BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai – 400058-India
Department of Computer Engineering

Name	Raj Kalpesh Mathuria
UID no.	2023300139
Experiment No.	3

AIM:	1) Calculate bigram and trigram probability from a given corpus. 2) Apply add-one smoothing on sparse bigram table 3) Find the probability of the given statement "I drink coffee every morning" and "I drink morning every coffee" by taking an example corpus into consideration.
PRE LAB QUESTIONS:	**1. What problem is solved by smoothing and how does it solve the problem?** In N-gram language models, probabilities are estimated from frequency counts in a training corpus. However, since the corpus is finite, many valid word combinations may never appear in it. When such unseen combinations occur, their frequency count becomes zero, resulting in zero probability. This creates the **sparsity problem**, where even a single unseen N-gram makes the entire sentence probability zero. Smoothing techniques address this issue by assigning small non-zero probabilities to unseen word sequences. In Add-One (Laplace) smoothing, 1 is added to every N-gram count and the denominator is increased accordingly. This redistributes probability mass from frequent events to unseen events, ensuring that no valid sentence receives zero probability. **2. What is Perplexity?** Perplexity is a metric used to evaluate how well a language model predicts a sequence of words. It measures the degree of uncertainty the model has while predicting the next word in a sentence. Mathematically, perplexity is the inverse probability of a test sentence, normalized by the number of words. A lower perplexity value indicates that the model is better at predicting the sequence, while a higher perplexity indicates poor predictive performance. In simple terms, perplexity tells us how "confused" a model is when generating or evaluating text — the lower the confusion, the better the model.

3. Can N-Gram models understand syntax?

No, N-gram models do not truly understand syntax. They operate purely on statistical patterns of word sequences based on fixed-length context. A bigram model considers only one previous word, while a trigram model considers two previous words. Although they may capture some local grammatical patterns, they cannot understand deeper syntactic structures, long-distance dependencies, or hierarchical sentence relationships. Therefore, N-gram models rely on frequency-based approximations rather than actual grammatical understanding.

4. What are the limitations of the N-Gram model? What are the alternatives?

The N-gram model has several limitations. First, it suffers from the sparsity problem because many possible word combinations do not appear in a limited training corpus. Second, it has a fixed context window and cannot capture long-range dependencies beyond $N-1$ words. Third, it requires large storage space for higher-order N-grams due to the exponential growth in possible word combinations. Finally, it does not understand semantics or syntax; it only models surface-level statistical patterns.

To overcome these limitations, modern alternatives such as Neural Network Language Models (NNLMs), Recurrent Neural Networks (RNNs), Long Short-Term Memory networks (LSTMs), and Transformer-based models (such as GPT and BERT) are used. These models can capture long-range dependencies, learn distributed word representations, and model deeper contextual relationships in language more effectively than traditional N-gram models.

THEORY:

Experiment: Statistical Language Modeling using N-Gram Techniques

This experiment focuses on building probabilistic language models using N-gram techniques to analyze and predict word sequences in Natural Language Processing (NLP). A statistical language model estimates the probability of a sequence of words based on patterns learned from a training corpus. By implementing unigram, bigram, and trigram models, the experiment demonstrates how contextual dependency improves sentence probability estimation. N-gram models are based on the **Markov assumption**, which states that the probability of a word depends only on a limited number of preceding words rather than the entire sentence history. This assumption simplifies computation while still capturing useful contextual information.

1. Text Corpus Preparation and Preprocessing

Before constructing language models, the raw text corpus must be processed to ensure uniformity and accurate probability calculations. The preprocessing stage standardizes the dataset and removes inconsistencies.

Key preprocessing operations include:

- Converting all characters to lowercase to avoid case-sensitive mismatches
- Removing punctuation symbols and unnecessary special characters
- Tokenizing text into individual words
- Introducing special boundary markers such as `<s>` (start of sentence) and `</s>` (end of sentence)

The inclusion of sentence boundary tokens helps the model understand sentence structure and improves probability estimation for complete sentences.

2. N-Gram Modeling Approaches

An N-gram is a contiguous sequence of **N words** from a text corpus. Different values of N determine how much contextual information is used.

