NLP Assignment 2

Faqeha Nadeem 18I-0674

Libraries and Corpora used

For starters I imported all the relevant libraries that I needed and downloaded the corpora I intended to use. I have used nltk for functions of tokenization and ngrams.

The corpora I used are both parts of the Gutenberg corpus, which are two different novels from different time periods, that is, Jane Austen's "Persuasion" and William Shakespeare's "Hamlet", due to being from vastly different time periods I am treating the two as separate corpora.

Persuasion was published in the year 1818 whereas Hamlet was published in 1603. Both texts are fictional stories written in English by British authors almost two hundred years apart.

As a language evolves over time its vocabulary, slang and usage change drastically. Which is why I thought it would be interesting to compare the unigrams,bigrams,Trigrams generated by these two corpora.

Storing words/tokens in a corpus to a list

I stored all the words in Hamlet in a list called words and all the words in Persuasion in a list called words2, here we do not need to write a separate functionality for tokenization as nltk allows you to get all the words of a text including punctuation marks. Through the .words() functionality. All of these words/tokens can be stored in a list and the list can be passed as a parameter in the ngrams function.

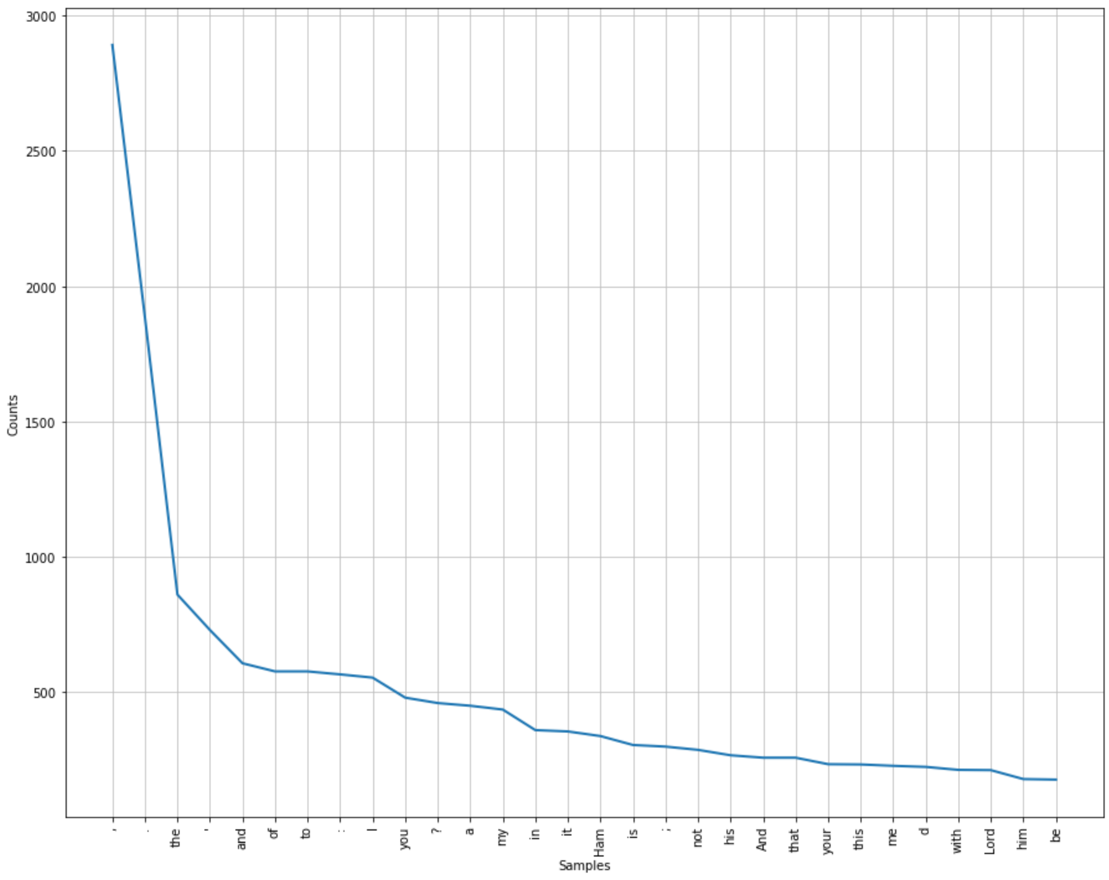
Comparison of unigrams generated from Hamlet and Persuasion

Despite the texts having been written almost two centuries apart we can see that the most frequent unigrams of both are extremely similar.

