SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**COURSE CODE: DJ19ITL504** **DATE:03/12/2023**

**COURSE NAME: Artificial Intelligence Laboratory** **CLASS: I2-1**

## EXPERIMENT NO.09

**CO/LO:** Apply NLP techniques to domain-specific problems.

**AIM / OBJECTIVE:** To implement the N-gram model and calculate the probability of a sentence.

**DESCRIPTION OF EXPERIMENT:**

A combination of words forms a sentence. However, such a formation is meaningful only when the words are arranged in some order.

Eg: Sit I car in the

Such a sentence is not grammatically acceptable. However, some perfectly grammatical sentences can be nonsensical too!

Eg: Colorless green ideas sleep furiously

One easy way to handle such unacceptable sentences is by assigning probabilities to the strings of words i.e, how likely the sentence is.

### Probability of a sentence

If we consider each word occurring in its correct location as an independent event, the probability of the sentences is P(w(1), w(2)..., w(n-1), w(n))

Using chain rule: = P(w(1)) * P(w(2) | w(1)) * P(w(3) | w(1)w(2)) ... P(w(n) | w(1)w(2) ... w(n-1))

### Bigrams

We can avoid this very long calculation by approximating that the probability of a given word depends only on the probability of its previous words. This assumption is called the Markov assumption and such a model is called the Markov model- bigrams. Bigrams can be generalized to the n-gram which looks at (n-1) words in the past. A bigram is a first-order Markov model.

Therefore , P(w(1), w(2)..., w(n-1), w(n)) = P(w(2)|w(1)) P(w(3)|w(2)) ... P(w(n)|w(n-1))

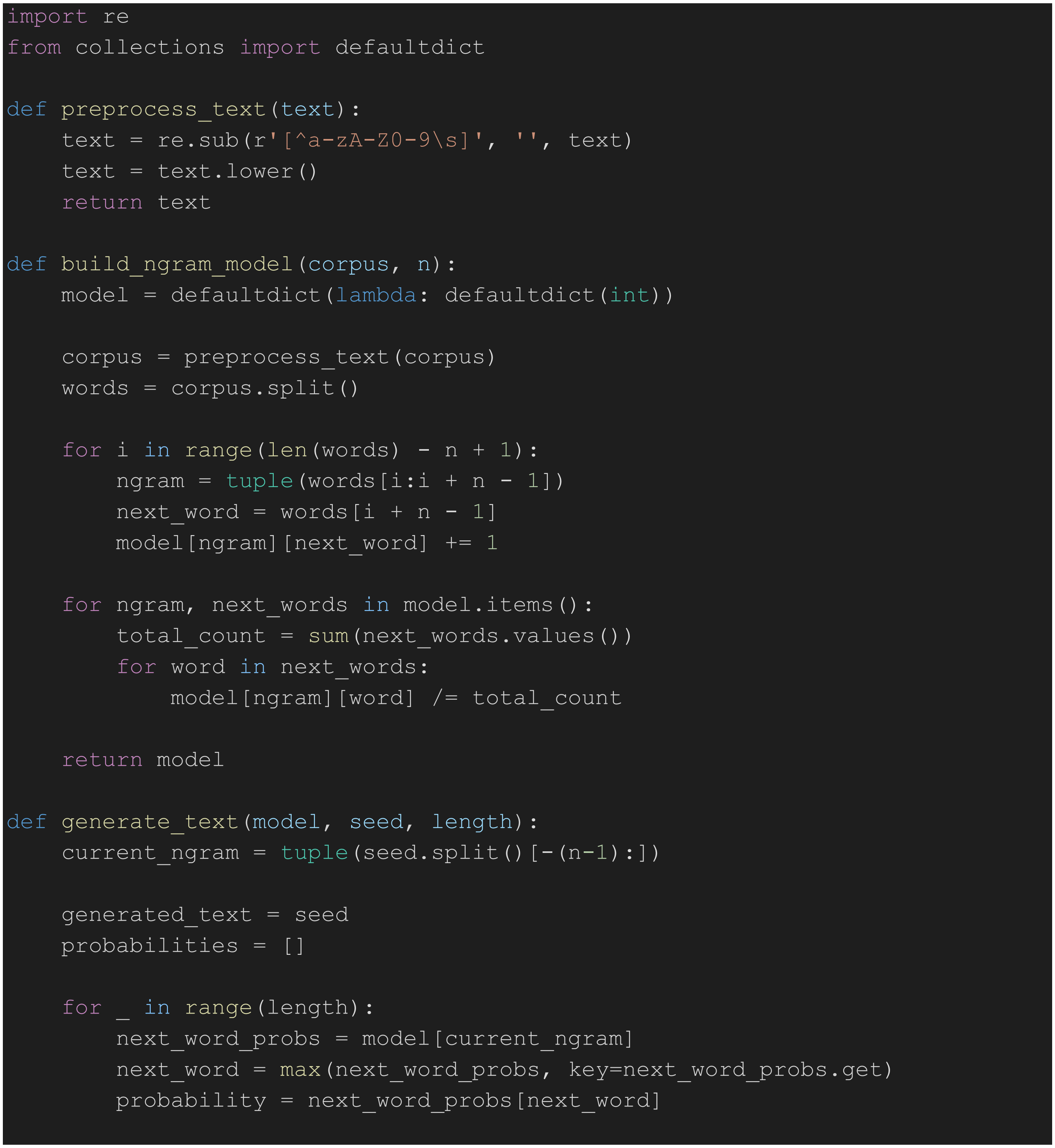
Objective:

