CS 565 Assignment 1 Report

UDBHAV CHUGH: 170101081

The colab notebook with the code for the assignment is available at the following link:
https://colab.research.google.com/drive/1qKjq67I53cOGSlmWn2uCRrhB_06Gemf3?usp=sharing

Note:

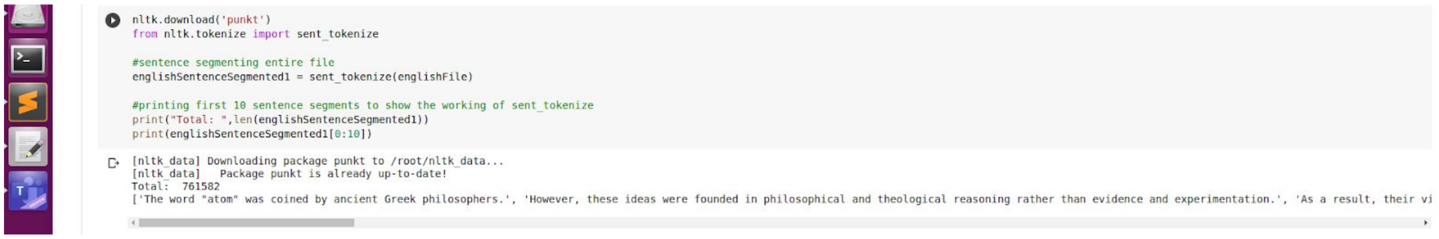
- Instructions to run the code are given alongside the colab notebook. Since the two corpus are very large, at some instances only parts of the corpus are used, details of which can be found in the below mentioned report.
- Since the corpus is large, there is an issue of exceeding RAM. Hence it is advised to run only necessary cells for which testing is required at any particular time.
- The entire notebook has been run in parts and outputs can be seen in the corresponding cells.
- Required outputs are copy pasted from the notebook in the report as text. Additionally, small screenshots are also attached for reference.

Question 1.3.1

1. Sentence segmentation and word tokenization on English and Hindi corpora.
Tool used for English Corpora: NLTK

Methods for sentence tokenization in NLTK:

- sent_tokenize: It is a wrapper over PunktSentenceTokenizer from the nltk.tokenize.punkt module, which has already been trained and thus very well knows to mark the end and beginning of sentences. Using the sent_tokenize function on the given corpus a total of **761582** sentences are generated.



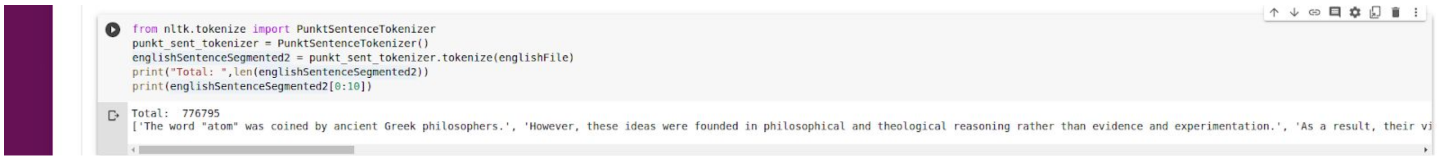
```
nltk.download('punkt')
from nltk.tokenize import sent_tokenize

#sentence segmenting entire file
englishSentenceSegmented1 = sent_tokenize(englishFile)

#printing first 10 sentence segments to show the working of sent_tokenize
print("Total: ",len(englishSentenceSegmented1))
print(englishSentenceSegmented1[0:10])
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Total:  761582
['The word "atom" was coined by ancient Greek philosophers.', 'However, these ideas were founded in philosophical and theological reasoning rather than evidence and experimentation.', 'As a result, their vi
```

- PunktSentenceTokenizer: This tool loads PunktSentenceTokenizer using English pickle file. With huge chunks of data then it is efficient to use PunktSentenceTokenizer. Using the sent_tokenize function on the given corpus a total of **776795** sentences are generated.



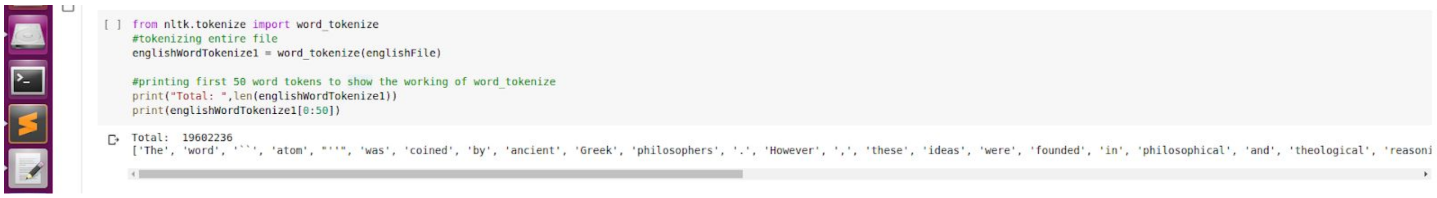
```
from nltk.tokenize import PunktSentenceTokenizer
punkt_sent_tokenizer = PunktSentenceTokenizer()
englishSentenceSegmented2 = punkt_sent_tokenizer.tokenize(englishFile)
print("Total: ",len(englishSentenceSegmented2))
print(englishSentenceSegmented2[0:10])
```

```
Total:  776795
['The word "atom" was coined by ancient Greek philosophers.', 'However, these ideas were founded in philosophical and theological reasoning rather than evidence and experimentation.', 'As a result, their vi
```

One difference that is evident is that in the name J.J. Thompson sent_tokenize is able to recognize that the period here is not sentence end while PunktSentenceTokenizer puts J.J. in a separate sentence and Thompson in another. Similar examples result in PunktSentenceTokenizer having more sentences over the same corpus

Methods for word tokenization in NLTK:

- word_tokenize: word_tokenize() function is a wrapper function that calls tokenize() on an instance of the TreebankWordTokenizer class. Using the word_tokenize function on the given corpus a total of **19602236** tokens are generated.

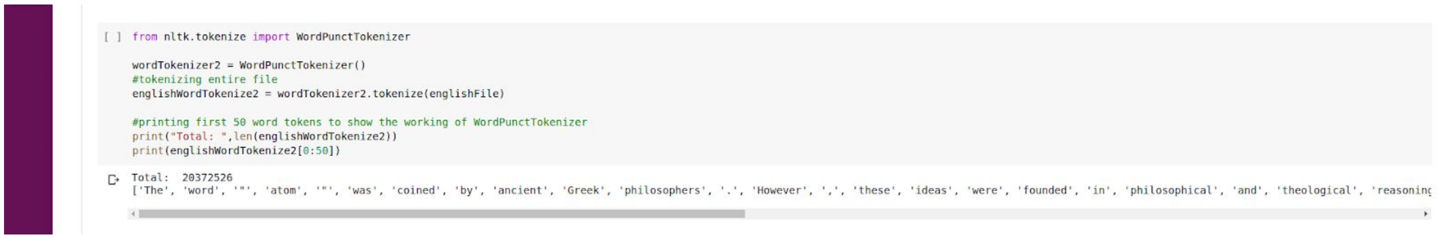


```
[ ] from nltk.tokenize import word_tokenize
    #tokenizing entire file
    englishWordTokenize1 = word_tokenize(englishFile)

    #printing first 50 word tokens to show the working of word_tokenize
    print("Total: ",len(englishWordTokenize1))
    print(englishWordTokenize1[0:50])

    Total:  19602236
    ['The', 'word', '``', 'atom', "''", 'was', 'coined', 'by', 'ancient', 'Greek', 'philosophers', '.', 'However', ',', 'these', 'ideas', 'were', 'founded', 'in', 'philosophical', 'and', 'theological', 'reasoni
```

- WordPunctTokenizer: This tokenizer separates all types of punctuations from words. Using the WordPunctTokenizer function on the given corpus a total of **20372526** tokens are generated.