Unigram Model

- Assumes each word occurs independently of others
- Probability is calculated as:

P(w)=Count(w)/Total WordsP(w)

Bigram Model

- Considers one preceding word as context
- Conditional probability formula:

$P(wi \mid wi-1)=Count(wi-1,wi)/Count(wi-1)$

- Captures short-term dependencies
- More accurate than unigram for sentence modeling

Trigram Model

- Uses two preceding words as context
- Probability estimation:

$P(wi \mid wi-2,wi-1)=Count(wi-2,wi-1,wi)/Count(wi-2,wi-1)$

- Provides richer contextual understanding
- Requires larger datasets due to increased sparsity

As the value of N increases, contextual understanding improves, but computational and memory requirements also increase.

3. Probability Estimation using Maximum Likelihood

Maximum Likelihood Estimation (MLE) is used to compute probabilities from observed frequency counts in the corpus. It estimates probabilities strictly based on training data without making additional assumptions.

However, MLE suffers from a major drawback:
If a particular N-gram does not appear in the corpus, its probability becomes zero. This leads to incorrect sentence probability calculation because multiplication with zero nullifies the entire result.

This issue is referred to as the **zero-frequency problem**.

4. Laplace (Add-One) Smoothing

To overcome zero probabilities, Laplace smoothing is applied by adding one to each count.

Smoothed Bigram Probability:

$P(wi \mid wi-1)=Count(wi-1,wi)+1/Count(wi-1)+V$

Smoothed Trigram Probability:

$P(wi \mid wi-2,wi-1)=Count(wi-2,wi-1,wi)+1Count(wi-2,wi-1)+V$

Where:

- V = Vocabulary size

Advantages of smoothing:

- Prevents zero probabilities
- Improves generalization
- Allows prediction of unseen word sequences