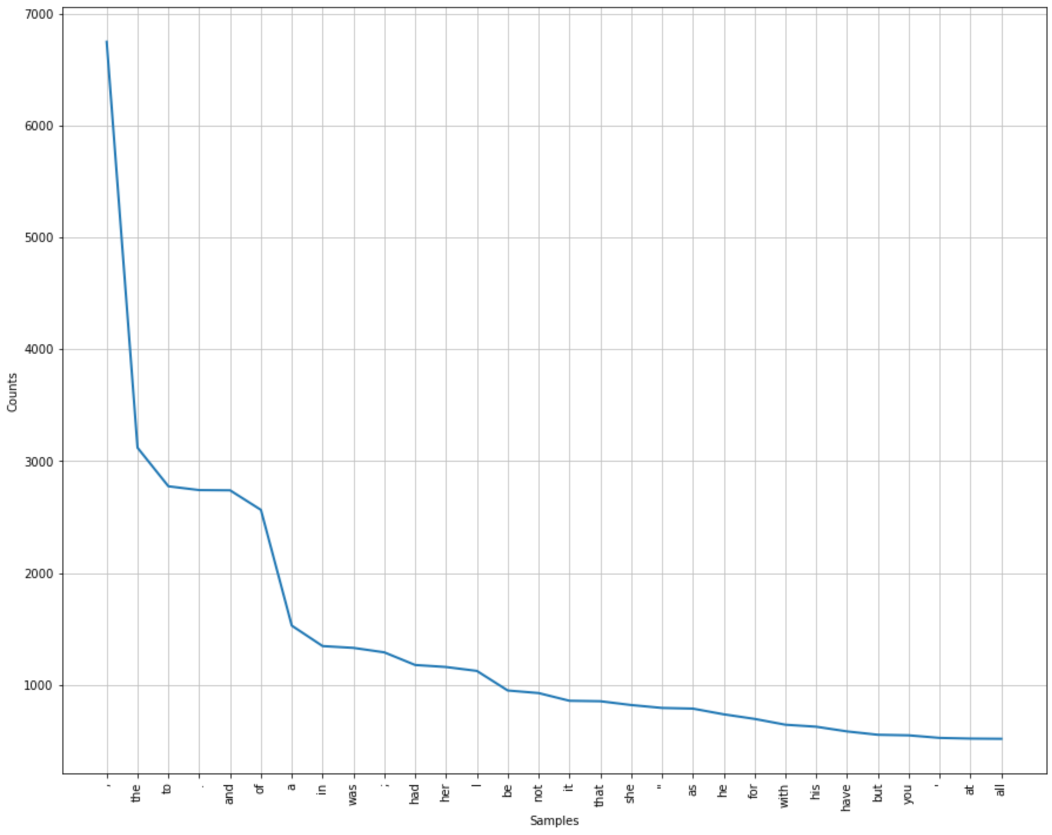
The most frequent unigrams of both include punctuation marks like commas,colons and hyphens. They also include words like the,of,to,a, with,that,my etc. These words seem to be pronouns and the foundational parts of English text that are a part of most sentences and can be related to how sentences are structured. The same goes for punctuation marks.

This makes a lot of sense as while the vocabulary and slang of a language can change a lot over the course of 200 years the structure and the most basic words used to form any sentence remain almost entirely the same.

Hamlet (corpus 1):



Persuasion (Corpus 2):

