Vidyavardhini's College of Engineering and Technology, Vasai

Department of Computer Science & Engineering (Data Science)

AY: 2025-26

Class:	**BE-CSE(DS)**	**Semester:**	**VII**
Course Code:	**CSDOL7011**	**Course Name:**	**NLP Lab**

Name of Student:	**Sahil Salunke**
Roll No. :	**45**
Experiment No.:	**4**
Title of the Experiment:	**Building Unigram, Bigram, and Trigram Language Models for Word Prediction**
Date of Performance:	
Date of Submission:	

Evaluation

Performance Indicator	Max. Marks	Marks Obtained
Performance	5	
Understanding	5	
Journal work and timely submission	10	
Total	20	

Performance Indicator	Exceed Expectations (EE)	Meet Expectations (ME)	Below Expectations (BE)
Performance	4-5	2-3	1
Understanding	4-5	2-3	1
Journal work and timely submission	8-10	5-8	1-4

Checked by

Name of Faculty :

Signature :

Date :

Aim: To build and evaluate N-gram language models (Unigram, Bigram, and Trigram) to analyze word-level dependencies in text.

Objective: To build N-gram language models for predicting words based on context and analyzing word-level dependencies.

Tools Required:

1. Python (Jupyter Notebook or Google Colab)
2. Nltk
3. collections (for frequency counting)
4. random (for sampling words)

Procedure:

1. Import libraries:
    a. Import nltk, collections.Counter, and any preprocessing modules.

2. Load a sample corpus:
    a. Use nltk.corpus.gutenberg, brown, or a custom dataset.
    b. Preprocess the text (lowercasing, tokenization, stopword removal optional).

3. Build N-gram models:
    a. Unigram: Compute frequency of each word.
    b. Bigram: Count frequency of consecutive word pairs.
    c. Trigram: Count frequency of word triplets.

4. Estimate probabilities:

For bigrams: $P(w_n|w_{n-1}) = \frac{\text{Count}(w_{n-1},w_n)}{\text{Count}(w_{n-1})}$

For trigrams: $P(w_n|w_{n-2},w_{n-1}) = \frac{\text{Count}(w_{n-2},w_{n-1},w_n)}{\text{Count}(w_{n-2},w_{n-1})}$

5. Implement word prediction:
    a. Prompt user to input one or two words.
    b. Predict the next word using the most probable bigram/trigram.

6. Evaluate results:

Print predicted words or top-k probable next words.

Description of the Experiment:

This experiment introduces students to statistical language modeling using N-grams. By computing word sequence probabilities, students observe how simple models can predict the likelihood of a word given its context. This helps them understand foundational ideas behind more advanced models like RNNs or transformers.

Detailed Description of the NLP Technique:

N-gram Language Models:

An N-gram is a contiguous sequence of n words. N-gram language models estimate the probability of a word given its n-1 previous words.

Types:

Unigram Model (n=1): Assumes each word is independent.

Bigram Model (n=2): Considers one preceding word.

Trigram Model (n=3): Considers two preceding words.

Formula for Probability Estimation:

Unigram: $P(w_n) = \frac{\text{Count}(w_n)}{N}$

Bigram: $P(w_n|w_{n-1})$

Trigram: $P(w_n|w_{n-2}, w_{n-1})$

Challenges with N-gram Models:

1. Data sparsity: Many possible combinations may never occur in training data.
2. Smoothing: Techniques like Laplace Smoothing are used to assign non-zero probabilities to unseen N-grams.
3. Limited context: Longer dependencies can't be captured effectively.

Applications:

1. Word prediction
2. Spelling correction
3. Speech recognition

Code:

1. Import Libraries

```
import nltk
from nltk.corpus import gutenberg
from nltk import word_tokenize
from collections import Counter, defaultdict
import random
import string
nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('gutenberg')
```

OUTPUT -->

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
[nltk_data] Downloading package gutenberg to /root/nltk_data...
[nltk_data]   Package gutenberg is already up-to-date!
True
```

2. Load and Preprocess the Corpus

```
raw_text = gutenberg.raw('austen-emma.txt')
tokens = word_tokenize(raw_text.lower())
tokens = [word for word in tokens if word.isalpha()]
```

3. Build N-gram Models

```
# Unigram
unigram_counts = Counter(tokens)

# Bigram
bigrams = list(nltk.bigrams(tokens))
bigram_counts = Counter(bigrams)
```

```
# Trigram
trigrams = list(nltk.trigrams(tokens))
trigram_counts = Counter(trigrams)
```

4. Estimate Probabilities

```
# For unigrams: P(w) = count(w) / total_words
total_unigrams = sum(unigram_counts.values())
unigram_prob = {word: count / total_unigrams for word, count in
unigram_counts.items()}

# For bigrams: P(w2|w1) = count(w1, w2) / count(w1)
bigram_prob = defaultdict(dict)
for (w1, w2), count in bigram_counts.items():
    bigram_prob[w1][w2] = count / unigram_counts[w1]
```

```
# For trigrams: P(w3|w1,w2) = count(w1, w2, w3) / count(w1, w2)
trigram_prob = defaultdict(dict)
bigram_pairs = Counter(list(nltk.bigrams(tokens)))
```

```
for (w1, w2, w3), count in trigram_counts.items():
    trigram_prob[(w1, w2)][w3] = count / bigram_pairs[(w1, w2)]
```

5. Implement Word Prediction

```
def predict_next_word(input_text, top_k=3):
    words = input_text.lower().split()
    if len(words) == 1:
        w1 = words[0]
        if w1 in bigram_prob:
            sorted_probs = sorted(bigram_prob[w1].items(), key=lambda x:
x[1], reverse=True)
            return sorted_probs[:top_k]
    elif len(words) >= 2:
        w1, w2 = words[-2], words[-1]
        if (w1, w2) in trigram_prob:
            sorted_probs = sorted(trigram_prob[(w1, w2)].items(),
key=lambda x: x[1], reverse=True)
            return sorted_probs[:top_k]
    return [("No prediction", 0.0)]
```

6. Evaluate Results

```
while True:
    user_input = input("Enter one or two words (or 'exit' to quit): ")
    if user_input.lower() == 'exit':
        break
    predictions = predict_next_word(user_input)
    print("Predicted next words:")
    for word, prob in predictions:
        print(f"{word} (Prob: {prob:.4f})")
```

OUTPUT -->

```
Enter one or two words (or 'exit' to quit): I am
Predicted next words:
sure (Prob: 0.2716)
not (Prob: 0.0812)
very (Prob: 0.0711)
Enter one or two words (or 'exit' to quit): How are
Predicted next words:
you (Prob: 0.5000)
they (Prob: 0.5000)
Enter one or two words (or 'exit' to quit): He is not
Predicted next words:
a (Prob: 0.0887)
the (Prob: 0.0645)
it (Prob: 0.0565)
Enter one or two words (or 'exit' to quit): exit
```

Conclusion:

The N-gram word prediction system provides contextually relevant results, especially with bigrams and trigrams trained on structured text like "Emma" by Jane Austen. Predictions like "I am → sure/afraid" show the model captures short-term word dependencies well. However, it struggles with unseen or modern words due to limited vocabulary and corpus bias. While it's lightweight and fast, it lacks deeper context understanding. Overall, it's useful for small applications or educational purposes but not ideal for dynamic or diverse language use cases.