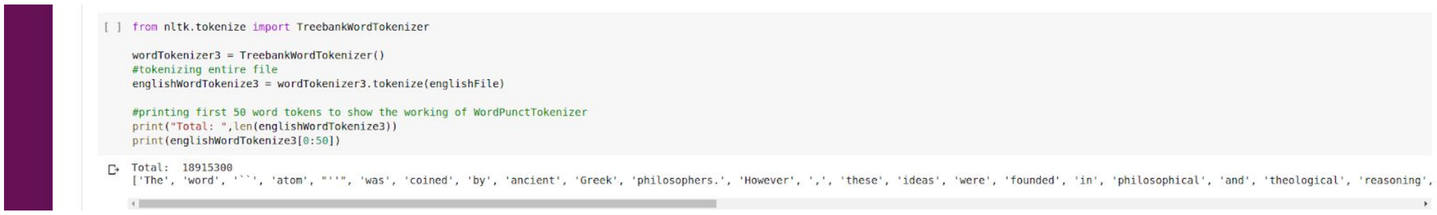
```
[ ] from nltk.tokenize import WordPunctTokenizer

    wordTokenizer2 = WordPunctTokenizer()
    #tokenizing entire file
    englishWordTokenize2 = wordTokenizer2.tokenize(englishFile)

    #printing first 50 word tokens to show the working of WordPunctTokenizer
    print("Total: ",len(englishWordTokenize2))
    print(englishWordTokenize2[0:50])

    Total:  20372526
    ['The', 'word', '"', 'atom', '"', 'was', 'coined', 'by', 'ancient', 'Greek', 'philosophers', '.', 'However', ',', 'these', 'ideas', 'were', 'founded', 'in', 'philosophical', 'and', 'theological', 'reasoninc
```

- TreebankWordTokenizer: Using the TreebankWordTokenizer function on the given corpus a total of **18915300** tokens are generated.



```
[ ] from nltk.tokenize import TreebankWordTokenizer

    wordTokenizer3 = TreebankWordTokenizer()
    #tokenizing entire file
    englishWordTokenize3 = wordTokenizer3.tokenize(englishFile)

    #printing first 50 word tokens to show the working of WordPunctTokenizer
    print("Total: ",len(englishWordTokenize3))
    print(englishWordTokenize3[0:50])

    Total:  18915300
    ['The', 'word', '``', 'atom', "''", 'was', 'coined', 'by', 'ancient', 'Greek', 'philosophers.', 'However', ',', 'these', 'ideas', 'were', 'founded', 'in', 'philosophical', 'and', 'theological', 'reasoning',
```

One difference that is evident in word_tokenize(and TreebankWordTokenizer) from WordPunctTokenizer can be seen in the tokenization of text like "Bohr's theory"

word_tokenizer and TreebankWordTokenizer tokenizes it as: "**Bohr**" "**'s**" "**theory**" (3 tokens)
WordPunctTokenizer tokenizes it as: "**Bohr**" "'" "**s**" "**theory**" (4 tokens)

One difference that is evident in word_tokenize and WordPunctTokenizer from TreebankWordTokenizer can be seen in the tokenization of text like "by philosophers."
word_tokenizer and WordPunctTokenizer tokenizes it as: "**by**" "**philosophers**" "**.**" (3 tokens)
TreebankWordTokenizer tokenizes it as: "**by**" "**philosophers.**" (2 tokens)

Tools used for **Hindi** Corpus: indic-nlp-library and Stanford Stanza library
For Hindi, 2 different libraries are used as no library has multiple tokenization methods.

Methods for sentence segmentation for Hindi Corpus:

- indic-nlp: sentence_tokenize in indic-nlp is used for sentence segmentation. On the given Hindi corpus, **348593** sentences were generated using this tool.
- Stanford stanza: the nlp-pipeline of Stanford Stanza is used for sentence segmentation. Since this tool takes more time it was run on 10% of the corpora. On the given 10% Hindi corpus, **36425** sentences were generated using this tool.

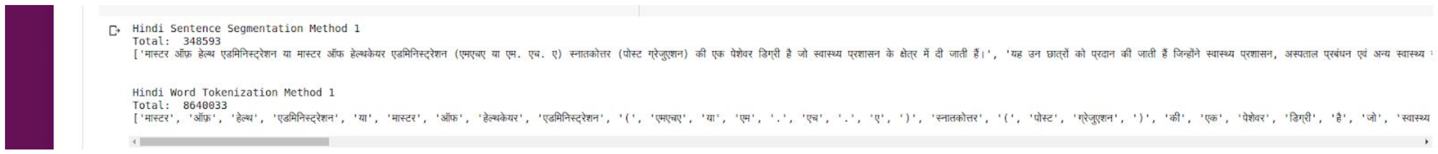
One difference in the two sentence segmentation is the separation of text सकता हैं, हालांकि
IndicNLP doesn't separate at हैं and considers entire text as one sentence which is correct
Stanford stanza classifies this as two sentences with sentence boundary at हैं which is incorrect.

Method for word tokenization for Hindi Corpus:

- indic-nlp: indic_tokenize.trivial_tokenize in indic-nlp tool is used for word tokenization. On the given Hindi corpus, **8640033** tokens were generated using this tool.
- Stanford stanza: the nlp-pipeline of Stanford Stanza is used for token segmentation. Since this tool takes more time it was run on 10% of the corpora. On the given 10% Hindi corpus, **840135** tokens were generated using this tool.

One difference in the tokenization of the two tools is in tokenization of **एम. एच. ए**
indic-nlp classifies this as 5 tokens **'एम' '.' 'एच' '.' 'ए'**
Stanford stanza classifies this as 1 token **'एम. एच. ए'**

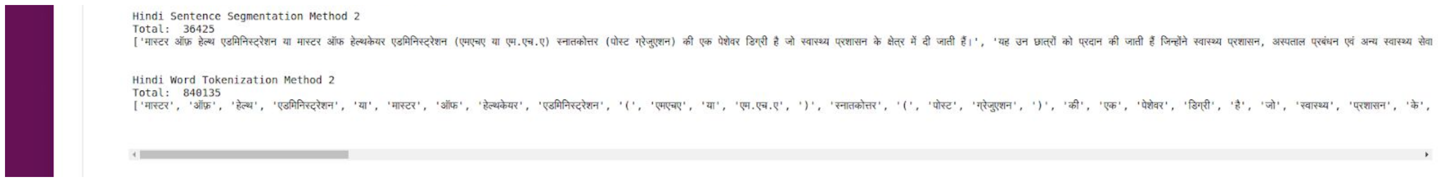
IndicNLP



```
Hindi Sentence Segmentation Method 1
Total:  348593
['मास्टर ऑफ़ हेल्थ एडमिनिस्ट्रेशन या मास्टर ऑफ हेल्थकेयर एडमिनिस्ट्रेशन (एमएचए या एम. एच. ए) स्नातकोत्तर (पोस्ट ग्रेजुएशन) की एक पेशेवर डिग्री है जो स्वास्थ्य प्रशासन के क्षेत्र में दी जाती हैं।', 'यह उन छात्रों को प्रदान की जाती हैं जिन्होंने स्वास्थ्य प्रशासन, अस्पताल प्रबंधन एवं अन्य स्वास्थ्य

Hindi Word Tokenization Method 1
Total:  8640033
['मास्टर', 'ऑफ़', 'हेल्थ', 'एडमिनिस्ट्रेशन', 'या', 'मास्टर', 'ऑफ', 'हेल्थकेयर', 'एडमिनिस्ट्रेशन', '(', 'एमएचए', 'या', 'एम', '.', 'एच', '.', 'ए', ')', 'स्नातकोत्तर', '(', 'पोस्ट', 'ग्रेजुएशन', ')', 'की', 'एक', 'पेशेवर', 'डिग्री', 'है', 'जो', 'स्वास्थ्य
```

Stanford Stanza



```
Hindi Sentence Segmentation Method 2
Total:  36425
['मास्टर ऑफ़ हेल्थ एडमिनिस्ट्रेशन या मास्टर ऑफ हेल्थकेयर एडमिनिस्ट्रेशन (एमएचए या एम.एच.ए) स्नातकोत्तर (पोस्ट ग्रेजुएशन) की एक पेशेवर डिग्री है जो स्वास्थ्य प्रशासन के क्षेत्र में दी जाती हैं।', 'यह उन छात्रों को प्रदान की जाती हैं जिन्होंने स्वास्थ्य प्रशासन, अस्पताल प्रबंधन एवं अन्य स्वास्थ्य सेवा

Hindi Word Tokenization Method 2
Total:  840135
['मास्टर', 'ऑफ़', 'हेल्थ', 'एडमिनिस्ट्रेशन', 'या', 'मास्टर', 'ऑफ', 'हेल्थकेयर', 'एडमिनिस्ट्रेशन', '(', 'एमएचए', 'या', 'एम.एच.ए', ')', 'स्नातकोत्तर', '(', 'पोस्ट', 'ग्रेजुएशन', ')', 'की', 'एक', 'पेशेवर', 'डिग्री', 'है', 'जो', 'स्वास्थ्य', 'प्रशासन', 'के',
```

1.3.1 Question 2,3,4

Find unigrams, bigrams, trigrams, their frequencies and plot frequency distribution.
Unigrams are simply tokens of the corpora.
For the purpose of this question,

- For **English** corporus, unigrams are generated from word_tokenize tool in NLTK, bigrams are generated using nltk.bigrams tool in NLTK, trigrams are generated using nltk.trrigrams tool in NLTK.
- For **Hindi** corporus, unigrams are generated using indic_tokenize in indic-nlp tool. Bigrams are generated by combining 2 consecutive unigrams while trigrams are generated by combining 3 consecutive unigrams.