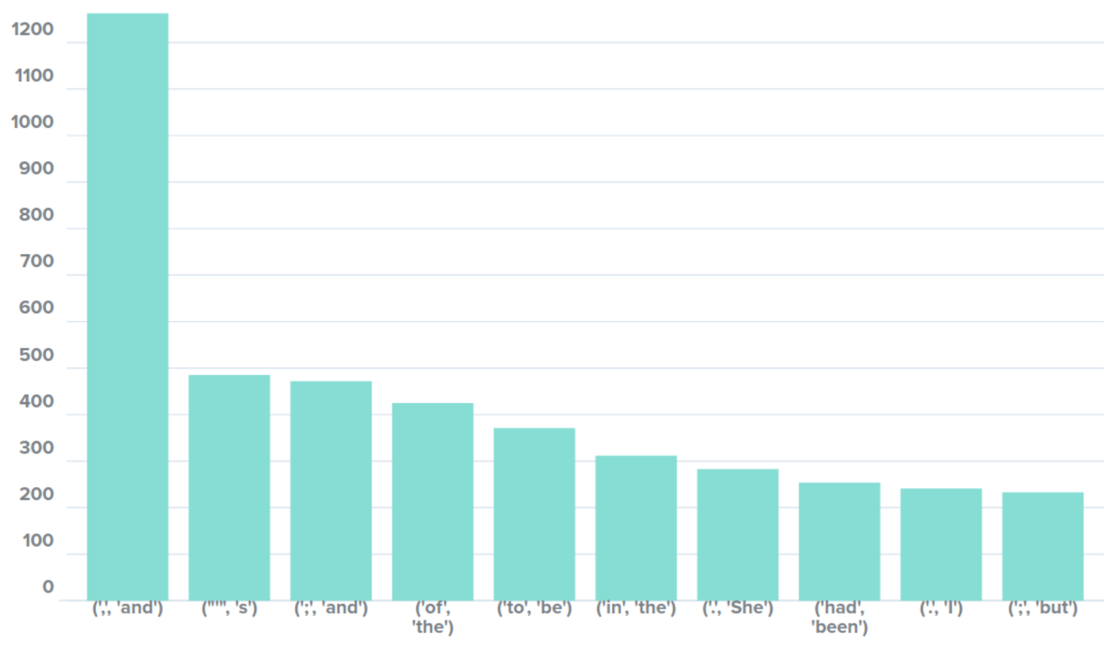


Comparison between Most frequent Bigrams of both corpora

While still there are similarities between most frequent bigrams generated by both, they are not as similar as the unigrams generated were. We can see a trend of pairs between a punctuation mark and some word like I, she etc. But now there are more instances of context specific language such as 'king' 'Lord' 'Ham' 'Hor' in Hamlet which all refer to particular characters in the story. There are also more variations in frequency of different punctuation marks, and conjunction of commonly used words like 'had' 'been' a possible indicator of the two texts being written in different eras or an indicator of the differing writing styles of the two authors.

