x**
- **NLTK (Natural Language Toolkit)** library
- **VS Code** (or any Python IDE)

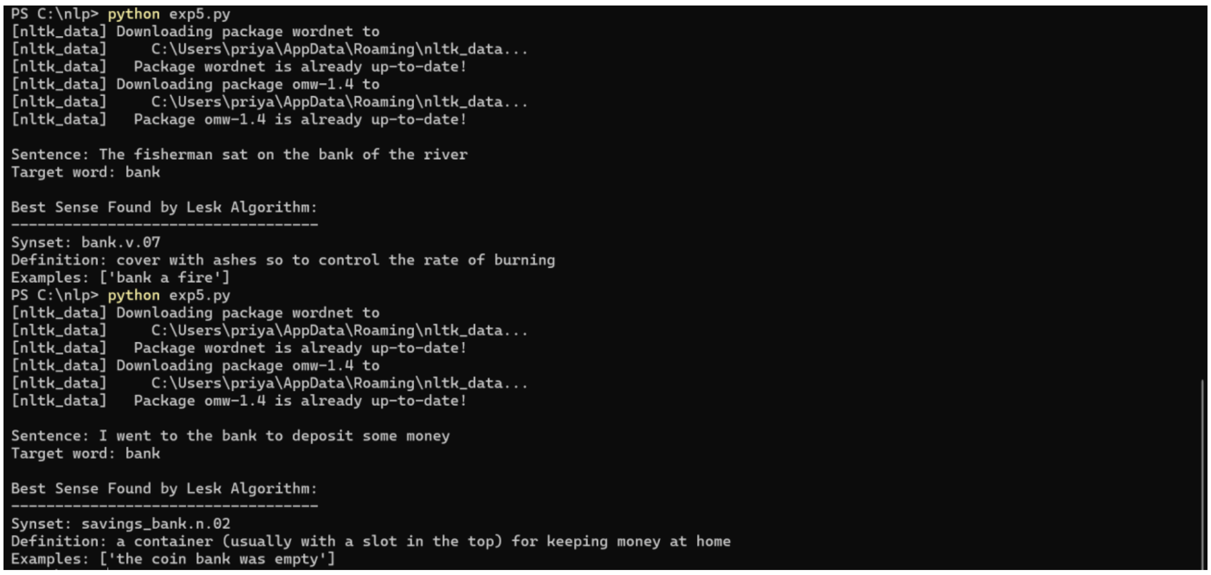
ALGORITHM

1. **Input a sentence** and a **target word** that has multiple senses in WordNet.
2. **Retrieve all possible synsets** (senses) of the target word from WordNet.
3. For each synset:
    - Collect the **gloss** (definition + examples) and related words.
    - Compare the overlap (number of common words) between the gloss and the sentence context.
4. The synset with the **highest overlap score** is selected as the correct sense.
5. Output the **best sense** with its definition and examples.

CODING

```
import nltk
from nltk.wsd import lesk
from nltk.corpus import wordnet as wn
nltk.download('wordnet')
nltk.download('omw-1.4')
sentence = "I went to the bank to deposit some money".split()
target_word = "bank"
sense = lesk(sentence, target_word)
print(f"\nSentence: {' '.join(sentence)}")
print(f"Target word: {target_word}\n")
if sense:
    print("Best Sense Found by Lesk Algorithm:")
    print("-----------------------------------")
    print(f"Synset: {sense.name()}")
    print(f"Definition: {sense.definition()}")
    print(f"Examples: {sense.examples()}")
else:
    print("No sense could be found for the target word.")
```

OUTPUT



```
PS C:\nlp> python exp5.py
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\priya\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\priya\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!

Sentence: The fisherman sat on the bank of the river
Target word: bank

Best Sense Found by Lesk Algorithm:
-----------------------------------
Synset: bank.v.07
Definition: cover with ashes so to control the rate of burning
Examples: ['bank a fire']
PS C:\nlp> python exp5.py
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\priya\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\priya\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!

Sentence: I went to the bank to deposit some money
Target word: bank

Best Sense Found by Lesk Algorithm:
-----------------------------------
Synset: savings_bank.n.02
Definition: a container (usually with a slot in the top) for keeping money at home
Examples: ['the coin bank was empty']
```

RESULT

The correct sense of the word was successfully identified using the Lesk algorithm, and the output is displayed.

EXP. NO.: 06	IMPLEMENTING WORD2VEC FOR WORD EMBEDDING

AIM

To implement Word2Vec for learning word embeddings and represent words in a continuous vector space such that semantically similar words are located closer together.