Few observations regarding frequencies of ngrams:

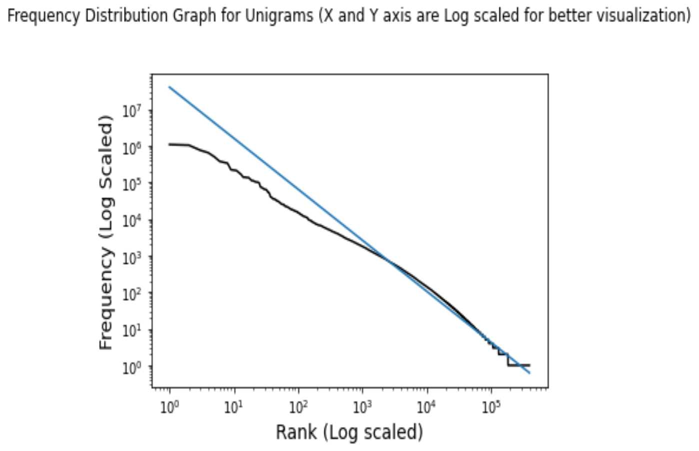
- As it can be noticed the most frequent ngrams are mainly punctuations symbols and prepositions, articles and conjunctions
- The least frequent ngrams are either few scientific words which are rare or some are a few mathematical values indicating that they occur only once in the entire corpus.
- The graphs of frequency distribution are plotted with log scale on both x and y axis for better visualization since there are drastic differences in most frequent and least frequent ngrams.
- Log(frequency) vs log(rank) is an indicator of frequency vs rank graph. If f is proportional to 1/r then ideally log(f) = -log(r) + b. So a slope close to -1 **(m=-1)** in the graph log(f) vs log(r) indicates correctness of Zipf's Law.

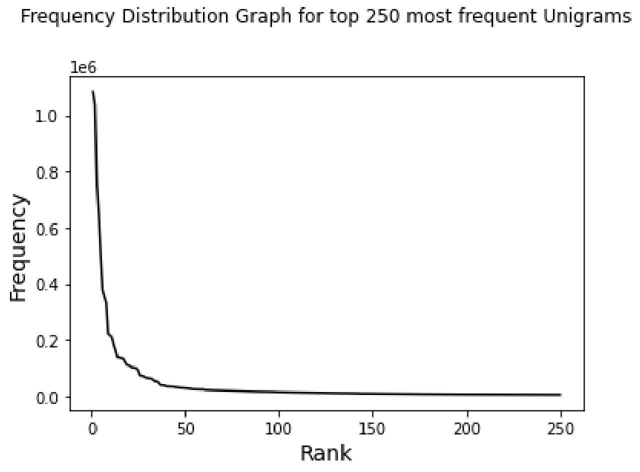
- The graphs plotted show a somewhat similar trend for unigrams and bigrams(m is close to -1) and deviate more for trigrams (m is around -0.4). They overlap with the fit line **(blue line)** towards the centre of the graph while deviating from it towards the end points showing a trend similar to **Zipf's** law.

The following page shows the most and least frequent ngrams, and frequency distribution of ngrams based on rank for both **English** and **Hindi** corpus. For left graph both x and y axis are log scaled and all ngrams are taken and for right graph top 250 most frequent ngrams are taken.

Frequnecy distribution for English Corpus unigrmas.

m = - `1.3942748169093626 for left graph`

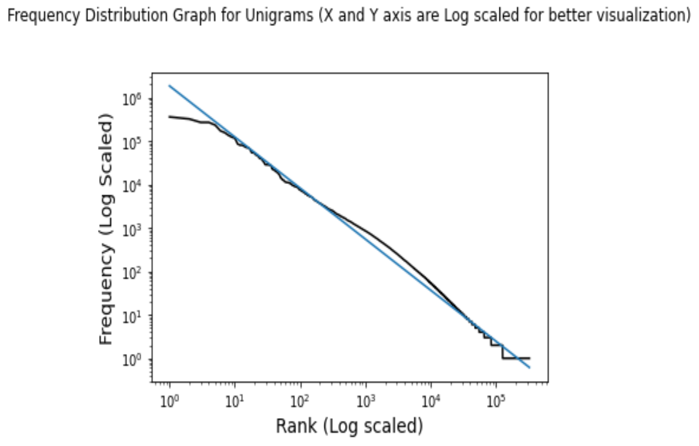


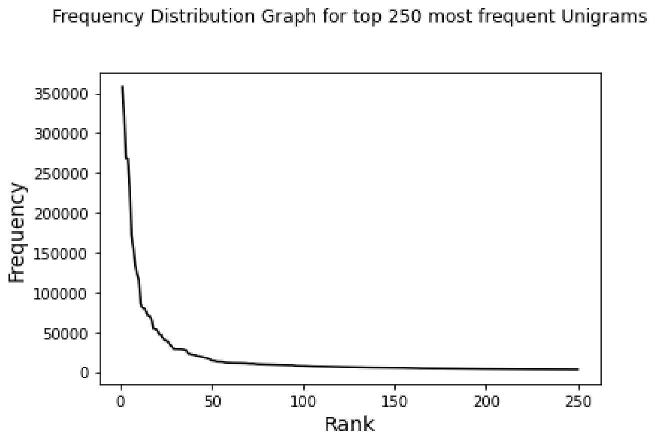


```
10 most frequent Unigrams: the :  1083392, , :  1037808, . :  757183, of :  646178,
and :  498233, in :  377175, to :  354649, a :  333421, '' :  222661, `` :  216424,
10 least frequent Unigrams: 100g :  1, 13.5g :  1, 27g :  1, units—in :  1, corpuscle
:  1, radiochemist :  1, transuranium :  1, alpha-particle :  1, dimensions—on :  1,
range-properties :  1,
```

Frequency distribution for Hindi Corpus unigrmas.

m = -`1.175844035172983 for left graph`



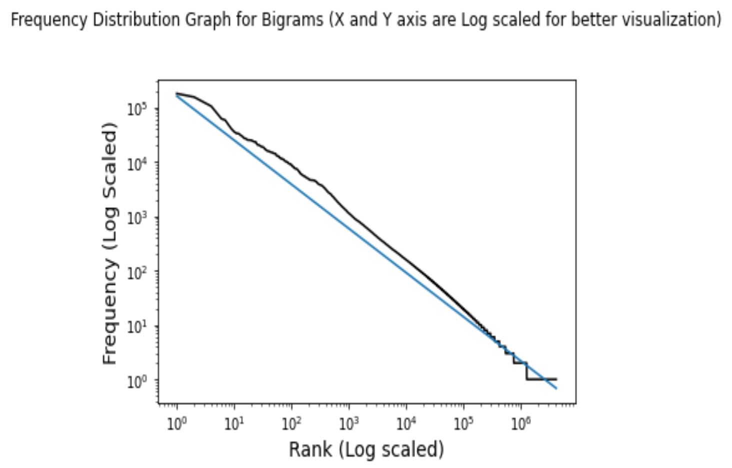


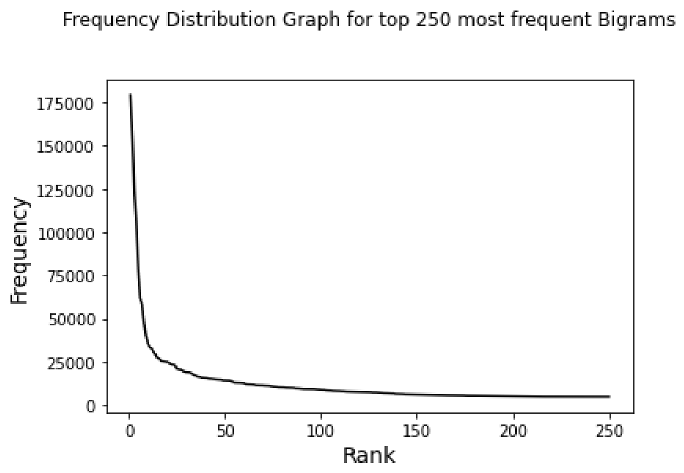
```
10 most frequent Unigrams:के :  357833, । :  321526, में :  268249, , :  267743, है :
232156, की :  171779, और :  155743, से :  134901, का :  122948, को :  115876,

10 least frequent Unigrams:एमएचए :  1, “कमीशन :  1, अक्रेदिसन :  1, एजुकेशन” :  1,
सीएएचएमई :  1, एचओएम :  1, आषधियों :  1, पाठ्शालाओं :  1, डिग्रीधारीयों :  1, हैल्थकेयर :  1,
```

Frequency distribution for English Corpus bigrams.

