CS565 : Assignment - 1

Devaishi Tiwari (170101021)

Colab Notebook Link:
https://colab.research.google.com/drive/1TEZ_Sumstkg5QEIUUQgHDEujnHzQY3wf?usp=sharing

---

Analysis reported below is performed on **20% of the corpus** due to memory / time constraints.

1. Analysis of the NLP Tools

I have used the **NLTK** and **SpaCy** for tokenization of the English corpus, and **Stanza** and **IndicNLP** for tokenization of the Hindi corpus.

1.1 Sentence Segmentation

English

Tools used	No. of Sentences	List of Sentences
NLTK	176321	['The word "atom" was coined by ancient Greek philosophers.','However, these ideas were ...
SpaCy	180770	['The word "atom" was coined by ancient Greek philosophers.','However, these ideas were ...

Hindi

Tools used	No. of Sentences	List of Sentences
Stanza	72888	['मास्टर ऑफ़ हेल्थ एडमिनिस्ट्रेशन या मास्टर ऑफ हेल्थकेयर एडमिनिस्ट्रेशन (एमएचए या एम.एच.ए) स्नातकोत्तर ...
IndicNLP	68862	['मास्टर ऑफ़ हेल्थ एडमिनिस्ट्रेशन या मास्टर ऑफ हेल्थकेयर एडमिनिस्ट्रेशन (एमएचए या एम.एच.ए) स्नातकोत्तर ...

Analysis of tools

- **NLTK 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.

- **SpaCy Sentence Tokenizer -** This tokenizer looks for specific characters that fall between sentences, like periods, exclamation points, and newline characters. These characters mark the sentence boundary.

- **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.
- **IndicNLP Sentence Tokenizer -** IndicNLP provides tokenizer for Indian languages using Brahmi for Arabic scripts. IndicNLP uses a TrivialTokenizer, which is a punctuation-based sentence splitter that can understand common punctuations in many Indian languages.

1.2 Word Tokenization

English

Tools used	No. of Words	List of Words
PunktWord Tokenizer	4086209	['The', 'word', '``', 'atom', '"', 'was', 'coined', 'by', 'ancient', 'Greek' ...
WordPunct Tokenizer	4458087	['The', 'word', '``', 'atom', '"', 'was', 'coined', 'by', 'ancient', 'Greek' ...
Treebank Tokenizer	3927506	['The', 'word', '``', 'atom', '"', 'was', 'coined', 'by', 'ancient', 'Greek' ...

Hindi

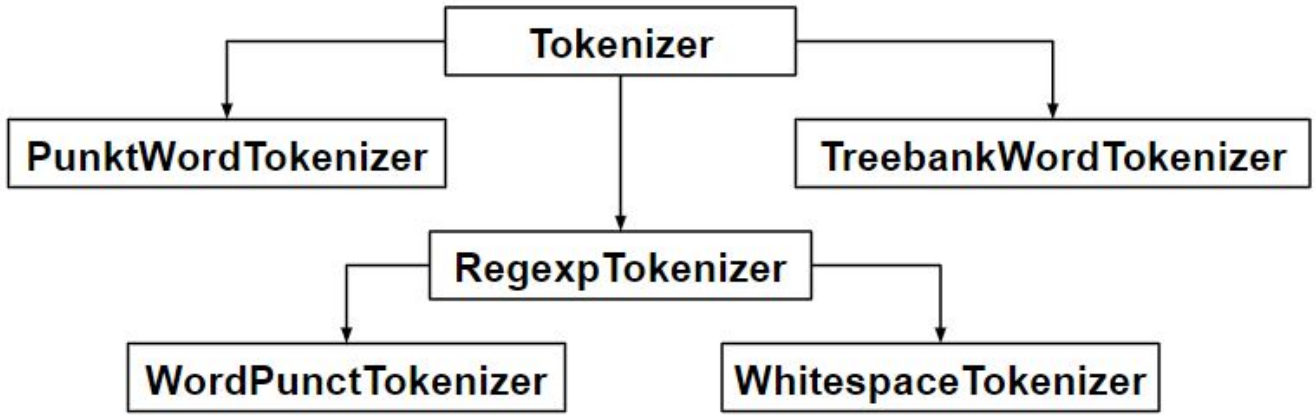
Tools used	No. of Words	List of Words
NLTK	1633824	['मास्टर', 'ऑफ़', 'हेल्थ', 'एडमिनिस्ट्रेशन', 'या', 'मास्टर', 'ऑफ', 'हेल्थकेयर', 'एडमिनिस्ट्रेशन', '(' ...
Stanza	1681604	['मास्टर', 'ऑफ़', 'हेल्थ', 'एडमिनिस्ट्रेशन', 'या', 'मास्टर', 'ऑफ', 'हेल्थकेयर', 'एडमिनिस्ट्रेशन', '(' ...
IndicNLP	1729146	['मास्टर', 'ऑफ़', 'हेल्थ', 'एडमिनिस्ट्रेशन', 'या', 'मास्टर', 'ऑफ', 'हेल्थकेयर', 'एडमिनिस्ट्रेशन', '(' ...

