



Vidyavardhini's College of Engineering & Technology

Department of Computer Science and Engineering (Data Science)

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



Aim: Implement Bi-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 $N-1$ 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 k th-order Markov model, the next state only depends on the k most recent states, therefore an N-gram model is a $(N-1)$ -order Markov model.

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



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

Fig. Example of Trigrams in a sentence

Implementation:

```
Exp-4.ipynb
File Edit View Insert Runtime Tools Help Last saved at 8:01 PM
+ Code + Text
Experiment 04 ~ NLP DLOG ~ Sahil Gujral ~ Roll No. 15 ~ Dept. of CSE (DS) ~ VCET

Necessary Imports
[] import nltk, re, pprint, string
from nltk import word_tokenize, sent_tokenize
string.punctuation = string.punctuation + "'\">
string.punctuation = string.punctuation.replace('.', '')
file = open('dataset.txt', encoding = 'utf8').read()

Preprocess of the Data
[] file_nl_removed = ""
for line in file:
    line_nl_removed = line.replace("\n", " ")
    file_nl_removed += line_nl_removed
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))
```

```
Exp-4.ipynb
File Edit View Insert Runtime Tools Help Last saved at 8:01 PM
+ Code + Text
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))

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

Building the N-Gram Model
[] from nltk.util import ngrams
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))

[] unigram=[]
bigram=[]
trigram=[]
fourgram=[]
tokenized_text = []

[] for sentence in sents:
    sentence = sentence.lower()
```



Vidyavardhini's College of Engineering & Technology

Department of Computer Science and Engineering (Data Science)

```
Exp-4.ipynb
File Edit View Insert Runtime Tools Help Last saved at 8:01 PM
+ Code + Text
for word in sequence:
    if (word != '.'):
        sequence.remove(word)
    else:
        unigram.append(word)
        tokenized_text.append(sequence)
        bigram.extend(list(ngrams(sequence, 2)))
        trigram.extend(list(ngrams(sequence, 3)))
        fourgram.extend(list(ngrams(sequence, 4)))

#removes ngrams containing only stopwords
def removal(x):
    y = []
    for pair in x:
        count = 0
        for word in pair:
            if word in stop_words:
                count = count + 1
            else:
                count = count + 1
        if (count==1):
            y.append(pair)
    return(y)

bigram = removal(bigram)
trigram = removal(trigram)
fourgram = removal(fourgram)
freq_bi = nltk.FreqDist(bigram)
freq_tri = nltk.FreqDist(trigram)
freq_four = nltk.FreqDist(fourgram)
```

```
Exp-4.ipynb
File Edit View Insert Runtime Tools Help Last saved at 8:01 PM
+ Code + Text
[ ] Most common trigrams: [('the', 'mock', 'turtle'), 51], [('the', 'march', 'hare'), 30], [('said', 'the', 'king'), 29], [('the', 'white', 'rabbit'), 21], [('said', 'the', 'hatter'), 21]
Most common fourgrams: [('said', 'the', 'mock', 'turtle'), 19], [('she', 'said', 'to', 'herself'), 16], [('a', 'minute', 'on', 'two'), 11], [('said', 'the', 'march', 'hare'), 8], [('

Script for downloading the stopwords using NLTK

from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))

Print 10 Unigrams and Bigrams after removing stopwords

[ ] print("Most common n-grams with stopword removal and without add-1 smoothing:\n")
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 = []
bigram_sw_removed.extend(list(ngrams(unigram_sw_removed, 2)))
fdist = nltk.FreqDist(bigram_sw_removed)
print("\nMost common bigrams: ", fdist.most_common(10))

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

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'), 94], [('march', 'hare'), 31], [('said', 'king'), 29], [('thought', 'alice'), 26], [('white', 'rabbit'), 22], [('
```

```
Exp-4.ipynb
File Edit View Insert Runtime Tools Help Last saved at 8:01 PM
+ Code + Text
Add-1 smoothing

[ ] ngrams_all = {}
for i in range(4):
    for each in tokenized_text:
        for j in ngrams(each, i+1):
            ngrams_all[i+1].append(j)
ngrams_voc = {}
for i in range(4):
    for gram in ngrams_all[i+1]:
        if gram not in ngrams_voc[i+1]:
            ngrams_voc[i+1].add(gram)
total_ngrams = {}
total_voc = {}
for i in range(4):
    total_ngrams[i+1] = len(ngrams_all[i+1])
    total_voc[i+1] = len(ngrams_voc[i+1])
ngrams_prob = {}
for i in range(4):
    for gram in ngrams_voc[i+1]:
        tlist = [ngram]
        tlist.append(ngrams_all[i+1].count(ngram))
        ngrams_prob[i+1].append(tlist)
for i in range(4):
    for ngram in ngrams_prob[i+1]:
        ngram[-1] = (ngram[-1]+1)/(total_ngrams[i+1]+total_voc[i+1])
```



