



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.

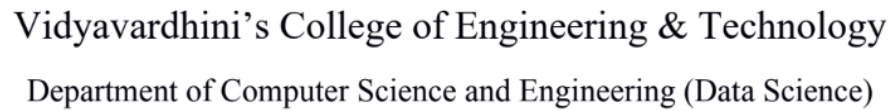
N-gram Models:

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



...

Output:

CSDL7013: Natural Language Processing Lab



Vidyavardhini's College of Engineering & Technology

Department of Computer Science and Engineering (Data Science)

```
29_Dinesh Madhav_Exp04_NLP.ipynb
File Edit View Insert Runtime Tools Help Last edited on October 11
Code + Text
print(string 1: ", ngram_1: ", ngram_2: ", ngram_3: ")
string 1: (1: 'the', 2: 'said', 'the'), 3: ('alice', 'said', 'the')
string 2: (1: 'said', 2: 'the', 'was'), 3: ('that', 'the', 'was')

for i in range(1):
    ngram_prob[i+1] = sorted(ngram_prob[i+1], key = lambda x:x[1], reverse = True)
    pred_1 = (1:1), 2:1, 3:1
    for i in range(1):
        count = 0
        for each in ngram_prob[i+1]:
            if each[0][1] == ngram_1[i]:
                #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")
                    count -=1
                #if no word prediction is found, replace with NOT FOUND
                count +=1
        for i in range(1):
            ngram_prob[i+1] = sorted(ngram_prob[i+1], key = lambda x:x[1], reverse = True)
    pred_2 = (1:1), 2:1, 3:1
    for i in range(2):
        count = 0
        for each in ngram_prob[i+1]:
            if each[0][1] == ngram_2[i]:
                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("UP")
                    count -=1
    count +=1

for i in range(2):
    print("next word predictions for the strings using the probability models of bigrams, trigrams, and fourgrams is")
    print(string 1 - after that alice said the: ")
    print("bigram model predictions: {}".format(pred_1[i], pred_1[i], pred_1[i]))
    print("string 2 - Alice felt so desperate that she was: ")
    print("bigram model predictions: {}".format(pred_2[i], pred_2[i], pred_2[i]))

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: ['floor', 'lightend', 'sides', 'not found', 'not found']
trigram model predictions: ['not found', 'not found', 'not found', 'not found', 'not found']
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: ['not found', 'not found', 'not found', 'not found', 'not found']
```

Conclusion:

N-gram language models are statistical models that predict the next word in a sequence based on the previous N-1 words. They are often used in NLP tasks such as speech recognition, machine translation, and text generation.

The results of N-gram language models depend on the size and quality of the training corpus, the order of the N-gram model, and the smoothing algorithm used.

In general, N-gram language models are effective in a variety of NLP tasks, but they can be computationally expensive to train and use, and they may not perform well on data that is different from the training corpus.