Most Frequent Bigrams of Hamlet

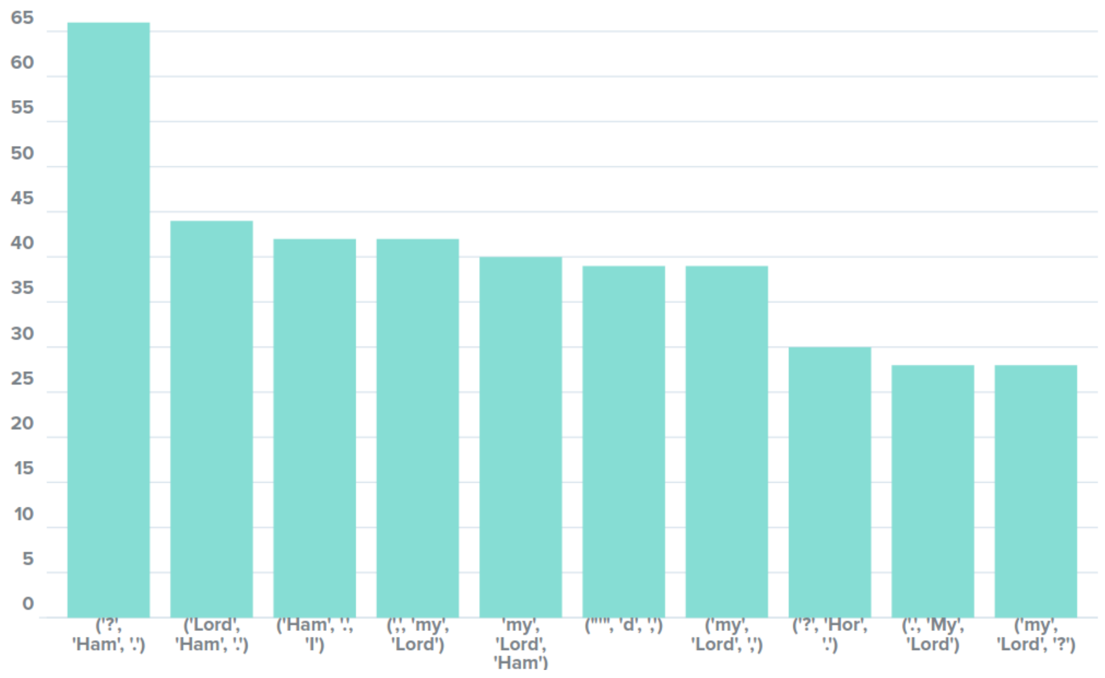
Most Frequent Bigrams Persuasion



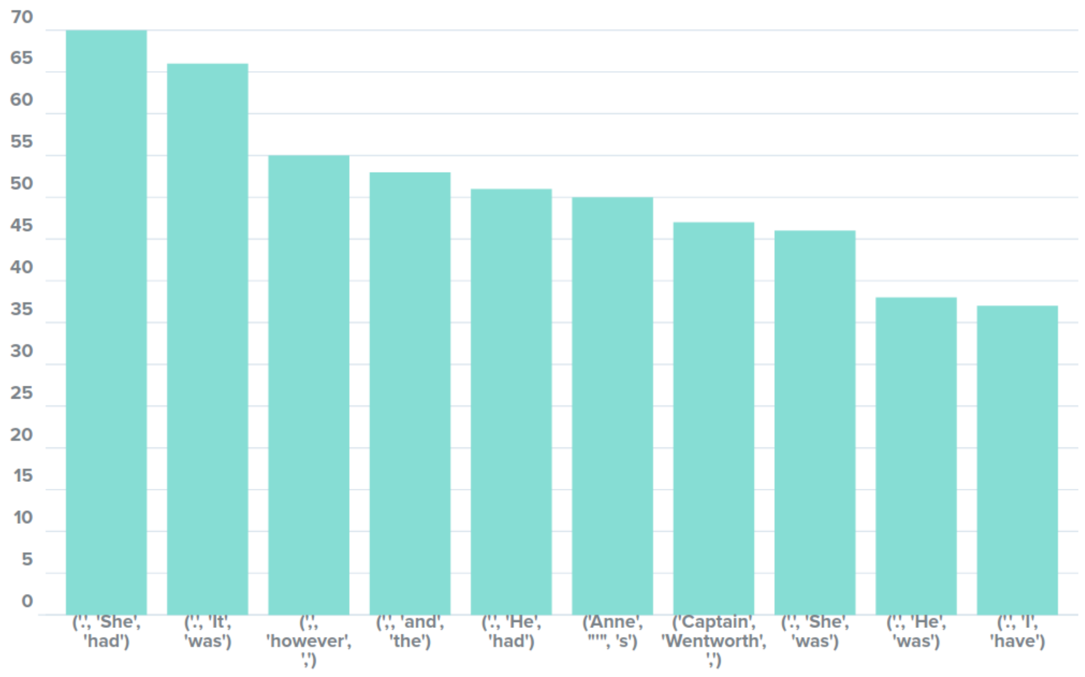
Comparison between Most frequent Trigrams of both corpora

Now the most frequent values we have obtained have begun to differ even more and are clearly from distinct texts, Specific names of characters are mentioned in the most frequent trigrams of both, the pronouns used and the sequence of words are more distinct and you can get a slight sense that the first corpus is much older than the second from phrases like "My Lord"

Most Frequent Trigrams Hamlet



Most Frequent Trigrams Persuasion



Text-Generation with Bigrams and Trigrams

I have generated text using Bigrams and Trigrams. Each method starts off by assigning a random word from the list of words in the corpus as the starting or leading word in the generated text.

Then a list of possibilities is initialized as empty. And then we initialize a string “result” with the value of head (which is currently the leading value generated randomly)

Then we enter our external loop that runs a certain number of times within it we have an internal loop that goes through the list of all ngrams, when it encounters the “head” value as the first value in an ngram tuple the remaining n-1 words in tuple after first one are stored in the list of possibilities (as a tuple for n>2)

The possibilities list is similar to using a context dictionary but instead of providing context or proving possible next words for all words in corpus we only generate possibilities for the last word generated by the program (i-e the head).

Once we exit this internal loop we randomly generate an element from the list of possibilities and append all words in this tuple to the result, the head is updated to be the last word in the result string.

The external loop moves on to the next iteration and the process is repeated a number of times.

This text generation uses the concept of Markov Chains, where it looks at one state and then records all possible next states of that one state. And then generates a next state accordingly.

Instead of calculating the conditional probabilities I chose to generate the text randomly from the possibilities, I didn't adjust the list of possibilities for redundancies i-e allow multiple entries of an element with same contents.

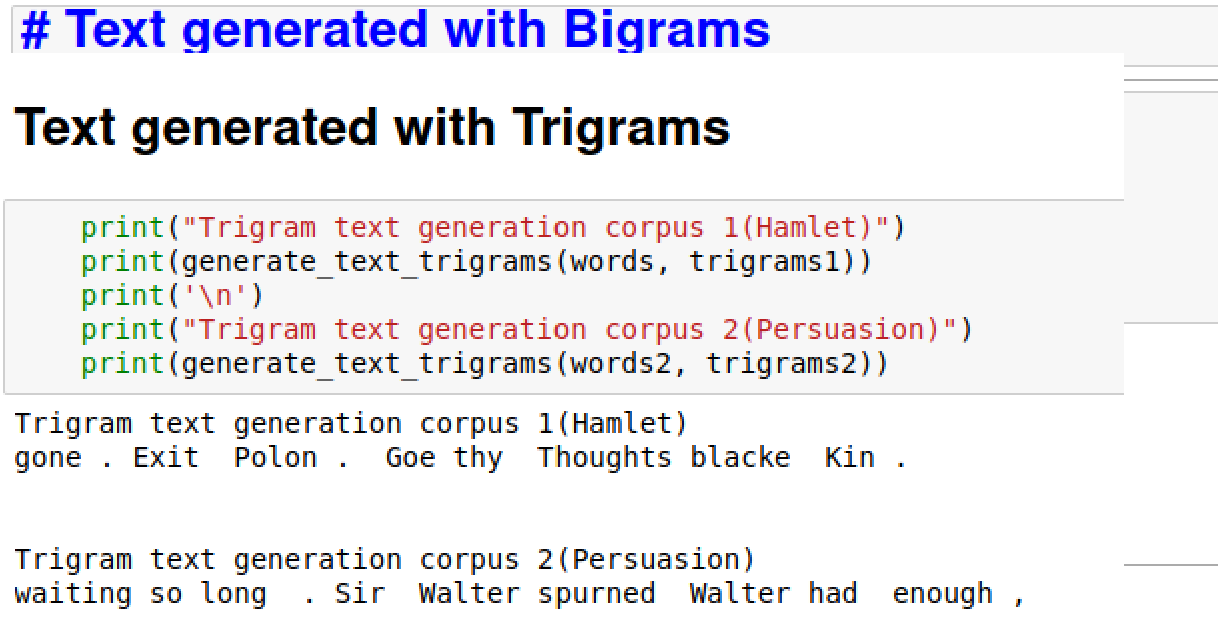
This simulates semi-random behavior, the next state is generated randomly but from a list where some possibilities exist more frequently than others due to this the **likelihood** of a certain entry being generated will be more. However there is still a chance of other words with less frequency being generated as well,which is different from generating a word with the highest conditional probability every time as in that case if n is small we can get stuck in a loop and repeat states.

As n increases the text generated starts to resemble actual correct text as at that point we are picking up actual phrases or sentences from the text as a possible next state.

That is why the Trigram text generation appears to be more sensible than the Bigram one.

The text generated is also clearly affected by the content of the corpus.

Sample Output:



Text generated with Bigrams

Text generated with Trigrams

```
print("Trigram text generation corpus 1(Hamlet)")
print(generate_text_trigrams(words, trigrams1))
print('\n')
print("Trigram text generation corpus 2(Persuasion)")
print(generate_text_trigrams(words2, trigrams2))
```

```
Trigram text generation corpus 1(Hamlet)
gone . Exit  Polon .  Goe thy  Thoughts blacke  Kin .


Trigram text generation corpus 2(Persuasion)
waiting so long  . Sir  Walter spurned  Walter had  enough ,
```