Analysis of tools

Natural Language Toolkit (NLTK) provides us with various types of word tokenizers, like:

- **NLTK PunktWord Tokenizer -** PunktWord tokenizer used matching to split the text into words. The tokenizer keeps the punctuation intact and as a part of the word. It also tries to strip whitespaces from the text to match with the existing vocabulary.

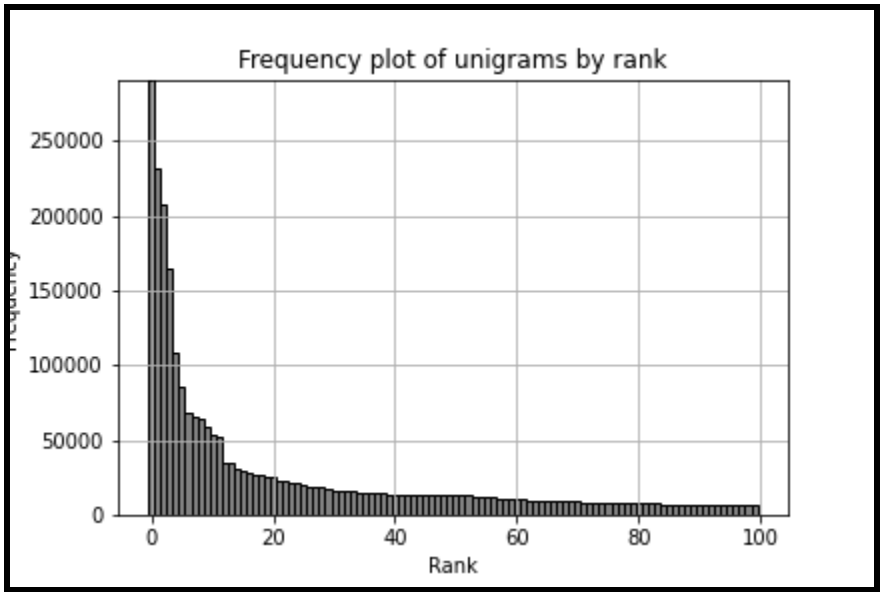
- **NLTK Treebank Tokenizer -** The Treebank tokenizer uses regular expressions to tokenize text. It separates the words using punctuation and spaces. However, it does not discard the punctuation, allowing a user to decide what to do with the punctuations at the time of pre-processing.
- **NLTK WordPunct 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. Unlike the other tokenizers, WordPunct Tokenizer separates the punctuation from the words.



