Experiment No.2
Apply Tokenization on given English and Indian Language Text
Date of Performance:
Date of Submission:

Aim: Apply Tokenization on given English and Indian Language Text

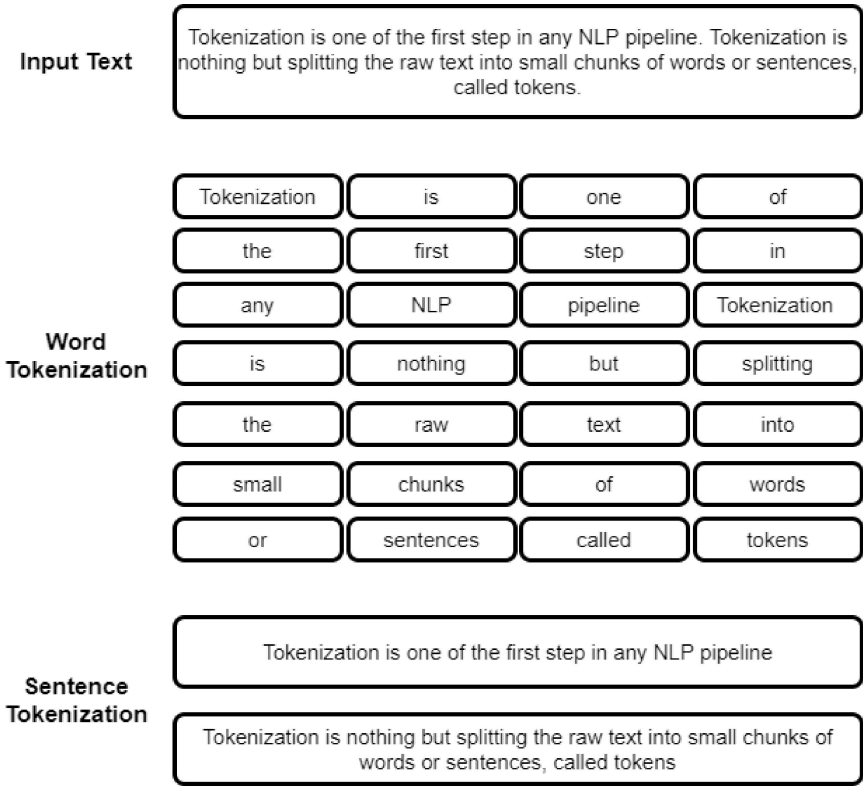
Objective: Able to perform sentence and word tokenization for the given input text for English and Indian Language.

Theory:

Tokenization is one of the first steps in any NLP pipeline. Tokenization is nothing but splitting the raw text into small chunks of words or sentences, called tokens. If the text is split into words, then it's called 'Word Tokenization' and if it's split into sentences then it's called 'Sentence Tokenization'. Generally 'space' is used to perform the word tokenization and characters like 'periods, exclamation point and newline char are used for Sentence Tokenization. We have to choose the appropriate method as per the task in hand. While performing the tokenization of a few characters like spaces, punctuations are ignored and will not be part of the final list of tokens.

Why Tokenization is Required?

Every sentence gets its meaning by the words present in it. So by analyzing the words present in the text we can easily interpret the meaning of the text. Once we have a list of words we can also use statistical tools and methods to get more insights into the text. For example, we can use word count and word frequency to find out the importance of words in that sentence or document.



Code:

```
import nltk

nltk.download()

from nltk.tokenize import sent_tokenize

text = '''Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book.'''

text

sentences = sent_tokenize (text)

sentencesfrom nltk.tokenize import word_tokenize

words = word_tokenize (text)

words

for w in words:

    print (w)

sent_tokenize (text)

[word_tokenize (text) for t in sent_tokenize(text)]

from nltk.tokenize import wordpunct_tokenize

wordpunct_tokenize (text)
```

SENTENCE TOKENIZATION(*OUTPUT)

```
[ ] sentences

    ['Lorem Ipsum is simply dummy text of the printing and typesetting industry.',
     "Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when an unknown printer took a galley of type and scrambled it to make a type specimen book."]
```

WORD TOKENIZATION(*OUTPUT)

```
words

['Lorem',
 'Ipsum',
 'is',
 'simply',
 'dummy',
 'text',
 'of',
 'the',
 'printing',
 'and',
 'typesetting',
 'industry',
 '.',
 'Lorem',
 'Ipsum',
 'has',
 'been',
 'the',
 'industry',
 "'s",
 'standard',
 'dummy',
 'text',
 'ever',
 'since',
 'the',
 '1500s',
 ',',
 'when',
```

Conclusion:

Here are some common tools and techniques used for tokenization of Indian language input:

- Whitespace Tokenization: This is a basic form of tokenization where text is split based on spaces. While this method is not language-specific, it can work reasonably well for languages with clear word boundaries, such as Hindi or Tamil.
- Language-Specific Libraries: Many Indian languages have their own tokenization libraries. For example, Python libraries like Indic NLP and Malayalam Morphological Analyzer offer tokenization tools for Indian languages.
- NLP Libraries: General-purpose NLP libraries like NLTK, SpaCy, and Hugging Face's Transformers provide tokenizers that support various languages, including some Indian languages. These libraries often require language-specific models for better performance.
- Custom Rule-Based Tokenization: Developing custom tokenization rules based on the linguistic characteristics of the specific Indian language can be effective. Regular expressions and rule-based approaches can be employed for this purpose.
- Machine Learning Tokenization: Machine learning models like Byte Pair Encoding (BPE) and WordPiece can be trained to tokenize text effectively. Transformers-based models, such as the ones developed by Facebook AI (e.g., mBERT and XLM-R), can be fine-tuned for tokenization of Indian languages.
- Pre-trained Language Models: Many pre-trained language models, like BERT and GPT, have tokenizers that work well for a wide range of languages, including some Indian languages. Fine-tuning these models on Indian language-specific data can improve their performance.
- Hybrid Approaches: In some cases, a combination of methods might work best. For instance, you can start with a general-purpose tokenizer and then apply custom rules or language-specific models to handle nuances.