m = -0.8123473678166971 for left graph



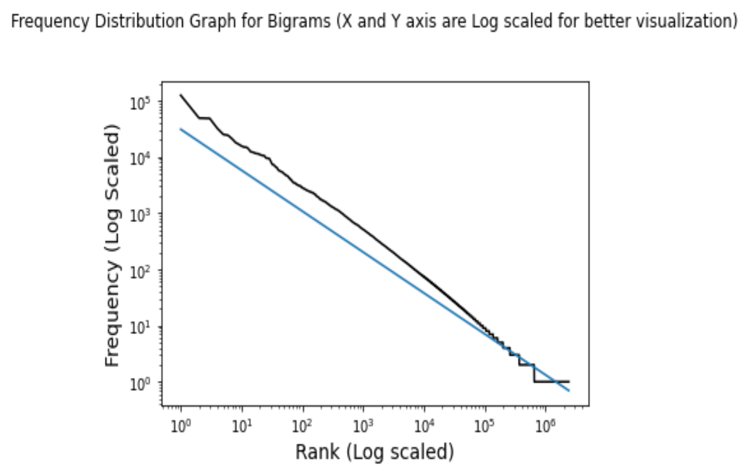


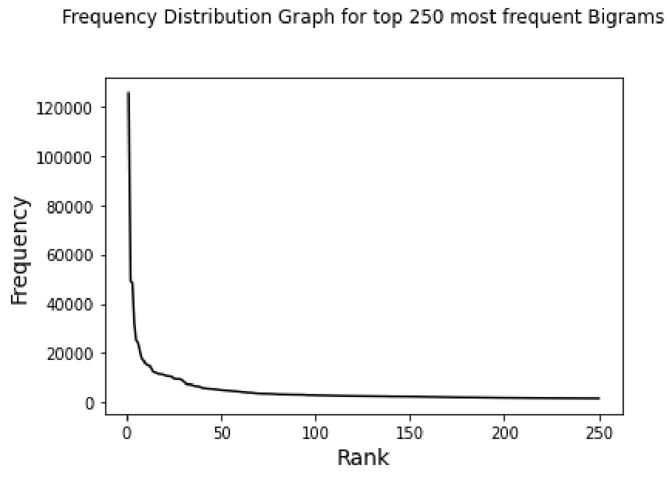
```
10 most frequent Bigrams:('of', 'the') :  179411, ('.', 'The') :  154963, (',', 'and') :  123570,
('in', 'the') :  104796, (',', 'the') :  78291, ('.', 'In') :  61881, ('to', 'the') :  58223,
('and', 'the') :  47437, ("''", ',') :  40118, (')', '.') :  35668,

10 least frequent Bigrams:('theological', 'reasoning') :  1, ('than', 'evidence') :  1, ('atoms',
'look') :  1, ('behave', 'were') :  1, ('convince', 'everybody') :  1, ('so', 'atomism') :  1,
('blossoming', 'science') :  1, ('chemistry', 'produced') :  1, ('produced', 'discoveries') :  1,
('elements', 'always') :  1,
```

Frequnecy distribution for Hindi Corpus bigrams.

m = -0.7286811063537385 for left graph



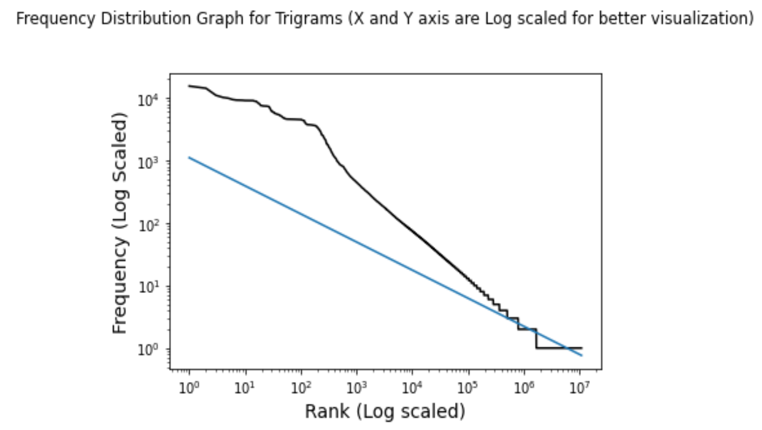


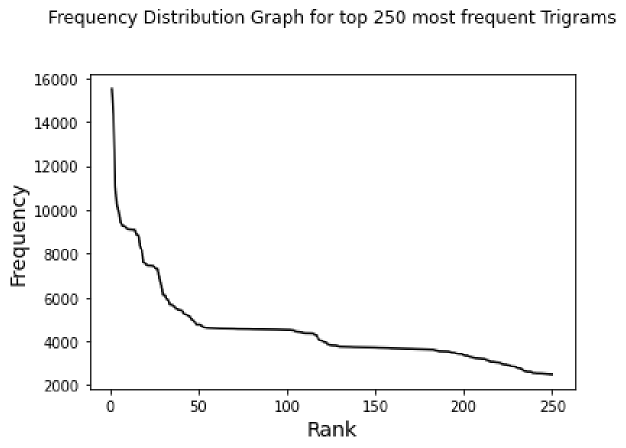
```
10 most frequent Bigrams:('है', '।') :  125369, ('के', 'लिए') :  49250, ('हैं', '।') :  48556, ('है',
',') :  32259, ('जाता', 'है') :  25250, ('था', '।') :  24258, ('के', 'साथ') :  20999, ('रूप', 'में') :
18031, ('के', 'रूप') :  16812, ('है', 'कि') :  15652,

10 least frequent Bigrams:('(', 'एमएचए') :  1, ('एमएचए', 'या') :  1, (')', 'स्नातकोत्तर') :  1,
('जिन्होंने', 'स्वास्थ्य') :  1, ('इन', 'पाठ्यक्रमो') :  1, ('इनके', 'सरंचना') :  1, ('हालांकि', 'व्यवसायी') :  1,
('शिक्षक', 'मॉडल') :  1, ('मॉडल', 'कार्यक्रम') :  1, ('स्वास्थ्य', 'व्यवसायों') :  1,
```

Frequency distribution for English Corpus trigrmas.

m = - 0.44948189732604676 for left graph



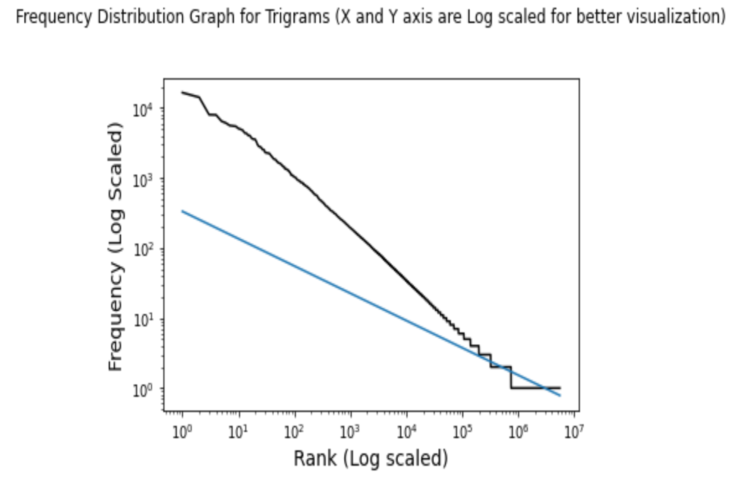


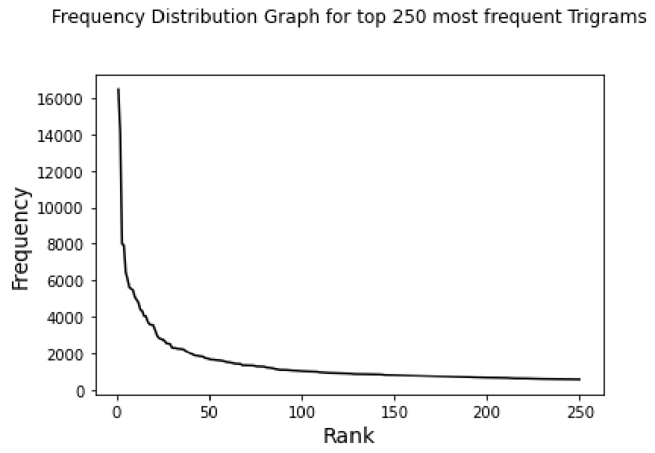
```
10 most frequent Trigrams: (',', 'and', 'the') :  15519, ('.', 'In', 'the') :  14311, ('the',
'age', 'of') :  11064, ('.', 'There', 'were') :  10236, ('under', 'the', 'age') :  9928, ('the',
'United', 'States') :  9454, ('age', 'of', '18') :  9262, ('years', 'of', 'age') :  9247, (')',
'.', 'The') :  9195, ('or', 'older', '.') :  9118,

10 least frequent Trigrams: ('atom', "''", 'was') :  1, ('coined', 'by', 'ancient') :  1, ('Greek',
'philosophers', '.') :  1, ('philosophers', '.', 'However') :  1, ('these', 'ideas', 'were') :  1,
('ideas', 'were', 'founded') :  1, ('founded', 'in', 'philosophical') :  1, ('in', 'philosophical',
'and') :  1, ('and', 'theological', 'reasoning') :  1, ('theological', 'reasoning', 'rather') :  1,
```

Frequnecy distribution for Hindi Corpus trigrmas.

m = -0.38964029287065705 for left graph





```
10 most frequent Trigrams: ('के', 'रूप', 'में') :  16475, ('जाता', 'है', '।') :  14139, ('करने', 'के',
'लिए') :  7987, ('होता', 'है', '।') :  7927, ('किया', 'जाता', 'है') :  6457, ('है', '।', 'यह') :  6068,
('सकता', 'है', '।') :  5596, ('गया', 'था', '।') :  5528, ('गया', 'है', '।') :  5446, ('जा', 'सकता', 'है')
:  5082,


10 least frequent Trigrams: ('हेल्थ', 'एडमिनिस्ट्रेशन', 'या') :  1, ('मास्टर', 'ऑफ', 'हेल्थकेयर') :  1,
('ऑफ', 'हेल्थकेयर', 'एडमिनिस्ट्रेशन') :  1, ('हेल्थकेयर', 'एडमिनिस्ट्रेशन', '(') :  1, ('एडमिनिस्ट्रेशन', '(',
'एमएचए') :  1, ('(', 'एमएचए', 'या') :  1, ('एमएचए', 'या', 'एम') :  1, ('ए', ')', 'स्नातकोत्तर') :  1,
(')', 'स्नातकोत्तर', '(') :  1, ('स्नातकोत्तर', '(', 'पोस्ट') :  1,
```

Log(frequency) vs log(rank) is an indicator of frequency vs rank graph. If f is proportional to 1/r then ideally log(f) = -log(r) + b. So a slope close to -1 **(m=-1)** in the graph log(f) vs log(r) indicates correctness of Zipf's Law.
The graphs plotted show a somewhat similar trend for unigrams and bigrams(m is close to -1) and deviate more for trigrams (m is around -0.4). They overlap with the fit line **(blue line)** towards the centre of the graph while deviating from it towards the end points showing a trend similar to **Zipf's** law.