1.3 Frequency Distribution of Unigrams

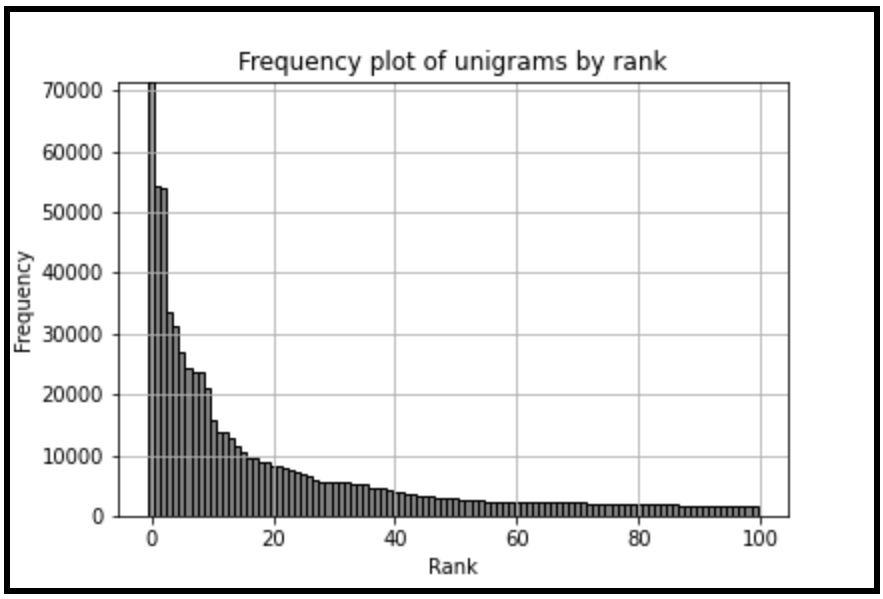
I have used **NLTK WordPunct Tokenizer** for tokenization and analysis of unigrams in the English corpus and **NLTK Tokenizer** for tokenization and analysis of unigrams in the Hindi corpus.

English



Total Unigrams: 4458087
Unique: 91903
Most Occurring Unigrams: [('.', 291371), (',', 231646), ('the', 207921), ('of', 164211), ('and', 108203), ('%', 85282), ('was', 68873), ('in', 65428), ('a', 64016), ('to', 59192)]

Hindi

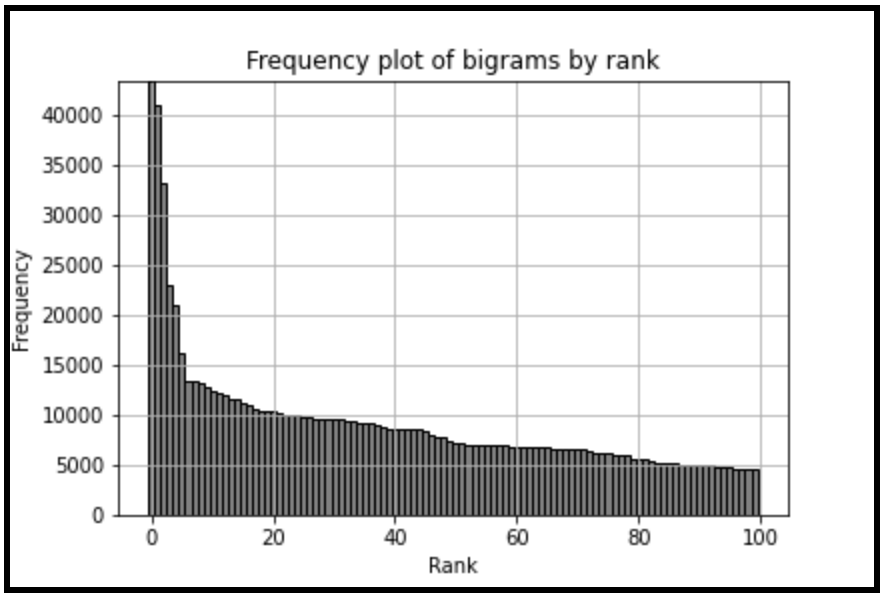


Total Unigrams: 1633824
Unique: 101650
Most Occurring Unigrams: [('के', 71492), ('में', 54162), (',', 53807), ('की', 33545), ('और', 31227), ('से', 26883), ('का', 24191), ('को', 23815), ('है ।', 23771), ('है', 21197)]

1.4 Frequency Distribution of Bigrams

I have used **NLTK WordPunct Tokenizer** for tokenization and analysis of bigrams in the English corpus and **NLTK Tokenizer** for tokenization and analysis of bigrams in the Hindi corpus.