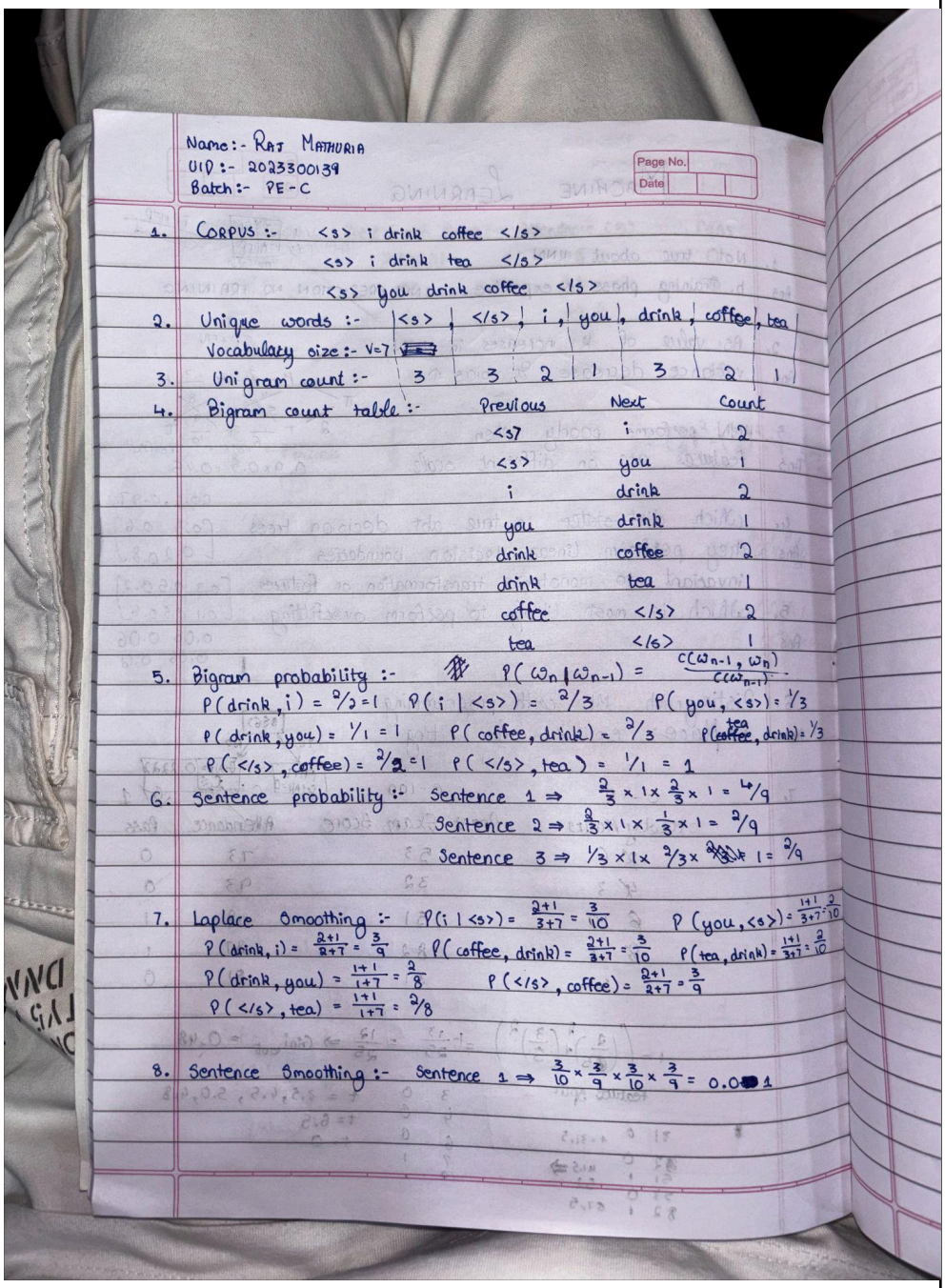
6. Vocabulary and Data Sparsity

Vocabulary size significantly affects model performance:

- Larger vocabulary increases sparsity
- Smaller corpus leads to insufficient N-gram coverage
- Higher-order N-grams require exponentially more data

Data sparsity is one of the biggest challenges in statistical language modeling.

PROBLEM SOLVING:



Name:- Raj Mathuria
UID:- 2023300139
Batch:- PE-C

Page No.
Date

1. Corpus :- <s> i drink coffee </s>
<s> i drink tea </s>
<s> you drink coffee </s>

2. Unique words :- <s>, </s>, i, you, drink, coffee, tea
Vocabulary size :- V=7

3. Unigram count :-

<s>	</s>	i	you	drink	coffee	tea
3	3	2	1	3	2	1

4. Bigram count table :-

Previous	Next	Count
<s>	i	2
<s>	you	1
i	drink	2
you	drink	1
drink	coffee	2
drink	tea	1
coffee	</s>	2
tea	</s>	1

5. Bigram probability :- $P(w_n | w_{n-1}) = \frac{C(w_{n-1}, w_n)}{C(w_{n-1})}$
$P(drink, i) = 2/2 = 1$ $P(i \mid <s>) = 2/3$ $P(you, <s>) = 1/3$
$P(drink, you) = 1/1 = 1$ $P(coffee, drink) = 2/3$ $P(tea, drink) = 1/3$
$P(</s>, coffee) = 2/2 = 1$ $P(</s>, tea) = 1/1 = 1$

6. Sentence probability :- Sentence 1 $\Rightarrow \frac{2}{3} \times 1 \times \frac{2}{3} \times 1 = 4/9$
Sentence 2 $\Rightarrow \frac{2}{3} \times 1 \times \frac{1}{3} \times 1 = 2/9$
Sentence 3 $\Rightarrow 1/3 \times 1 \times 2/3 \times 1 = 2/9$

7. Laplace Smoothing :- $P(i \mid <s>) = \frac{2+1}{3+7} = \frac{3}{10}$ $P(you, <s>) = \frac{1+1}{3+7} = \frac{2}{10}$
$P(drink, i) = \frac{2+1}{2+7} = \frac{3}{9}$ $P(coffee, drink) = \frac{2+1}{3+7} = \frac{3}{10}$ $P(tea, drink) = \frac{1+1}{3+7} = \frac{2}{10}$
$P(drink, you) = \frac{1+1}{1+7} = \frac{2}{8}$ $P(</s>, coffee) = \frac{2+1}{2+7} = \frac{3}{9}$
$P(</s>, tea) = \frac{1+1}{1+7} = 2/8$

