Lavish Gulati - 170101082

CS565 - Intelligent Systems and Interfaces - Assignment 1

Link to Google Colab Notebook: Click here

1.3.1 - Analysis using existing NLP tools

1 - Sentence Segmentation and Word Tokenization

Sentence Segmentation

English

- ➔ (nltk) Punkt Sentence Tokenizer: This tokenizer divides a text into a list of sentences by using an unsupervised algorithm to build a model for abbreviation words, collocations and words that start sentences.
  Number of sentence tokens: 761582
  Sample Output:

```
['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 views on what atoms look like and how they behave were incorrect.']
```

- ➔ (nltk) Regular-Expression Tokenizer using regex: Here, we perform the sentence tokenization based on a regex expression. Here the regex is constructed in a way such that sentence does not break at Dr. and Mr. and hence provides for better tokenization. The regex used is given below:
  `'(?<!\w\.\w.)(?<![A-Z][a-z]\.)(?<=\.|\?)\s'`
  Number of sentence tokens: 750330
  Sample Output:

```
['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 views on what atoms look like and how they behave were incorrect.']
```

Hindi

- ➔ (indicnlp) Sentence Tokenizer: A rule-based sentence splitter for Indian languages written in Brahmi-derived scripts. The text is split at sentence delimiter boundaries. The delimiters can be configured by passing appropriate parameters. The sentence splitter can identify non-breaking phrases like single letter, common abbreviations/honorifics for some Indian languages.
  Number of sentence tokens: 348593
  Sample Output:

```
['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 संगठनों के क्षेत्र में जरूरी ज्ञान और दक्षता हासिल की हैं।', 'इन पाठ्यक्रमो में परिस्थितियों के अनुसार इनके सरंचना में अंतर हो
```

```
सकता हैं हालांकि व्यवसायी-शिक्षक मॉडल कार्यक्रम आमतौर पर चिकित्सा, स्वास्थ्य व्यवसायों या संबद्ध स्वास्थ्य के कॉलेजों में पाए जाते हैं, कक्षा-आधारित कार्यक्रम व्यवसाय या सार्वजनिक स्वास्थ्य के कॉलेजों में होते हैं।']
```

➔ (stanza) Sentence Tokenizer: Tokenization and sentence segmentation in Stanza are jointly performed by the TokenizeProcessor. This processor splits the raw input text into tokens and sentences, so that downstream annotation can happen at the sentence level. Segments a Document into Sentences, each containing a list of Tokens. This processor also predicts which tokens are multi-word tokens, but leaves expanding them to the MWTProcessor.

NOTE: Taking 10% data due to RAM issues.

Number of sentence tokens (10% data): 36436

Number of sentence tokens (approx.) = 364360

Sample Output:

```
['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 संगठनों के क्षेत्र में जरूरी ज्ञान और दक्षता हासिल की हैं।', 'इन पाठ्यक्रमो में परिस्थितियों के अनुसार इनके सरंचना में अंतर हो सकता हैं']
```

Word Tokenization

English

➔ (nltk) Treebank Word Tokenizer: The Treebank tokenizer uses regular expressions to tokenize text. It assumes that text has already been split into sentences. It splits common English contractions eg. ‘don't’ is tokenized into ‘do’, ‘n't’. It handles punctuation characters as separate tokens and splits commas and single quotes off from words, when they are followed by whitespace. It also splits off periods that occur at the end of the sentence.

Number of word tokens: 18915300

Sample Output:

```
['The', 'word', '``', 'atom', "''", 'was', 'coined', 'by', 'ancient', 'Greek',
'philosophers.', 'However', ',', 'these', 'ideas', 'were', 'founded', 'in',
'philosophical', 'and']
```

➔ (nltk) Punct Word Tokenizer: With this method we are able to extract the tokens from a string of words or sentences in the form of Alphabetic or Non-Alphabetic character. It is based on a simple regex tokenization. It used the regular expression `'\w+|[^\w\s]+'` to split the input.

Number of word tokens: 20372526

Sample Output:

```
['The', 'word', '"', 'atom', '"', 'was', 'coined', 'by', 'ancient', 'Greek',
'philosophers', '.', 'However', ',', 'these', 'ideas', 'were', 'founded', 'in',
'philosophical']
```

Hindi

- ➔ (indicnlp) Word Tokenizer: A trivial tokenizer which just tokenizes on the punctuation boundaries. This also includes punctuations for the Indian language scripts (the purna virama and the deergha virama). It returns a list of tokens.
  Number of word tokens: 8640033
  Sample Output:

```
['मास्टर', 'ऑफ़', 'हेल्थ', 'एडमिनिस्ट्रेशन', 'या', 'मास्टर', 'ऑफ', 'हेल्थकेयर', 'एडमिनिस्ट्रेशन',
'(', 'एमएचए', 'या', 'एम', '.', 'एच', '.', 'ए', ')', 'स्नातकोतर', '(']
```

- ➔ (stanza) Word Tokenizer: The TokenizeProcessor is usually the first processor used in the pipeline. It performs tokenization and sentence segmentation at the same time. After this processor is run, the input document will become a list of Sentences. Each Sentence contains a list of Tokens, which can be accessed with the property tokens.
  NOTE: Taking 10% data due to RAM issues.
  Number of word tokens (10% data): 840150
  Number of word tokens (approx.) = 8401500
  Sample Output:

```
['मास्टर', 'ऑफ़', 'हेल्थ', 'एडमिनिस्ट्रेशन', 'या', 'मास्टर', 'ऑफ', 'हेल्थकेयर', 'एडमिनिस्ट्रेशन',
'(', 'एमएचए', 'या', 'एम.एच.ए', ')', 'स्नातकोतर', '(', 'पोस्ट', 'ग्रेजुएशन', ')', 'की']
```

Analysis

Sentence Segmentation

- ➔ **English:** The Regular-Expression based Tokenizer uses a simple heuristic to split the given text into sentences whereas the Punkt Sentence Tokenizer uses a trained unsupervised model (trained on tokenizers/punkt/english.pickle), hence giving more sentence split than the former one.
- ➔ **Hindi:** The reasoning for Hindi is similar to English, since the stanza sentence tokenizer uses a neural-based model to split sentences (thus, also resulting in RAM issues), whereas the indicnlp uses a rule-based model. Hence, we get more sentence splits with the former one.

