



Experiment No.5
Implement N-Gram model for the given text input.
Date of Performance:
Date of Submission:



Aim: Implement N-Gram model for the given text input.

Objective: To study and implement N-gram Language Model.

Theory:

A language model supports predicting the completion of a sentence.

Eg:

- Please turn off your cell _____
- Your program does not _____

Predictive text input systems can guess what you are typing and give choices on how to complete it.

N-gram Models:

Estimate probability of each word given prior context.

$P(\text{phone} \mid \text{Please turn off your cell})$

- Number of parameters required grows exponentially with the number of words of prior context.
- An N-gram model uses only N1 words of prior context.
 - Unigram: $P(\text{phone})$
 - Bigram: $P(\text{phone} \mid \text{cell})$
 - Trigram: $P(\text{phone} \mid \text{your cell})$
- The Markov assumption is the presumption that the future behavior of a dynamical system only depends on its recent history. In particular, in a kth-order Markov model, the next state only depends on the k most recent states, therefore an N-gram model is a (N1)-order Markov model.

N-grams: a contiguous sequence of n tokens from a given piece of text

Fig. Example of Trigrams in a sentence

Mary was scared because of the terrifying noise. ...

Code:

```
import nltk
import re
import pprint
import string
from nltk import word_tokenize, sent_tokenize
from nltk.util import ngrams
from nltk.corpus import stopwords
# Include additional punctuation marks for processing
```



```
string.punctuation += "'\"_-'_'"
string.punctuation = string.punctuation.replace('.', '')
# Load and preprocess the data
file = open('./dataset.txt', encoding='utf8').read()
file_nl_removed = " ".join(file.splitlines()) # Remove newlines and join lines
file_p = "".join([char for char in file_nl_removed if char not in string.punctuation])
### Statistics of the Data
sents = nltk.sent_tokenize(file_p)
print("The number of sentences is", len(sents))
words = nltk.word_tokenize(file_p)
print("The number of tokens is", len(words))
average_tokens = round(len(words) / len(sents))
print("The average number of tokens per sentence is", average_tokens)
unique_tokens = set(words)
print("The number of unique tokens are", len(unique_tokens))
### Building the N-Gram Model
stop_words = set(stopwords.words('english'))
unigram = []
bigram = []
trigram = []
fourgram = []
tokenized_text = []
# Process each sentence for n-grams
for sentence in sents:
    sequence = word_tokenize(sentence.lower())
    unigram.extend(word for word in sequence if word not in ('.',)) # Skip periods
    tokenized_text.append(sequence)
    bigram.extend(list(ngrams(sequence, 2)))
    trigram.extend(list(ngrams(sequence, 3)))
    fourgram.extend(list(ngrams(sequence, 4)))
# Function to remove n-grams containing only stopwords
def removal(x):
    return [pair for pair in x if any(word not in stop_words for word in pair)]
# Remove stopwords from n-grams
bigram = removal(bigram)
trigram = removal(trigram)
fourgram = removal(fourgram)
# Frequency distribution of n-grams
freq_bi = nltk.FreqDist(bigram)
freq_tri = nltk.FreqDist(trigram)
freq_four = nltk.FreqDist(fourgram)
print("Most common n-grams without stopword removal and without add-1 smoothing: \n")
print("Most common bigrams: ", freq_bi.most_common(5))
print("\nMost common trigrams: ", freq_tri.most_common(5))
print("\nMost common fourgrams: ", freq_four.most_common(5))
### Print 10 Unigrams and Bigrams after removing stopwords
unigram_sw_removed = [p for p in unigram if p not in stop_words]
fdist = nltk.FreqDist(unigram_sw_removed)
print("Most common unigrams: ", fdist.most_common(10))
```



```
bigram_sw_removed = list(ngrams(unigram_sw_removed, 2))
fdist = nltk.FreqDist(bigram_sw_removed)
print("\nMost common bigrams: ", fdist.most_common(10))
### Add-1 smoothing
ngrams_all = {1: [], 2: [], 3: [], 4: []}
ngrams_voc = {1: set(), 2: set(), 3: set(), 4: set()}
for i in range(4):
    for each in tokenized_text:
        ngrams_all[i + 1].extend(ngrams(each, i + 1))
for i in range(4):
    for gram in ngrams_all[i + 1]:
        ngrams_voc[i + 1].add(gram)
total_ngrams = {i + 1: len(ngrams_all[i + 1]) for i in range(4)}
total_voc = {i + 1: len(ngrams_voc[i + 1]) for i in range(4)}
ngrams_prob = {1: [], 2: [], 3: [], 4: []}
for i in range(4):
    for ngram in ngrams_voc[i + 1]:
        count = ngrams_all[i + 1].count(ngram)
        prob = (count + 1) / (total_ngrams[i + 1] + total_voc[i + 1])
        ngrams_prob[i + 1].append((ngram, prob))
### Sort probabilities
for i in range(4):
    ngrams_prob[i + 1].sort(key=lambda x: x[1], reverse=True)
### Prints top 10 unigram, bigram, trigram, fourgram after smoothing
print("Most common n-grams without stopword removal and with add-1 smoothing: \n")
print("Most common unigrams: ", str(ngrams_prob[1][:10]))
print("\nMost common bigrams: ", str(ngrams_prob[2][:10]))
print("\nMost common trigrams: ", str(ngrams_prob[3][:10]))
print("\nMost common fourgrams: ", str(ngrams_prob[4][:10]))
### Next word Prediction
def next_word_prediction(ngram_input, ngrams_prob):
    predictions = []
    for i in range(3):
        count = 0
        for each in ngrams_prob[i + 2]:
            if each[0][-1] == ngram_input:
                predictions.append(each[0][-1])
                count += 1
            if count == 5:
                break
        while count < 5:
            predictions.append("NOT FOUND")
            count += 1
    return predictions
str1 = 'after that alice said the'
str2 = 'alice felt so desperate that she was'
token_1 = word_tokenize(str1.lower())
token_2 = word_tokenize(str2.lower())
ngram_1 = {i + 1: list(ngrams(token_1, i + 1))[-1] for i in range(3)}
```



```
ngram_2 = {i + 1: list(ngrams(token_2, i + 1))[-1] for i in range(3)}  
print("Next word predictions for the strings using the probability models of bigrams, trigrams,  
and fourgrams\n")  
print("String 1 - after that alice said the-\n")  
pred_1 = next_word_prediction(ngram_1[1], ngrams_prob)  
print("Bigram model predictions: {} \n Trigram model predictions: {} \n Fourgram model  
predictions: {} \n".format(pred_1[0], pred_1[1], pred_1[2]))  
print("String 2 - alice felt so desperate that she was-\n")  
pred_2 = next_word_prediction(ngram_2[1], ngrams_prob)  
print("Bigram model predictions: {} \n Trigram model predictions: {} \n Fourgram model  
predictions: {}".format(pred_2[0], pred_2[1], pred_2[2]))
```