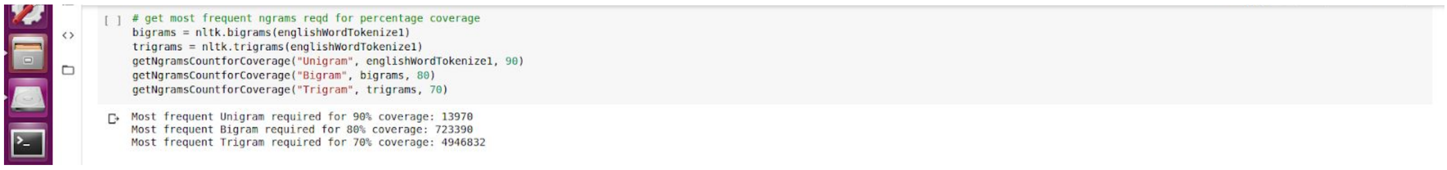
Question 1.3.2

For the purpose of this question,

- For **English** corporus, unigrams are generated from word_tokenize tool in NLTK, bigrams are generated using nltk.bigrams tool in NLTK, trigrams are generated using nltk.trrigrams tool in NLTK.
- Note for stemming of English corpora PorterStemmer in NLTK.stem tool is used.
- For **Hindi** corporus, unigrams are generated using Stanford Stanza library. Bigrams are generated by combining 2 consecutive unigrams while trigrams are generated by combining 3 consecutive unigrams. Stemming is performed using Stanford Stanza library as well.
- Since Stanford Stanza library is slow on processing, for this task 10% of Hindi corporus is used.

English Copropa:

1) Number of Most frequent Unigram required for 90% coverage: **13970**
2) Number of Most frequent Bigram required for 80% coverage: **723390**
3) Number of Most frequent Trigram required for 70% coverage: **4946832**

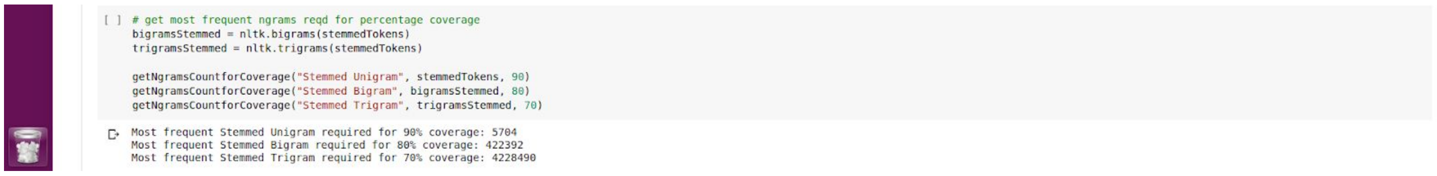


```
# get most frequent ngrams reqd for percentage coverage
bigrams = nltk.bigrams(englishWordTokenize1)
trigrams = nltk.trigrams(englishWordTokenize1)
getNgramsCountforCoverage("Unigram", englishWordTokenize1, 90)
getNgramsCountforCoverage("Bigram", bigrams, 80)
getNgramsCountforCoverage("Trigram", trigrams, 70)
```

```
Most frequent Unigram required for 90% coverage: 13970
Most frequent Bigram required for 80% coverage: 723390
Most frequent Trigram required for 70% coverage: 4946832
```

4) After Stemming:

- Number of Most frequent Unigram required for 90% coverage: **5704**
- Number of Most frequent Bigram required for 80% coverage: **422392**
- Number of Most frequent Trigram required for 70% coverage: **4228490**



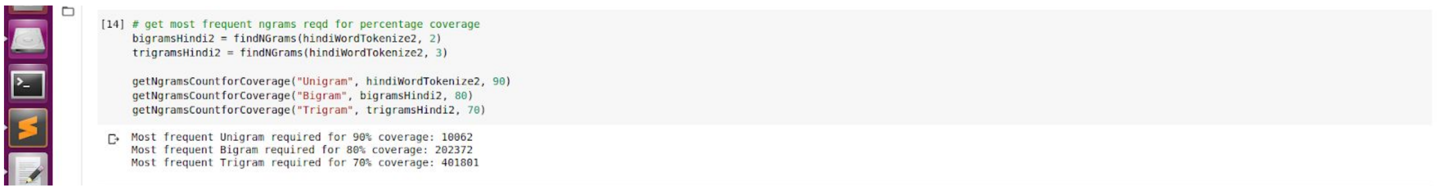
```
# get most frequent ngrams reqd for percentage coverage
bigramsStemmed = nltk.bigrams(stemmedTokens)
trigramsStemmed = nltk.trigrams(stemmedTokens)

getNgramsCountforCoverage("Stemmed Unigram", stemmedTokens, 90)
getNgramsCountforCoverage("Stemmed Bigram", bigramsStemmed, 80)
getNgramsCountforCoverage("Stemmed Trigram", trigramsStemmed, 70)
```

```
Most frequent Stemmed Unigram required for 90% coverage: 5704
Most frequent Stemmed Bigram required for 80% coverage: 422392
Most frequent Stemmed Trigram required for 70% coverage: 4228490
```

Hindi Copropa (10% corporus is used for this task)**:**

1) Number of Most frequent Unigram required for 90% coverage: **10062**
2) Number of Most frequent Bigram required for 80% coverage: **202372**
3) Number of Most frequent Trigram required for 70% coverage: **401801**



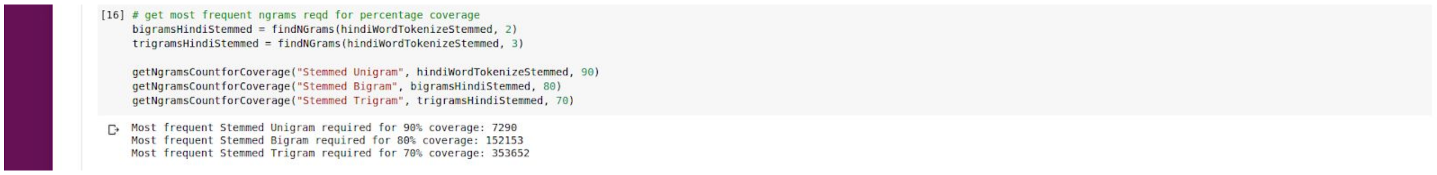
```
[14] # get most frequent ngrams reqd for percentage coverage
     bigramsHindi2 = findNGrams(hindiWordTokenize2, 2)
     trigramsHindi2 = findNGrams(hindiWordTokenize2, 3)

     getNgramsCountforCoverage("Unigram", hindiWordTokenize2, 90)
     getNgramsCountforCoverage("Bigram", bigramsHindi2, 80)
     getNgramsCountforCoverage("Trigram", trigramsHindi2, 70)

     Most frequent Unigram required for 90% coverage: 10062
     Most frequent Bigram required for 80% coverage: 202372
     Most frequent Trigram required for 70% coverage: 401801
```

4) After Stemming:

- Number of Most frequent Unigram required for 90% coverage: **7290**
- Number of Most frequent Bigram required for 80% coverage: **152153**
- Number of Most frequent Trigram required for 70% coverage: **353652**



```
[16] # get most frequent ngrams reqd for percentage coverage
     bigramsHindiStemmed = findNGrams(hindiWordTokenizeStemmed, 2)
     trigramsHindiStemmed = findNGrams(hindiWordTokenizeStemmed, 3)

     getNgramsCountforCoverage("Stemmed Unigram", hindiWordTokenizeStemmed, 90)
     getNgramsCountforCoverage("Stemmed Bigram", bigramsHindiStemmed, 80)
     getNgramsCountforCoverage("Stemmed Trigram", trigramsHindiStemmed, 70)

     Most frequent Stemmed Unigram required for 90% coverage: 7290
     Most frequent Stemmed Bigram required for 80% coverage: 152153
     Most frequent Stemmed Trigram required for 70% coverage: 353652
```

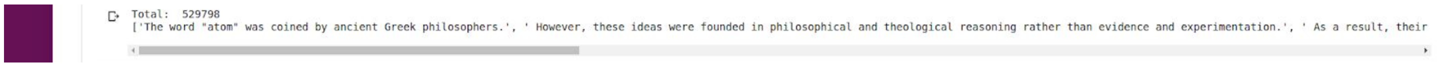
It can be seen that since stemming reduces a few tokens to the same token (which have the same stem), the number of most frequent ngram required to cover the given percentage is reduced.

