EDGE INTELLIGENCE LAB

G.PALLAVI(25MML0033)

Dataset Link : https://www.gutenberg.org/cache/epub/77444/pg77444.txt

Introduction

In this exercise, a novel from Project Gutenberg was taken as the dataset and processed using several Natural Language Processing (NLP) steps. The text was first downloaded and cleaned by removing unwanted parts and standardizing the formatting. After this, the data was prepared for analysis through important preprocessing tasks such as tokenizing sentences and words, eliminating stopwords, and applying stemming.
These steps helped convert the raw text into a usable form. Later, various analysis techniques like frequency counts, POS tagging, Named Entity Recognition (NER), and N-gram extraction were carried out to study the structure and linguistic patterns of the novel.

Approach

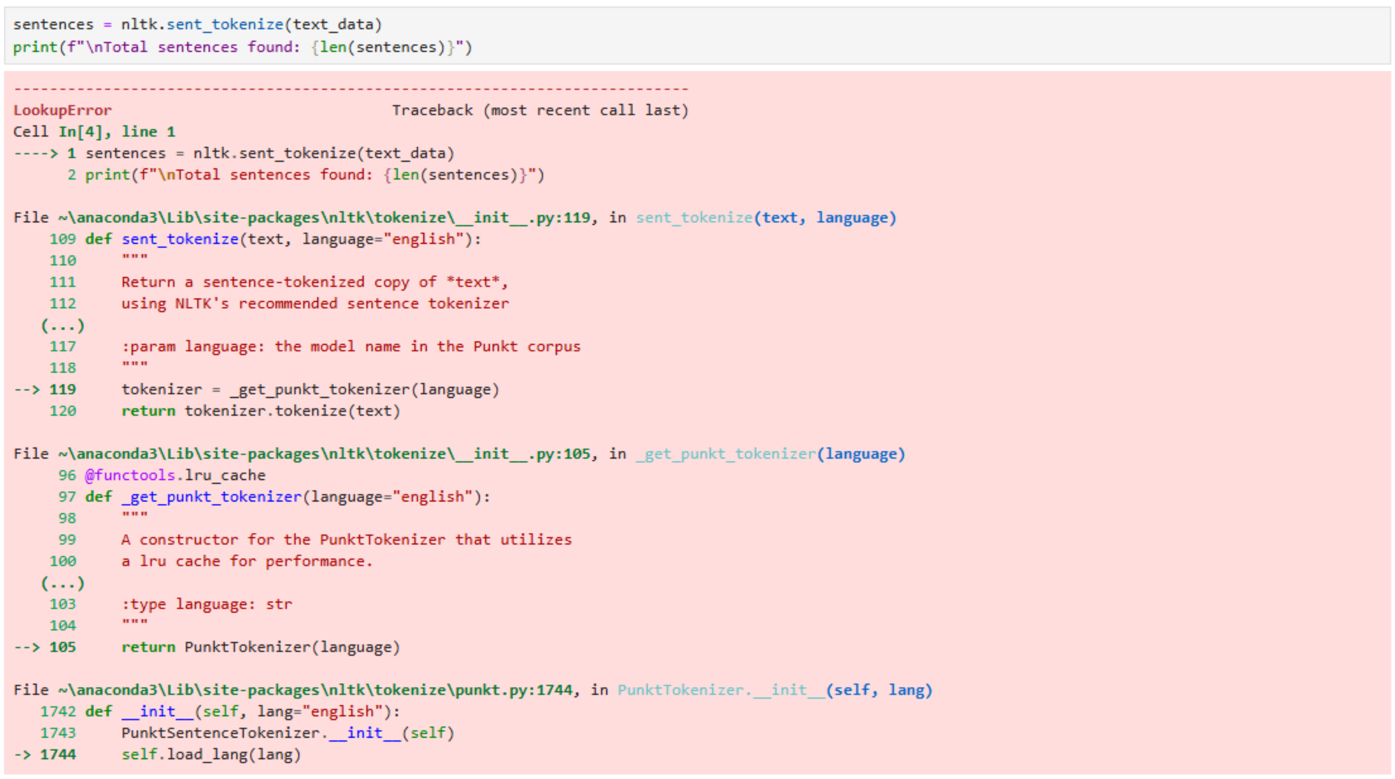
Handling NLTK Error Exceptions

While carrying out multiple NLP operations in the notebook, a few errors appeared because some necessary NLTK packages were not available. Each problem was identified and fixed one by one to ensure the workflow ran smoothly.

1. Error: Missing punkt tokenizer

Issue:
When running nltk.sent_tokenize(text_data), the program stopped and showed an error saying **punkt** was not available.
This happened because the tokenizer model required for sentence splitting was not downloaded.