1. Take a corpus (minimum6/7 sentences).

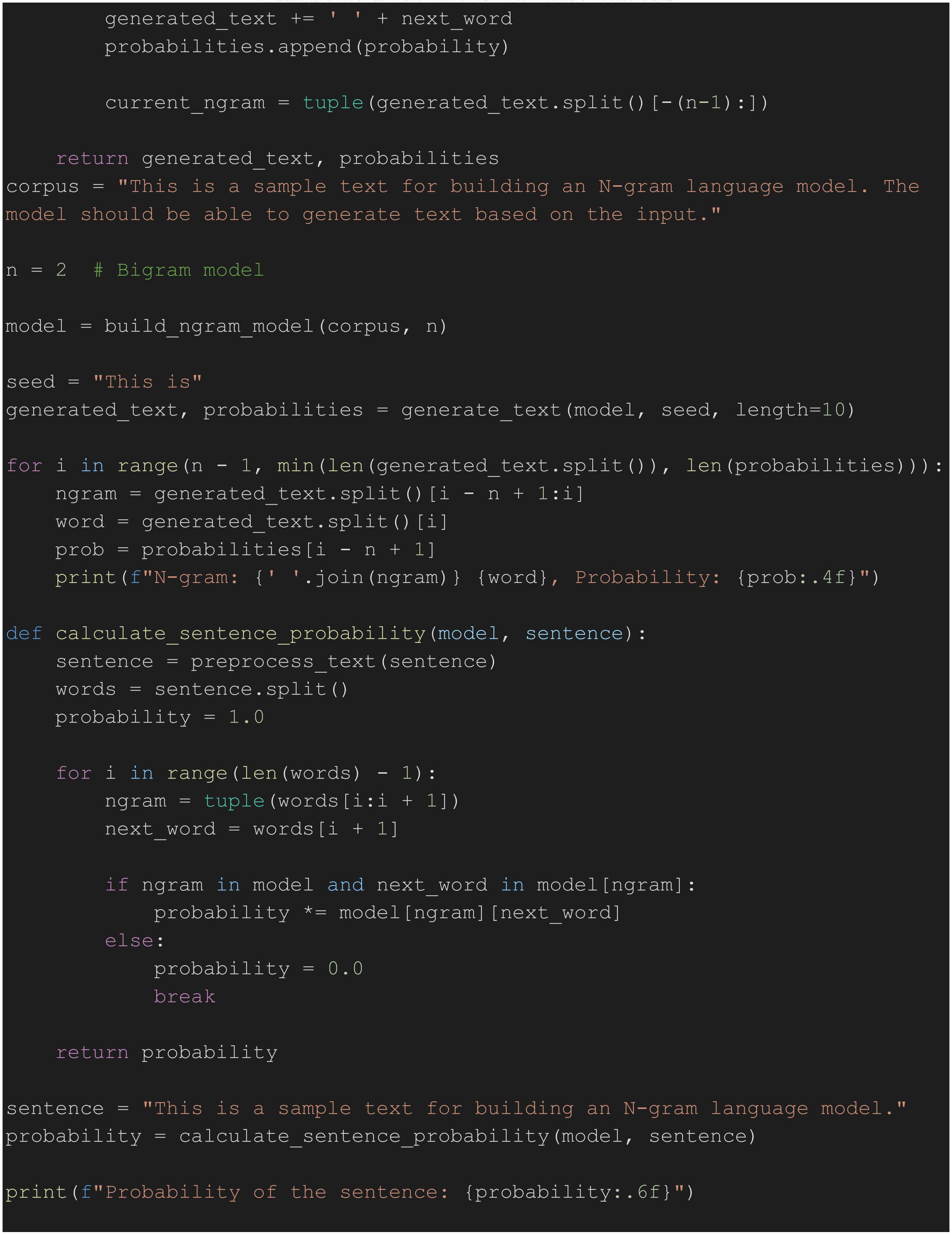
2. Calculate the bi-gram and Trigram probabilities.

