**Lab-3**

[**https://www.analyticsvidhya.com/blog/2022/01/building-language-models-in-nlp/#:~:text=A%20language%20model%20in%20NLP,appear%20next%20in%20the%20sentence**](https://www.analyticsvidhya.com/blog/2022/01/building-language-models-in-nlp/#:~:text=A%20language%20model%20in%20NLP,appear%20next%20in%20the%20sentence)**.**

**Building Language Models in NLP**

* A language model in NLP is a probabilistic statistical model
* Determines the probability of a given sequence of words occurring in a sentence based on the previous words
* It predicts which word is more likely to appear next in the sentence.
* Widely used in predictive text input systems, speech recognition, machine translation, spelling correction.
* The input to a language model is usually a training set of example sentences.
* The output is a probability distribution over sequences of words.
* can use the last one word (unigram), last two words (bigram), last three words (trigram) or last n words (n-gram) to predict the next word

**Steps**

* **Reading the Raw Text Corpus**
  + file = open("rawCorpus.txt", "r")
  + rawReadCorpus = file.read()
  + import the nltk library to perform some basic text processing tasks
* **Preprocessing the Raw Text**
* remove all new lines and special characters from the text corpus.
  + break up the corpus to obtain the words and the sentences using sent\_tokenize and word\_tokenize from nltk.tokenize.
  + remove stopwords from the corpus.
* **Creating Unigram, Bigram and Trigram Language Models**
  + We can create n-grams using the ngrams module from nltk.util.
  + obtain unigrams, bigrams and trigrams from the corpus

from collections import Counter

from nltk.util import ngrams

unigrams=[]

for content in (sentences): # \*\*\* Write code \*\*\*

content = content.lower()

content = word\_tokenize(content)

for word in content:

if (word =='.'):

content.remove(word)

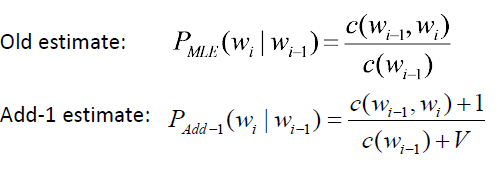
else:

unigrams.append(word)

print ("Sample of n-grams:n" + "-------------------------")

print ("--> UNIGRAMS: n" + str(unigrams[:5]) + " ...n")

* + obtain those unigrams, bigrams and trigrams from the corpus which do not have stopwords like articles, prepositions or determiners
  + obtain the count or frequency of each n-gram appearing in the corpus
* **Predicting Next Three words using Bigram and Trigram Models**
  + The chain rule is used to compute the probability of a sentence in a language model.
  + the probability of the sentence “I love dogs” is given by :
* P(I love dogs) = P(I)P(love | I)P(dogs | I love)
* Now the individual probabilities can be obtained in the following way :
* P(I) = Count(‘I’) / Total no. of words
* P(love | I) = Count(‘I love’) / Count(‘I’)
* P(dogs | I love) = Count(‘I love dogs’) / Count(‘I love’)
  + bigram model to compute the probabilities.
* Suppose there is some bigram that does not appear in the training set but appears in the test set.
* Then we will assign a probability of 0 to that bigram, making the overall probability of the test sentence 0, which is undesirable.
* Add-one smoothing is performed by adding 1 to all bigram counts and V (no. of unique words in the corpus) to all unigram counts.



* write the code to compute smoothed bigram and trigram models
* try to predict the next three words of three test sentences using the computed smoothed bigram and trigram language models.

testSent1 = "There was a sudden jerk, a terrific convulsion of the limbs; and there he"

testSent2 = "They made room for the stranger, but he sat down"

testSent3 = "The hungry and destitute situation of the infant orphan was duly reported by"

* tokenize the test sentences into component words and obtain the last unigrams and bigrams.
* we write functions to predict the next word and the next 3 words respectively of the three test sentences using the smoothed bigram model.