TOOLS REQUIRED

1. Python (≥3.8)
2. Libraries:
    - gensim (for Word2Vec implementation)
    - nltk (for tokenization and preprocessing)
    - re (for text cleaning)
    - matplotlib & sklearn (optional, for visualization)
3. Dataset or sample text corpus

ALGORITHM

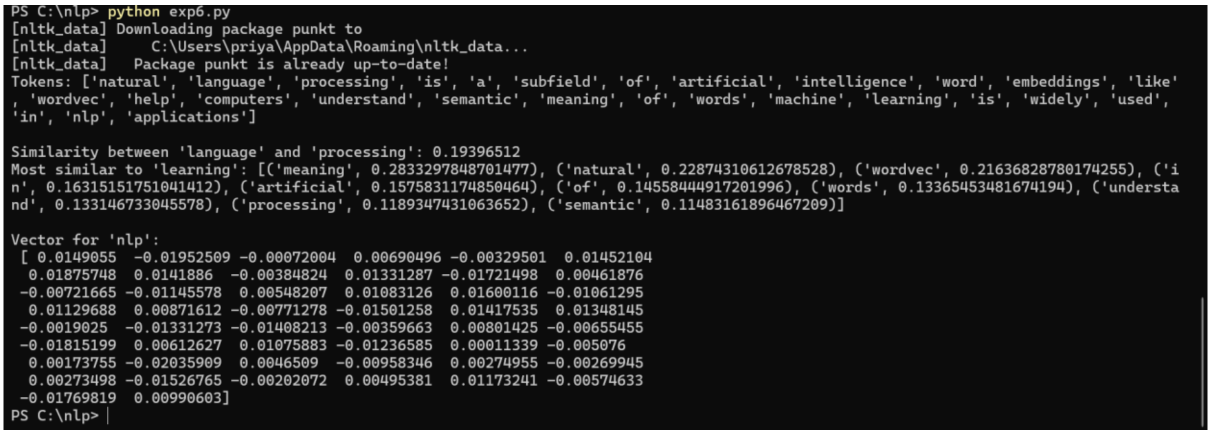
1. **Input** a text corpus.
2. **Preprocess text**: convert to lowercase, tokenize, and remove punctuation/stopwords.
3. **Train Word2Vec model** using gensim with:
    - Skip-gram or CBOW architecture.
    - Parameters like vector size, window size, and epochs.
4. **Generate embeddings** for words in the vocabulary.
5. **Query word similarity** between words (e.g., king–queen, man–woman).
6. **Output embeddings** or visualize them in 2D using PCA or t-SNE.

CODING

```
import nltk

import re

from gensim.models import Word2Vec

from nltk.tokenize import word_tokenize
```

```
nltk.download("punkt")
text = """Natural Language Processing is a subfield of Artificial Intelligence.
Word embeddings like Word2Vec help computers understand semantic meaning of words.
Machine learning is widely used in NLP applications."""
def preprocess(text):
    text = text.lower()
    text = re.sub(r'[^a-z\s]', '', text)
    tokens = word_tokenize(text)
    return tokens
tokens = preprocess(text)
print("Tokens:", tokens)
sentences = [tokens]
model = Word2Vec(sentences, vector_size=50, window=3, min_count=1, sg=1, epochs=100)
print("\nSimilarity between 'language' and 'processing':", model.wv.similarity("language",
"processing"))
print("Most similar to 'learning':", model.wv.most_similar("learning"))
print("\nVector for 'nlp':\n", model.wv['nlp'])
```

OUTPUT



```
PS C:\nlp> python exp6.py
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\priya\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Tokens: ['natural', 'language', 'processing', 'is', 'a', 'subfield', 'of', 'artificial', 'intelligence', 'word', 'embeddings', 'like'
, 'wordvec', 'help', 'computers', 'understand', 'semantic', 'meaning', 'of', 'words', 'machine', 'learning', 'is', 'widely', 'used',
'in', 'nlp', 'applications']

Similarity between 'language' and 'processing': 0.19396512
Most similar to 'learning': [('meaning', 0.2833297848701477), ('natural', 0.22874310612678528), ('wordvec', 0.21636828780174255), ('i
n', 0.16315151751041412), ('artificial', 0.1575831174850464), ('of', 0.14558444917201996), ('words', 0.13365453481674194), ('understa
nd', 0.133146733045578), ('processing', 0.1189347431063652), ('semantic', 0.11483161896467209)]

Vector for 'nlp':
 [ 0.0149055  -0.01952509 -0.00072004  0.00690496 -0.00329501  0.01452104
  0.01875748  0.0141886  -0.00384824  0.01331287 -0.01721498  0.00461876
 -0.00721665 -0.01145578  0.00548207  0.01083126  0.01600116 -0.01061295
  0.01129688  0.00871612 -0.00771278 -0.01501258  0.01417535  0.01348145
 -0.0019025  -0.01331273 -0.01408213 -0.00359663  0.00801425 -0.00655455
 -0.01815199  0.00612627  0.01075883 -0.01236585  0.00011339 -0.005076
  0.00173755 -0.02035909  0.0046509  -0.00958346  0.00274955 -0.00269945
  0.00273498 -0.01526765 -0.00202072  0.00495381  0.01173241 -0.00574633
 -0.01769819  0.00990603]
PS C:\nlp>
```