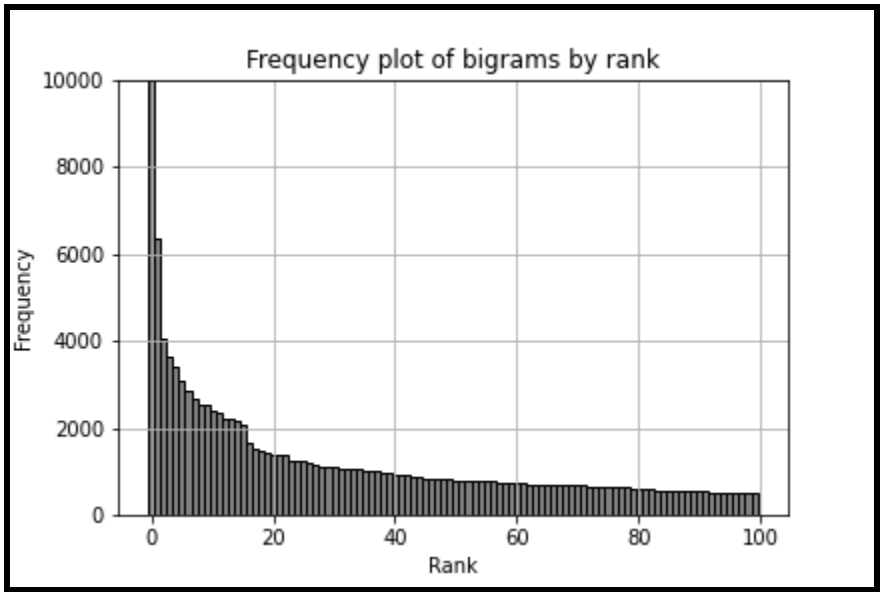
English



Total Bigrams: 4458087
Unique: 867225
Most Occurring Bigrams: [(('.', 'The'), 43567), (('of', 'the'), 41015), ((',', 'and'), 33359), (('in', 'the'), 23068), (('%', 'of'), 21010), (('%', 'from'), 16251), (('%', 'had'), 13426), (('and', 'the'), 13335), ((',', 'the'), 13210), (("'", 's'),

12755)]

Hindi

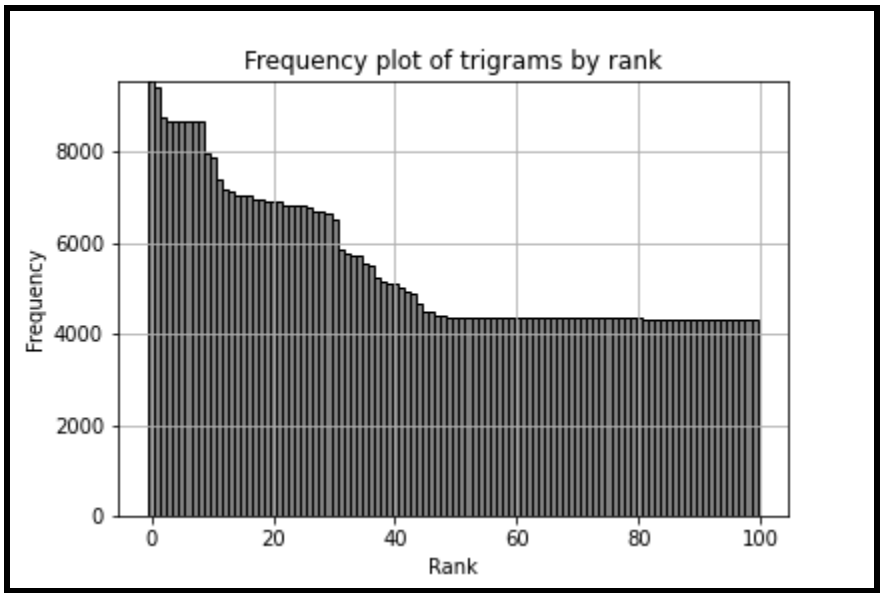


Total Bigrams: 1633824
Unique: 671375
Most Occurring Bigrams: [(('के', 'लिए'), 10008), (('है', ','), 6377), (('के', 'साथ'), 4062), (('रूप', 'में'), 3651), (('के', 'रूप'), 3392), (('है', 'कि'), 3104), (('है', 'और'), 2841), (('जाता', 'है।'), 2658), (('में', ','), 2552), (('हैं', ','), 2527)]

1.5 Frequency Distribution of Trigrams

I have used **NLTK WordPunct Tokenizer** for tokenization and analysis of trigrams in the English corpus and **NLTK Tokenizer** for tokenization and analysis of trigrams in the Hindi corpus.