Output:

```
(venv) PS D:\Vartak college\sem 7\NLP\EXP\New folder> python .\exp5.py
```

The number of sentences is 981

The number of tokens is 27361

The average number of tokens per sentence is 28

The number of unique tokens are 3039

Most common n-grams without stopword removal and without add-1 smoothing:

Most common bigrams: [((('said', 'the'), 209), (('said', 'alice'), 115), (('the', 'queen'), 65), (('the', 'king'), 60), (('a', 'little'), 59)]

Most common trigrams: [((('the', 'mock', 'turtle'), 51), (('said', 'alice', '.'), 33), (('the', 'march', 'hare'), 30), (('said', 'the', 'king'), 29), (('the', 'white', 'rabbit'), 21)]

Most common fourgrams: [((('said', 'the', 'mock', 'turtle'), 19), (('she', 'said', 'to', 'herself'), 16), (('said', 'the', 'caterpillar', '.'), 12), (('a', 'minute', 'or', 'two'), 11), (('the', 'march', 'hare', '.'), 10)]

Most common unigrams: [('said', 462), ('alice', 385), ('little', 128), ('one', 101), ('like', 85), ('know', 85), ('would', 83), ('went', 83), ('could', 77), ('thought', 74)]

Most common bigrams: [((('said', 'alice'), 122), (('mock', 'turtle'), 54), (('march', 'hare'), 31), (('said', 'king'), 29), (('thought', 'alice'), 26), (('white', 'rabbit'), 22), (('said', 'hatter'), 22), (('said', 'mock'), 20), (('said', 'caterpillar'), 18), (('said', 'gryphon'), 18)]

Most common n-grams without stopword removal and with add-1 smoothing:

Most common unigrams: [((('the',), 0.05416085541608554), (('.',), 0.03257621040047818), (('and',), 0.028060038520289567), (('to',), 0.023975559540413097), (('a',), 0.020854087799694495), (('she',), 0.01786544464368732), (('it',), 0.017500166035730888), (('of',), 0.016902437404529454), (('said',), 0.01537490868034801), (('i',), 0.013316065617320847)]