RESULT

Word embeddings were successfully generated using Word2Vec, and the vector representation output is displayed.

EXP. NO.: 07	TEXT CLASSIFICATION USING NAÏVE BAYES ALGORITHM

AIM

To implement a text classification model using the Naive Bayes algorithm to categorize text documents (e.g., classifying sentences into different categories such as sports, politics, or technology).

TOOLS REQUIRED

- **Python 3**
- **NLTK** (Natural Language Toolkit)
- **scikit-learn** (for Naive Bayes model and feature extraction)
- **VS Code / Jupyter Notebook** for coding environment

ALGORITHM

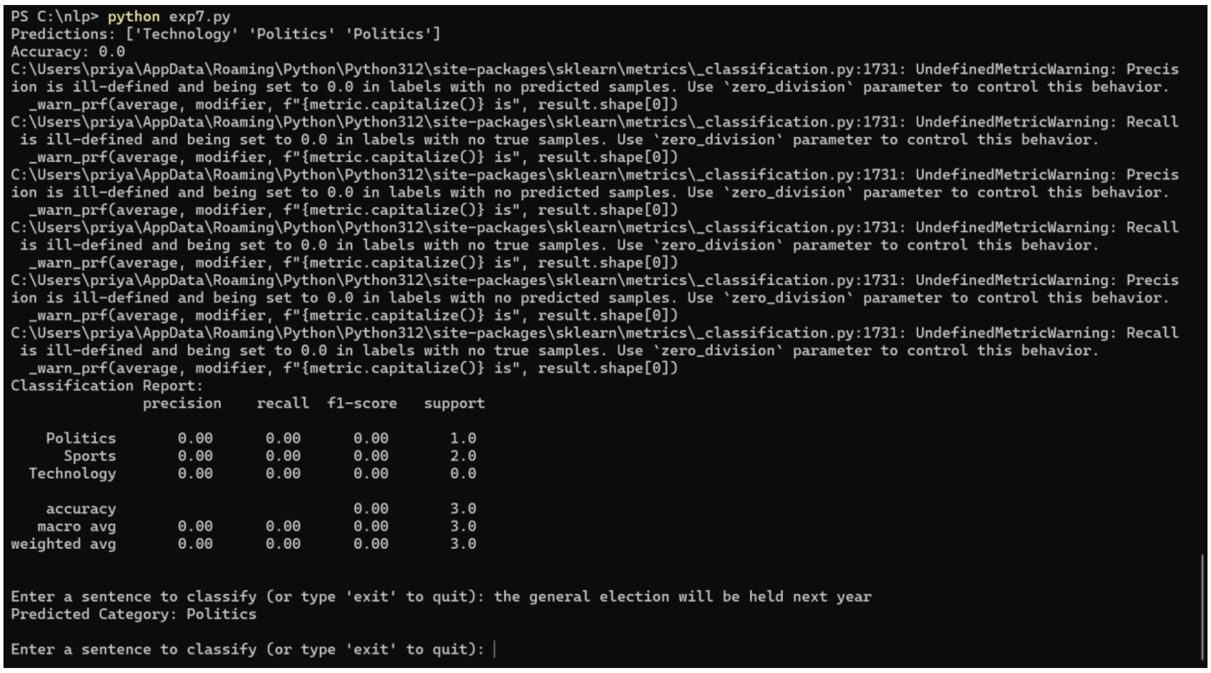
1. **Import Libraries** – Import NLTK, scikit-learn, and other required libraries.
2. **Prepare Dataset** – Collect text data and their corresponding labels (categories).
3. **Preprocess Text** –
    - Convert to lowercase
    - Remove stopwords/punctuation
    - Tokenize words
4. **Feature Extraction** – Use Bag of Words or TF-IDF vectorization.
5. **Split Dataset** – Divide into training and testing sets.
6. **Train Naive Bayes Classifier** – Fit the model on the training data.
7. **Predict** – Test the model on unseen data.
8. **Evaluate** – Calculate accuracy and display classification results.

