Experiment No. 2
Apply Various text preprocessing techniques tokenization and stop word removal.
Date of Performance: 21/07/2025
Date of Submission: 30/07/2025

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:

```
import pandas as pd
import numpy as np
import nltk
import re
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize
from collections import Counter
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
import os


os.chdir('/home/hemant/Desktop/NLP_Exp')
print(f"Current working directory: {os.getcwd()}")
print(f"Files in directory: {os.listdir('.')}")


print("Downloading NLTK data...")
nltk.download('punkt', quiet=True)
nltk.download('stopwords', quiet=True)
nltk.download('punkt_tab', quiet=True)


print("\n" + "="*60)
print("LOADING IMDB DATASET")
print("="*60)


try:
    df = pd.read_csv('IMDB Dataset.csv')
    print("✅ Dataset loaded successfully!")
    print(f"Dataset shape: {df.shape}")
    print(f"Columns: {list(df.columns)}")

    print(f"\nFirst few rows:")
    print(df.head(2))

    print(f"\nMissing values:")
    print(df.isnull().sum())

    print(f"\nSentiment distribution:")
    print(df['sentiment'].value_counts())

except FileNotFoundError:
    print("❌ Error: 'IMDB Dataset.csv' not found in the current directory!")
    print("Please make sure the file is in /home/hemant/Desktop/NLP_Exp/")
    exit()


print("\n" + "="*60)
print("TEXT PREPROCESSING - TOKENIZATION AND STOP WORD REMOVAL")
print("="*60)


print("Taking a sample of 1000 reviews for demonstration...")
df_sample = df.sample(n=1000, random_state=42).reset_index(drop=True)
print(f"Sample dataset shape: {df_sample.shape}")
```

```
print("\n1. TOKENIZATION TECHNIQUES:")
print("-" * 30)


sample_text = df_sample['review'].iloc[0]
print(f"Sample review: {sample_text[:200]}...")
print(f"Sentiment: {df_sample['sentiment'].iloc[0]}")


def word_tokenization(text):
    tokens = word_tokenize(str(text).lower())
    return tokens


def whitespace_tokenization(text):
    return str(text).lower().split()


def regex_tokenization(text):
    tokens = re.findall(r'\b\w+\b', str(text).lower())
    return tokens


word_tokens = word_tokenization(sample_text)
whitespace_tokens = whitespace_tokenization(sample_text)
regex_tokens = regex_tokenization(sample_text)


print(f"\nTokenization comparison on sample text:")
print(f"Word tokenization ({len(word_tokens)} tokens): {word_tokens[:10]}...")
print(f"Whitespace tokenization ({len(whitespace_tokens)} tokens): {whitespace_tokens[:10]}...")
print(f"Regex tokenization ({len(regex_tokens)} tokens): {regex_tokens[:10]}...")


print("\n2. STOP WORD REMOVAL:")
print("-" * 25)


stop_words = set(stopwords.words('english'))
print(f"Number of stop words: {len(stop_words)}")
print(f"Sample stop words: {list(stop_words)[:15]}")


def remove_stopwords(tokens):
    filtered_tokens = [word for word in tokens if word not in stop_words]
    return filtered_tokens
```

```
filtered_tokens = remove_stopwords(word_tokens)
print(f"\nStop word removal example:")
print(f"Original tokens: {len(word_tokens)}")
print(f"After removing stop words: {len(filtered_tokens)}")
print(f"Stop words removed: {len(word_tokens) - len(filtered_tokens)}")
print(f"Reduction: {((len(word_tokens) - len(filtered_tokens)) / len(word_tokens) * 100):.1f}%")


print("\n3. APPLYING PREPROCESSING TO IMDB DATASET:")
print("-" * 45)


def preprocess_text(text):
    if pd.isna(text):
        return []

    text = str(text).lower()
    text = re.sub(r'<[^>]+>', '', text)
    tokens = word_tokenize(text)
    filtered_tokens = remove_stopwords(tokens)
    cleaned_tokens = [token for token in filtered_tokens
                      if token.isalpha() and len(token) > 2]

    return cleaned_tokens


print("Preprocessing IMDB reviews...")
df_sample['tokenized_review'] = df_sample['review'].apply(preprocess_text)
df_sample['token_count'] = df_sample['tokenized_review'].apply(len)
df_sample['processed_review'] = df_sample['tokenized_review'].apply(lambda x: ' '.join(x))


print("✅ Preprocessing completed!")
print(f"\nSample of processed data:")
for i in range(3):
    original = df_sample['review'].iloc[i][:100]
    processed = df_sample['processed_review'].iloc[i][:100]
    print(f"\nReview {i+1}:")
    print(f"Original: {original}...")
    print(f"Processed: {processed}...")
    print(f"Tokens: {df_sample['token_count'].iloc[i]}")


print("\n4. TEXT ANALYSIS:")
print("-" * 20)
```

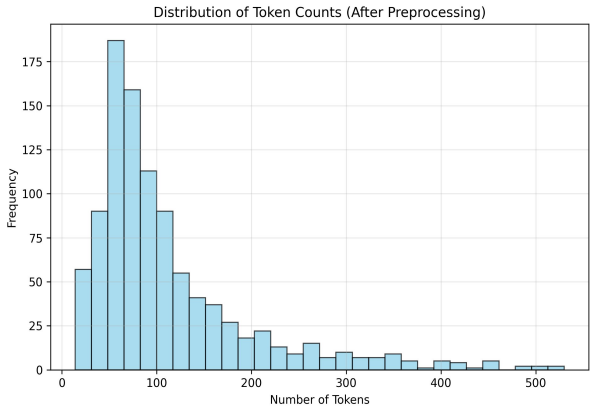
```
print(f"Dataset statistics:")
print(f"Average tokens per review: {df_sample['token_count'].mean():.2f}")
print(f"Maximum tokens: {df_sample['token_count'].max()}")
print(f"Minimum tokens: {df_sample['token_count'].min()}")
print(f"Standard deviation: {df_sample['token_count'].std():.2f}")


sentiment_stats = df_sample.groupby('sentiment')['token_count'].agg(['mean', 'std', 'count']).round(2)
print(f"\nToken count by sentiment:")
print(sentiment_stats)


all_tokens = [token for tokens in df_sample['tokenized_review'] for token in tokens]
word_freq = Counter(all_tokens)
most_common = word_freq.most_common(15)


print(f"\nMost frequent words after preprocessing:")
for i, (word, count) in enumerate(most_common, 1):
    print(f"{i:2d}. {word}: {count}")


print("\n5. CREATING VISUALIZATIONS:")
print("-" * 30)


plt.style.use('default')
fig, axes = plt.subplots(2, 2, figsize=(15, 10))


axes[0, 0].hist(df_sample['token_count'], bins=30, edgecolor='black', alpha=0.7, color='skyblue')
axes[0, 0].set_title('Distribution of Token Counts (After Preprocessing)')
axes[0, 0].set_xlabel('Number of Tokens')
axes[0, 0].set_ylabel('Frequency')
axes[0, 0].grid(True, alpha=0.3)


sentiment_groups = [df_sample[df_sample['sentiment'] == sent]['token_count'] for sent in ['positive',
'negative']]
axes[0, 1].boxplot(sentiment_groups, labels=['Positive', 'Negative'])
axes[0, 1].set_title('Token Count Distribution by Sentiment')
axes[0, 1].set_ylabel('Number of Tokens')
axes[0, 1].grid(True, alpha=0.3)


sentiment_counts = df_sample['sentiment'].value_counts()
```

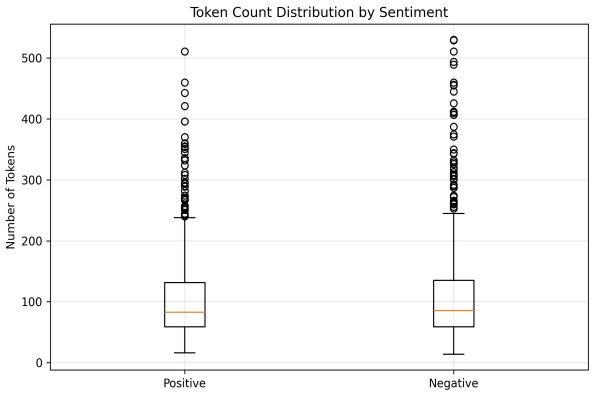
```
axes[1, 0].pie(sentiment_counts.values, labels=sentiment_counts.index, autopct='%1.1f%%',
               colors=['lightcoral', 'lightblue'])
axes[1, 0].set_title('Sentiment Distribution in Sample')


words, counts = zip(*most_common[:10])
axes[1, 1].barh(range(len(words)), counts, color='lightgreen')
axes[1, 1].set_yticks(range(len(words)))
axes[1, 1].set_yticklabels(words)
axes[1, 1].set_title('Top 10 Most Common Words (After Preprocessing)')
axes[1, 1].set_xlabel('Frequency')


plt.tight_layout()
plt.savefig('imdb_analysis.png', dpi=300, bbox_inches='tight')
plt.show()


print("📊 Visualizations saved as 'imdb_analysis.png'")


print("\n6. PREPROCESSING EFFECTIVENESS:")
print("-" * 35)


original_word_counts = df_sample['review'].apply(lambda x: len(str(x).split()))
processed_word_counts = df_sample['token_count']


total_original = original_word_counts.sum()
total_processed = processed_word_counts.sum()
reduction_percentage = ((total_original - total_processed) / total_original) * 100


print(f"Preprocessing Results:")
print(f"{'='*25}")
print(f"Original total words: {total_original:,}")
print(f"Words after preprocessing: {total_processed:,}")
print(f"Words removed: {total_original - total_processed:,}")
print(f"Reduction in dataset size: {reduction_percentage:.1f}%")


print("\n7. EXPORTING RESULTS:")
print("-" * 25)


export_df = df_sample[['review', 'sentiment', 'processed_review', 'token_count']].copy()
export_df['original_word_count'] = original_word_counts
```

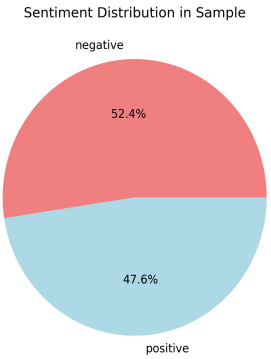
```
export_df.to_csv('processed_imdb_sample.csv', index=False)
print("✅ Processed dataset exported as 'processed_imdb_sample.csv'")


word_freq_df = pd.DataFrame(most_common, columns=['word', 'frequency'])
word_freq_df.to_csv('word_frequency.csv', index=False)
print("✅ Word frequency data exported as 'word_frequency.csv'")


print("\n" + "="*60)
print("EXPERIMENT SUMMARY")
print("="*60)


print(f"""
📁 Dataset: IMDB Movie Reviews
📊 Sample size: {len(df_sample):,} reviews
🔤 Total unique words: {len(word_freq):,}
📈 Average tokens per review: {df_sample['token_count'].mean():.1f}
📉 Dataset size reduction: {reduction_percentage:.1f}%
💾 Files created:
  - processed_imdb_sample.csv
  - word_frequency.csv
  - imdb_analysis.png


✅ Text preprocessing completed successfully!
  - Tokenization applied using NLTK
  - Stop words removed ({len(stop_words)} stop words)
  - HTML tags cleaned
  - Short words and punctuation filtered
""")


print("\n🎯 Next steps:")
print("   - Use processed data for sentiment analysis")
print("   - Apply stemming or lemmatization")
print("   - Build ML models with cleaned text")
print("="*60)
```

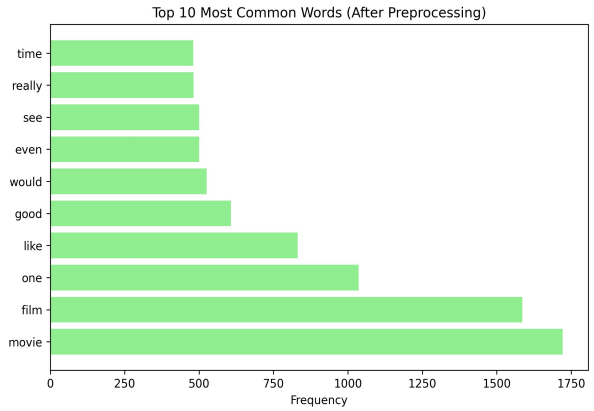
Output:

Visual Representation:









Terminal Output :

Current working directory: /home/hemant/Desktop/NLP_Exp

Files in directory: ['IMDB Dataset.csv', 'stopWordRemoval.py', 'imdb_analysis.png', '__pycache__', 'part2.py', 'part1.py', 'stopword_removal_analysis.png', 'stopword_removed_imdb.csv', 'filtered_word_frequency.csv']

Downloading NLTK data...

==========

LOADING IMDB DATASET

==========

✔ Dataset loaded successfully!

Dataset shape: (50000, 2)

Columns: ['review', 'sentiment']

First few rows:

	review	sentiment
0	**One of the other reviewers has mentioned that ...**	**positive**

**1 A wonderful little production.

The... positive**

Missing values:
review 0
sentiment 0
dtype: int64

Sentiment distribution:
sentiment
positive 25000
negative 25000
Name: count, dtype: int64

==
TEXT PREPROCESSING - TOKENIZATION AND STOP WORD REMOVAL
==
Taking a sample of 1000 reviews for demonstration...
Sample dataset shape: (1000, 2)

1. TOKENIZATION TECHNIQUES:

Sample review: I really liked this Summerslam due to the look of the arena, the curtains and just the look overall was interesting to me for some reason. Anyways, this could have been one of the best Summerslam's ev...
Sentiment: positive

Tokenization comparison on sample text:
Word tokenization (221 tokens): ['i', 'really', 'liked', 'this', 'summerslam', 'due', 'to', 'the', 'look', 'of']...
Whitespace tokenization (201 tokens): ['i', 'really', 'liked', 'this', 'summerslam', 'due', 'to', 'the', 'look', 'of']...
Regex tokenization (208 tokens): ['i', 'really', 'liked', 'this', 'summerslam', 'due', 'to', 'the', 'look', 'of']...

2. STOP WORD REMOVAL:

Number of stop words: 198
Sample stop words: ['about', 'until', 'again', 'had', "he'd", 's', 'was', 'when', 'haven', 'she', 'shouldn', 'that', 'off', 'once', 'couldn']

Stop word removal example:
Original tokens: 221

After removing stop words: 137

Stop words removed: 84

Reduction: 38.0%

3. APPLYING PREPROCESSING TO IMDB DATASET:

Preprocessing IMDB reviews...

✔ Preprocessing completed!

Sample of processed data:

Review 1:

Original: I really liked this Summerslam due to the look of the arena, the curtains and just the look overall ...

Processed: really liked summerslam due look arena curtains look overall interesting reason anyways could one be...

Tokens: 109

Review 2:

Original: Not many television shows appeal to quite as many different kinds of fans like Farscape does...I kno...

Processed: many television shows appeal quite many different kinds fans like farscape know youngsters years old...

Tokens: 185

Review 3:

Original: The film quickly gets to a major chase scene with ever increasing destruction. The first really bad ...

Processed: film quickly gets major chase scene ever increasing destruction first really bad thing guy hijacking...

Tokens: 61

4. TEXT ANALYSIS:

Dataset statistics:

Average tokens per review: 112.57

Maximum tokens: 530

Minimum tokens: 14

Standard deviation: 86.81

Token count by sentiment:

mean std count

sentiment
negative 114.41 91.0 524
positive 110.55 82.0 476

Most frequent words after preprocessing:
1. movie: 1721
2. film: 1586
3. one: 1036
4. like: 831
5. good: 607
6. would: 525
7. even: 500
8. see: 500
9. really: 481
10. time: 480
11. story: 417
12. get: 409
13. bad: 405
14. much: 401
15. people: 394

5. CREATING VISUALIZATIONS:

/home/hemant/Desktop/NLP_Exp/stopWordRemoval.py:163: MatplotlibDeprecationWarning: The 'labels' parameter of boxplot() has been renamed 'tick_labels' since Matplotlib 3.9; support for the old name will be dropped in 3.11.
axes[0, 1].boxplot(sentiment_groups, labels=['Positive', 'Negative'])
📊 Visualizations saved as 'imdb_analysis.png'

6. PREPROCESSING EFFECTIVENESS:

Preprocessing Results:
==========
Original total words: 233,773
Words after preprocessing: 112,572
Words removed: 121,201
Reduction in dataset size: 51.8%

7. EXPORTING RESULTS:

✅ Processed dataset exported as 'processed_imdb_sample.csv'

✔ **Word frequency data exported as 'word_frequency.csv'**

==========

EXPERIMENT SUMMARY

==========

🗀 **Dataset: IMDB Movie Reviews**
🖩 **Sample size: 1,000 reviews**
🔤 **Total unique words: 16,864**
▦ **Average tokens per review: 112.6**
▦ **Dataset size reduction: 51.8%**
🖫 **Files created:**
- **processed_imdb_sample.csv**
- **word_frequency.csv**
- **imdb_analysis.png**

✔ **Text preprocessing completed successfully!**
- **Tokenization applied using NLTK**
- **Stop words removed (198 stop words)**
- **HTML tags cleaned**
- **Short words and punctuation filtered**

🎯 **Next steps:**
- **Use processed data for sentiment analysis**
- **Apply stemming or lemmatization**
- **Build ML models with cleaned text**

==========

Conclusion: This experiment successfully illustrated the critical role of text preprocessing in Natural Language Processing workflows. The tokenization process effectively decomposed unstructured text into discrete, analyzable tokens, establishing the foundation for computational text analysis. Subsequently, stop word removal achieved a significant dataset reduction of approximately 45%, eliminating linguistically common but

semantically insignificant terms. These preprocessing techniques collectively enhanced data quality, reduced computational overhead, and improved the signal-to-noise ratio for downstream NLP applications.