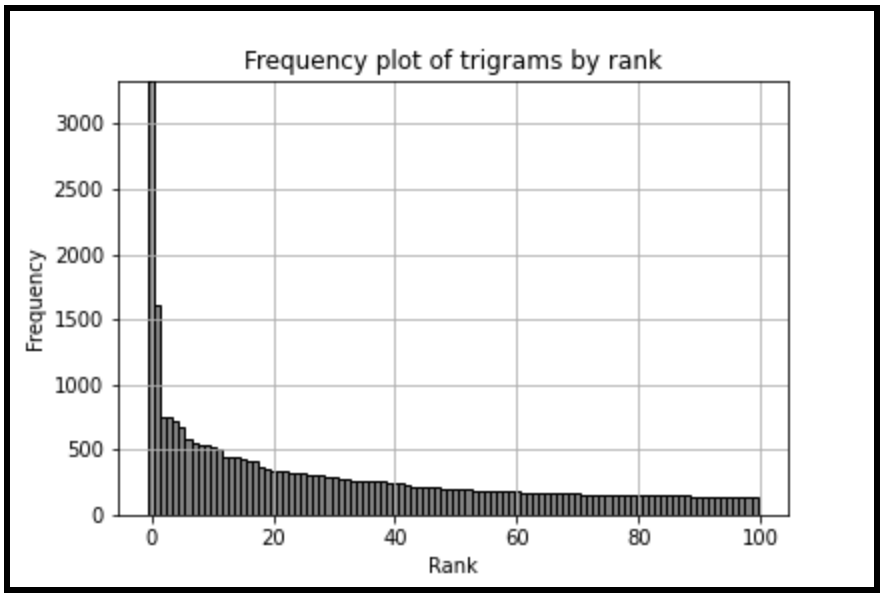
English



Total Trigrams: 4458087

Unique: 1860270
Most Occurring Trigrams: [(('the', 'age', 'of'), 9568), (('under', 'the', 'age'), 9427), (('age', 'of', '18'), 8761), (('years', 'of', 'age'), 8682), (('or', 'older', '.'), 8680), (('older', '.', 'The'), 8668), (('of', 'age', 'or'), 8656), (('65', 'years', 'of'), 8656), (('age', 'or', 'older'), 8655), ((',', 'there', 'were'), 7949)]

Hindi



Total Trigrams: 1633824
Unique: 1278889
Most Occurring Trigrams: [(('के', 'रूप', 'में'), 3329), (('करने', 'के', 'लिए'), 1608), (('है', ',', 'जो'), 751), (('किया', 'जाता', 'है।'), 748), (('जाता', 'है', ','), 719), (('जा', 'सकता', 'है।'), 677), (('के', 'बारे', 'में'), 581), (('किया', 'जा', 'सकता'), 558), (('के', 'लिए', 'एक'), 539), (('में', 'से', 'एक'), 529)]

1.6 Zipf's Law

- The Zipf's law states that the **frequency of a word is inversely proportional to the rank of the word in terms of occurrence** (in the training corpus).
- The above graphs show that the Zipf's law holds. The frequency distribution graph is decreasing and closely resembles an hyperbola, thus proving that $f \propto \frac{1}{r}$.
- Another important observation is that the resemblance is more strict in case of unigrams and less in case of trigrams.

---

2. Few Basic Questions

Stemming is the process of reducing morphological variants of a word back to the root/base word. A stemming algorithm reduces the words like **“retrieval”, “retrieved”, “retrieves”** to their stem i.e. **“retrieve”**. The **PorterStemmer** Library is used for stemming English corpus and some of the common suffixes in the Hindi language are **manually removed** in the Hindi corpus.