Most common bigrams: [((('said', 'the'), 0.00514718498002402), (('of', 'the'), 0.0032108630113483172), (('said', 'alice'), 0.002843206941346602), (('in', 'a'), 0.0024020196573445425), (('and', 'the'), 0.001985342778009265), (('in', 'the'), 0.0019363219686757028), (('it', 'was'), 0.0018382803500085786), (('to', 'the'), 0.0017157283266746733), (('the', 'queen'), 0.0016176867080075492), (('as', 'she'), 0.001519645089340425)]

Most common trigrams: [((('the', 'mock', 'turtle'), 0.0011025358324145535), (('said', 'alice', 'the'), 0.001985342778009265), (('in', 'the'), 0.0019363219686757028), (('it', 'was'), 0.0018382803500085786), (('to', 'the'), 0.0017157283266746733), (('the', 'queen'), 0.0016176867080075492), (('as', 'she'), 0.001519645089340425)]



Most common trigrams: [(('the', 'mock', 'turtle'), 0.0011025358324145535), (('said', 'alice', '5492'), (('as', 'she'), 0.001519645089340425)]

Most common trigrams: [(('the', 'mock', 'turtle'), 0.0011025358324145535), (('said', 'alice', '.'), 0.0007208888135018235), (('the', 'march', 'hare'), 0.0006572809770163684), (('said', 'the', 'king'), 0.0006360783648545501), (('said', 'the', 'hatter'), 0.0004664574675600034), (('the', 'white', 'rabbit'), 0.0004664574675600034), (('said', 'to', 'herself'), 0.0004240522432363667

Most common trigrams: [(('the', 'mock', 'turtle'), 0.0011025358324145535), (('said', 'alice', '.'), 0.0007208888135018235), (('the', 'march', 'hare'), 0.0006572809770163684), (('said', 'the', 'king'), 0.0006360783648545501), (('said', 'the', 'hatter'), 0.0004664574675600034), (('the', 'white', 'rabbit'), 0.0004664574675600034), (('said', 'to', 'herself'), 0.0004240522432363667), (('said', 'the', 'mock'), 0.0004240522432363667), (('said', 'the', 'caterpillar'), 0.0004028496310745484), (('said', 'the', 'gryphon'), 0.00038164701891273004)]

), (('said', 'the', 'mock'), 0.0004240522432363667), (('said', 'the', 'caterpillar'), 0.0004028496310745484), (('said', 'the', 'gryphon'), 0.00038164701891273004)]
, (('said', 'the', 'mock'), 0.0004240522432363667), (('said', 'the', 'caterpillar'), 0.0004028496310745484), (('said', 'the', 'gryphon'), 0.00038164701891273004)]
496310745484), (('said', 'the', 'gryphon'), 0.00038164701891273004)]

Most common fourgrams: [(('said', 'the', 'mock', 'turtle'), 0.00041860270417346895), (('she', 'said', 'to', 'herself'), 0.0003558122985474486), (('said', 'the', 'caterpillar', '.'), 0.0002720917577127548), (('a', 'minute', 'or', 'two'), 0.0002511616225040814), (('said', 'the', 'king', '.'), 0.00023023148729540792), (('the', 'march', 'hare', '.'), 0.00023023148729540792), (('said', 'the', 'hatter', '.'), 0.00020930135208673448), (('will', 'you', 'wont', 'you'), 0.00018837121687806103), (('said', 'the', 'march', 'hare'), 0.00018837121687806103), (('the', 'mock', 'turtle', '.'), 0.00018837121687806103)]

Next word predictions for the strings using the probability models of bigrams, trigrams, and fourgrams

String 1 - after that alice said the-

Bigram model predictions: queen

Trigram model predictions: king

Fourgram model predictions: mock

String 2 - alice felt so desperate that she was-

Bigram model predictions: a

Trigram model predictions: the

Fourgram model predictions: not

Conclusion:

The N-gram language model was implemented for text analysis, generating unigrams, bigrams, trigrams, and fourgrams from a dataset. Key statistics included 981 sentences, 27,361 tokens, and 3,039 unique tokens. Most common n-grams were identified, with bigrams like ('said', 'the') and trigrams like ('the', 'mock', 'turtle') being the most frequent. Add-1 smoothing was applied to enhance the probability distribution. Next-word predictions were made for two input strings, yielding various bigram and trigram predictions. The model effectively captures word patterns but showed limitations in fourgram predictions, indicating a need for more data or refinement for improved accuracy.