---

2 - Frequency Distribution of Unigrams

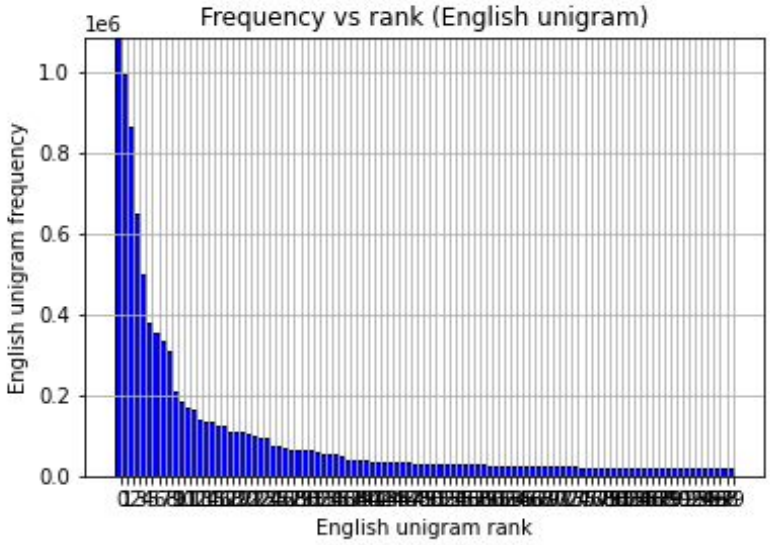
English

Number of word tokens as input: 20372526

Number of unigrams obtained: 20372526

Sample Output:

```
['The', 'word', '"', 'atom', '"', 'was', 'coined', 'by', 'ancient', 'Greek',
'philosophers', '.', 'However', ',', 'these', 'ideas', 'were', 'founded', 'in',
'philosophical']
```



Type of unigrams: 286686

Sample Output:

```
[('the', 1084635), (',', 994858), ('.', 862133), ('of', 646923), ('and', 499999),
('in', 379124), ('to', 356109), ('a', 334452), ('"', 308618), ('was', 210438)]
```

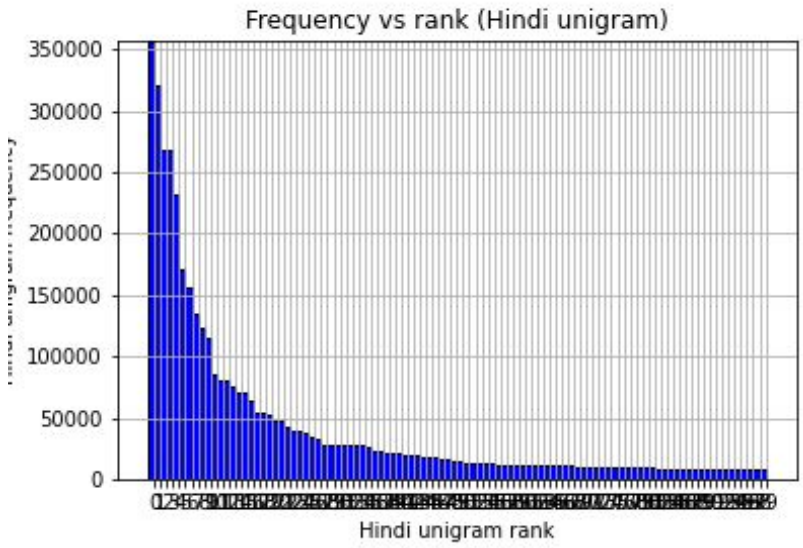
Hindi

Number of word tokens as input: 8640033

Number of unigrams obtained: 8640033

Sample Output:

```
['मास्टर', 'ऑफ़', 'हेल्थ', 'एडमिनिस्ट्रेशन', 'या', 'मास्टर', 'ऑफ', 'हेल्थकेयर', 'एडमिनिस्ट्रेशन',
'(', 'एमएचए', 'या', 'एम', '.', 'एच', '.', 'ए', ')', 'स्नातकोत्तर', '(']
```



Type of unigrams: 322350

Sample Output:

```
[('के', 357833), ('।', 321526), ('में', 268249), (',', 267743), ('है', 232156), ('की', 171779), ('और', 155743), ('से', 134901), ('का', 122948), ('को', 115876)]
```

