AY: 2025-26

Class:	BE	**Semester:**	VII
Course Code:	CSDOL7011	**Course Name:**	Natural Language Processing

Name of Student:	Konisha Jayesh Thakare
Roll No. :	71
Experiment No.:	4
Title of the Experiment:	Building Unigram, Bigram, and Trigram Language Models for Word Prediction
Date of Performance:	05.08.2025
Date of Submission:	12.08.2025

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 : Dr. Tatwadarshi P. Nagarhalli

Signature :

Date :

Aim: To build Unigram, Bigram, and Trigram language models and use them for word prediction in a given text corpus.

Theory:

Traditional Language models estimate the probability of a sequence of words in a language, which is essential for tasks like text prediction, speech recognition, and machine translation.

- **Unigram Model:**

Assumes each word occurs independently. Probability of a word sequence is the product of individual word probabilities:

$P(w1,w2,...,wn)=P(w1)\cdot P(w2)\cdot ...\cdot P(wn)$

Simple but ignores context.

- **Bigram Model:**

Considers the probability of a word given the previous word:

$P(w1,w2,...,wn)=P(w1)\cdot P(w2\mid w1)\cdot ...\cdot P(wn\mid wn-1)$

Captures local context but limited to one preceding word.

- **Trigram Model:**

Considers the probability of a word given the previous two words:

$P(w1,w2,...,wn)=P(w1)\cdot P(w2\mid w1)\cdot P(w3\mid w1,w2)\cdot ...$

Provides better context, improving word prediction accuracy.

N-gram models are widely used in predictive text keyboards, autocomplete, and speech recognition systems.

Procedure:

1. Prepare a sample text corpus.
2. Preprocess the corpus: lowercase, tokenize sentences and words.
3. Build frequency distributions for unigram, bigram, and trigram models.
4. Compute probabilities for each n-gram.
5. Use the models to predict the next word given previous word(s).
6. Compare predictions from unigram, bigram, and trigram models.

Algorithm:

1. Tokenize the text corpus.
2. Generate unigram, bigram, and trigram counts.
3. Compute probabilities for each n-gram.

4. For word prediction:

   Unigram: Choose word with highest unigram probability.

   Bigram: Choose word with highest probability following previous word.

   Trigram: Choose word with highest probability following previous two words.

5. Output predicted words.

Implementation:

```
import nltk
from nltk import word_tokenize
from nltk.util import ngrams
from collections import Counter
nltk.download('punkt')
corpus="""Natural language processing allows computers to understand human
language.
Language models are essential for predicting the next word in a sentence.
Unigram, Bigram, and Trigram models provide different levels of context
for word prediction."""
tokens=word_tokenize(corpus.lower())
unigram_counts=Counter(tokens)
total_unigrams=sum(unigram_counts.values())
unigram_probs={word:count/total_unigrams for word,count in
unigram_counts.items()}
print("Top Unigrams:",unigram_probs)
bigrams=list(ngrams(tokens,2))
bigram_counts=Counter(bigrams)
bigram_probs={bg:count/unigram_counts[bg[0]] for bg,count in
bigram_counts.items()}
print("\nTop Bigrams:",bigram_probs)
trigrams=list(ngrams(tokens,3))
trigram_counts=Counter(trigrams)
trigram_probs={tg:count/bigram_counts[tg[:2]] for tg,count in
trigram_counts.items()}
print("\nTop Trigrams:",trigram_probs)
def predict_next_word(prev_words,ngram_probs,n=2):
            candidates={k[-1]:v  for  k,v  in  ngram_probs.items()  if
k[:-1]==tuple(prev_words[-(n-1):])}
    return max(candidates,key=candidates.get) if candidates else None
print("\nBigram                    Prediction                    for
```

```
'natural':",predict_next_word(['natural'],bigram_probs,n=2))
print("Trigram Prediction for 'natural
language':",predict_next_word(['natural','language'],trigram_probs,n=3))
```