Question 1.3.3

1) **Self Implementation** of Sentence tokenization and word tokenization heuristics:

For **English** Corpus Sentence segmentation:

- The following symbols are used as separators = ['.','?','!',';']
- If a separator is followed by a lowercase letter, it is not a sentence separator.
- If a period is preceded by an abbreviation = = ['Mr', 'Mrs', 'Ms', 'Jr', 'Dr', 'vs', 'etc']. It is not a sentence separator.
- In case of Jr. and etc. if they are followed by upper case letter it is a sentence separator despite being an abbreviation.
- If any of the separators are present within quotes then it is not a sentence segmentation. If “ is preceded by \ then it is not a quote symbol.

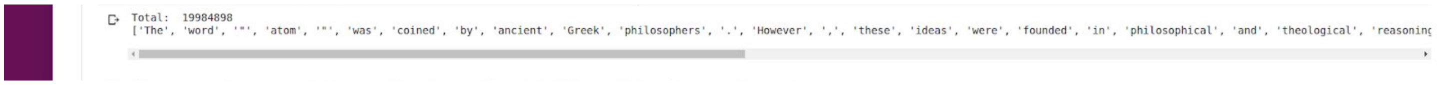


```
Total:  529798
['The word "atom" was coined by ancient Greek philosophers.', ' However, these ideas were founded in philosophical and theological reasoning rather than evidence and experimentation.', ' As a result, their
```

A total of **529798** sentences were generated using above heuristics compared to **761582** in sent_tokenize and **776795** in PunktSentenceTokenizer. A difference that can be noted from sent_tokenize is it classifies (J.J. Thompson) as a single sentence while my heuristic classifies it as 3 sentences ‘J.’ ‘J.’ ‘Thompson’

For **English** Corpus Word Tokenization:

- The white spaces are ignored and marked as seaprators between words: separators = [' ','\t','\n']
- If a punctuation mark is encountered punctuationSeparators = ['.',',',';', '!','?', '"', '"', '(', ')', '[‘, ']', '{', '}'], it also acts as a separator
- In case of punctuation mark, I have divided ‘a Punctuation b’ as ‘a’ ‘Punctuation’ ‘b’



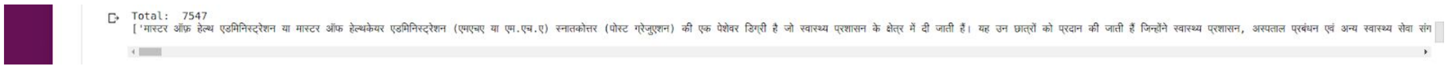
```
Total:  19984898
['The', 'word', '"', 'atom', '"', 'was', 'coined', 'by', 'ancient', 'Greek', 'philosophers', '.', 'However', ',', 'these', 'ideas', 'were', 'founded', 'in', 'philosophical', 'and', 'theological', 'reasoning
```

A total of **19984898** tokens were generated using above heuristics compared to **19602236** in word_tokenize, **20372526** in WordPunctTokenizer and **18915300** in TreebankWordTokenizer. Difference can be seen in tokenization of “Bohr’s”
word_tokenizer and TreebankWordTokenizer tokenizes it as: “Bohr”   “‘s” (2 tokens)
My heuristic tokenizes it as: “Bohr   “‘“   “s”   (3 tokens)

For **Hindi** Corpus Sentence segmentation:

- The following symbols are used as separators = ['|','?','!',';']
- If any of the separators are present within quotes then it is not a sentence segmentation. If “ is preceded by \ then it is not a quote symbol.

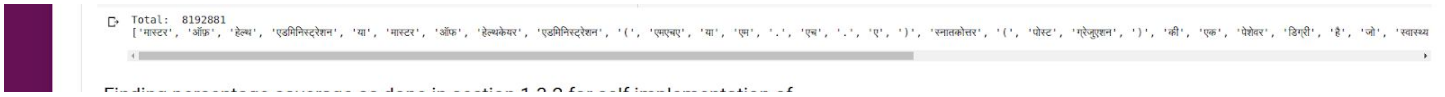
A total of **7547** sentences were generated using above heuristics compared to **348593** in indic-nlp and **36425** in Stanford Stanza (last one was on 10% corpus). One difference can be seen in the separation of text सकता हैं, हालांकि
Stanford Stanza classifies हैं as sentence boundary while my heuristic considers this as one sentence.



```
Total:  7547
['मास्टर ऑफ़ हेल्थ एडमिनिस्ट्रेशन या मास्टर ऑफ हेल्थकेयर एडमिनिस्ट्रेशन (एमएचए या एम.एच.ए) स्नातकोत्तर (पोस्ट ग्रेजुएशन) की एक पेशेवर डिग्री है जो स्वास्थ्य प्रशासन के क्षेत्र में दी जाती हैं। यह उन छात्रों को प्रदान की जाती हैं जिन्होंने स्वास्थ्य प्रशासन, अस्पताल प्रबंधन एवं अन्य स्वास्थ्य सेवा संग
```

For **Hindi** Corpus Word Tokenization:

- The white spaces are ignored and marked as seaprators between words: separators = [' ','\t','\n']
- If a punctuation mark is encountered punctuationSeparators = ['.',',',';', '!','?', '"', '"', '(', ')', '[', ']', '{', '}'], it also acts as a separator
- In case of punctuation mark, I have divided ‘a Punctuation b’ as ‘a’ ‘Punctuation’ ‘b’

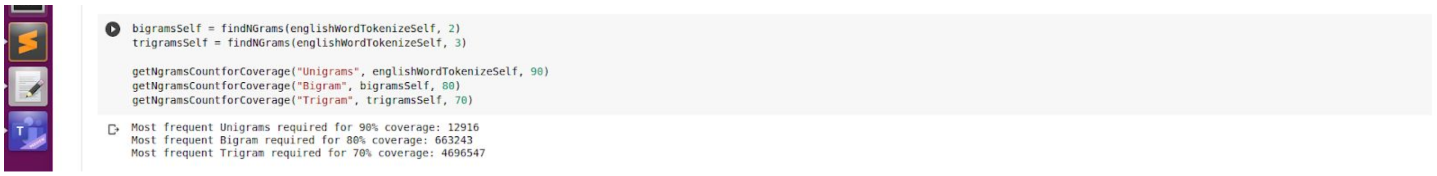


```
Total:  8192881
['मास्टर', 'ऑफ़', 'हेल्थ', 'एडमिनिस्ट्रेशन', 'या', 'मास्टर', 'ऑफ', 'हेल्थकेयर', 'एडमिनिस्ट्रेशन', '(', 'एमएचए', 'या', 'एम', '.', 'एच', '.', 'ए', ')', 'स्नातकोत्तर', '(', 'पोस्ट', 'ग्रेजुएशन', ')', 'की', 'एक', 'पेशेवर', 'डिग्री', 'है', 'जो', 'स्वास्थ्य
```

Finding percentage coverage as done in section 1.3.2 for self implementation of

A total of **8192881** tokens were generated using above heuristics compared to **8640033** in indic-nlp, **8640033** in Stanford Stanza (last one was on 10% corpus). One difference can be seen in tokenization of एम. एच. ए
Stanford stanza classifies this as 1 token ‘एम. एच. ए’ while my heuristic classifies it as 5 tokens ‘एम’ ‘.’ ‘एच’ ‘.’ ‘ए’

Since question has asked to repeat tasks of section 1.3.2 for self tokenization, results are as follows:
Using self tokenization in **English**(entire corpus), results for percentage coverage are:
Most frequent Unigrams required for 90% coverage: **12916**
Most frequent Bigrams required for 80% coverage: **663243**
Most frequent Trigrams required for 70% coverage: **4696547**



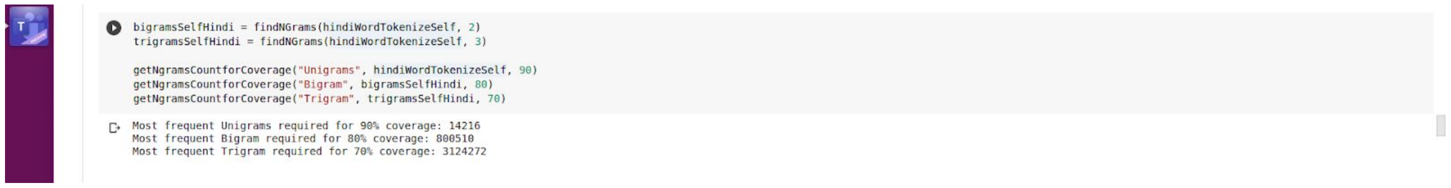
```
bigramsSelf = findNGrams(englishWordTokenizeSelf, 2)
trigramsSelf = findNGrams(englishWordTokenizeSelf, 3)

getNgramsCountforCoverage("Unigrams", englishWordTokenizeSelf, 90)
getNgramsCountforCoverage("Bigram", bigramsSelf, 80)
getNgramsCountforCoverage("Trigram", trigramsSelf, 70)
```

```
Most frequent Unigrams required for 90% coverage: 12916
Most frequent Bigram required for 80% coverage: 663243
Most frequent Trigram required for 70% coverage: 4696547
```

