Experiment No. 2
Apply Various text preprocessing techniques tokenization and stop word removal.
Date of Performance:
Date of Submission:

Exp. No.: 2

Title: Apply Various text preprocessing techniques tokenization and stop word removal.

Theory:

The process of converting data to something a computer can understand is referred to as pre-processing.

Tokenization in Text Preprocessing:

Tokenization is a fundamental step in Natural Language Processing (NLP), where a large piece of text is broken down into smaller units called tokens. These tokens can be words, subwords, or even characters, depending on the level of granularity required. Tokenization simplifies the analysis of text by making it easier for machines to process and understand.

Definition: Tokenization is the process of splitting a text into smaller, manageable units (tokens).

- **Word-level Tokenization**: Divides text into individual words.
- **Subword Tokenization**: Breaks down words into meaningful subword units (used in models like BERT).
- **Character-level Tokenization**: Treats each character as a token.

Input to Models: Tokenization is essential for feeding data into NLP models.

Handling Variability: Deals with issues like punctuation, contractions, and special symbols.

Basis for Further Preprocessing: After tokenization, other steps like stopword removal, stemming, and lemmatization can be performed.

Types of Tokenization:

1. **Whitespace Tokenization**:
    - The simplest method that splits tokens based on spaces.
    - Fast but doesn't handle punctuation well.

2. **Punctuation-Based Tokenization**:
    - Uses punctuation as delimiters for tokenization.
    - Handles punctuation effectively but may split relevant subparts (e.g., hyphenated words).
3. **Regular Expression Tokenization**:
    - Uses patterns to define how to split text into tokens (e.g., splitting by spaces, punctuation, or digits).
    - Flexible but requires custom regex patterns.
4. **Subword Tokenization (Byte-Pair Encoding, WordPiece)**:
    - Breaks down rare or unknown words into smaller subword units.
    - Useful in transformer models like BERT and GPT.

Stop Word Removal in Text Preprocessing:

Stopwords are common words in any language (e.g., "the," "in," "a") that are typically filtered out in Natural Language Processing (NLP) tasks. They don't contribute significantly to the meaning of the text and are often removed to improve model efficiency.

Why Remove Stopwords?

- **Reduced dataset size**: Fewer words mean faster model training.
- **Increased accuracy**: Focus shifts to more meaningful tokens.
- **Improved search results**: Search engines like Google exclude stopwords for faster data retrieval.

Use Cases for Stopword Removal:

- **Text Classification** (e.g., Spam Filtering, Language, and Genre Classification)
- **Caption Generation**

- **Auto-Tag Generation**

When to Avoid Stopword Removal:

- Machine Translation
- Language Modeling
- Text Summarization
- Question-Answering

Methods for Removing Stopwords:

1. **NLTK**: A popular library offering stopword lists in 16 languages.
2. **spaCy**: Efficient stopword removal via the STOP_WORDS class.
3. **Gensim**: Provides the remove_stopwords method for preprocessing.

Code:

```
!pip install nltk
import nltk
nltk.download('punkt')
from nltk.tokenize import sent_tokenize

text = "Hello everyone. Welcome to College. You are studying NLP article."
sent_tokenize(text)
```

```
['Hello everyone.', 'Welcome to College.', 'You are studying NLP article.']
```

```
import nltk.data

# Loading PunktSentenceTokenizer using English pickle file
tokenizer = nltk.data.load('tokenizers/punkt/PY3/english.pickle')
tokenizer.tokenize(text)
```

```
['Hello everyone.', 'Welcome to College.', 'You are studying NLP article.']
```

```
import nltk.data

spanish_tokenizer = nltk.data.load('tokenizers/punkt/PY3/spanish.pickle')
```

```
text = 'Hola amigo. Estoy bien.'
spanish_tokenizer.tokenize(text)
```

```
['Hola amigo.', 'Estoy bien.']
```

```
from nltk.tokenize import word_tokenize

text = "Hello everyone. Welcome to college."
word_tokenize(text)
```

```
['Hello', 'everyone', '.', 'Welcome', 'to', 'college', '.']
```

```
from nltk.tokenize import TreebankWordTokenizer

tokenizer = TreebankWordTokenizer()
tokenizer.tokenize(text)
```

```
'Hello', 'everyone.', 'Welcome', 'to', 'college', '.']
```

```
from nltk.tokenize import RegexpTokenizer

tokenizer = RegexpTokenizer(r'\w+')
text = "Let's see how it's working."
tokenizer.tokenize(text)
```

```
['Let', 's', 'see', 'how', 'it', 's', 'working']
```

Conclusion:

In conclusion, text prepossessing techniques like tokenization and stop word removal are crucial steps in preparing text data for natural language processing tasks. Tokenization helps break down text into manageable units, whether at the word, subword, or character level, enabling machines to better process and analyze text. Stopword removal further refines the dataset by eliminating common, non-essential words, thereby enhancing model performance and efficiency. Together, these techniques lay the foundation for more advanced processing and improved accuracy in NLP applications.