S. B. JAIN INSTITUTE OF TECHNOLOGY, MANAGEMENT & RESEARCH, NAGPUR.

Practical No. 7

Aim: Apply Text processing, tokenization on given text file & create bigram, and trigram.

Name of Student: **Shrutika Pradeep Bagdi**

Roll No.: **CS22130**

Semester/Year: **IV/VII**

Academic Session: **2025-2026**

Date of Performance: _____

Date of Submission: _____

AIM: Apply Text processing, tokenization on given text file & create bigram, and trigram.

OBJECTIVE/EXPECTED LEARNING OUTCOME:

- Understanding word tokenization
- Understanding N Gram data modelling

HARDWARE AND SOFTWARE REQUIRMENTS:

Hardware Requirement:

Software Requirement:

THEORY:

Tokenization is essentially splitting a phrase, sentence, paragraph, or an entire text document into smaller units, such as individual words or terms. Each of these smaller units are called tokens. These tokens help in understanding the context or developing the model for the NLP. The tokenization helps in interpreting the meaning of the text by analyzing the sequence of the words. For example, the text "It is raining" can be tokenized into 'It', 'is', 'raining
There are different methods and libraries available to perform tokenization. NLTK, Gensim, Keras are some of the libraries that can be used to accomplish the task. Tokenization can be done to either separate words or sentences. If the text is split into words using some separation technique it is called word tokenization and same separation done for sentences is called sentence tokenization.

Purpose:
Tokenization is performed on the corpus to obtain tokens. The tokens are then used to prepare a vocabulary. Vocabulary refers to the set of unique tokens in the corpus. Remember that vocabulary can be constructed by considering each unique token in the corpus or by considering the top K Frequently Occurring Words.

Steps:

1. Load the demotext.txt text file into a variable and then close the file
2. Do word wise tokenization list out generated tokens
3. Transform each token into a small case
4. Remove stop words from the generated token list
5. Remove extra symbols like commas, full stops, and question marks using a regular expression tokenizer and store them in another variable
6. Do bigram and trigram for generated tokens

CODE:

```
from google.colab import drive
drive.mount('/content/drive')
import nltk
import re
from nltk.corpus import stopwords
from nltk.util import ngrams

# Download required NLTK data
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('punkt_tab')

# 1. Load the file into a variable
with open(r"/demotext.txt", "r", encoding="utf-8-sig") as file:
    text = file.read()
print("Original Text:\n", text)

# 2. Word-wise tokenization
tokens = nltk.word_tokenize(text)
print("Original Tokens:", tokens)

# 3. Transform each token into lowercase
tokens_lower = [token.lower() for token in tokens]
print("\nLowercase Tokens:", tokens_lower)

# 4. Remove stopwords
stop_words = set(stopwords.words('english'))
tokens_no_stop = [word for word in tokens_lower if word not in stop_words]
print("\nTokens without Stopwords:", tokens_no_stop)

# 5. Remove extra symbols using regex tokenizer
regex_tokens = nltk.RegexpTokenizer(r'\w+').tokenize(text.lower())
regex_tokens_no_stop = [word for word in regex_tokens if word not in stop_words]
print("\nRegex Clean Tokens:", regex_tokens_no_stop)

# 6. Generate bigram
bigrams = list(ngrams(regex_tokens_no_stop, 2))
print("\nBigrams:", bigrams)

# 7. Generate trigram
trigrams = list(ngrams(regex_tokens_no_stop, 3))
print("\nTrigrams:", trigrams)
```