Using self tokenization in **Hindi**(entire corpus), results for percentage coverage are:
Most frequent Unigrams required for 90% coverage: **14216**
Most frequent Bigrams required for 80% coverage: **800510**

Most frequent Trigrams required for 70% coverage: **3124272**



```
bigramsSelfHindi = findNGrams(hindiWordTokenizeSelf, 2)
trigramsSelfHindi = findNGrams(hindiWordTokenizeSelf, 3)

getNgramsCountforCoverage("Unigrams", hindiWordTokenizeSelf, 90)
getNgramsCountforCoverage("Bigram", bigramsSelfHindi, 80)
getNgramsCountforCoverage("Trigram", trigramsSelfHindi, 70)
```

```
Most frequent Unigrams required for 90% coverage: 14216
Most frequent Bigram required for 80% coverage: 800510
Most frequent Trigram required for 70% coverage: 3124272
```

Compared with values that were obtained using tools in section 1.3.2, the values are close and differ only by +- 10% due to difference in implementation of the library and my heuristics.

Question 2) **Likelihood Ratio test** for finding Bigram Collocation
Algorithm and Implementation Details:

In applying the likelihood ratio test to collocation discovery, the following two alternative explanations for the occurrence frequency of a bigram collocation w1w2 are examined:
Hypothesis 1. P (w2 | w1) = p = P (w2 | ~w1)
Hypothesis 2. P (w2 | w1) = p1 != p2 = P (w2 | ~w1)
Taking c1, c2 and c12 as number of occurrences of w1, w2 and w1w2 and N as total tokens in corpus
p = c2/N (probability of w2 in corpus irrespective of position of w1)
p1 = c12/c1 (probability of w1 followed by w2 given presence of w1)
p2 = (c2-c12)/N (probability of w2 without being preceded by w1)

log(Lambda)
= log L(H1) / L(H2)
= log [(b(c12; c1; p) * b(c2-c12; N-c1; p))/ (b(c12; c1; p1) * b(c2-c12; N-c1; p2))]
= log L(c12; c1; p) + log L(c2-c12; N c1; p) + log L(c12; c1; p1) + log L(c2-c12; N-c1; p2)
where L(k; n; x) = (x^k) * ((1-x)^(n-k)) and b is the standard binomial distribution

If Lambda is a likelihood ratio of a particular form, then the quantity -2* log(Lambda) is asymptotically (Psi)^2 distributed. The critical value (for one degree of freedom) is approximately 7.88
Hence after implementing the above formula if -2*(log(Lambda)) >= 7.88 the bigram can be classified as collocation.

For English bigrams, I additionally remove those bigrams which contain separators like punctuation marks to get more accurate collocations.

A few top collocations found for **English** Corpus are:

```
('of', 'the')
('United', 'States')
('in', 'the')
('median', 'income')
('such', 'as')
('the', 'the')
('65', 'years')
('there', 'were')
('every', '100')
```

```
('100', 'females')
('For', 'every')
('or', 'older')
('income', 'for')
('square', 'mile')
('There', 'were')
('to', 'be')
('can', 'be')
('racial', 'makeup')
('housing', 'units')
('married', 'couples')
```

A few top collocations found for **Hindi** Corpus are:

```
('है', '।')
('के', 'लिए')
('हैं', '।')
('जाता', 'है')
('था', '।')
('के', 'साथ')
('किया', 'गया')
('होता', 'है')
('रूप', 'में')
('के', 'रूप')
('जा', 'सकता')
('है', 'कि')
('सकता', 'है')
('करता', 'है')
('गया', 'था')
('रूप', 'से')
('थे', '।')
('होते', 'हैं')
('है', ',')
('करते', 'हैं')
('होती', 'है')
('किया', 'जाता')
('के', 'बाद')
('कर', 'दिया')
```

Observations:

- As it can be seen the algorithm is able to very well determine collocations like (United States), (square mile), (housing units).
- It also categorises some of the tokens that often exist together as collocation specifically in Hindi corpus like `('of', 'the'), ('such', 'as') ('है', 'कि') ('सकता', 'है')('करता', 'है')`

Question 1.3.4 Morphological Parsing

Five words were randomly sampled from the least frequent and most frequent 100 tokens in both English and Hindi Corpora.

For Morphological Analysis on **English** Corpora, polyglot tool is used which generated the following Morphemes for the randomly sampled words:

```
5 random words of 100 most frequently occuring tokens
under : ['under'] #under is a verb
income : ['in', 'come'] #income is broken into in (preposition) + come(verb)
living : ['liv', 'ing'] # living is broken into liv(verb) + ing(inflicted form)
be : ['be'] # be is in its simplest form
households :   # house(Noun) + hold(verb) + s(plural)


5 random words of 100 least frequently occuring tokens
AlC : ['Al', 'C'] #Al (Proper oun) + C(Proper Noun)
fugacity : ['fu', 'ga', 'city'] #fu (prefix) + ga + city(Noun, singular)
bayerite : ['bay', 'er', 'ite'] # bay(noun) + er(suffix) + ite(suffix)
metalline : ['metal', 'line'] # metal(noun) + line(noun)
hyperpolarization : ['hy', 'per', 'polar', 'ization'] # hy(prefix) + per(prefix) +
polar(noun) + ization(suffix)
```

About Polyglot

- **Polyglot** uses trained morfessor model (Morpho project) to generate morphemes from words.
- The goal of the Morpho project is to develop unsupervised data-driven methods that discover the regularities behind word forming in natural languages.
- Morpho project is focussing on the discovery of morphemes. Morphemes are important in automatic generation and recognition of a language, especially in languages in which words may have many different inflected forms.
- Raw training data is fed into the unsupervised learning model to generate a vocabulary of morphs. Then, these morphemes are used to find the optimal word segmentation based on the “Minimum Description Length” principle.
- Note that such an analyzer is primarily restricted to concatenative morphology.

For Morphological Analysis on **Hindi** Corpora, unsupervised_morph tool from indic-nlp is used which generated the following Morphemes for the randomly sampled words:

```
5 random words of 100 most frequently occuring tokens
कुछ :  ['कुछ']
करता :  ['कर', 'ता']
और :  ['और']
कारण :  ['कारण']
सबसे :  ['सबसे']


5 random words of 100 least frequently occuring tokens
सूत्रसाहित्य :  ['सूत्र', 'साहित्य']
जकीउर :  ['ज', 'की', 'उर']
उपलब्धा :  ['उपलब्ध', 'ा']
स्त्रीलिंगवाचक :  ['स्त्री', 'लिंग', 'वाचक']
```

```
हैल्थकेयर :  ['हैल्थ', 'केयर']
```

About Unsupervised Morphological analyser

- Unsupervised Morphological analyser is built using Morfessor 2.0 with an Indic language input stream.
- Morfessor 2.0 is applicable to any string segmentation task including morphological segmentation tasks. The task of the algorithm is to find a set of constructions that describe the provided training corpus efficiently and accurately.
- The training corpus contains a collection of compounds, which are the largest sequences that a single construction can hold. The smallest pieces of constructions and compounds are called atoms (in our case morphemes).
- The input data makes a large difference as using Polyglot on Hindi words generates individual characters as morphemes.

Question 1.3.5 Sub Word Tokenization

About algorithm and implementation:
The reference for the algorithm is taken from Jurafsky and Martin's book [SLP 3rd Edition] Section 2.3.4 which was to be studied in lecture 4.

- Initially each character in the words of the corpus is separate with a </w> character at end of each word
- For a fixed number of iterations the following process is repeated:
  - Get the two consecutive pairs which have the maximum frequency as pairs among all consecutive pairs.
  - Insert this in a list used later to find tokenization of unknown words. And also use this pair to merge the two consecutive characters in the current vocabulary
- Once this process is done, we have the vocabulary separated as sub tokens along with the pairs used for this process.
- For tokenizing any new word, separate it as single characters and then we can run it through all pairs in the order they were generated and keep merging two consecutive sub tokens as they occur in pairs list.

For **English** Corporus (Approximate 1% of corporus is used for training due to long training steps of the algorithm. A total of 28000 iterations of finding the maxim pair and merging according to the pair were run):
</w> indicates end of word

```
50 most frequent tokens after BPE in the corpus

the</w>: 9501, of</w>: 6136, and</w>: 4539, in</w>: 3875, to</w>: 3551, a</w>: 3213,
is</w>: 1983, as</w>: 1637, that</w>: 1384, was</w>: 1299, The</w>: 1190, with</w>:
1173, by</w>: 1148, for</w>: 1103, his</w>: 988, on</w>: 877, are</w>: 868, an</w>:
861, from</w>: 792, be</w>: 762, or</w>: 684, he</w>: 680, In</w>: 662, which</w>:
642, at</w>: 617, it</w>: 493, have</w>: 468, not</w>: 450, ": 421, were</w>: 410,
has</w>: 387, also</w>: 375, but</w>: 357, had</w>: 342, first</w>: 341, can</w>:
```

```
341, "The</w>: 331, this</w>: 324, her</w>: 317, such</w>: 302, one</w>: 300, (: 283,
their</w>: 282, been</w>: 282, other</w>: 276, its</w>: 274, more</w>: 259, He</w>:
257, who</w>: 256, they</w>: 254,

50 least frequent tokens after BPE in the corpus
philosophers.</w>: 1, experimentation.</w>: 1, convince</w>: 1, everybody,</w>: 1,
atomism</w>: 1, scientists,</w>: 1, blossoming</w>: 1, discoveries</w>: 1,
explain.</w>: 1, 1800s,</w>: 1, proportions).</w>: 1, oxide:</w>: 1, 88.1%</w>: 1,
11.9%</w>: 1, 78.7%</w>: 1, 21.3%</w>: 1, (tin(II)</w>: 1, 100g</w>: 1, 13.5g</w>: 1,
27g</w>: 1, oxygen.</w>: 1, 13.5</w>: 1, 1:2,</w>: 1, units—in</w>: 1,
proportions.</w>: 1, hypothesized</w>: 1, gases'</w>: 1, (CO)</w>: 1, (N).</w>: 1,
1827,</w>: 1, floating</w>: 1, erratically,</w>: 1, "Brownian</w>: 1, motion".</w>:
1, knocking</w>: 1, motions</w>: 1, Statistical</w>: 1, Brownian</w>: 1, Perrin</w>:
1, experimentally</w>: 1, verifying</w>: 1, Dalton's</w>: 1, particle,</w>: 1,
"subatomic"</w>: 1, ""corpuscle""</w>: 1, "electron",</w>: 1, Johnstone</w>: 1,
Stoney</w>: 1, 1874.</w>: 1, photoelectric</w>: 1,

Tokenization of 10 unknown words:

sibling:          ['sibl', 'ing</w>']
jentacular:       ['j', 'ent', 'acular</w>']
phobia:           ['phob', 'ia</w>']
preprocessing:    ['pre', 'processing</w>']
handpick:         ['hand', 'pick</w>']
restitution:      ['restitution</w>']
lamprophony:      ['lam', 'proph', 'ony</w>']
dutiful:          ['du', 'ti', 'ful</w>']
prejudicing:      ['prejud', 'ic', 'ing</w>']
unethically:      ['un', 'eth', 'ically</w>']
```

Tokenization of 10 words used during morphological analysis:

```
BPE on most frequent words used during morphological analysis
under :  ['under</w>']
income :  ['income</w>']
living :  ['living</w>']
be :  ['be</w>']
households :  ['house', 'holds</w>']

BPE on least frequent words used during morphological analysis
AlC :  ['Al', 'C</w>']
fugacity :  ['fugacity</w>']
bayerite :  ['bay', 'er', 'ite</w>']
metalline :  ['metalline</w>']
hyperpolarization :  ['hyper', 'polari', 'zation</w>']
```

It can be seen BPE performs very well in sub tokenizing words into morphemes like hyperpolarization, households, and bayerite. There are some differences compared to morphological analysis using polyglot in section 1.3.4 as we see in hyperpolarization and households

Morphemes: ['hy', 'per', 'polar', 'ization'] and ['house', 'hold', 's']

Using BPE : ['hyper', 'polari', 'zation</w>'] and ['house', 'holds</w>']

The result can vary based on the number of iterations the BPE is run.

For **Hindi** Corporus (Approximate 5% of corporus is used for training due to long training steps of the algorithm. A total of 28000 iterations of finding the maxim pair and merging according to the pair were run):
</w> indicates end of word

```
50 most frequent tokens after BPE in the corpus

के</w>: 8853, में</w>: 6113, की</w>: 4015, और</w>: 3755, से</w>: 3331, है।</w>: 3122,
को</w>: 2973, का</w>: 2963, एक</w>: 1964, है</w>: 1923, पर</w>: 1592, भी</w>: 1279,
लिए</w>: 1265, हैं।</w>: 1234, ने</w>: 1191, कि</w>: 1099, यह</w>: 994, किया</w>: 948,
रूप</w>: 860, है,</w>: 837, इस</w>: 823, जो</w>: 789, करने</w>: 767, जाता</w>: 742,
ही</w>: 735, कर</w>: 735, या</w>: 722, नहीं</w>: 681, हो</w>: 647, साथ</w>: 588, हैं</w>:
584, गया</w>: 570, द्वारा</w>: 564, तथा</w>: 546, अपने</w>: 534, था।</w>: 455, होता</w>:
442, तक</w>: 410, वह</w>: 406, कुछ</w>: 401, (: 390, तो</w>: 379, जा</w>: 377,
करते</w>: 373, सकता</w>: 355, नाम</w>: 351, हैं,</w>: 350, में,</w>: 343, वे</w>: 338,
बाद</w>: 333,

50 least frequent tokens after BPE in the corpus

(एमएचए</w>: 1, एम.एच.ए)</w>: 1, स्नातकोत्तर</w>: 1, ग्रेजुएशन)</w>: 1, पाठ्यक्रमो</w>: 1,
सरंचना</w>: 1, व्यवसायी-शिक्षक</w>: 1, कक्षा-आधारित</w>: 1, विद्यार्थियों</w>: 1, अर्थशास्त्र,</w>: 1,
संगठनात्मक</w>: 1, अमेरिकी,</w>: 1, यूरोपीय,</w>: 1, ऑस्ट्रेलियाई,</w>: 1, ग्रेजुएट</w>: 1,
मेडिसिन-</w>: 1, स्थानीय,</w>: 1, गैर-लाभकारी</w>: 1, प्रबंधकीय</w>: 1, स्नातकों</w>: 1,
प्रबंधकों,</w>: 1, योजनाकारों,</w>: 1, वित्तपोषण,</w>: 1, यदयपि</w>: 1, वाणिज्य,</w>: 1,
“कमीशन</w>: 1, ओन</w>: 1, अक्रेदिसन</w>: 1, एजुकेशन”</w>: 1, (सीएएचएमई)</w>: 1,
पूर्वस्नातक</w>: 1, एएसीएसबी</w>: 1, मान्यता,</w>: 1, एचओएम</w>: 1, समवर्ती</w>: 1,
इंटर्नशिप</w>: 1, फेलोशिप</w>: 1, व्यवसाय,</w>: 1, आषधियों</w>: 1, पाठ्शालाओं</w>: 1,
डिग्रीधारीयों</w>: 1, कोर्स</w>: 1, हैल्थकेयर</w>: 1, बढती</w>: 1, समेस्टर</w>: 1, (6</w>: 1,
समेस्टर)</w>: 1, बिभाजित</w>: 1, (एचआर)</w>: 1, सीखाया</w>: 1,
```

Tokenization of 10 unknown words:

```
शरीर:        ['शरीर</w>']
आत्मा:               ['आत्मा</w>']
इच्छा:        ['इच्छा</w>']
सच्चाई:       ['सच्चाई</w>']
ससुर:        ['स', 'सुर</w>']
गायक:        ['गायक</w>']
विचारक:              ['विचारक</w>']
पंजाब:        ['पंजाब</w>']
मदिरा:        ['म', 'दि', 'रा</w>']
विभाजन:              ['विभाजन</w>']
```

Tokenization of 10 words used during morphological analysis:

```
BPE on most frequent words used during morphological analysis Hindi
```

```
कुछ :  ['कुछ</w>']
करता :  ['करता</w>']
और :  ['और</w>']
कारण :  ['कारण</w>']
सबसे :  ['सबसे</w>']


BPE on least frequent words used during morphological analysis Hindi
सूत्रसाहित्य :  ['सूत्र', 'साहित्य</w>']
जकीउर :  ['ज', 'की', 'उ', 'र</w>']
उपलब्धा :  ['उपलब्', 'धा</w>']
स्त्रीलिंगवाचक :  ['स्त्री', 'लिंगवाचक</w>']
हैल्थकेयर :  ['है', 'ल्', 'थकेयर</w>']
```

It can be seen that BPE performs very well in sub tokenizing words like सूत्रसाहित्य, स्त्रीलिंगवाचक. There are some differences compared to morphological analysis using polyglot in section 1.3.4 as we see in स्त्रीलिंगवाचक and करता
Morphemes: ['स्त्री', 'लिंग', 'वाचक'] and ['कर', 'ता']
Using BPE : ['स्त्री', 'लिंगवाचक</w>'] and ['करता</w>']
The result can vary based on the number of iterations the BPE is run. It can also be noticed for more frequent words, it is able to recognize it as full token and does not break it into sub tokens. For large and infrequent words, it is able to find the morphemes with some accuracy.