FIX : Install the missing resource using: nltk.download("punkt")

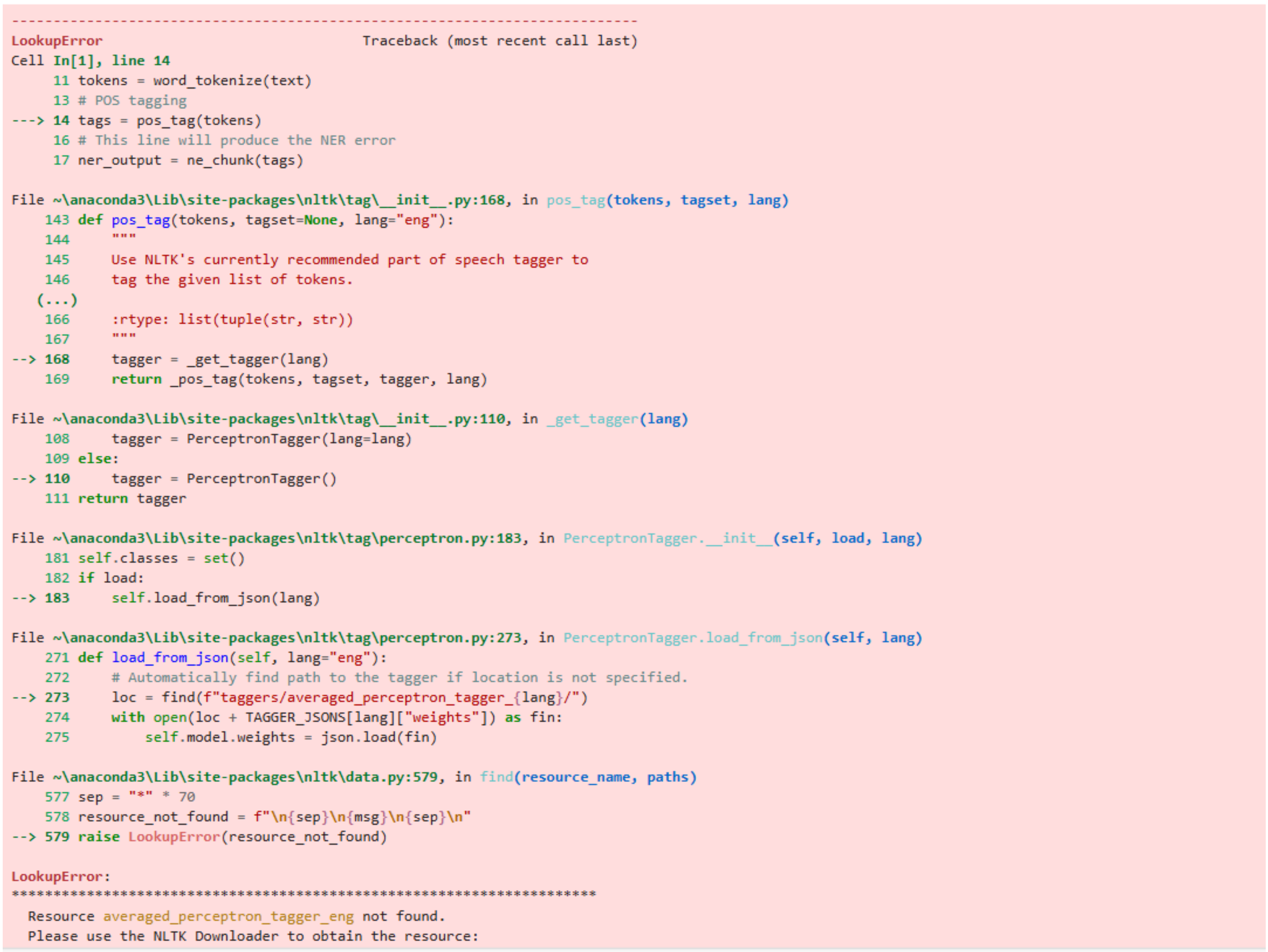


```
sentences = nltk.sent_tokenize(text_data)
print(f"\nTotal sentences found: {len(sentences)}")
```

```
---------------------------------------------------------------------------
LookupError                               Traceback (most recent call last)
Cell In[4], line 1
----> 1 sentences = nltk.sent_tokenize(text_data)
      2 print(f"\nTotal sentences found: {len(sentences)}")

File ~\anaconda3\Lib\site-packages\nltk\tokenize\__init__.py:119, in sent_tokenize(text, language)
    109 def sent_tokenize(text, language="english"):
    110     """
    111     Return a sentence-tokenized copy of *text*,
    112     using NLTK's recommended sentence tokenizer
   (...)
    117     :param language: the model name in the Punkt corpus
    118     """
--> 119     tokenizer = _get_punkt_tokenizer(language)
    120     return tokenizer.tokenize(text)

File ~\anaconda3\Lib\site-packages\nltk\tokenize\__init__.py:105, in _get_punkt_tokenizer(language)
     96 @functools.lru_cache
     97 def _get_punkt_tokenizer(language="english"):
     98     """
     99     A constructor for the PunktTokenizer that utilizes
    100     a lru cache for performance.
   (...)
    103     :type language: str
    104     """
--> 105     return PunktTokenizer(language)

File ~\anaconda3\Lib\site-packages\nltk\tokenize\punkt.py:1744, in PunktTokenizer.__init__(self, lang)
   1742 def __init__(self, lang="english"):
   1743     PunktSentenceTokenizer.__init__(self)
-> 1744     self.load_lang(lang)
```