---

3 - Frequency Distribution of Bigrams

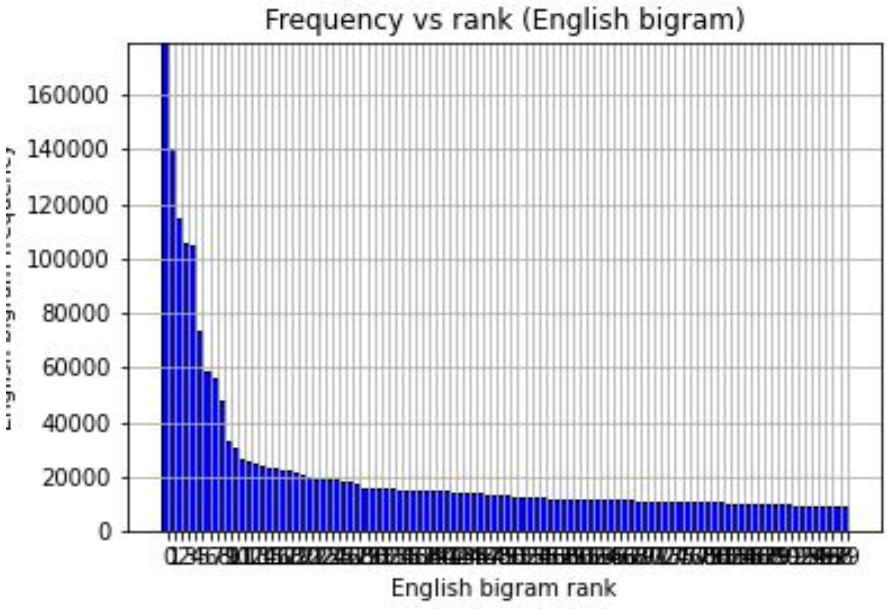
English

Number of word tokens as input: 20372526

Number of bigrams obtained: 20372525

Sample Output:

```
['The word', 'word "', '" atom', 'atom "', '" was', 'was coined', 'coined by', 'by ancient', 'ancient Greek', 'Greek philosophers']
```



Type of bigrams: 3977129

Sample Output:

```
[('of the', 179431), ('. The', 140543), (', and', 114438), ("' s", 105543), ('in the', 104836), (', the', 73479), ('to the', 58240), ('. In', 56002), ('and the', 47484), ('on the', 32972)]
```

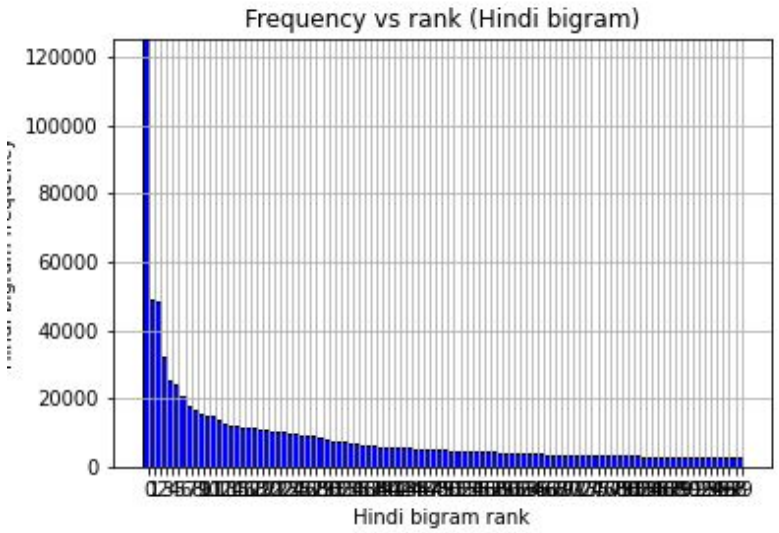
Hindi

Number of word tokens as input: 8640033

Number of bigrams obtained: 8640032

Sample Output:

```
['मास्टर ऑफ़', 'ऑफ़ हेल्थ', 'हेल्थ एडमिनिस्ट्रेशन', 'एडमिनिस्ट्रेशन या', 'या मास्टर', 'मास्टर ऑफ', 'ऑफ हेल्थकेयर', 'हेल्थकेयर एडमिनिस्ट्रेशन', 'एडमिनिस्ट्रेशन (', '( एमएचए']
```



Type of bigrams: 2392979

Sample Output:

```
[('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)]
```

---

4 - Frequency Distribution of Trigrams

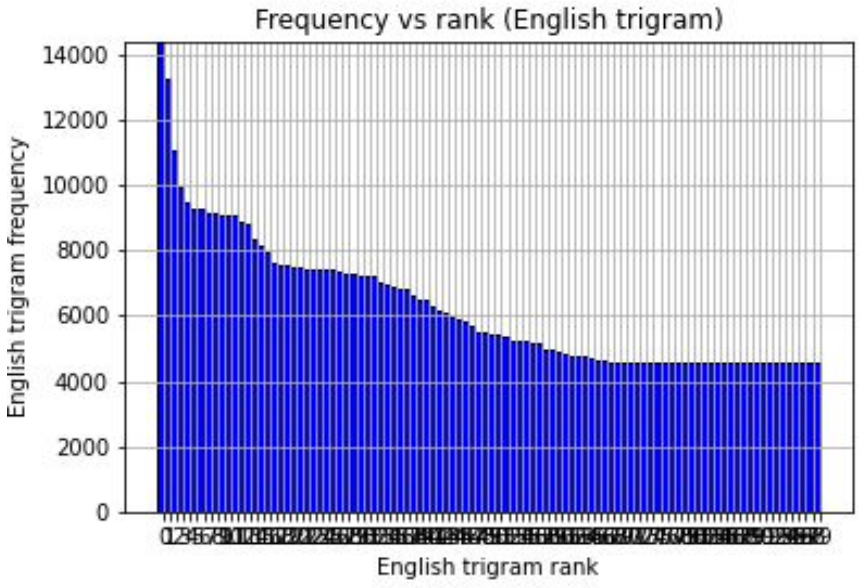
English

Number of word tokens as input: 20372526

Number of trigrams obtained: 20372524

Sample Output:

```
['The word "', 'word " atom', '" atom "', 'atom " was', '" was coined', 'was coined
by', 'coined by ancient', 'by ancient Greek', 'ancient Greek philosophers', 'Greek
philosophers .']
```



Type of trigrams: 10878682

Sample Output:

```
[(', and the', 14397), ('. In the', 13244), ('the age of', 11064), ('under the age', 9928), ('the United States', 9467)]
```