Vidyavardhini's College of Engineering & Technology

Department of Computer Science and Engineering (Data Science)

```
Exp-4.ipynb
File Edit View Insert Runtime Tools Help Last saved at 8:01 PM
+ Code + Text
Prints top 10 unigram, bigram, trigram, fourgram after smoothing

[ ] print("Most common n-grams without stopword removal and with add-1 smoothing: \n")
for i in range(4):
    ngrams_prob[i+1] = sorted(ngrams_prob[i+1], key = lambda x:x[1], reverse = True)

print ("Most common unigrams: ", str(ngrams_prob[1][:10]))
print ("Most common bigrams: ", str(ngrams_prob[2][:10]))
print ("Most common trigrams: ", str(ngrams_prob[3][:10]))
print ("Most common fourgrams: ", str(ngrams_prob[4][:10]))

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

Most common unigrams: [['the',), 0.05598462224968249], [['and',), 0.02900490852298081], [['to',), 0.02478289225277177], [['a',), 0.02155631071293722], [['she',), 0.018467030515223287]
Most common bigrams: [['said', 'the',), 0.0053395713087035016], [['of', 'the',), 0.0033308754354293268], [['said', 'alice',), 0.0029494774848076483], [['in', 'a',), 0.002491799944061634]
Most common trigrams: [['the', 'mock', 'turtle',), 0.00114383757064341], [['the', 'march', 'hare',), 0.0006819031697498955], [['said', 'the', 'king',), 0.0006599062933063505], [['the', 'said', 'to', 'herself',), 0.0003699351525438482], [['a', 'minute', 'or', 'two',), 0.000261]

Next word Prediction

[ ] str1 = 'after that alice said the'
str2 = 'alice felt so desperate that she was'
```

```
Exp-4.ipynb
File Edit View Insert Runtime Tools Help Last saved at 8:01 PM
+ Code + Text

[ ] token_1 = word_tokenize(str1)
token_2 = word_tokenize(str2)
ngram_1 = {1:[], 2:[], 3:[]} #to store the n-grams formed
ngram_2 = {1:[], 2:[], 3:[]}
for i in range(3):
    ngram_1[i+1] = list(ngrams(token_1, i+1))[-1]
    ngram_2[i+1] = list(ngrams(token_2, i+1))[-1]
print("String 1: ", ngram_1, "\nString 2: ", ngram_2)

String 1: {1: ('the',), 2: ('said', 'the',), 3: ('alice', 'said', 'the',)}
String 2: {1: ('was',), 2: ('she', 'was',), 3: ('that', 'she', 'was',)}

[ ] for i in range(4):
    ngrams_prob[i+1] = sorted(ngrams_prob[i+1], key = lambda x:x[1], reverse = True)

pred_1 = {1:[], 2:[], 3:[]}
for i in range(3):
    count = 0
    for each in ngrams_prob[i+2]:
        if each[0][:-1] == ngram_1[i+1]:
            #to find predictions based on highest probability of n-grams

            count +=1
            pred_1[i+1].append(each[0][:-1])
            if count ==5:
                break
    if count<5:
        while(count!=5):
            pred_1[i+1].append("NOT FOUND")
            #if no word prediction is found, replace with NOT FOUND
```



Vidyavardhini's College of Engineering & Technology

Department of Computer Science and Engineering (Data Science)

```
Exp-4.ipynb
File Edit View Insert Runtime Tools Help Last saved at 8:01 PM
+ Code + Text
[ ] #if no word prediction is found, replace with NOT FOUND
    count +=1
    for i in range(4):
        ngrams_prob[i+1] = sorted(ngrams_prob[i+1], key = lambda x:x[1], reverse = True)

    pred_2 = {1:[], 2:[], 3:[]}
    for i in range(3):
        count = 0
        for each in ngrams_prob[i+2]:
            if each[0][:-1] == ngram_2[i+1]:
                count +=1
                pred_2[i+1].append(each[0][:-1])
            if count ==5:
                break

        if count<5:
            while(count!=5):
                pred_2[i+1].append("")
                count +=1

    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")
    print("Bigram model predictions: {}\\nTrigram model predictions: {}\\nFourgram model predictions: {}".format(pred_1[1], pred_1[2], pred_1[3]))
    print("String 2 - alice felt so desperate that she was-\n")
    print("Bigram model predictions: {}\\nTrigram model predictions: {}\\nFourgram model predictions: {}".format(pred_2[1], pred_2[2], pred_2[3]))

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', 'king', 'gryphon', 'mock', 'hatter']
```

```
Exp-4.ipynb
File Edit View Insert Runtime Tools Help Last saved at 8:01 PM
+ Code + Text
[ ] 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")
    print("Bigram model predictions: {}\\nTrigram model predictions: {}\\nFourgram model predictions: {}".format(pred_1[1], pred_1[2], pred_1[3]))
    print("String 2 - alice felt so desperate that she was-\n")
    print("Bigram model predictions: {}\\nTrigram model predictions: {}\\nFourgram model predictions: {}".format(pred_2[1], pred_2[2], pred_2[3]))

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', 'king', 'gryphon', 'mock', 'hatter']
Trigram model predictions: ['king', 'hatter', 'mock', 'caterpillar', 'gryphon']
Fourgram model predictions: ['NOT FOUND', 'NOT FOUND', 'NOT FOUND', 'NOT FOUND', 'NOT FOUND']

String 2 - alice felt so desperate that she was-
Bigram model predictions: ['a', 'the', 'not', 'that', 'going']
Trigram model predictions: ['now', 'quite', 'a', 'beginning', 'walking']
Fourgram model predictions: ['now', 'ready', 'quite', 'dozing', 'in']
```

Conclusion: