**Introduction**

**N gram models**

N-gram is a function that is defined in ‘NLTK’ library. This model is mainly used to develop application in NLP. Some of the applications of NLP are text input prediction, voice recognition. The N-gram model can be used as nltk.ngrams(tokens,n). Where tokens are the taken from a text that is provided and tokenized, n is to determine the value of which kind of N gram models to generate. For example if we want to generate bigrams we can provide n as 2, for trigrams n as 3 and so on.

We can also create bigrams by passing the tokens to nltk.bigrams(tokens) , nltk.trigrams(tokens). We can develop the model using any one of the ways.

**Implementation**

**Creating virtual Environment in Anaconda[7].**

For completing the project, I have used jupyter Notebook. I have created the virtual environment in Anaconda. The name of my virtual environment is tf-gpu.

A picture containing diagram

Description automatically generated

This tf-gpu environment utilizes the graphic card(CUDA cores) that is available in the laptop/Desktop which is exclusive to Nvidia GPU’s.

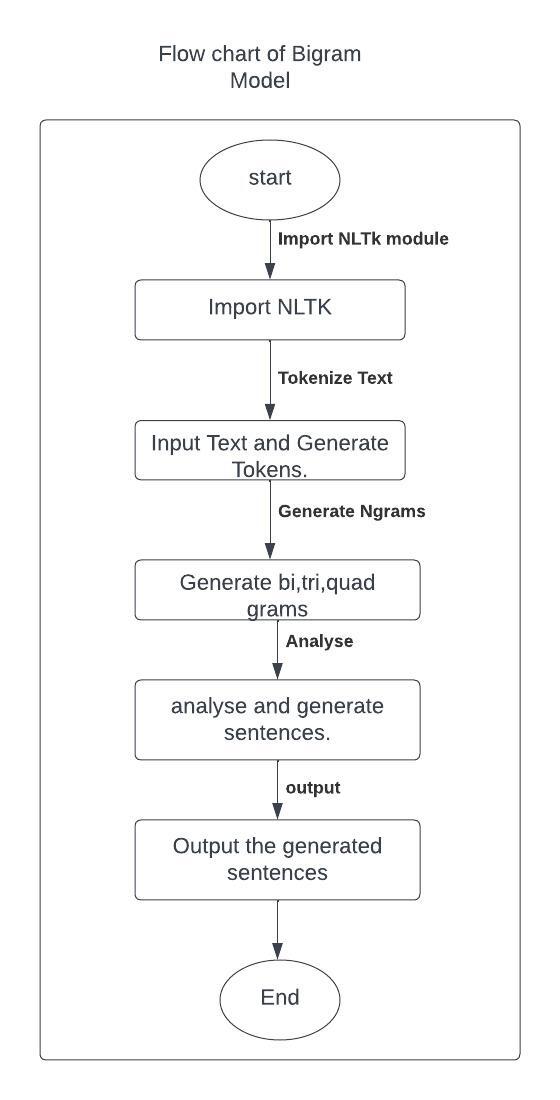
Commands to create the environment

1. conda create -n tf-gpu tensorflow-gpu (creates the environment)
2. conda activate tf-gpu (enters into the environment)
3. Install required dependencies from requirements.txt file
4. write this code statement after the import nltk statement. nltk.download('punkt')
5. If working with docx file -> pip install python-docx

if its not working or issues exits while running

First run “pip install docx”, then use pip install python-docx.

**Flow of the project**



**Code of the project**

Graphical user interface, text

Description automatically generated

Importing the required libraries

Text

Description automatically generated

In line 3,4 the data is read from the docx file. From line 7 to 28 the text variable is getting updated as the data is being cleaned in each step.

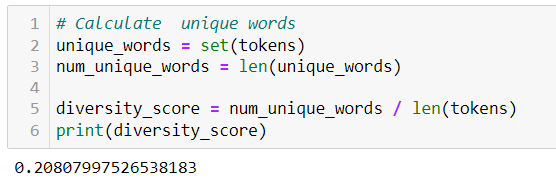
In line 31, 32 the data in the text variable is stored in the input.txt file. The data in the input.txt file is free from punctuations, special symbols, numbers etc., which does not have any meaning.

Graphical user interface, text, application

Description automatically generated

Reading the file input.txt which is generated in the previous case. Graphical user interface, text, application, email

Description automatically generated

The above code generates the tokens. The lower cell has the count of tokens. 

Calculating the unique words using set function. Find the length of unique words. The unique-words over the total length gives the diversity score.

Text

Description automatically generated with medium confidence

This plot shows the 30 most frequent tokens present in the corpus. Chart, line chart

Description automatically generated

A picture containing company name

Description automatically generatedAssigning the conditional frequency distribution for the variable cfd\_bigram. Text

Description automatically generated

The code creates a heatmap visualization of the bigram counts in the cfd\_bigram object using the seaborn and pandas libraries.  
Chart

Description automatically generated with medium confidence

Text

Description automatically generatedcfd\_bigram.items() returns a list of tuples, where each tuple contains a first word and its corresponding FreqDist object. Each FreqDist object contains the counts of all bigrams that start with the corresponding first word. Each tuple in the list represents a unique first word in the text.

**Calculating perplexity score of bigrams**Graphical user interface, text, application

Description automatically generated

This code calculates the perplexity score of a bigram model using the bigram counts stored in a ConditionalFreqDist object (cfd\_bigram) and a list of tokens (tokens).

**Calculating diversity score of bigramsText

Description automatically generated**This code calculates the diversity of bigrams in a text. This line first creates a list of all bigrams in the text using nltk.bigrams(tokens), and then creates a set of unique bigrams using set().

**Text

Description automatically generated**

This function takes in a bigram frequency distribution (data) which is cdf\_bigram and generates a random sentence using that distribution. First, it selects random word from the list of all words in the frequency distribution using random.choice(list(data.keys())). Then, it generates the rest of the sentence by selecting the most common next word from the bigram frequency distribution of the current word using freq\_dist.most\_common(1)[0][0]. This is repeated for 9 times to generate sentence. Graphical user interface, text, application

Description automatically generated

Generating frequency conditional pairs for trigrams.

**Calculating perplexity score of trigrams**

Text

Description automatically generatedGraphical user interface, text, application

Description automatically generatedText

Description automatically generated

Heat map for trigrams

Chart

Description automatically generated

A picture containing text

Description automatically generated

The above is the items in the cdf\_trigrams.

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generatedGenerating quadgrams.

Text

Description automatically generatedGraphical user interface, text, application

Description automatically generatedText

Description automatically generatedA picture containing diagram

Description automatically generated

Text

Description automatically generated

**Analysis**

Diversity score of entire dataset : 0.20807997526538183

Perplexity score of bigrams : 10.965655218617805

Diversity score of bigrams : 0.7062461348175634

Perplexity score of trigrams : infinity

Diversity score of trigrams : 0.920832903824348

perplexity score of quadgrams : 1.0

Diversity score of quadgrams : 0.0

The perplexity score of quadgrams is 1.0 and diversity score of quadgrams is 0. i.e., frequency is 0 or the unique values are zero.

The generated sentences

Bigrams

Sentence : 1

architecture for the blockchain technology and the blockchain technology and

Sentence : 2

reads data and the blockchain technology and the blockchain technology

Sentence : 3

modules is a blockchain technology and the blockchain technology and

Sentence : 4

though the blockchain technology and the blockchain technology and the

Sentence : 5

faults in the blockchain technology and the blockchain technology and

Trigrams

Sentence : 1

final developers feasibility institutes the the regulatory digital the at

Sentence : 2

of blockchain generate with not system aspects peer security and

Sentence : 3

brains a development systemic feasible this application controlled a number

Sentence : 4

time questions protocol affect a the industrial database a structure

Sentence : 5

framework popular tremendous incorporating aspects in can to tolerance consensus

quadgrams

Sentence : 1

technology and application development white paper blockchain reference framework and

Sentence : 2

produced by the network as well as a connectivity manager

Sentence : 3

with enough cryptocurrency to pay for the user gas costs

Sentence : 4

will not be conducive to the development trend of blockchain

Sentence : 5

have also demonstrated what can be achieved while pushing the

The above three are the outputs that are generated from the 3 different models.

For generating the sentences, we use conditional frequencies because the words in the sentences you generate should follow those probabilities.

When we use bigrams then the first word is randomly picked. The second word should be the most frequent second word for the given first word. The third word would be the most frequent word given the second word and so on until 10 words are generated.

For trigram, pick the first word at random. Then pick the second word from bigram. Then use those two words to pick the most frequent third word. Then use the second and third word to pick the fourth word and so on until you generate sentence. The same idea is followed for quadgrams as well.

In bigrams the words are repeated many times in a sentence, where as in the trigrams and quadgrams the diversity of the words in the sentence is more.

**References:**

[1] https://docs.anaconda.com/anaconda/user-guide/tasks/tensorflow/

[2] https://www.nltk.org/api/nltk.html

[3] https://pypi.org/project/doc2text/

[4] https://www.nltk.org/\_modules/nltk/tokenize/punkt.html

[5] https://anaconda.org/conda-forge/docx2txt

[6] https://stackoverflow.com/questions/66540825/unable-to-encode-a-unicode-into-a-txt-file-in-python

[7] https://docs.anaconda.com/anaconda/user-guide/tasks/tensorflow/

[8] [MyFlowchart](https://lucid.app/lucidchart/c0e716a1-cfbe-4f2b-931f-e117075c3be2/edit?viewport_loc=-10%2C122%2C1707%2C779%2C0_0&invitationId=inv_778ed5fc-3404-480f-a8b4-26ed3ad168d8)

[9] <https://seaborn.pydata.org/>

[10] <https://matplotlib.org/>

[11] https://stackoverflow.com/questions/23747756/python-plotting-conditional-frequency-distributions

[12] https://stackoverflow.com/questions/66890249/how-to-find-perplexity-of-bigram-if-probability-of-given-bigram-is-0

[13]

[14]