Output:

```
PS C:\Users\Konisha Thakare\OneDrive\Desktop\Python>  & 'c:\Users\Konisha Thakare\AppData\Local\Programs\Python\Python312\python.exe' 'c:\Users\Konisha Thakare\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '51816' '--' 'c:\Users\Konisha Thakare\OneDrive\Desktop\Python\ngram_word_prediction.py'
Top Unigrams: {'natural': 0.02564102564102564, 'language': 0.07692307692307693, 'processing': 0.02564102564102564, 'allows': 0.02564102564102564, 'computers': 0.02564102564102564, 'to': 0.02564102564102564, 'understand': 0.02564102564102564, 'human': 0.02564102564102564, '.': 0.07692307692307693, 'models': 0.05128205128205128, 'are': 0.02564102564102564, 'essential': 0.02564102564102564, 'for': 0.05128205128205128, 'predicting': 0.02564102564102564, 'the': 0.02564102564102564, 'next': 0.02564102564102564, 'word': 0.05128205128205128, 'in': 0.02564102564102564, 'a': 0.02564102564d', 'in'): 0.5, ('in', 'a'): 1.0, ('a', 'sentence'): 1.0, ('sentence', '.'): 1.0, ('.', 'unigram'): 0.3333333333333333, ('unigram', ','): 1.0, (',', 'bigram'): 0.5, ('bigram', ','): 1.0, (',', 'and'): 0.5, ('and', 'trigram'): 1.0, ('trigram', 'models'): 1.0, ('moded', 'in'): 0.5, ('in', 'a'): 1.0, ('a', 'sentence'): 1.0, ('sentence', '.'): 1.0, ('.', 'unigram'): 0.3333333333333333, ('unigram', ','): 1.0, (',', 'bigram')d', 'in'): 0.5, ('in', 'a'): 1.0, ('a', 'sentence'): 1.0, ('sentence', '.'): 1.0, ('.', 'unigram'): 0.3333333333333333, ('unigram', ','): 1.0, (',', 'bigram'): 0d', 'in'): 0.5, ('in', 'a'): 1.0, ('a', 'sentence'): 1.0, ('sentence', '.'): 1.0, ('.', 'unigram'): 0.3333333333333333, ('unigram', ','): 1.0, (',', 'bigram'): d', 'in'): 0.5, ('in', 'a'): 1.0, ('a', 'sentence'): 1.0, ('sentence', '.'): 1.0, ('.', 'unigram'): 0.3333333333333333, ('unigram', ','): 1.0, (',', 'bigram'): 0.5, ('bigram', ','): 1.0, (',', 'and'): 0.5, ('and', 'trigram'): 1.0, ('trigram', 'models'): 1.0, ('models', 'provide'): 0.5, ('provide', 'different'): 1.0, ('different', 'levels'): 1.0, ('levels', 'of'): 1.0, ('of', 'context'): 1.0, ('context', 'for'): 1.0, ('for', 'word'): 0.5, ('word', 'prediction'): 0.5, ('prediction', '.'): 1.0}

Top Trigrams: {('natural', 'language', 'processing'): 1.0, ('language', 'processing', 'allows'): 1.0, ('processing', 'allows', 'computers'): 1.0, ('allows', 'computers', 'to'): 1.0, ('computers', 'to', 'understand'): 1.0, ('to', 'understand', 'human'): 1.0, ('understand', 'human', 'language'): 1.0, ('human', 'language', '.'): 1.0, ('language', '.', 'language'): 1.0, ('.', 'language', 'models'): 1.0, ('language', 'models', 'are'): 1.0, ('models', 'are', 'essential'): 1.0, ('are', 'essential', 'for'): 1.0, ('essential', 'for', 'predicting'): 1.0, ('for', 'predicting', 'the'): 1.0, ('predicting', 'the', 'next'): 1.0, ('the', 'next', 'word'): 1.0, ('next', 'word', 'in'): 1.0, ('word', 'in', 'a'): 1.0, ('in', 'a', 'sentence'): 1.0, ('a', 'sentence', '.'): 1.0, ('sentence', '.', 'unigram'): 1.0, ('.', 'unigram', ','): 1.0, ('unigram', ',', 'bigram'): 1.0, (',', 'bigram', ','): 1.0, ('bigram', ',', 'and'): 1.0, (',', 'and', 'trigram'): 1.0, ('and', 'trigram', 'models'): 1.0, ('trigram', 'models', 'provide'): 1.0, ('models', 'provide', 'different'): 1.0, ('provide', 'different', 'levels'): 1.0, ('different', 'levels', 'of'): 1.0, ('levels', 'of', 'context'): 1.0, ('of', 'context', 'for'): 1.0, ('context', 'for', 'word'): 1.0, ('for', 'word', 'prediction'): 1.0, ('word', 'prediction', '.'): 1.0}

Bigram Prediction for 'natural': language
Trigram Prediction for 'natural language': processing
PS C:\Users\Konisha Thakare\OneDrive\Desktop\Python>
```

Conclusion:

This experiment demonstrated the construction and application of unigram, bigram, and trigram language models for word prediction. While unigram models treat each word independently and provide limited context, bigram and trigram models incorporate one and two previous words respectively, allowing for more accurate predictions. The results illustrate that higher-order n-grams capture more contextual information, improving the reliability of predicted words. Such models form the foundation of applications like predictive keyboards, autocomplete features, and speech recognition, though they can become computationally expensive with large corpora due to the combinatorial increase in n-grams.