OUTPUT (SCREENSHOT):

```
What is Lorem Ipsum?
```

```
Lorem Ipsum is simply dummy text of the printing and
typesetting industry. Lorem Ipsum has been the industry's
standard dummy text ever since the 1500s, when an unknown
printer took a galley of type and scrambled it to make a
type specimen book. It has survived not only five centuries,
but also the leap into electronic typesetting,
remaining essentially unchanged.


It was popularised in the 1960s with the release of Letraset
sheets containing Lorem Ipsum passages, and more recently
with desktop publishing software like Aldus PageMaker
including versions of Lorem Ipsum.
```

```
[ ] # 2. Word-wise tokenization
    tokens = nltk.word_tokenize(text)
    print("Original Tokens:", tokens)
```

```
Original Tokens: ['What', 'is', 'Lorem', 'Ipsum', '?', 'Lorem', 'Ipsum', 'is', 'simply', 'dummy', 'text', 'of', 'the', 'printing', 'and', 'typesetting',
```

```
[ ] # 3. Transform each token into lowercase
    tokens_lower = [token.lower() for token in tokens]
    print("\nLowercase Tokens:", tokens_lower)
```

```
Lowercase Tokens: ['what', 'is', 'lorem', 'ipsum', '?', 'lorem', 'ipsum', 'is', 'simply', 'dummy', 'text', 'of', 'the', 'printing', 'and', 'typesetting',
```

```
[ ] # 4. Remove stopwords
    stop_words = set(stopwords.words('english'))
    tokens_no_stop = [word for word in tokens_lower if word not in stop_words]
    print("\nTokens without Stopwords:", tokens_no_stop)
```

```
Tokens without Stopwords: ['lorem', 'ipsum', '?', 'lorem', 'ipsum', 'simply', 'dummy', 'text', 'printing', 'typesetting', 'industry'
```

```
[ ] # 5. Remove extra symbols using regex tokenizer
    regex_tokens = nltk.RegexpTokenizer(r'\w+').tokenize(text.lower())
    regex_tokens_no_stop = [word for word in regex_tokens if word not in stop_words]
    print("\nRegex Clean Tokens:", regex_tokens_no_stop)
```

```
Regex Clean Tokens: ['lorem', 'ipsum', 'lorem', 'ipsum', 'simply', 'dummy', 'text', 'printing', 'typesetting', 'industry', 'lorem',
```

```
[ ] # 6. Generate bigram
    bigrams = list(ngrams(regex_tokens_no_stop, 2))
    print("\nBigrams:", bigrams)
```

```
Bigrams: [('lorem', 'ipsum'), ('ipsum', 'lorem'), ('lorem', 'ipsum'), ('ipsum', 'simply'), ('simply', 'dummy'), ('dummy', 'text'), ('text', 'printing'), ('pri
```

```
[ ] # 7. Generate trigram
    trigrams = list(ngrams(regex_tokens_no_stop, 3))
    print("\nTrigrams:", trigrams)
```

```
Trigrams: [('lorem', 'ipsum', 'lorem'), ('ipsum', 'lorem', 'ipsum'), ('lorem', 'ipsum', 'simply'), ('ipsum', 'simply', 'dummy'), ('simply', 'dummy', 'text'),
```

POS Tagging - Viterbi Decoding

Corpus A

Book a car. Park the car. The book is in the car. The car is in a park.

EMISSION MATRIX

	book	park	car	is	in	a	the
determiner	0	0	0	0	0	1	1
noun	0.5	0.5	1	0	0	0	0
verb	0.5	0.5	0	1	0	0	0
preposition	0	0	0	0	1	0	0

TRANSITION MATRIX

	eos	determiner	noun	verb	preposition
eos	0	0.33	0	0.5	0
determiner	0	0	1	0	0
noun	1	0	0	0.5	0
verb	0	0.33	0	0	1
preposition	0	0.33	0	0	0

Viterbi Decoding

Sentence: Book a park

You can answer a column only if you have completed the previous ones.(note:answer can also be upto three decimals)

	Book	a	park
determiner	0	0.165	0
noun	0	0	0.165
verb	0.5	0	0
preposition	0	0	0

Check Part of Speech

Viterbi Decoding

Sentence: Book a park

You can answer a column only if you have completed the previous ones.(note:answer can also be upto three decimals)

Bookaparkdeterminer0 0.1650noun00 0.165verb 0.500preposition000POS tags for the words in the sentence are as following:

Book is verb

a is determiner

park is noun

CONCLUSION:

DISCUSSION AND VIVA VOCE:

- What is Tokenization?
- Why tokenization is done?
- Elaborate N gram modeling.

REFERENCE: