Edge Intelligence Laboratory **Oviya R**

Assessment-1 **25MML0007**

Dataset URL: https://www.gutenberg.org/cache/epub/77439/pg77439.txt

Introduction:

I downloaded a raw English novel from Project Gutenberg to perform end-to-end Natural Language Processing (NLP) using Python and NLTK. The analysis involved text cleaning, tokenization, stop-word removal, frequency analysis, part-of-speech tagging, named entity recognition. This workflow demonstrates how raw text can be transformed into structured linguistic insights using standard NLP methods.

Approach:

The novel was fetched using the requests library. A timeout mechanism was included to handle network delays, and an HTTP status check ensured successful download. While downloading the novel, I faced errors such as Module Not Found Error when requests was missing, and Attribute Error when I mistakenly used response.code instead of response.status_code. I corrected these by installing the library, the URL, and using the correct attribute.

While running this code, I faced the issue like None Type object is not iterable also the library for frequency count is not imported.

Error Code:

```
from nltk.util import ngrams
words_no_sw = None
# Generate all bigrams (pairs of consecutive words) from words_no_sw and store them as a list of tuples.
bigrams = list(ngrams(words_no_sw, 2))

# Generate all trigrams (triplets of consecutive words) from words_no_sw and store them as a list of tuples.
trigrams = list(ngrams(words_no_sw, 3))
#printing the most common bigram and trigram along with its frequency count
for bg, freq in fdist_bi.most_common(10):
    print(f"{' '.join(bg)}: {freq}")
for bg, freq in fdist_tri.most_common(10):
    print(f"{' '.join(bg)}: {freq}")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipython-input-2444498275.py in <cell line: 0>()
      2 words_no_sw = None
      3 # Generate all bigrams (pairs of consecutive words) from words_no_sw and store them as a list of tuples.
----> 4 bigrams = list(ngrams(words_no_sw, 2))
      5
      6 # Generate all trigrams (triplets of consecutive words) from words_no_sw and store them as a list of tuples.

                                 1 frames
/usr/local/lib/python3.12/dist-packages/nltk/util.py in pad_sequence(sequence, n, pad_left, pad_right, left_pad_symbol, right_pad_symbol)
    898     :rtype: sequence or iter
    899     """
--> 900     sequence = iter(sequence)
    901     if pad_left:
    902         sequence = chain((left_pad_symbol,) * (n - 1), sequence)

TypeError: 'NoneType' object is not iterable
```

Corrected Code:

```
from nltk.util import ngrams
from nltk import FreqDist

# Generate all bigrams (pairs of consecutive words) from words_no_sw and store them as a list of tuples.
bigrams = list(ngrams(words_no_sw, 2))

# Generate all trigrams (triplets of consecutive words) from words_no_sw and store them as a list of tuples.
trigrams = list(ngrams(words_no_sw, 3))

fdist_bi = FreqDist(bigrams)
fdist_tri = FreqDist(trigrams)

#printing the most common bigram and trigram along with its frequency count
for bg, freq in fdist_bi.most_common(10):
    print(f"{' '.join(bg)}: {freq}")
for bg, freq in fdist_tri.most_common(10):
    print(f"{' '.join(bg)}: {freq}")
```

```
de la: 50
lo que: 46
senyor joseph: 34
que es: 34
que l: 26
hi ha: 24
ja ho: 23
es que: 23
en joanet: 21
la meva: 21
la meva na: 8
que en joanet: 6
senyor joseph joseph: 6
qu vol dir: 6
lo que es: 6
se n va: 5
donya mercedes mercedes: 5
llegir el diari: 5
que es el: 5
la senyora rosa: 4
```

Here while doing tokenisation, I passed the clean_text in the form of list, which showed error. Then I converted and passed it as a tuple

Error Code:

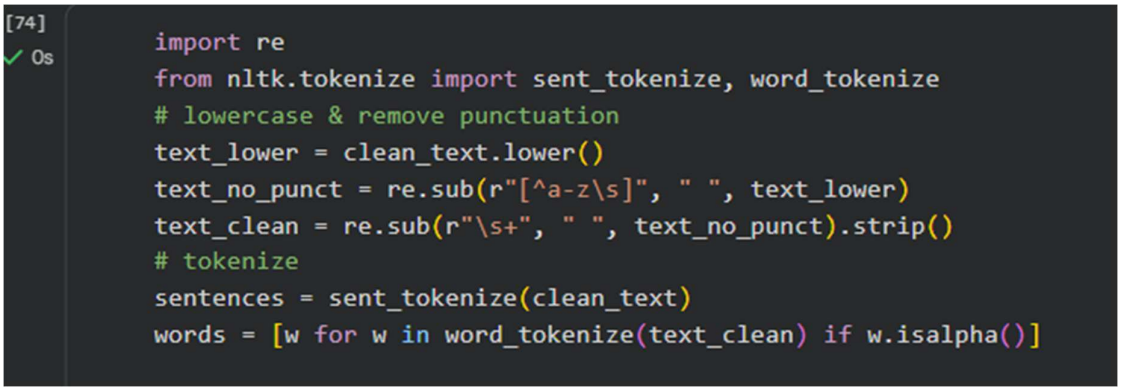
```
import re
from nltk.tokenize import sent_tokenize, word_tokenize
# lowercase & remove punctuation
text_lower = clean_text.lower()
text_no_punct = re.sub(r"[^a-z\s]", " ", text_lower)
text_clean = re.sub(r"\s+", " ", text_no_punct).strip()
# tokenize
sentences = sent_tokenize([clean_text])
words = [w for w in word_tokenize(text_clean) if w.isalpha()]
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/tmp/ipython-input-3030811973.py in <cell line: 0>()
      6 text_clean = re.sub(r"\s+", " ", text_no_punct).strip()
      7 # tokenize
----> 8 sentences = sent_tokenize([clean_text])
      9 words = [w for w in word_tokenize(text_clean) if w.isalpha()]

                              7 frames
/usr/local/lib/python3.12/dist-packages/nltk/tokenize/punkt.py in _match_potential_end_contexts(self, text)
   1392         previous_slice = slice(0, 0)
   1393         previous_match = None
-> 1394         for match in self._lang_vars.period_context_re().finditer(text):
   1395             # Get the slice of the previous word
   1396             before_text = text[previous_slice.stop : match.start()]

TypeError: expected string or bytes-like object, got 'list'
```

Corrected Code:



```
[74]
✓ 0s
    import re
    from nltk.tokenize import sent_tokenize, word_tokenize
    # lowercase & remove punctuation
    text_lower = clean_text.lower()
    text_no_punct = re.sub(r"[^a-z\s]", " ", text_lower)
    text_clean = re.sub(r"\s+", " ", text_no_punct).strip()
    # tokenize
    sentences = sent_tokenize(clean_text)
    words = [w for w in word_tokenize(text_clean) if w.isalpha()]
```

Removing Gutenberg Metadata

Project Gutenberg files contain non-novel content at the beginning and end.
I identified and removed this using:

- "*** START OF" as the beginning marker
- "*** END OF" as the ending marker

Text Preprocessing

Several preprocessing steps were applied to clean and structure the text:

- **(a) Lowercasing**
- The entire text was converted to lowercase to maintain consistency.
- **(b) Removing Punctuation**
- Regular expressions were used to retain only alphanumeric characters and whitespace.
- **(c) Whitespace Normalization**
- Multiple spaces and newlines were replaced with a single space to remove noise from the text.

Tokenization

Tokenization was performed using NLTK's prebuilt tokenizers:

Sentence Tokenization

➔ sent_tokenize() was applied to divide the novel into individual sentences.

Word Tokenization

➔ word_tokenize() was applied to split the cleaned text into word-level tokens. Non-alphabetic tokens were removed for better linguistic clarity.

Frequency Distribution

A FreqDist object was created to compute term frequencies.
The top-occurring words (after stopword removal) were printed, giving insight into:

- The novel's vocabulary
- The dominant themes
- Character names and repeated concepts

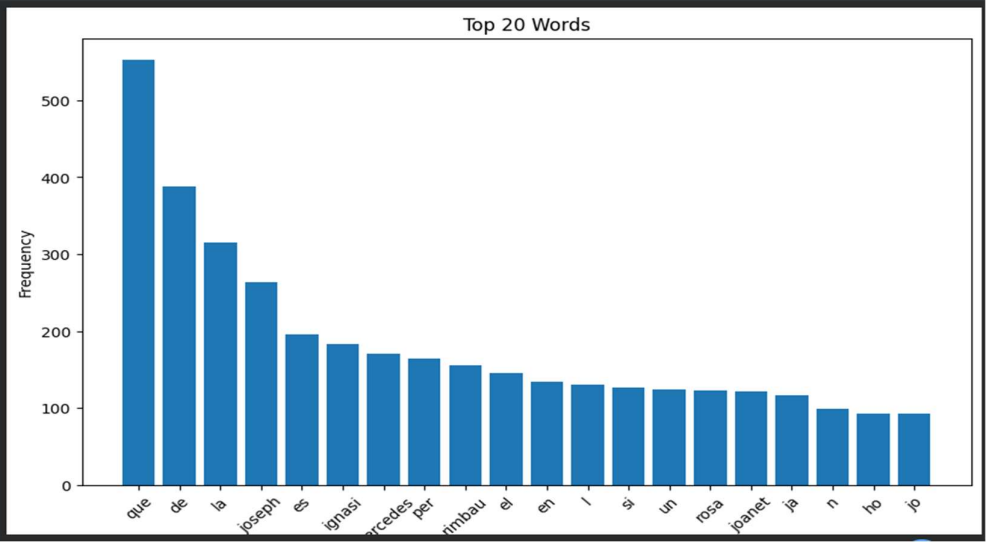
I used Matplotlib to clearly visualize the top 20 words which are occurring too often

```
import matplotlib.pyplot as plt
from nltk import FreqDist

fdist = FreqDist(words_no_sw)
top_words = fdist.most_common(20)

words, freqs = zip(*top_words)

plt.figure(figsize=(10,6))
plt.bar(words, freqs)
plt.xticks(rotation=45)
plt.title("Top 20 Words")
plt.xlabel("Words")
plt.ylabel("Frequency")
plt.show()
```



Named Entity Recognition (NER)

NER was performed using ne_chunk() on POS-tagged sentences.
Entities such as:

- **PERSON** (character names)
- **GPE** (cities, countries)
- **ORGANIZATION**

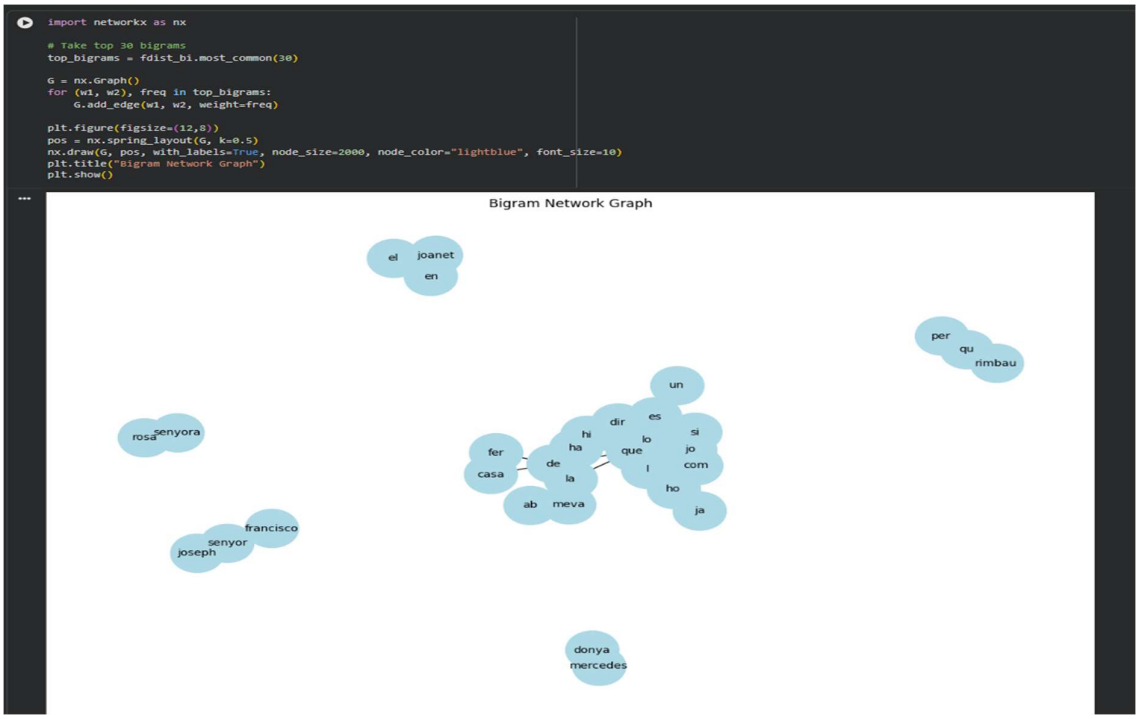
were extracted.
This step provides deeper semantic information about important figures and places mentioned in the novel.

Bigrams and Trigrams

To examine phrase-level patterns:

- Bigrams (2-word combinations)
- Trigrams (3-word combinations)
  The most frequent bigrams and trigrams show repeated phrases or writing patterns characteristic of the novel.

I generated visualisation for understanding the bigram words occurrence or distribution



Summary

- The novel contained thousands of sentences and tens of thousands of word tokens.
- After stop-word removal, the frequency distribution highlighted the most important vocabulary.
- Stemming grouped related words together.
- POS tagging revealed the mix of grammatical structures.
- NER identified key characters and locations.
- Bigrams and trigrams exposed frequent linguistic patterns.

Conclusion

This experiment demonstrated a complete NLP pipeline applied to a raw, unprocessed literary text. Starting from downloading the novel to performing linguistic analysis, every major step of text preprocessing was covered. The process shows how textual data can be converted into meaningful insights using NLTK, and highlights foundational NLP techniques such as tokenization, stop-word filtering, POS tagging and NER.