2. Named Entity Recognition (NER) Error

When attempting to run the NER step using nltk.ne_chunkthe execution stopped with a LookupError.
The error message showed that NLTK could not find the maxent_ne_chunker_tab file, which is required for loading the pre-trained named entity recognition model.
Without this resource, NER could not identify persons, locations, or organizations in the text.



```
---------------------------------------------------------------------------
LookupError                               Traceback (most recent call last)
Cell In[1], line 14
     11 tokens = word_tokenize(text)
     13 # POS tagging
---> 14 tags = pos_tag(tokens)
     16 # This line will produce the NER error
     17 ner_output = ne_chunk(tags)

File ~\anaconda3\Lib\site-packages\nltk\tag\__init__.py:168, in pos_tag(tokens, tagset, lang)
    143 def pos_tag(tokens, tagset=None, lang="eng"):
    144     """
    145     Use NLTK's currently recommended part of speech tagger to
    146     tag the given list of tokens.
   (...)
    166     :rtype: list(tuple(str, str))
    167     """
--> 168     tagger = _get_tagger(lang)
    169     return _pos_tag(tokens, tagset, tagger, lang)

File ~\anaconda3\Lib\site-packages\nltk\tag\__init__.py:110, in _get_tagger(lang)
    108     tagger = PerceptronTagger(lang=lang)
    109 else:
--> 110     tagger = PerceptronTagger()
    111 return tagger

File ~\anaconda3\Lib\site-packages\nltk\tag\perceptron.py:183, in PerceptronTagger.__init__(self, load, lang)
    181 self.classes = set()
    182 if load:
--> 183     self.load_from_json(lang)

File ~\anaconda3\Lib\site-packages\nltk\tag\perceptron.py:273, in PerceptronTagger.load_from_json(self, lang)
    271 def load_from_json(self, lang="eng"):
    272     # Automatically find path to the tagger if location is not specified.
--> 273     loc = find(f"taggers/averaged_perceptron_tagger_{lang}/")
    274     with open(loc + TAGGER_JSONS[lang]["weights"]) as fin:
    275         self.model.weights = json.load(fin)

File ~\anaconda3\Lib\site-packages\nltk\data.py:579, in find(resource_name, paths)
    577 sep = "*" * 70
    578 resource_not_found = f"\n{sep}\n{msg}\n{sep}\n"
--> 579 raise LookupError(resource_not_found)

LookupError:
**********************************************************************
  Resource averaged_perceptron_tagger_eng not found.
  Please use the NLTK Downloader to obtain the resource:
```

Solution : nltk.download("maxent_ne_chunker_tab")

Summary :

- A novel from Project Gutenberg was downloaded and prepared by removing unnecessary portions and organizing the text into a clean format.
- The text underwent essential preprocessing steps such as converting to lowercase, eliminating punctuation, and performing sentence and word tokenization.
- The dataset was further improved by filtering out stopwords and applying stemming techniques to simplify the vocabulary.
- Various analytical methods like word frequency computation, POS tagging, and Named Entity Recognition were used to understand how words function and identify important entities in the text.
- Additional exploration was done using N-grams and concordance views to observe repeated patterns and understand words in their surrounding context.
- Multiple NLTK-related errors were encountered and successfully resolved by installing the necessary language resources, allowing the entire workflow to run without interruptions.

Conclusion:

This work demonstrates a complete end-to-end NLP pipeline applied to a real text, starting from downloading and preparing the data to performing deeper linguistic analysis.
The combination of preprocessing steps and different NLP techniques—such as frequency analysis, POS tagging, entity extraction, and N-gram study—offered a detailed understanding of the text's structure and linguistic behavior. By resolving the missing NLTK resource errors, the analysis became smooth and reliable. Overall, this workflow forms a strong base for extracting meaningful insights from textual data and for progressing toward more advanced NLP or machine-learning applications.