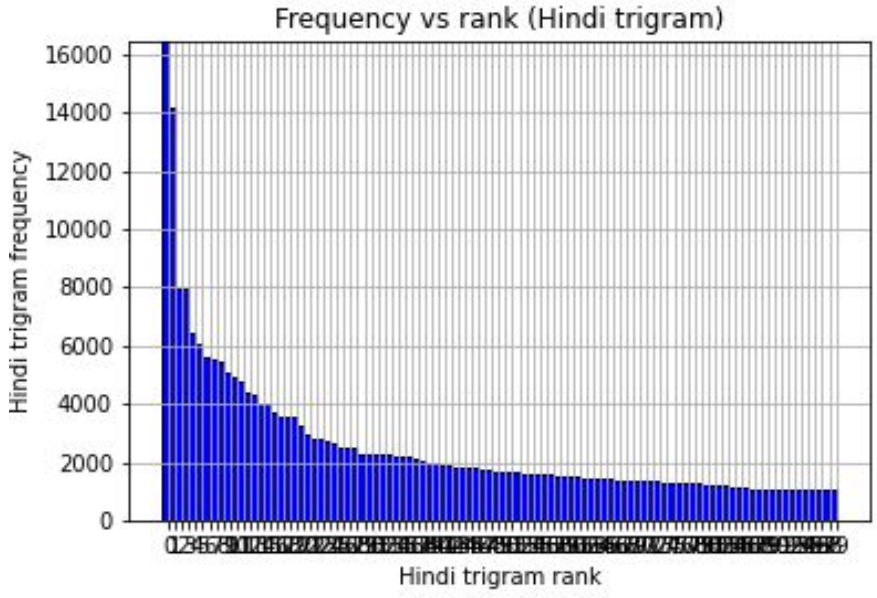
Hindi

Number of word tokens as input: 20372526

Number of trigrams obtained: 8640031

Sample Output:

```
['मास्टर ऑफ़ हेल्थ', 'ऑफ़ हेल्थ एडमिनिस्ट्रेशन', 'हेल्थ एडमिनिस्ट्रेशन या', 'एडमिनिस्ट्रेशन या मास्टर', 'या मास्टर ऑफ', 'मास्टर ऑफ हेल्थकेयर', 'ऑफ हेल्थकेयर एडमिनिस्ट्रेशन', 'हेल्थकेयर एडमिनिस्ट्रेशन (', 'एडमिनिस्ट्रेशन ( एमएचए', '( एमएचए या']
```



Type of trigrams: 5508737

Sample Output:

```
[('के रूप में', 16475), ('जाता है ।', 14139), ('करने के लिए', 7987), ('होता है ।', 7927), ('किया जाता है', 6457)]
```

---

5 - Analysis

According to Zipf's Law, the relationship between frequency f of word type and its rank r based on frequency is given as:

$$f \propto \frac{1}{r}$$

This resembles a hyperbola in geometry with an equation:

$$f.r = k \qquad \text{where k is a constant}$$

The frequency vs rank graphs obtained by the frequency distribution of the ngrams also resemble a hyperbola closely. The resemblance is more profound in case of unigrams, and skewed in case of trigrams. This is because

trigrams enhance the effect of uniqueness because several words are joined together (e.g. three words such as ‘the’, ‘of’ and ‘an’ can form six types of trigrams). Thus, the frequency remains constant for specific ranges of rank forming groups, as shown in the graph of English trigrams.
It is shown that for a large corpus, Zipf 's law for both words in English and characters in Chinese does not hold for all ranks. The frequency falls below the frequency predicted by Zipf's law for English words for rank greater than about 5,000 and for Chinese characters for rank greater than about 1,000.

1.3.2 - Few Basic Questions

1 - 90% Coverage By Unigrams

English
Type of unigrams as input: 286686
Type of unigrams required for 90% coverage: 11224

Hindi
Type of unigrams as input: 322350
Type of unigrams required for 90% coverage: 13339

---

2 - 80% Coverage By Bigrams

English
Type of bigrams as input: 3977129
Type of bigrams required for 80% coverage: 1939876

Hindi
Type of bigrams as input: 2392979
Type of bigrams required for 80% coverage: 1528975

---

3 - 70% Coverage By Trigrams

English
Type of trigrams as input: 10878682
Type of trigrams required for 70% coverage: 8841429

Hindi
Type of trigrams as input: 5508737
Type of trigrams required for 70% coverage: 4644733

---

4 - Coverage After Stemming

Stemming: It is basically the process of producing morphological variants of a root/base word.

➔ English [using (nltk) Porter Stemmer]: It is based on the idea that the suffixes in the English language are made up of a combination of smaller and simpler suffixes. This stemmer is known for its speed and simplicity. The main applications of Porter Stemmer include data mining and Information retrieval. However, its applications are only limited to English words. Also, the group of stems is mapped on to the same stem and the output stem is not necessarily a meaningful word. The algorithms are fairly lengthy in nature and are known to be the oldest stemmer.

Number of word tokens as input: 20372526

Sample Output:

```
['the', 'word', '"', 'atom', '"', 'wa', 'coin', 'by', 'ancient', 'greek', 'philosoph',
'.', 'howev', ',', 'these', 'idea', 'were', 'found', 'in', 'philosoph']
```

