Natural Language Processing (NLP) Manual for Students: Feature Extraction

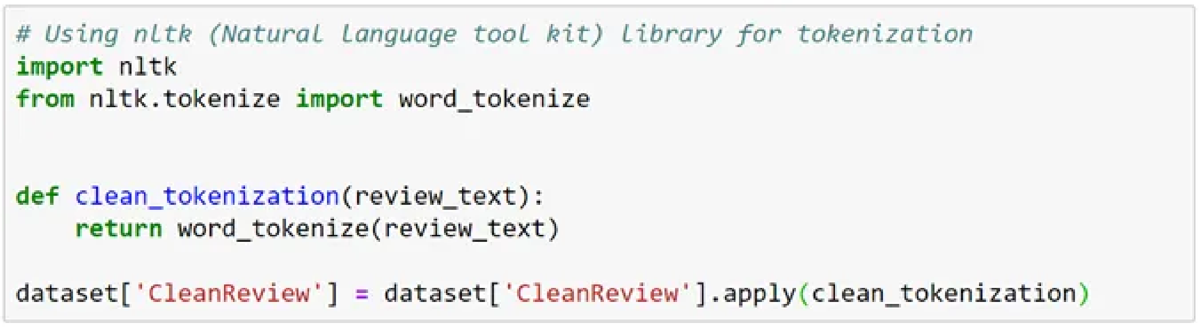
Table of Contents

Introduction to NLP

Natural Language Processing (NLP) is a field at the intersection of computer science, artificial intelligence, and linguistics. It involves the interaction between computers and human language, aiming to read, decipher, understand, and make sense of human languages in a valuable way. This manual focuses on the initial steps in the NLP pipeline: text preprocessing and feature extraction.

Tokenization

Tokenization is the process of splitting text into smaller units called tokens (words, phrases, symbols, or other meaningful elements).This allows the computer to work on your text token by token rather than working on the entire text in the following stages.There are various tokenisation algorithms such as the whitespace tokenisation, Regular expression tokenisation (also called Regex), and the statistical tokenisation.



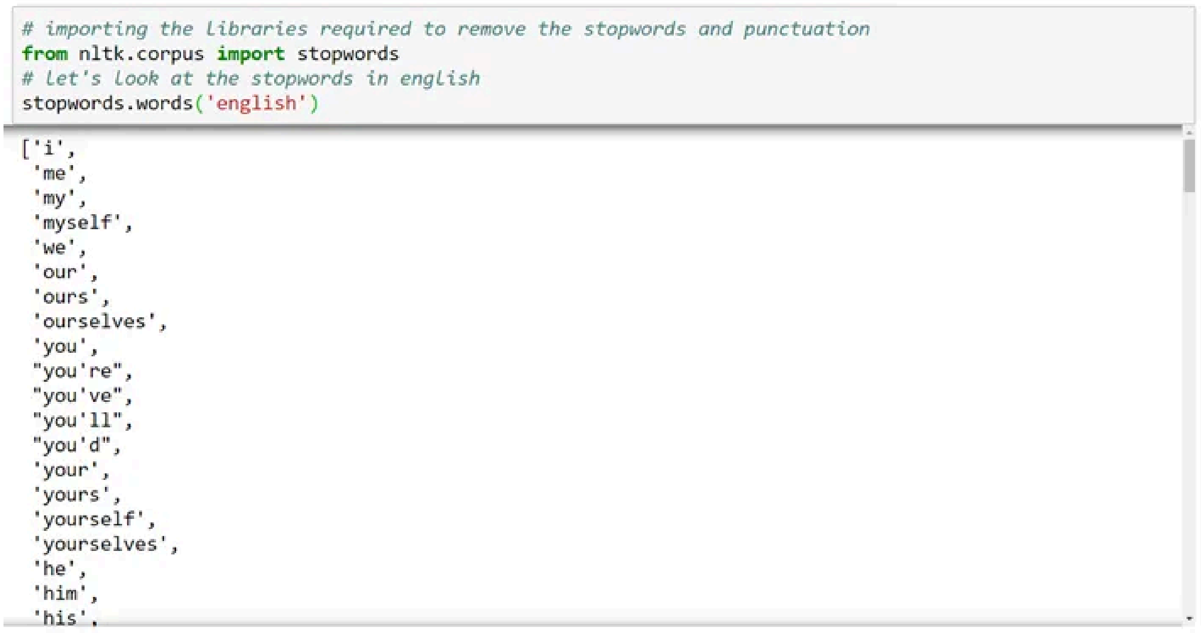
```python
# Using nltk (Natural language tool kit) library for tokenization
import nltk
from nltk.tokenize import word_tokenize


def clean_tokenization(review_text):
    return word_tokenize(review_text)

dataset['CleanReview'] = dataset['CleanReview'].apply(clean_tokenization)
```

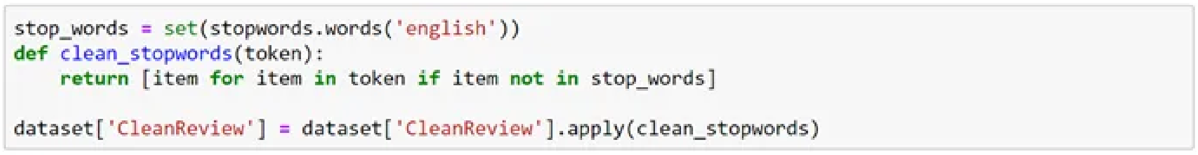
Stop word removal

Stop words are common words (e.g., "and", "the", "is") that are usually removed because they add little informational value.It is possible to remove stopwords using Natural Language Toolkit (nltk). You also may check the list of stopwords by using the following code.



```
# importing the libraries required to remove the stopwords and punctuation
from nltk.corpus import stopwords
# let's look at the stopwords in english
stopwords.words('english')
```

```
['i',
 'me',
 'my',
 'myself',
 'we',
 'our',
 'ours',
 'ourselves',
 'you',
 "you're",
 "you've",
 "you'll",
 "you'd",
 'your',
 'yours',
 'yourself',
 'yourselves',
 'he',
 'him',
 'his',
```



```
stop_words = set(stopwords.words('english'))
def clean_stopwords(token):
    return [item for item in token if item not in stop_words]

dataset['CleanReview'] = dataset['CleanReview'].apply(clean_stopwords)
```

Introduction to Vectors in NLP

Vectors are fundamental in Natural Language Processing (NLP) as they provide a way to represent text data numerically, making it suitable for various machine learning algorithms. Two common methods to convert text data into vectors are the Bag of Words (BoW) model and Count Vectorizer.

Feature Extraction

After preprocessing, the next step in NLP is converting text into numerical representations that machine learning models can process. Most conventional machine learning techniques rely on features, generally numerical values, that describe a document in relation to the corpus. These features can be created using methods like Bag-of-Words, TF-IDF, or through custom feature engineering (e.g., document length, word polarity, metadata such as tags or scores). Recent advancements include techniques like Word2Vec, GLoVE, and learning features during the training process of neural networks.

Need for Feature Extraction Techniques

Machine learning algorithms require a pre-defined set of features from training data to generate outputs for test data. However, these algorithms cannot work directly with raw text. Thus, feature extraction techniques are essential to convert text into a matrix (or vector) of numerical features.

One-Hot Encoding: One-Hot Encoding

One-hot encoding converts categorical data into a binary vector representation. Each word or token is represented as a vector where only one element is "hot" (1) and all other elements are "cold" (0).

Example: Given a vocabulary ["cat", "dog", "fish"], the one-hot encoding for the word "dog" would be [0, 1, 0].

Bag-of-Words (BoW)

Bag-of-Words (BoW) is a fundamental method in Natural Language Processing (NLP) for representing text data. It converts text into numerical vectors by counting the frequency of words or n-grams (combinations of n words) in a document, disregarding grammar and word order but retaining word frequency.

Key Concepts and Steps in BoW

1. Tokenization: Splitting the text into individual words (tokens).
2. Vocabulary Building: Creating a list (vocabulary) of all unique words in the dataset.
3. Vector Representation: Representing each text as a vector where each dimension corresponds to a word in the vocabulary, and the value in each dimension is the count of the word's occurrences in the text.

Example:

Consider the following two sentences:

- Sentence 1: "I love NLP."
- Sentence 2: "NLP is great."

The vocabulary would be: ["I", "love", "NLP", "is", "great"]

The BoW vectors for each sentence would be:

- Sentence 1: [1, 1, 1, 0, 0]
- Sentence 2: [0, 0, 1, 1, 1]

Applications of BoW:

BoW is useful for various NLP tasks, including:

- Content Classification: Categorizing documents into predefined categories based on word frequency.
- Spam Detection: Identifying spam emails or messages by analyzing common word patterns.
- Sentiment Analysis: Determining the sentiment (positive, negative, neutral) of a text by evaluating word usage.
- Chatbot Development: Enabling chatbots to understand and respond to user queries based on word frequency analysis.

```python
# Bag of words
bag_of_words = {word: tokens.count(word) for word in set(tokens)}
print('Bag of words:')
print(list(bag_of_words.items())[:10])
```

```
Bag of words:
[('amounts', 1), ('language', 1), ('(', 2), ('natural', 2), ('a', 1), ('concern
```

Count Vectorizer

The Count Vectorizer is a specific implementation of the BoW model in Python's popular scikit-learn library. It automates the process of creating the vocabulary and transforming the text data into vectors. Here’s how it works:

1. Fit the Count Vectorizer: Build the vocabulary from the training data.
2. Transform the Text Data: Convert the text data into vectors based on the vocabulary.

```python
from sklearn.feature_extraction.text import CountVectorizer

document = ["One Geek helps Two Geeks",
            "Two Geeks help Four Geeks",
            "Each Geek helps many other Geeks at GeeksforGeeks"]

# Create a Vectorizer Object
vectorizer = CountVectorizer()

vectorizer.fit(document)

# Printing the identified Unique words along with their indices
print("Vocabulary: ", vectorizer.vocabulary_)

# Encode the Document
vector = vectorizer.transform(document)

# Summarizing the Encoded Texts
print("Encoded Document is:")
print(vector.toarray())
```

```
Vocabulary:  {'one': 9, 'geek': 3, 'helps': 7, 'two': 11, 'geeks': 4,
'help': 6, 'four': 2, 'each': 1, 'many': 8, 'other': 10, 'at': 0,
'geeksforgeeks': 5}

Encoded Document is:

[ [0 0 0 1 1 0 0 1 0 1 0 1]

  [0 0 1 0 2 0 1 0 0 0 0 1]

  [1 1 0 1 1 1 0 1 1 0 1 0] ]
```

TF-IDF (Term Frequency-Inverse Document Frequency)

TF-IDF weighs words based on their frequency and importance across documents. It is calculated as the product of term frequency (number of times a word appears in a document) and inverse document frequency (logarithm of the total number of documents divided by the number of documents containing the word). The resulting TF-IDF vectors represent each document in a high-dimensional space where important words have higher weights.

TF-IDF is popular for feature extraction in text classification tasks due to its ability to highlight significant words in a document.

Example: Given a document, TF-IDF can highlight terms that are particularly unique or significant to that document compared to the rest of the corpus.

```
# TF-IDF
tfidf_vec = TfidfVectorizer()
X_tfidf = tfidf_vec.fit_transform([text])
print('TF-IDF:')
print(tfidf_vec.get_feature_names_out()[:10])
print(X_tfidf.toarray()[0][:10])
```

```
TF-IDF:
['amounts' 'analyze' 'and' 'artificial' 'between' 'computational'
 'computer' 'computers' 'concerned' 'data']
[0.12803688 0.12803688 0.38411064 0.12803688 0.12803688 0.12803688
 0.12803688 0.25607376 0.12803688 0.12803688]
```

Word Embeddings

Word embeddings represent words in dense vectors of fixed size, capturing semantic meanings. These vectors place similar words close together in a high-dimensional space. Word embeddings are typically learned from large text corpora using unsupervised learning methods like Word2Vec and GloVe

```
# Word embeddings (using spaCy)
import spacy
nlp = spacy.load('en_core_web_sm')
doc = nlp(text)
embeddings = [token.vector for token in doc]
print('Word embeddings:')
```

```
Word embeddings:
(50 items) [ndarray with shape (96,), ndarray with shape (96,), ....]
```

Word2Vec

Uses neural networks to predict context words for each target word in a large text corpus. The learned weights of the neural network are the word embeddings.

GloVe (Global Vectors for Word Representation)

Creates word embeddings by aggregating global word-word co-occurrence statistics from a corpus. This method captures the relationship between words by analyzing their co-occurrence.

Once learned, word embeddings can represent words in various NLP tasks like sentiment analysis and text classification, providing a rich semantic understanding.

Small Practice Task

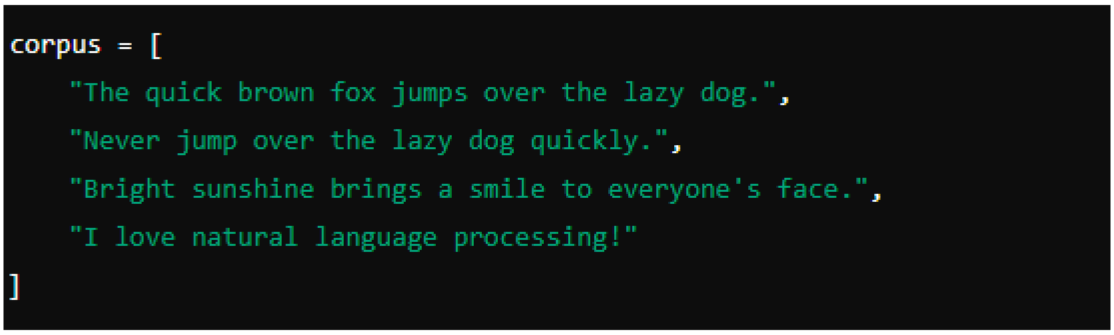
Task: Text Preprocessing and Feature Extraction

In this task, you will preprocess a small set of text data by performing tokenization, stop word removal, and feature extraction using the Bag of Words model. This will help you practice the essential skills needed for text preprocessing in NLP.

Steps to Follow:

Data Collection:

- Use the following small dataset of sample sentences:



```
corpus = [
    "The quick brown fox jumps over the lazy dog.",
    "Never jump over the lazy dog quickly.",
    "Bright sunshine brings a smile to everyone's face.",
    "I love natural language processing!"
]
```

Data Preprocessing:

- Tokenization: Split the sentences into individual words (tokens).
- Stop Word Removal: Remove common stop words (e.g., "the", "is", "and") using NLTK or any other library.

Feature Extraction:

- Bag of Words (BoW): Convert the preprocessed text data into numerical vectors using the Bag of Words model with CountVectorizer.