Experiment: 6

AIM: Write a Program on Bi-grams.

OBJECTIVES: To understand and implement **bi-grams**, which are pairs of consecutive words in a given text.

REQUIREMENTS: Python (version 3.x or above), NLTK library.

THEORY:

In Language modeling is the way of determining the probability of any sequence of words. Language modeling is used in various applications such as Speech Recognition, Spam filtering, etc. Language modeling is the key aim behind implementing many state-of-the-art Natural Language Processing models.

For instance, N-grams can be unigrams like ("This", "article", "is", "on", "NLP") or bigrams ("This article", "article is", "is on", "on NLP").

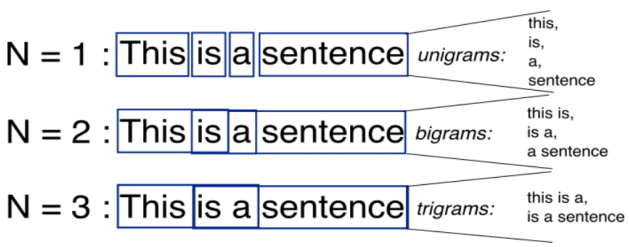
Examples:

- **Unigrams (1-grams):** Single words, e.g., "cat," "dog."
- **Bigrams (2-grams):** Pairs of consecutive words, e.g., "natural language," "deep learning."
- **Trigrams (3-grams):** Triplets of consecutive words, e.g., "machine learning model," "data science approach."
- **N-Gram :** An N-gram is a contiguous sequence of n items (usually words) from a given text.

What are Bigrams?

In a sequence of text, bigrams are pairs of consecutive words or tokens. Bigrams allow us to see which words commonly co-occur within a given dataset, which can be particularly useful for:

- Predictive text and autocomplete features, where the next word is predicted based on the previous word.
- Speech recognition systems to improve accuracy by considering two words at a time.
- Information retrieval systems to enhance search accuracy.



Implementing Bi-grams using NLTK library:

Step 1: Importing necessary libraries.

```
import nltk
from nltk.util import bigrams
from nltk.tokenize import word_tokenize
# Download required tokenizer models
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\tdhan\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

Step 2: Taking Input text and dividing them in form of tokens.

```
text = "Natural Language Processing with Python is powerful and \
interesting to train Various Machine learning and Deep learning models."

# Word tokenization
tokens = word_tokenize(text)

print("Word Tokens:")
print(tokens)
```

```
Word Tokens:
['Natural', 'Language', 'Processing', 'with', 'Python', 'is', 'powerful', 'and', 'interesting', 'to', 'train', 'Various',
achine', 'learning', 'and', 'Deep', 'learning', 'models', '.']
```

Step 3: Generating Bi-grams using the above text data.

```
# Generate bi-grams
bi_grams = list(bigrams(tokens))
print("\nBi-grams:")
for bg in bi_grams:
    print(bg)
```

```
Bi-grams:
('Natural', 'Language')
('Language', 'Processing')
('Processing', 'with')
('with', 'Python')
('Python', 'is')
('is', 'powerful')
('powerful', 'and')
('and', 'interesting')
('interesting', 'to')
('to', 'train')
('train', 'Various')
('Various', 'Machine')
('Machine', 'learning')
('learning', 'and')
('and', 'Deep')
('Deep', 'learning')
('learning', 'models')
('models', '.')
```

Conclusion

In this experiment, we successfully implemented **bi-gram generation** using the NLTK library in Python.