➔ Hindi [Using heuristics-based stemmer]: We use a heuristics-based stemmer. The full implementation is given in the Google Colab notebook. Here we define some common suffixes and try to remove that suffix from the words present in the Hindi corpora. As Hindi language follows more of a general rule in terms of suffix as compared to English, hence this way of stemming gives us desired results.

```
def generate_hin_stem_words(word):
 suffixes = {
   1: [u"ो", u"े", u"ू", u"ु", u"ी", u"ि", u"ा"],
   2: [u"कर", u"ाओ", u"िए", u"ाई", u"ाए", u"ने", u"नी", u"ना", u"ते", u"ीं", u"ती", u"ता",
u"ाँ", u"ां", u"ों", u"ें"],
   3: [u"ाकर", u"ाइए", u"ाईं", u"ाया", u"ेगी", u"ेगा", u"ोगी", u"ोगे", u"ाने", u"ाना",
u"ाते", u"ाती", u"ाता", u"तीं", u"ाओं", u"ाएं", u"ुओं", u"ुएं", u"ुआं"],
   4: [u"ाएगी", u"ाएगा", u"ाओगी", u"ाओगे", u"एंगी", u"ेंगी", u"एंगे", u"ेंगे", u"ूंगी", u"ूंगा",
u"ातीं", u"नाओं", u"नाएं", u"ताओं", u"ताएं", u"ियाँ", u"ियों", u"ियां"],
   5: [u"ाएंगी", u"ाएंगे", u"ाऊंगी", u"ाऊंगा", u"ाइयाँ", u"ाइयों", u"ाइयां"],
 }

  for L in 5, 4, 3, 2, 1:
   if len(word) > L + 1:
     for suf in suffixes[L]:
       if word.endswith(suf):
         return word[:-L]
 return word
```

Number of word tokens as input: 8640033

Sample Output:

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

Unigrams

➔ **English**

Type of unigrams obtained: 200158

Type of unigrams required for 90% coverage: 4499

➔ **Hindi**

Type of unigrams obtained: 279722

Type of unigrams required for 90% coverage: 8033

Bigrams

➔ **English**

Type of bigrams obtained: 2117887

Type of bigrams required for 80% coverage: 1146847

➔ **Hindi**

Type of bigrams obtained: 3184100

Type of bigrams required for 80% coverage: 1253883

Trigrams

➔ **English**

Type of trigrams obtained: 10145300

Type of trigrams required for 70% coverage: 8108047

➔ **Hindi**

Type of trigrams obtained: 5242642

Type of trigrams required for 70% coverage: 4378638

---

5 - Analysis

English

	Coverage after using tool		Coverage after stemming	
ngram	**Reqd. / Total**	**% of required ngrams**	**Reqd. / Total**	**% of required ngrams**
unigram (90%)	11224 / 286686	3.915	4499 / 200158	2.248
bigram (80%)	1939876 / 3977129	48.776	1146847 / 2117887	54.150
trigram (70%)	8841429 / 10878682	81.273	8108047 / 10145300	79.919

Hindi

	Coverage after using tool		Coverage after stemming	
ngram	**Reqd. / Total**	**% of required ngrams**	**Reqd. / Total**	**% of required ngrams**
unigram (90%)	13339 / 322350	4.138	8033 / 279722	2.872
bigram (80%)	1528975 / 2392979	63.894	1253883 / 3184100	39.379

trigram (70%)	4644733 / 5508737	84.316	4378638 / 5242642	83.520

Observations

As observed by the two tables, the percentage of required ngrams for the coverage of the respective corporus is generally lesser after stemming. This is because during stemming, all the words are converted to their root words (after removing the affixes), thus increasing the frequency of the root words and removing derived words from the corpus. Hence, the frequency of more common words increases (like ‘coding’, ‘coded’, ‘code’; all change to ‘code’) whereas the frequency of less common words such as proper nouns like places (‘Delhi’) and names (‘John’) remain the same. Thus, a lesser percentage of the ngrams are required for the coverage.

1.3.3 - Writing some of your basic codes and comparing with results obtained using tools

1 - Heuristics For Sentence Segmentation And Word Tokenization

English

The heuristics is based on a more complex regular expression than the one used in 1.3.1.1. It handles the following cases explained in the regex. After the split on regex, all the words are converted to lower case to provide a more uniform tokenized segmentation.

```
pattern = r'''(?x)          # set flag to allow verbose regexps
 (?:[A-Z]\.)+        # abbreviations, e.g. U.S.A.
 | (?:\s\w\w\.)+         # Dr. Mr. Ms.
 | \w+(?:-\w+)*        # words with optional internal hyphens
 | \$?\d+(?:\.\d+)?%?  # currency and percentages, e.g. $12.40, 82%
 | \.\.\.              # ellipsis
 | [][.,;"'?():_`-]    # these are separate tokens; includes ], [
 '''
```

ngram	Coverage after using tool	Coverage after stemming	Coverage after heuristics
unigram (90%)	11224	4499	9296
bigram (80%)	1939876	1146847	1681281
trigram (70%)	8841429	8108047	8385969

As we can observe from the table, the proposed heuristics is better than the tool (essentially due to the complex regular expression and conversion to lowercase), but worse than stemmed tokenization because during stemming, all the words are converted to their root words (after removing the affixes), thus increasing the frequency of the root words and removing derived words from the corpus. This results in less type of ngrams, and thus, less are required for the coverage of the corpus.

Hindi

The heuristics is based on a simple notion that in Hindi language, a sentence always ends in a poorna viram ("।").

```
text.split(u"।")
```

ngram	Coverage after using tool	Coverage after stemming	Coverage after heuristics
unigram (90%)	13339	8033	13860
bigram (80%)	1528975	1253883	1591769
trigram (70%)	4644733	4378638	4704830

Since the heuristics applied was the most basic one, the coverage is also worse as compared to this-party tool and after stemming. This can be observed easily from the table. This is because complex words are formed as tokens (e.g. 'है?') which creates a sense of false differentiation between other forms (such as 'है'), thus resulting in more ngrams requirement for coverage of the corpus.

---

2 - Likelihood Ratio Test

Implementation

```
from math import log10

def getVal(k, n, x):
 temp = log10(x) * k
 temp2 = log10(1-x) * (n-k)
 temp += temp2
 return temp

def constructCollocations(bigram_dist, unigram_dist, number_tokens):
   collocation = []
   i = 0
   for bigram, freq in bigram_dist.items():
     bigramSplit = bigram.split()
     c12 = freq
     c1 = unigram_dist[bigramSplit[0]]
     c2 = unigram_dist[bigramSplit[1]]
     n = number_tokens
     p = c2/n
     p1 = c12/c1
     p2 = (c2 - c12)/(n-c1)
     if(p2 == 0 or p1==0 or p==0):
         continue

     if(p2 == 1 or p1==1 or p==1):
         continue

     val = getVal(c12, c1, p) + getVal(c2 - c12, n-c1, p) - getVal(c12, c1, p1) - getVal(c2
- c12, n-c1, p2)
     val *= -2
```

```
    if(val >= 7.88):
        collocation.append(bigram)

  return collocation
```

English

Type of bigrams as input: 3977129

Type of unigrams as input: 286686

Number of unigrams as input: 20372526

Type of collocations obtained: 491686

Sample Output:

```
['The word', 'word "', '" was', 'was coined', 'coined by', 'ancient Greek', 'Greek philosophers', '. However', 'However ,', ', these', 'these ideas', 'ideas were', 'were founded', 'founded in', 'philosophical and', 'and theological', 'rather than', 'and experimentation', '. As', 'As a']
```

Hindi

Type of bigrams as input: 2392979

Type of unigrams as input: 322350

Number of unigrams as input: 8640033

Type of collocations obtained: 262625

Sample Output:

```
['मास्टर ऑफ़', 'ऑफ़ हेल्थ', 'हेल्थ एडमिनिस्ट्रेशन', 'मास्टर ऑफ', 'हेल्थकेयर एडमिनिस्ट्रेशन', 'एडमिनिस्ट्रेशन (', 'एम .', '. एच', 'एच .', '. ए', 'स्नातकोत्तर (', 'पोस्ट ग्रेजुएशन', ') की', 'की एक', 'एक पेशेवर', 'है जो', 'स्वास्थ्य प्रशासन', 'प्रशासन के', 'के क्षेत्र', 'क्षेत्र में']
```

Analysis

Roughly 11% of bigram collocations were found by the Likelihood Ratio Test for both English and Hindi corpora (12.363% for English and 10.975% for Hindi). Some interesting collocations were found in Hindi language which often go together in general text like 'हेल्थकेयर एडमिनिस्ट्रेशन', 'पोस्ट ग्रेजुएशन' and 'स्वास्थ्य प्रशासन', and similarly in English like 'Greek philosophers'.

Likelihood ratios are one of the approaches to hypothesis testing. This test is generally more appropriate for sparse data. Also this test is more interpretable as compared to other hypothesis tests as this simply returns a number.

1.3.4 - Morphological parsing

1 - English

Polyglot offers trained morfessor models 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.

In the so-called Morfessor Baseline method we use raw text as training data in order to learn a model, i.e., a vocabulary of morphs. We construct a morph vocabulary, or a lexicon of morphs, so that it is possible to form any word in the data by the concatenation of some morphs. Each word in the data is rewritten as a sequence of morph pointers, which point to entries in the lexicon. This model is inspired by the Minimum Description Length (MDL) principle.

Type of unigrams as input: 286686

Most frequent words

Random words chosen:

```
['can', 'than', 'their', 'or', 'been']
```

Output:

Word	Morphemes
can	'can'
than	'th', 'an'
their	't', 'heir'
or	'or'
been	'be', 'en'

Least frequent words

Random words chosen:

```
['Karun', 'ellipticities', 'Thalaimurai', 'DECsystem', 'Gorillas']
```

Output:

Word	Morphemes
Karun	'Ka', 'run'
ellipticities	'elliptic', 'ities'
Thalaimurai	'Th', 'al', 'aim', 'urai'
DECsystem	'DE', 'C', 'system'
Gorillas	'Gor', 'illa', 's'

---

2 - Hindi

indicnlp: Unsupervised morphological analysers for various Indian languages. Given a word, the analyzer returns the component morphemes. The analyzer can recognize inflectional and derivational morphemes. This also uses a Morfessor Baseline approach.
Type of unigrams as input: 322350

Most frequent words

Random words chosen:

```
['ही', 'हुई', 'क्षेत्र', 'थे', 'गए']
```

Output:

Word	Morphemes
ही	'ही'
हुई	'हुई'
क्षेत्र	'क्षेत्र'
थे	'थे'
गए	'गए'

Least frequent words

Random words chosen:

```
['उहेल्स', 'केंद्रकसूत्रों', 'थेलिओटोकी', 'स्त्रीजनन', 'फ़्ली']
```

Output:

Word	Morphemes
उहेल्स	'उ', 'हेल्स'
केंद्रकसूत्रों	'केंद्र', 'क', 'सूत्रों'
थेलिओटोकी	'थे', 'लिओ', 'टोकी'
स्त्रीजनन	'स्त्री', 'जन', 'न'
फ़्ली	'फ़्', 'ली'

1.3.5 - Sub-word tokenization

1 - Byte-pair-encoding (BPE) Implementation

```
def build_vocab(freq_dist):
 vocab = nltk.FreqDist([])
 for word, freq in freq_dist.items():
```

```
    w = ''
    for c in word:
      w += c + ' '
    vocab[w+'</w>'] = freq
  return vocab

def get_stats(vocab):
    pairs = defaultdict(int)
    for word, frequency in vocab.items():
        symbols = word.split()
        for i in range(len(symbols) - 1):
            pairs[symbols[i], symbols[i + 1]] += frequency
    return pairs

def merge_vocab(pair, v_in):
    v_out =  nltk.FreqDist([])
    bigram = re.escape(' '.join(pair))
    p = re.compile(r'(?<!\S)' + bigram + r'(?!\S)')
    for word in v_in:
        w_out = p.sub(''.join(pair), word)
        v_out[w_out] = v_in[word]
    return v_out

def byte_pair_encoding(freq_dist, num):
  vocab = build_vocab(freq_dist)
  num_merges = num
  print('Vocabulary size:', num_merges)
  fin_encodings = []

  for i in range(num_merges):
      pairs = get_stats(vocab)

      if not pairs:
          break

      best = max(pairs, key=pairs.get)
      fin_encodings.append(best)
      vocab = merge_vocab(best, vocab)

  return fin_encodings
```

---

2 - English

NOTE: Taking 10% data due to RAM issues.

Type of words as input: 58731

Number of merges: 4000

Sample Output of byte encodings:

```
[('e', '</w>'), ('s', '</w>'), ('t', 'h'), ('e', 'r'), ('.', '</w>'), ('d', '</w>'),
('n', '</w>'), (',', '</w>'), ('th', 'e</w>'), ('i', 'n'), ('a', 'n'), ('o', 'f'),
('of', '</w>'), ('y', '</w>'), ('t', '</w>'), ('o', 'r'), ('a', 'l'), ('a', 'r'),
('an', 'd</w>'), ('er', '</w>')]
```

Most frequent words:

```
[('.</w>', 153214), (',</w>', 116642), ('the</w>', 102691), ('of</w>', 87515),
('and</w>', 54311), ('%</w>', 48617), ('was</w>', 36350), ('in</w>', 32048), ('a</w>',
31982), ('were</w>', 30573), ('to</w>', 28767), ('The</w>', 27373), ('is</w>', 17501),
('age</w>', 17249), ('"</w>', 16173), ('from</w>', 15099), ('for</w>', 14914),
('or</w>', 14196), ('with</w>', 13567), ('(</w>', 13466), ('0</w>', 12440), ('2</w>',
12431), ('-</w>', 12147), ('had</w>', 12120), ('18</w>', 12067), ('1</w>', 11262),
('population</w>', 10763), ('township</w>', 10393), ('3</w>', 10224), ('$</w>', 10012),
('as</w>', 8989), ('years</w>', 8496), ('city</w>', 8414), ('median</w>', 8354),
('living</w>', 8141), ('4</w>', 8053), ('5</w>', 7896), ('which</w>', 7813),
('income</w>', 7806), ('average</w>', 7737), ('at</w>', 7661), ('households</w>',
7600), ('6</w>', 7544), ('8</w>', 7481), ('under</w>', 7459), ('7</w>', 7377),
('by</w>', 7335), ('there</w>', 7296), ('families</w>', 7079), ("'</w>", 6915)]
```

Least frequent words:

```
[('K ra use</w>', 1), ('Al bers</w>', 1), ('Fre der ick son</w>', 1), ('Mus cal a</w>',
1), ('A top</w>', 1), ('L in k ar a</w>', 1), ('Lov ha u g</w>', 1), ('G on e au</w>',
1), ('L or en</w>', 1), ('Bri m h all</w>', 1), ('Em met t</w>', 1), ('Par k view</w>',
1), ('R A M S</w>', 1), ('S n ail</w>', 1), ('O w as so</w>', 1), ('F em ales</w>', 1),
('An ce str ies</w>', 1), ('O d il ia</w>', 1), ('polic y making</w>', 1), ('Mar i ann
e</w>', 1), ('bro ad c ast ers</w>', 1), ('ant enn ae</w>', 1), ('T ele f arm</w>', 1),
('Re plac ement</w>', 1), ('lat er als</w>', 1), ('S ew er</w>', 1), ('W ast ew
ater</w>', 1), ('M C W S</w>', 1), ('P uc k</w>', 1), ('Mc M ill an</w>', 1), ('in a de
qu acy</w>', 1), ('out building</w>', 1), ('C er t ification</w>', 1), ('tr ic ol
our</w>', 1), ('dis cont igu ous</w>', 1), ('Ter re b on ne</w>', 1), ('Par ent e
au</w>', 1), ('bu t ter f at</w>', 1), ('sk im</w>', 1), ('wh ey</w>', 1), ('mon oc ul
ture</w>', 1), ('pre domin ated</w>', 1), ('bl ames</w>', 1), ('port ended</w>', 1),
('E mar d</w>', 1), ('I si ah</w>', 1), ('b lea k</w>', 1), ('ro ad house</w>', 1), ('t
ex ting</w>', 1), ('Min ot</w>', 1)]
```

Analysis on unknown words

Word	BPE output	polyglot output
serendipity	'ser en di p ity</w>'	's', 'er', 'en', 'dip', 'ity'
gobbledygook	'g ob b le d y g ook</w>'	'go', 'b', 'ble', 'dy', 'go', 'o', 'k'
scrumptious	'sc r um p t ious</w>'	's', 'c', 'rump', 't', 'ious'
agastopia	'ag as top ia</w>'	'a', 'ga', 'stop', 'ia'

halfpace	'hal f p ace</w>'	'half', 'pace'
impignorate	'imp ig nor ate</w>'	'imp', 'ignor', 'ate'
jentacular	'j ent ac ular</w>'	'je', 'nta', 'cular'
nudiustertian	'nu di u ster t ian</w>'	'nudi', 'us', 'ter', 'tian'
quire	'qu ire</w>'	'quire'
yarborough	'y ar bor ough</w>'	'y', 'ar', 'borough'

---

3 - Hindi

NOTE: Taking 10% data due to RAM issues.