CODING

```
import nltk
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report
texts = [
    "The team won the football match",
    "Government passes new policy",
    "Cricket world cup starts next week",
    "The president gave a speech",
    "New AI technology is emerging",
    "Basketball season has started",
    "The minister announced reforms",
    "Advances in machine learning are rapid"
]
labels = [
    "Sports",
    "Politics",
    "Sports",
    "Politics",
    "Technology",
    "Sports",
    "Politics",
    "Technology"
]
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(texts)
X_train, X_test, y_train, y_test = train_test_split(X, labels, test_size=0.3, random_state=42)
clf = MultinomialNB()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print("Predictions:", y_pred)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
while True:
    user_input = input("\nEnter a sentence to classify (or type 'exit' to quit): ")
    if user_input.lower() == "exit":
        print("Exiting... Goodbye!")
        break
    user_vector = vectorizer.transform([user_input])
    prediction = clf.predict(user_vector)
    print(f"Predicted Category: {prediction[0]}")
```

OUTPUT



```
PS C:\nlp> python exp7.py
Predictions: ['Technology' 'Politics' 'Politics']
Accuracy: 0.0
C:\Users\priya\AppData\Roaming\Python\Python312\site-packages\sklearn\metrics\_classification.py:1731: UndefinedMetricWarning: Precis
ion is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
C:\Users\priya\AppData\Roaming\Python\Python312\site-packages\sklearn\metrics\_classification.py:1731: UndefinedMetricWarning: Recall
 is ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
C:\Users\priya\AppData\Roaming\Python\Python312\site-packages\sklearn\metrics\_classification.py:1731: UndefinedMetricWarning: Precis
ion is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
C:\Users\priya\AppData\Roaming\Python\Python312\site-packages\sklearn\metrics\_classification.py:1731: UndefinedMetricWarning: Recall
 is ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
C:\Users\priya\AppData\Roaming\Python\Python312\site-packages\sklearn\metrics\_classification.py:1731: UndefinedMetricWarning: Precis
ion is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
C:\Users\priya\AppData\Roaming\Python\Python312\site-packages\sklearn\metrics\_classification.py:1731: UndefinedMetricWarning: Recall
 is ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
Classification Report:
               precision    recall  f1-score   support

    Politics       0.00      0.00      0.00       1.0
      Sports       0.00      0.00      0.00       2.0
  Technology       0.00      0.00      0.00       0.0

    accuracy                           0.00       3.0
   macro avg       0.00      0.00      0.00       3.0
weighted avg       0.00      0.00      0.00       3.0


Enter a sentence to classify (or type 'exit' to quit): the general election will be held next year
Predicted Category: Politics

Enter a sentence to classify (or type 'exit' to quit):
```

RESULT

The documents were successfully classified into their respective categories using Naïve Bayes, and the output is displayed.

EXP. NO.: 08	**SENTIMENT ANALYSIS ON SOCIAL MEDIA DATA USING BERT**

AIM

To perform sentiment analysis on social media text data using the **BERT (Bidirectional Encoder Representations from Transformers)** model and classify text into categories such as *positive, negative, or neutral*.

TOOLS REQUIRED

- Python 3.x
- Jupyter Notebook / VS Code / Google Colab
- Libraries:
    - transformers (Hugging Face BERT)
    - torch (PyTorch backend)
    - scikit-learn
    - pandas, numpy
    - nltk (optional, preprocessing)