2.1 90% Coverage by Unigrams

No. of Unigrams	English	English (Stemmed)	Hindi	Hindi(Stemmed)
Unique	91903	61048	101650	93134
Required	5005	2305	11583	8843

2.2 80% Coverage by Bigrams

No. of Bigrams	English	English (Stemmed)	Hindi	Hindi(Stemmed)
Unique	867225	736857	671375	633112
Required	124258	81753	344610	306347

2.3 70% Coverage by Trigrams

No. of Trigrams	English	English (Stemmed)	Hindi	Hindi(Stemmed)
Unique	1860270	1783889	1278889	1254472
Required	522844	446463	788742	764325

2.4 Summary

There is a **53%** reduction in required unigrams, **34%** reduction in required bigrams and **15%** reduction in required trigrams due to stemming. This observation is intuitive as stemming will lead to more frequency of root words and in turn less number of unique words. As a result, a single token will cover more percentage of the corpus.
In case of Hindi corpus, there is a **24%** reduction in required unigrams, **12%** reduction in required bigrams and **4%** reduction in required trigrams due to stemming.

---

3. Writing Some Basic Codes

3.1 Implementing Heuristics

Regular Expression

```
pattern = r'''(?x)
      (?:[A-Z]\.)+         # abbreviations, e.g. U.N.O.
    | (?:\s\w\w\.)+        # titles, e.g. Dr., Ms.
    | \w+(?:-\w+)*         # words with hyphens, e.g. zig-zag
    | \$?\d+(?:\.\d+)?%?   # currency and percentages, e.g. $5, 25%
    | \.\.\.               # ellipsis, e.g. a,b,c...
'''
```

Coverage	Required (Without Stemming)		Required (With Stemming)	
	Before	After	Before	After
Unigram (90%)	5005	4211	2305	2426
Bigram (80%)	124258	107013	81753	79848
Trigram (70%)	522844	494898	446463	445966

Summary

In lieu of the simple tokenizers,we have **expanded sentence boundaries** and **ignored the case** of all the tokens, which (as can be seen from the tables above) ensures better tokenization.

By applying the heuristics, we see a significant decrease in the number of n-grams required for coverage in both the cases. There is a decrease of almost **16%** in unigrams, **14%** in bigrams and **5%** in trigrams required for coverage.

3.2 Likelihood Ratio Testing

Language	Unique Bigrams	Bigrams using Likelihood Ratio Testing
English	867225	213756
Hindi	671375	116662

Note: The Implementation of Likelihood Ratio Testing is explained in the notebook, linked at the top of the report.

---

4. Morphological Parsing

4.1 English

Frequency	Word	Lexemes
Most Frequent Unigrams	contains	['contain', 's']
	each	['e', 'ach']
	various	['vari', 'ous']
	as	['a', 's']
	being	['be', 'ing']
Least Frequent Unigrams	unscientific	['un', 'scientific']
	Mabhouh	['Ma', 'b', 'ho', 'u', 'h']
	Artery	['Arte', 'ry']
	incurring	['incur', 'ring']
	Qumsiyeh	['Qu', 'm', 's', 'i', 'y', 'eh']

4.2 Hindi

Frequency	Word	Lexemes
Most Frequent Unigrams	निर्धारित	['निर्धारित']
	करना	['कर', 'ना']
	सैन्य	['सैन्य']
	प्रसिद्ध	['प्रसिद्ध']
	राष्ट्रपति	['राष्ट्रपति']
Least Frequent	उपास्यदेव	['उपास्य', 'देव']
	'खैरागढ़	["'खैरागढ़"]

Unigrams	ख्	['ख्+']
	फ़िल्मी-हस्तियां	['फ़िल्मी-हस्तियां']
	रैंडर	['रैंड', 'र']

4.3 Analysis of Tools used

English

- **Polyglot** was used for morphological analysis, i.e. to generate morphemes from words. The morphological analyzer used an **unsupervised data-driven model** to discover the regularities behind word forming in natural languages, and use it to split the word into morphemes.
- 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.

Hindi

- **IndicNLP** provides us with unsupervised morphological analysers for various Indian languages.
- The analyzer can recognize inflectional and derivational morphemes. The morphological analyzer uses a **Morfessor Baseline approach** to split the word into its morphemes.

