



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

Output:

```
29_Dinesh Madhav_Expt 05_NLP.ipynb
File Edit View Insert Runtime Tools Help Last edited on October 11
+ Code + Text
[ ] text = "TON 618 (short for Tonantzintla 618) is a hyperluminous, broad-absorption-line, radio-loud quasar and Lyman-alpha blob located near the border of the constellations Canes Venatici and ..."

Importing necessary dependencies
[ ] import nltk
    from nltk.tokenize import word_tokenize

Word Tokenization
[ ] import nltk
    nltk.download('punkt')
    words = word_tokenize(text)

[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Unzipping tokenizers/punkt.zip.

Parts of Speech Tagging
[ ] import nltk
    nltk.download('universal_tagset')
    nltk.download('averaged_perceptron_tagger')
    tagged_words = nltk.pos_tag(words, tagset = 'universal')

[nltk_data] Downloading package universal_tagset to /root/nltk_data...
[nltk_data] Package universal_tagset is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to /root/nltk_data...

tagged_words
[('TON', '.'),
 ('618', 'NUM'),
 ('(', '.'),
 ('short', 'ADJ'),
 ('for', 'ADP'),
 ('Tonantzintla', 'NOUN'),
 ('618', 'NUM'),
 ('(', '.'),
 ('is', 'VERB'),
 ('a', 'DET'),
 ('hyperluminous', 'ADJ'),
 ('(', '.'),
 ('broad-absorption-line', 'ADJ'),
 ('(', '.'),
 ('radio-loud', 'ADJ'),
 ('quasar', 'NOUN'),
 ('and', 'CONJ'),
 ('Lyman-alpha', 'NOUN'),
 ('blob', 'NOUN'),
 ('located', 'VERB'),
 ('near', 'ADP'),
 ('the', 'DET'),
 ('border', 'NOUN'),
 ('of', 'ADP'),
 ('the', 'DET'),
 ('constellations', 'NOUN'),
 ('Canes', 'NOUN'),
 ('Venatici', 'NOUN'),
 ('and', 'CONJ'),
 ('Coma', 'NOUN'),
 ('Berenicis', 'NOUN'),
 ('(', '.'),
 ('...', '.')]

```



```
for t in tagged_words:
    print(t)

('TON', '.')
('618', 'NUM')
(',', '.')
('short', 'ADJ')
('for', 'ADP')
('Tonantzintla', 'NOUN')
('618', 'NUM')
(',', '.')
('is', 'VERB')
('a', 'DET')
('hyperluminous', 'ADJ')
(',', '.')
('broad-absorption-line', 'ADJ')
(',', '.')
('radio-loud', 'ADJ')
('quasar', 'NOUN')
('and', 'CONJ')
('Lyman-alpha', 'NOUN')
('blob', 'NOUN')
('located', 'VERB')
('near', 'ADP')
('the', 'DET')
('border', 'NOUN')
('of', 'ADP')
('the', 'DET')
('constellations', 'NOUN')
('Canes', 'NOUN')
('Venatici', 'NOUN')
('and', 'CONJ')
('Coma', 'NOUN')
('Berenices', 'NOUN')
```

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.