8. Sentence Smoothing :- Sentence 1 $\Rightarrow \frac{3}{10} \times \frac{3}{9} \times \frac{3}{10} \times \frac{3}{9} = 0.0$ 1

PROGRAM:

```
from collections import Counter, defaultdict

corpus = [
    "I drink coffee",
    "I drink tea",
    "You drink coffee"
]

sentences = []
for s in corpus:
    sentences.append(["<s>"] + s.lower().split() + ["</s>"])

unigram_counts = Counter()
bigram_counts = defaultdict(Counter)

for sent in sentences:
    for i in range(len(sent)):
        unigram_counts[sent[i]] += 1
        if i > 0:
            bigram_counts[sent[i-1]][sent[i]] += 1

vocab = list(unigram_counts.keys())
V = len(vocab)

print("\nUNIGRAM COUNTS")
for w in unigram_counts:
    print(w, ":", unigram_counts[w])

print("\nBIGRAM COUNTS")
for prev in bigram_counts:
    for curr in bigram_counts[prev]:
        print(f"({prev}, {curr}) :", bigram_counts[prev][curr])

print("\nBIGRAM PROBABILITIES (NO SMOOTHING)")
for prev in bigram_counts:
    for curr in bigram_counts[prev]:
        print(f"P({curr}|{prev}) =", bigram_counts[prev][curr] /
unigram_counts[prev])

print("\nBIGRAM PROBABILITIES (ADD-ONE SMOOTHING)")
for prev in unigram_counts:
    for curr in vocab:
        print(
```

```
            f"P*({curr}|{prev}) =",
            (bigram_counts[prev][curr] + 1) / (unigram_counts[prev] + V)
        )

def sentence_probability(sentence, smoothing=False):
    tokens = ["<s>"] + sentence.lower().split() + ["</s>"]
    prob = 1
    for i in range(1, len(tokens)):
        prev = tokens[i-1]
        curr = tokens[i]
        if smoothing:
            p = (bigram_counts[prev][curr] + 1) / (unigram_counts[prev] + V)
            print(f"P*({curr}|{prev}) =", p)
        else:
            if bigram_counts[prev][curr] == 0:
                print(f"P({curr}|{prev}) = 0")
                return 0
            p = bigram_counts[prev][curr] / unigram_counts[prev]
            print(f"P({curr}|{prev}) =", p)
        prob *= p
    return prob

sentence = input("\nEnter a sentence: ")
threshold = float(input("Enter acceptance threshold: "))

print("\nWITHOUT SMOOTHING")
p1 = sentence_probability(sentence)
print("Final Probability =", p1)
if p1 >= threshold:
    print("Conclusion: ACCEPTED")
else:
    print("Conclusion: NOT ACCEPTED")

print("\nWITH ADD-ONE SMOOTHING")
p2 = sentence_probability(sentence, smoothing=True)
print("Final Probability =", p2)
if p2 >= threshold:
    print("Conclusion: ACCEPTED")
else:
    print("Conclusion: NOT ACCEPTED")
```