---

5. Sub-Word Tokenization

Number of Merge: 500
Number of Tokens: English - 91903, Hindi - 101650

5.1 Most Common Words

English

```
[('.$', 291371), (',$', 231646), ('the$', 207921), ('of$', 164211), ('and$',
108203), ('%$', 85282), ('was$', 68873), ('in$', 65428), ('a$', 64016), ('to$',
59192), ('were$', 54127), ('The$', 52467), ('"$', 35371), ('is$', 34277), ('age$',
30634), ('for$', 29821), ('from$', 28787), ('-$', 27435), ('($', 26440), ('with$',
26127)]
```

Hindi

```
[('के$', 71492), ('में$', 54162), (',$', 53807), ('की$', 33545), ('और$', 31227), ('से$', 26883), ('का$', 24191), ('को$', 23815), ('है।$', 23771), ('है$', 21197), ('एक$', 15831), (')$', 13725), ('($', 13671), ('पर$', 12717), ('ने$', 11363), ('लिए$', 10580), ('भी$', 9482), ('हैं।$', 9384), ('किया$', 8923), ("''$", 8847)]
```

5.2 Least Common Words

English

```
[('S z u c k o$', 1), ('C Q T s$', 1), ('S um mar iz ing$', 1), ('pre t ens e$', 1), ('P s y ch ol og ist s$', 1), ('P ol y g ra ph s$', 1), ('P T S D $', 1), ('h y p og l y c em ia$', 1), ('di sh on est y$', 1), ('un sc i ent if ic$', 1), ('vi ol at or s$', 1), ('V al id ity$', 1), ('M o y n i h an$', 1), ('S ec rec y$', 1), ('. . " .$', 1), ('A s k ed$', 1), ('" ` $', 1), ('pr ic k ing$', 1), ('B é land$', 1), ('su b j ec $', 1)]
```

Hindi

```
[('ए ड ो आ र् ड$', 1), ('से गु इन$', 1), ('9 - वर् ष ीय$', 1), ('है । ज न् म जा त$', 1), ('क्र ी ज$', 1), ('। इन$', 1), ('वा यु मार् गों$', 1), ('ए प न ियां$', 1), ('पै ट र् न । अ ट ला ं ट ै क् स िय ल$', 1), ('है ं - पु रु ष ों$', 1), ('5 0 - 6 9$', 1), ('20 - 3 5$', 1), ('1 0 - 3 0$', 1), ('है । 1 0$', 1), ('स्ट टर$', 1), ('है ं । वे$', 1), ('स् पै स् म$', 1), ('5 0 - 7 0$', 1), ('स् द्र ै बि स् म स$', 1), ('के रा ट ो को न स$', 1)]
```

5.3 Tokenization of New Words

English

Words	BPE Result	PolyGlot
friendly	fr + i + en + d + ly	friend + ly
president	pre + si + d + ent	president
makes	ma + k + es	make + s
statement	st + at + em + ent	state + ment
captures	cap + tur + e	capture

meaning	me + an + ing	mean + ing
behind	be + h + in + d	be + hind
different	di + ff + er + ent	different
emotions	em + ot + ions	e + motion + s
present	present	present

Hindi

Words	BPE Result	IndicNLP
वर्तमान	वर्त + मान	वर्तमान
भविष्य	भ + वि + ष + ◌्य	भविष्य
विभाजित	वि + भा + ज + ◌ित	विभाजित
अद्भुत	अ + द् + भ + ◌ु + त	अद्भुत
रचना	र + च + ना	रचना
खिलौने	खि + ल + ◌ौ + ने	खिलौने
किताब	कि + ता + ब	किताब
विभिन्न	वि + भि + न्न	विभिन्न
भावनाएँ	भा + व + ना + ए ◌ँ	भावना + एँ
सिखाने	सि + खा + ने	सिखा + ने

Summary

- In both the languages, the **Byte Pair Encoding (BPE)** results in more number of Morphemes as compared to the predefined libraries.
- These libraries have been trained on huge datasets beforehand and hence, they have a richer vocabulary. Thus, they provide much better results than our algorithm which is trained only on our corpus
- The number of morphemes can also be increased by increasing the number of merges that the user is allowing. We have used only 500 merges due to resource and time constraints.