Type of words as input: 63290

Number of merges: 4000

Sample Output of byte encodings:

```
[('े', '</w>'), ('ा', '</w>'), ('ी', '</w>'), ('ं', '</w>'), ('र', '</w>'), ('क', 'े</w>'), ('्', 'र'), ('न', '</w>'), ('ह', 'ै'), ('।', '</w>'), ('म', 'े'), ('मे', 'ं</w>'), ('क', '</w>'), (',', '</w>'), ('र', '्'), ('त', '</w>'), ('ो', '</w>'), ('है', '</w>'), ('्', 'य'), ('ि', 'य')]
```

Most frequent words:

```
[('के</w>', 34914), ('।</w>', 31925), (',</w>', 27641), ('मैं</w>', 26777), ('है</w>', 23026), ('की</w>', 16905), ('और</w>', 15511), ('से</w>', 13455), ('का</w>', 12385), ('को</w>', 11851), ('"</w>', 9234), ('हैं</w>', 8764), ('एक</w>', 8006), ('-</w>', 7802), (')</w>', 6683), ('(</w>', 6653), ('पर</w>', 6371), ('ने</w>', 5452), ('किया</w>', 5448), ('लिए</w>', 5008), ('भी</w>', 4939), ('.</w>', 4408), ('था</w>', 4189), ('कि</w>', 4175), ('गया</w>', 3824), ('यह</w>', 3758), ('इस</w>', 3501), ('रूप</w>', 3416), ('जो</w>', 3002), ('ही</w>', 2990), ('जाता</w>', 2957), ('नहीं</w>', 2836), ('करने</w>', 2827), ('साथ</w>', 2788), ('कर</w>', 2745), ('हो</w>', 2742), ('या</w>', 2471), ('द्वारा</w>', 2356), ('थे</w>', 2236), ('तथा</w>', 2136), ("'</w>", 2131), ('अपने</w>', 1990), ('दिया</w>', 1877), ('होता</w>', 1868), (':</w>', 1805), ('तक</w>', 1746), ('बाद</w>', 1730), ('थी</w>', 1668), ('वह</w>', 1660), ('करते</w>', 1503)]
```

Least frequent words:

```
[('अ सर दार</w>', 1), ('" प ण्ड ित</w>', 1), ('जग दी श च न्द्र</w>', 1), ('सम् मो हित</w>', 1), ('\n ज श पुर</w>', 1), ('\n \n छ त् तीस गढ़</w>', 1), ('नी चा घा ट</w>', 1), ('रो मं चक</w>', 1), ('उ रा ओं</w>', 1), ('ध ं धा</w>', 1), ('को टे बि रा</w>', 1), ('ए ब</w>', 1), ('\n \n सम र बार</w>', 1), ('बु झ ा</w>', 1), ('महा गि रि जा घर</w>', 1), ('नि हा र ने</w>', 1), ('कैम र</w>', 1), ('द नग री</w>', 1), ('टै क् सियों</w>', 1), ('\n पुष् प कवि मान</w>', 1), ('सु वर्ण म ण्ड ित</w>', 1), ('प्रार ु प</w>', 1), ('अं गि रा</w>', 1), ('भि ड ंत</w>', 1), ('उ ड ान</w>', 1), ('भर ते</w>', 1), ('उ डन</w>', 1),
```

```
('त श तर ियों</w>', 1), ('न भ चर</w>', 1), ('स्वय मे व</w>', 1), ('अल का पुरी</w>', 1), ('\n \n वर् त् त मान</w>', 1), ('वि मान क्षेत्रों</w>', 1), ('उ सा न गो ड़ा</w>', 1), ('गु रू लो पो था</w>', 1), ('तो तू पो ला क ंदा</w>', 1), ('वा रिया पो ला</w>', 1), ('\n \n ऋ ग वेद</w>', 1), ('कु मा रों</w>', 1), ('उदाह रण तया</w>', 1), ('कि ंग फि श र</w>', 1), ('जल यान</w>', 1), ('त्रि ता ला</w>', 1), ('त्रि चक्र</w>', 1), ('उ ड</w>', 1), ('प् की र ती त</w>', 1), ('\n \n सम रा ंग ण सू त्र धार</w>', 1), ('२ २ ५</w>', 1), ('शि ल् पा चार्य</w>', 1), ('योग सि द्ध ा</w>', 1)]
```

Analysis on unknown words

Word	BPE output	indicnlp output
आवेग	'आ वे ग</w>'	'आवेग'
अभियांत्रिकी	'अभ िया ं त्रि की</w>'	'अभि', 'यांत्रिक', 'ी'
वैतरणी	'वै तर णी</w>'	'वै', 'तरणी'
असहिष्णुता	'अस हि ष्ण ु ता</w>'	'अ', 'सहिष्णु', 'ता'
कुलाधिपति	'कु ला धि पति</w>'	'कुला', 'धिपति'
चिकित्सक	'चिकित् स क</w>'	'चिकित्सक'
स्थानांतरण	'स्थाना ंतरण</w>'	'स्थानांतर', 'ण'
दस्ता	'द स्ता</w>'	'दस्ता'
संपादन	'सं पादन</w>'	'संपादन'
प्रारूप	'प्रार ू प</w>'	'प्रारूप'

---

4 - Analysis

In both English and Hindi, Byte-Pair-Encoding algorithm provides more number of morphemes than using third-party tools for a given word. This is due to the following reasons:

- ➔ The BPE algorithm is trained only on 10% of given data with 4000 merges (due to limited amount of resources). This results in an insufficiently trained model with less number of pair encodings, hence resulting in more number of morphemes.
- ➔ The BPE is a more trivial algorithm than the Morfessor-based unsupervised model provided by the third party tools, which is also trained on huge amounts of data, thus providing better results and a more complete morphological analysis than the BPE algorithm.

However, one of the main advantages of the BPE algorithm is that this is language independent, and we can train almost any language using this algorithm. The real advantage of BPE over classical Word Embeddings is that it does not fall into the out-of-vocabulary error (when a word was not seen). At worst, we can always fall back to single bytes.
