

✓ Practical 1 - Text Processing

NLP Lab Task: Text Preprocessing with NLTK

The goal is to explore core NLTK preprocessing functions in Python. You will implement: Tokenization, Stopword Removal, Stemming, Lemmatization


Instructions

You are required to **visit and study** the contents from the following links. Then, create a **Jupyter Notebook** (`.ipynb`) where you implement and demonstrate the concepts covered in these tutorials using sample text(s) of your choice.

Resources to Study

Go through the following tutorials and implement the concepts covered:

1. [NLTK Tokenization Tutorial](#)
2. [Stopwords in NLTK](#)
3. [Stemming in NLTK](#)
4. [Lemmatization in NLTK](#)

 Postlab Assignment: In addition to implementing the tasks using **NLTK**, we encourage you to explore and experiment with other popular NLP libraries such as:

- [SpaCy](#)
- [TextBlob](#)
- [Gensim](#)
- [Transformers by HuggingFace](#)

Try replicating some preprocessing steps using any one of these libraries and observe the differences in functionality, ease of use, and output.

✓ Complete the Code: Preprocessing with NLTK

In this activity, some lines of code are intentionally left incomplete. Your task is to **fill in the #### placeholders** with the correct Python statements or function names based on your understanding of NLTK preprocessing techniques.

Instructions:

Read the following code and complete the placeholders.

- Replace all ##### with appropriate code.
- Run each block after completion and verify the output.
- Add comments wherever necessary to explain your logic.
- Use the NLTK documentation or tutorials if needed.

Let's see how well you've understood tokenization, stopwords removal, stemming, and more!

```
import nltk
nltk.download('punkt_tab')

#The Natural Language Toolkit (NLTK) is a platform used for
##building Python programs that work with human language data for applying in statistical
nltk.download('punkt') #punkt is the required package for tokenization.
nltk.download('wordnet') #WordNet is a lexical database for the English language, which w
nltk.download('averaged_perceptron_tagger') #averaged_perceptron_tagger is used for taggi
nltk.download('words')
import pandas as pd
import numpy as np

#
# Upload the file to your Colab environment:
# Click the folder icon in the left sidebar
# Click "Upload to session storage"
# Select your moby.txt file
# this file is available in classroom, download it in local machine

#The file will be accessible at /content/moby.txt
# If you would like to work with the raw text you can use 'moby_raw'
with open('/content/moby.txt', 'r') as f:
    moby_raw = f.read()

# If you would like to work with the novel in nltk.Text format you can use 'text1'
moby_tokens = nltk.word_tokenize(moby_raw)
text1 = nltk.Text(moby_tokens)
print(list(text1))

[ntlk_data] Downloading package punkt_tab to /root/nltk_data...
[ntlk_data] Unzipping tokenizers/punkt_tab.zip.
[ntlk_data] Downloading package punkt to /root/nltk_data...
[ntlk_data] Unzipping tokenizers/punkt.zip.
[ntlk_data] Downloading package wordnet to /root/nltk_data...
[ntlk_data] Downloading package averaged_perceptron_tagger to
[ntlk_data] /root/nltk_data...
[ntlk_data] Unzipping taggers/averaged_perceptron_tagger.zip.
[ntlk_data] Downloading package words to /root/nltk_data...
[ntlk_data] Unzipping corpora/words.zip.
['\ufe0f', '[', 'Moby', 'Dick', 'by', 'Herman', 'Melville', '1851', ']', 'ETYMOLOGY',
```

✓ Q1

How many tokens (words and punctuation symbols) are in text1?

This function should return an integer.

```
def total_tokens():  
    return len(moby_tokens)  
  
total_tokens()  
  
160751
```

✓ Q2

How many unique tokens (unique words and punctuation) does text1 have?

This function should return an integer.

```
def unique_tokens():  
    return len(set(moby_tokens))  
  
unique_tokens()  
  
16030
```

✓ Q3

After lemmatizing the verbs, how many unique tokens does text1 have?

This function should return an integer.

```
from nltk.stem import WordNetLemmatizer  
nltk.download('omw-1.4')# Open Multilingual Wordnet ("OMW")  
  
def verb_lemmatization(text1):  
    lemmatizer = WordNetLemmatizer()  
    lemmatized = (lemmatizer.lemmatize(w,'v') for w in text1)  
  
    return len(set(lemmatized))  
  
verb_lemmatization(text1)
```

```
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
13133
```

✓ Q4

What is the lexical diversity of the given text input? (i.e. ratio of unique tokens to the total number of tokens)

This function should return a float.

```
def diversity():

    return unique_tokens()/total_tokens()

diversity()

0.09971944186972398
```

✓ Q5

What percentage of tokens is 'whale' or 'Whale'?

This function should return a float.

```
def ratio():

    dist = nltk.FreqDist(text1) #The class FreqDist works like a dictionary where the
                                # keys are the words in the text and
                                # the values are the count(frequency) associated with tha

    return (dist['whale']+dist['Whale']) / len(nltk.word_tokenize(moby_raw)) * 100 # Your

ratio()

0.41492743435499624
```

The NLTK FreqDist class is used to encode “frequency distributions”, which count the number of times that something occurs, for example a token.

Its `most_common()` method returns a list of tuples where each tuple is of the form (token, frequency). The list is sorted in descending order of frequency.

✓ Q7

What tokens have a length of greater than 5 and frequency of more than 150?

This function should return an alphabetically sorted list of the tokens that match the above constraints. To sort your list, use `sorted()`

```
def word_list():

    dist = nltk.FreqDist(text1)
    vocab1 = dist.keys()


    freqwords = []
    for token in dist:
        if dist[token]>150 and len(token) > 5:
            freqwords.append(token)

    #freqwords = [##### if len(token) > 5 and freq > 150]
    return sorted(freqwords) # Your answer here

word_list()

['Captain', 'Queequeg', 'before', 'little', 'seemed', 'though']
```

Post Lab Activity (Pair Work)

 Based on the paper: *Evaluating the Effectiveness of Text Pre-Processing in Sentiment Analysis* by Palomino & Aider (2022)

You are required to read the paper provided and answer the following questions in **pairs**. Write answers in your own words, citing sections from the paper where necessary.

◆ Part 1: Introduction & Motivation

1. What challenges in social media text make pre-processing essential for sentiment analysis?
 2. Why did the authors choose Twitter as the platform for their study over others like Facebook?
-

◆ Part 2: Literature Review and Gaps

3. What does existing literature say about the importance of pre-processing in sentiment

analysis?

4. Which specific gaps in current pre-processing research does this paper aim to address?

◆ Part 3: Pre-processing Techniques

5. What are the 12 pre-processing components used in this study, and how does each contribute to sentiment analysis? The 12 pre-processing components are:

Lowercasing Converts all characters to lowercase to reduce variability and improve token matching.

Removal of URLs and Twitter features Removes URLs (which don't contribute to sentiment) and Twitter-specific metadata like RT and hashtags (but retains hashtag content words).

Removal of unnecessary spaces Cleans up excess spaces that don't carry sentiment but could interfere with tokenization.

Removal of punctuation Deletes punctuation (after emoticons are processed) to streamline text for analysis.

Negation handling Transforms negated expressions (e.g., not good → bad) to preserve sentiment polarity.

Stop-word removal Eliminates common words (like the, is) that are usually sentiment-neutral to reduce noise.

Emoticons and emojis translation Converts emoticons (e.g., :) and emojis into sentiment-bearing words using a dictionary.

Acronym and slang expansion Replaces microtext (e.g., lol, brb) with their full forms to clarify sentiment meaning.

Spelling correction Fixes misspellings to align words with lexicon entries used in sentiment detection.

Tokenisation Breaks text into individual words or tokens, forming the basis for most NLP analysis.

Short-word removal Removes words with 1–2 characters which usually don't contribute meaningful sentiment.

Lemmatisation Reduces words to their base form (e.g., running → run) to group different forms of a word under one term.

6. Why was lemmatization included in some flows and excluded from others, and what hypothesis was tested regarding it? Lemmatization was included in some flows and

hypothesis was tested regarding it? Lemmatization was included in some flows and excluded from others to test its real impact on sentiment analysis performance.

Hypothesis tested: The researchers hypothesized that lemmatization might not significantly improve sentiment classification accuracy, because while it helps with standardizing word forms for tasks like information retrieval, it does not necessarily change the sentiment conveyed (e.g., happy, happily, happiness all carry similar sentiment).

Findings: Their results confirmed the hypothesis: Lemmatization did not notably improve accuracy. In fact, Flow 4, which excluded both lemmatization and spelling correction, produced the best results, especially for their naïve Bayes classifier.

◆ Part 4: Pre-processing Flows

7. What are the five pre-processing flows constructed in the study, and how do they differ from each other?
 8. Why is the order of pre-processing components significant in determining classifier performance?
-

◆ Part 5: Evaluation and Results

9. How did each pre-processing flow impact the performance of the sentiment classifiers on the full dataset and the gold standard?
 10. Which classifier showed the highest sensitivity to pre-processing, and why?
-

◆ Part 6: Interpretation & Implications

11. What conclusions were drawn about the effectiveness of Flow 4, and what implications does this have for future sentiment analysis?
 12. What are the main limitations identified in the study, and what future directions do the authors suggest?
-



Submission Instructions:

- Work in pairs and submit a single document per team.
- You may use bullet points or short paragraphs for clarity.
- Include references to relevant figures or tables where applicable.