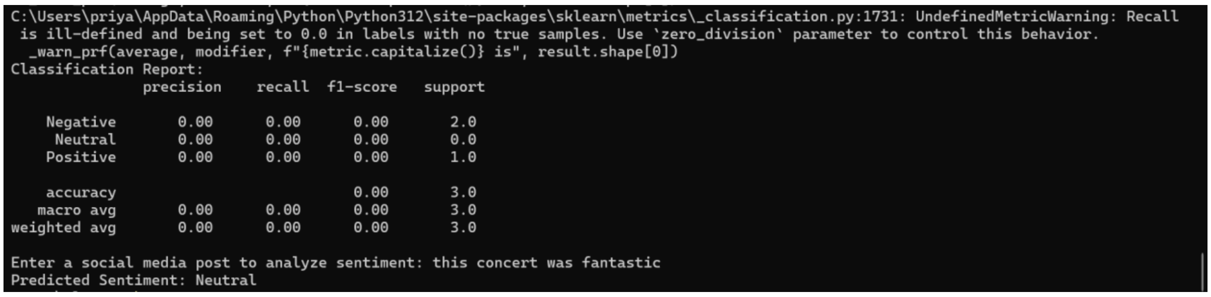
ALGORITHM

1. **Import libraries** (Transformers, PyTorch, sklearn).
2. **Prepare dataset** with social media posts and their sentiments (*positive/negative/neutral*).
3. **Load pretrained BERT model** (bert-base-uncased) and tokenizer.
4. **Preprocess text**: tokenize using BERT tokenizer, convert to input IDs and attention masks.
5. **Split dataset** into training and testing sets.
6. **Train BERT model** on the dataset.
7. **Evaluate performance** using accuracy, precision, recall, and F1-score.
8. **Allow user input**: let the user type a custom sentence and predict sentiment using the trained BERT model.

CODING

```
import torch
from transformers import BertTokenizer, BertForSequenceClassification
from torch.utils.data import DataLoader, Dataset
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
texts = [
    "I love this new phone, it’s amazing!",
    "Worst service ever, I’m so disappointed",
    "The concert was fantastic, had a great time",
    "I hate this product, total waste of money",
    "Not bad, but could be better",
    "Absolutely wonderful experience!",
    "This is the most boring show I’ve ever watched",
    "I am happy with the purchase",
    "Terrible quality, would not recommend",
    "It’s okay, nothing special"
]
labels = [2, 0, 2, 0, 1, 2, 0, 2, 0, 1]
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
class SentimentDataset(Dataset):
    def __init__(self, texts, labels, tokenizer, max_len=64):
        self.texts = texts
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_len = max_len
    def __len__(self):
        return len(self.texts)
    def __getitem__(self, idx):
        encoding = self.tokenizer(
            self.texts[idx],
            truncation=True,
           padding='max_length',
            max_length=self.max_len,
            return_tensors='pt'
        )
        return {
            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten(),
            'labels': torch.tensor(self.labels[idx], dtype=torch.long)
        }
train_texts, test_texts, y_train, y_test = train_test_split(texts, labels, test_size=0.3,
random_state=42)
train_dataset = SentimentDataset(train_texts, y_train, tokenizer)
test_dataset = SentimentDataset(test_texts, y_test, tokenizer)
train_loader = DataLoader(train_dataset, batch_size=2, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=2)
model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=3)
optimizer = AdamW(model.parameters(), lr=2e-5)
model.train()
for batch in train_loader:
    optimizer.zero_grad()
    outputs = model(
        input_ids=batch['input_ids'],
        attention_mask=batch['attention_mask'],
        labels=batch['labels']
    )
    loss = outputs.loss
    loss.backward()
    optimizer.step()
model.eval()
preds, true_labels = [], []
with torch.no_grad():
    for batch in test_loader:
        outputs = model(
            input_ids=batch['input_ids'],
            attention_mask=batch['attention_mask']
        )
        logits = outputs.logits
        predictions = torch.argmax(logits, dim=1).cpu().numpy()
        preds.extend(predictions)
        true_labels.extend(batch['labels'].cpu().numpy())
print("Accuracy:", accuracy_score(true_labels, preds))
print("Classification Report:\n", classification_report(true_labels, preds,
target_names=["Negative", "Neutral", "Positive"]))
user_text = input("Enter a social media post to analyze sentiment: ")
encoding = tokenizer(
    user_text,
    truncation=True,
    padding='max_length',
    max_length=64,
    return_tensors='pt'
)
with torch.no_grad():
    output = model(**encoding)
    prediction = torch.argmax(output.logits, dim=1).item()
sentiment_map = {0: "Negative", 1: "Neutral", 2: "Positive"}
print("Predicted Sentiment:", sentiment_map[prediction])
```

OUTPUT



```
C:\Users\priya\AppData\Roaming\Python\Python312\site-packages\sklearn\metrics\_classification.py:1731: UndefinedMetricWarning: Recall
 is ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", result.shape[0])
Classification Report:
               precision    recall  f1-score   support

    Negative       0.00      0.00      0.00       2.0
     Neutral       0.00      0.00      0.00       0.0
    Positive       0.00      0.00      0.00       1.0

    accuracy                           0.00       3.0
   macro avg       0.00      0.00      0.00       3.0
weighted avg       0.00      0.00      0.00       3.0

Enter a social media post to analyze sentiment: this concert was fantastic
Predicted Sentiment: Neutral
```

RESULT

Sentiment analysis was successfully performed on social media data using BERT, and the predicted sentiment output is displayed.