RESULT:

```
NLP on  main [!?] is 📦 v0.1.0 via 🐍 v3.10.13 (NLP)
❯ uv run Exp3/exp3.py

UNIGRAM COUNTS
<s> : 3
i : 2
drink : 3
coffee : 2
</s> : 3
tea : 1
you : 1

BIGRAM COUNTS
(<s>, i) : 2
(<s>, you) : 1
(i, drink) : 2
(drink, coffee) : 2
(drink, tea) : 1
(coffee, </s>) : 2
(tea, </s>) : 1
(you, drink) : 1

BIGRAM PROBABILITIES (NO SMOOTHING)
P(i|<s>) = 0.6666666666666666
P(you|<s>) = 0.3333333333333333
P(drink|i) = 1.0
P(coffee|drink) = 0.6666666666666666
P(tea|drink) = 0.3333333333333333
P(</s>|coffee) = 1.0
P(</s>|tea) = 1.0
P(drink|you) = 1.0

BIGRAM PROBABILITIES (ADD-ONE SMOOTHING)
P*(<s>|<s>) = 0.1
```

```
NLP on  main [!?] is 📦v0.1.0 via 🐍v3.10.13 (NLP)
❯ uv run Exp3/exp3.py
BIGRAM PROBABILITIES (ADD-ONE SMOOTHING)
P*(<s>|<s>) = 0.1
P*(i|<s>) = 0.3
P*(drink|<s>) = 0.1
P*(coffee|<s>) = 0.1
P*(</s>|<s>) = 0.1
P*(tea|<s>) = 0.1
P*(you|<s>) = 0.2
P*(<s>|i) = 0.1111111111111111
P*(i|i) = 0.1111111111111111
P*(drink|i) = 0.3333333333333333
P*(coffee|i) = 0.1111111111111111
P*(</s>|i) = 0.1111111111111111
P*(tea|i) = 0.1111111111111111
P*(you|i) = 0.1111111111111111
P*(<s>|drink) = 0.1
P*(i|drink) = 0.1
P*(drink|drink) = 0.1
P*(coffee|drink) = 0.3
P*(</s>|drink) = 0.1
P*(tea|drink) = 0.2
P*(you|drink) = 0.1
P*(<s>|coffee) = 0.1111111111111111
P*(i|coffee) = 0.1111111111111111
P*(drink|coffee) = 0.1111111111111111
P*(coffee|coffee) = 0.1111111111111111
P*(</s>|coffee) = 0.3333333333333333
P*(tea|coffee) = 0.1111111111111111
P*(you|coffee) = 0.1111111111111111
P*(<s>|</s>) = 0.1
P*(i|</s>) = 0.1
P*(drink|</s>) = 0.1
P*(coffee|</s>) = 0.1
```

```
NLP on  main [!?] is 📦v0.1.0 via 🐍v3.10.13 (NLP)
❯ uv run Exp3/exp3.py
P*(tea|</s>) = 0.1
P*(you|</s>) = 0.1
P*(<s>|tea) = 0.125
P*(i|tea) = 0.125
P*(drink|tea) = 0.125
P*(coffee|tea) = 0.125
P*(</s>|tea) = 0.25
P*(tea|tea) = 0.125
P*(you|tea) = 0.125
P*(<s>|you) = 0.125
P*(i|you) = 0.125
P*(drink|you) = 0.25
P*(coffee|you) = 0.125
P*(</s>|you) = 0.125
P*(tea|you) = 0.125
P*(you|you) = 0.125

Enter a sentence: drink tea coffee you
Enter acceptance threshold: 0.0005

WITHOUT SMOOTHING
P(drink|<s>) = 0
Final Probability = 0
Conclusion: NOT ACCEPTED

WITH ADD-ONE SMOOTHING
P*(drink|<s>) = 0.1
P*(tea|drink) = 0.2
P*(coffee|tea) = 0.125
P*(you|coffee) = 0.1111111111111111
P*(</s>|you) = 0.125
Final Probability = 3.472222222222223e-05
Conclusion: NOT ACCEPTED
```

CONCLUSION:	In this experiment, I gained practical insight into how N-gram language models function by implementing unigram, bigram, and trigram models manually using an actual text dataset. I explored how preprocessing steps such as cleaning the text, breaking it into tokens, and inserting special end-of-sentence markers help organize the corpus effectively for modeling. By computing word frequencies and forming bigram probability tables, I understood how probabilities are calculated using Maximum Likelihood Estimation (MLE) as well as Laplace (Add-One) smoothing. This process made it clear why smoothing techniques are important for handling unseen word pairs and avoiding zero probability issues. I also practiced calculating full sentence probabilities with both bigram and trigram approaches. By evaluating different sample sentences, I observed that trigram models capture contextual dependencies more effectively than bigram models. Overall, this activity deepened my understanding of statistical language modeling and the practical implementation of N-gram techniques in Natural Language Processing.