3. Analyze the results.

**CODE & OUTPUT:**

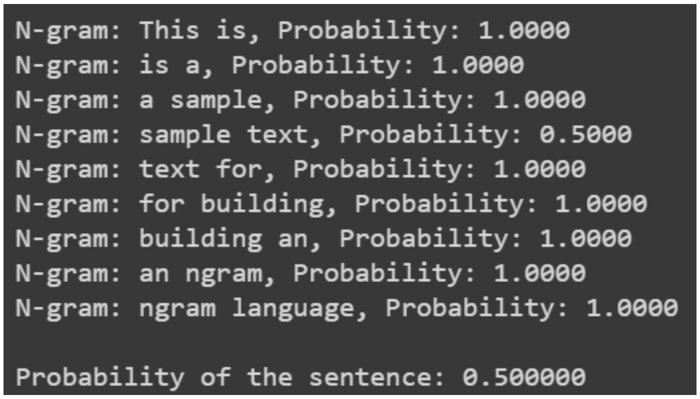


```
import re
from collections import defaultdict

def preprocess_text(text):
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text)
    text = text.lower()
    return text

def build_ngram_model(corpus, n):
    model = defaultdict(lambda: defaultdict(int))

    corpus = preprocess_text(corpus)
    words = corpus.split()

    for i in range(len(words) - n + 1):
        ngram = tuple(words[i:i + n - 1])
        next_word = words[i + n - 1]
        model[ngram][next_word] += 1

    for ngram, next_words in model.items():
        total_count = sum(next_words.values())
        for word in next_words:
            model[ngram][word] /= total_count

    return model

def generate_text(model, seed, length):
    current_ngram = tuple(seed.split()[-(n-1):])

    generated_text = seed
    probabilities = []

    for _ in range(length):
        next_word_probs = model[current_ngram]
        next_word = max(next_word_probs, key=next_word_probs.get)
        probability = next_word_probs[next_word]
```



```
        generated_text += ' ' + next_word
        probabilities.append(probability)

        current_ngram = tuple(generated_text.split()[-(n-1):])

    return generated_text, probabilities
corpus = "This is a sample text for building an N-gram language model. The model should be able to generate text based on the input."

n = 2  # Bigram model

model = build_ngram_model(corpus, n)

seed = "This is"
generated_text, probabilities = generate_text(model, seed, length=10)

for i in range(n - 1, min(len(generated_text.split()), len(probabilities))):
    ngram = generated_text.split()[i - n + 1:i]
    word = generated_text.split()[i]
    prob = probabilities[i - n + 1]
    print(f"N-gram: {' '.join(ngram)} {word}, Probability: {prob:.4f}")

def calculate_sentence_probability(model, sentence):
    sentence = preprocess_text(sentence)
    words = sentence.split()
    probability = 1.0

    for i in range(len(words) - 1):
        ngram = tuple(words[i:i + 1])
        next_word = words[i + 1]

        if ngram in model and next_word in model[ngram]:
            probability *= model[ngram][next_word]
        else:
            probability = 0.0
            break

    return probability

sentence = "This is a sample text for building an N-gram language model."
probability = calculate_sentence_probability(model, sentence)

print(f"Probability of the sentence: {probability:.6f}")
```

**OUTPUT :**



```
N-gram: This is, Probability: 1.0000
N-gram: is a, Probability: 1.0000
N-gram: a sample, Probability: 1.0000
N-gram: sample text, Probability: 0.5000
N-gram: text for, Probability: 1.0000
N-gram: for building, Probability: 1.0000
N-gram: building an, Probability: 1.0000
N-gram: an ngram, Probability: 1.0000
N-gram: ngram language, Probability: 1.0000

Probability of the sentence: 0.500000
```

**CONCLUSION**:

Hence, we have successfully implemented the N-gram model and calculate the probability of a sentence.

**REFERENCES:**

[1] Stuart Russell and Peter Norvig, “Artificial Intelligence: A Modern Approach”, 2nd Edition, Pearson Education, 2010