

Text Normalization and Tokenization

Why we do Tokenization and normalization?

- **Tokenization** helps in segmenting the text into smaller units (tokens) that are easier for machines to process.
- **Normalization** cleans and standardizes the text, reducing variability and improving consistency, enabling better performance in downstream tasks.

- **Better Matching and Search:** Normalization makes it easier to match similar words or phrases in tasks like information retrieval or search engines. For example, **normalizing "USA," "U.S.A.," and "United States"** to the same form ensures that all variations are recognized as the same entity.
- **Handling Noisy Data:** **In real-world text, especially from social media or informal sources, there may be misspellings, abbreviations, or other noise.** Normalization helps clean up this data so that it can be more effectively processed by NLP models.

Text normalization in Natural Language Processing (NLP) is the process of transforming text into a standardized format to improve its consistency and usability for further processing. This step is crucial for making text data comparable and analyzable, especially when dealing with diverse and unstructured data sources.

Key Components of Text Normalization

1. Lowercasing:

1. Converts all characters in the text to lowercase to ensure that words are treated uniformly regardless of their case.
2. Example: "The Cat Sat on the MAT" → "the cat sat on the mat"

2. Removing Punctuation:

1. Eliminates punctuation marks to focus on the core text, which helps in simplifying the analysis.
2. Example: "Hello, world!" → "Hello world"

Lowercasing text in Python is a straightforward process, and it can be easily done using built-in string methods. Here's how you can implement lowercasing for text in Python:

Basic Example

You can use the `lower()` method provided by Python's string class to convert text to lowercase.

```
# Example text
text = "Hello, World! This is a Test."

# Convert to lowercase
lowercased_text = text.lower()

print(lowercased_text)
```

Output:

hello, world! this is a test.

Lowercasing Multiple Texts

If you have a list of texts and want to convert each to lowercase, you can use a list comprehension.

```
# List of texts
texts = ["Hello, World!", "Python is Great!", "Text Normalization"]

# Convert each text to lowercase
lowercased_texts = [text.lower() for text in texts]

print(lowercased_texts)
```

Output:

```
['hello, world!', 'python is great!', 'text normalization']
```

Lowercasing text in Python can be done simply using the `lower()` method for individual strings, list comprehensions for multiple strings, or pandas methods for DataFrames. For more complex preprocessing, you can combine lowercasing with regular expressions or other string manipulation techniques.

Removing punctuation is a common step in text normalization for NLP tasks. This helps standardize the text by focusing on the words themselves. Here's how you can remove punctuation in Python to normalize text:

Using `str.translate()` with `str.maketrans()`

This method is efficient and straightforward:

```
import string

def normalize_text(text):
    # Create a translation table that maps each punctuation character to None
    translator = str.maketrans("", "", string.punctuation)
    # Translate the text using the translation table
    return text.translate(translator)

# Sample text
text = "Hello, world! How's everything?"
normalized_text = normalize_text(text)
print(normalized_text) # Output: Hello world Hows everything
```

Using Regular Expressions (re module)

For more control, such as removing punctuation while preserving certain characters, you can use regular expressions:

```
import re

def normalize_text(text):
    # Remove punctuation using regular expressions
    return re.sub(r'^\w\s', '', text)

# Sample text
text = "Hello, world! How's everything?"
normalized_text = normalize_text(text)
print(normalized_text) # Output: Hello world Hows everything
```

Using List Comprehension

For a more manual approach, you can filter out punctuation characters:

```
import string

def remove_punctuation(text):
    return ''.join(char for char in text if char not in string.punctuation)

# Example usage
text = "Hello, world! How's everything?"
cleaned_text = remove_punctuation(text)
print(cleaned_text) # Output: Hello world Hows everything
```

Using str.replace() Method

If you want to manually remove specific punctuation characters:

```
def remove_punctuation(text):
    for char in ",.!?:;\'":
        text = text.replace(char, "")
    return text

# Example usage
text = "Hello, world! How's everything?"
cleaned_text = remove_punctuation(text)
print(cleaned_text) # Output: Hello world Hows everything
```

Using a Custom Function with Set Operations

For a more advanced approach, you can use set operations to filter out punctuation:

```
import string

def remove_punctuation(text):
    punctuation_set = set(string.punctuation)
    return ''.join(char for char in text if char not in punctuation_set)

# Example usage
text = "Hello, world! How's everything?"
cleaned_text = remove_punctuation(text)
print(cleaned_text) # Output: Hello world Hows everything
```

- **Tokenization:**

- Splits text into individual tokens or words, which is essential for most NLP tasks.
- Example: "Text normalization is crucial." → ["Text", "normalization", "is", "crucial"]

- **Removing Stop Words:**

- Filters out common words that may not contribute significant meaning to the text analysis, such as "and," "the," "is."
- Example: "The quick brown fox jumps over the lazy dog" → "quick brown fox jumps lazy dog"

Tokenization is the process of splitting text into individual units, such as words or sentences. In Python, there are several methods and libraries available for tokenization. Here's a rundown of some common techniques and libraries:

1. Using Built-in String Methods

For basic tokenization, you can use Python's built-in string methods:

Word Tokenization

```
text = "Hello, world! How's everything?"
```

```
# Basic word tokenization
tokens = text.split()
print(tokens) # Output: ['Hello,', 'world!', "How's",
'everything?']
```

2. Using nltk (Natural Language Toolkit)

The nltk library provides more advanced tokenization tools.

Word Tokenization

```
import nltk  
nltk.download('punkt')  
from nltk.tokenize import word_tokenize  
  
text = "Hello, world! How's everything?"  
tokens = word_tokenize(text)  
print(tokens) # Output: ['Hello', ',', 'world', '!', 'How', "'s",  
'everything', '?']
```

Using spaCy

The spaCy library offers efficient tokenization and other NLP features.

Word Tokenization

```
import spacy

# Load the spaCy model
nlp = spacy.load('en_core_web_sm')

text = "Hello, world! How's everything?"
doc = nlp(text)
tokens = [token.text for token in doc]
print(tokens) # Output: ['Hello', ',', 'world', '!', 'How', "'", "s",
'everything', '?']
```

Sentence Tokenization

```
import spacy

# Load the spaCy model
nlp = spacy.load('en_core_web_sm')

text = "Hello, world! How's everything?"
doc = nlp(text)
sentences = [sent.text for sent in doc.sents]
print(sentences) # Output: ['Hello, world!', "How's everything?"]
```

Using TextBlob

TextBlob is another library that provides simple and effective tokenization.

Word Tokenization

```
from textblob import TextBlob  
  
text = "Hello, world! How's everything?"  
blob = TextBlob(text)  
tokens = blob.words  
print(tokens) # Output: WordList(['Hello', 'world', 'How', 's',  
'everything'])
```

Sentence Tokenization

```
from textblob import TextBlob

text = "Hello, world! How's everything?"
blob = TextBlob(text)
sentences = blob.sentences
print(sentences) # Output: [Sentence('Hello, world!'),
Sentence("How's everything?")]
```

- **Stemming:**

- Reduces words to their base or root form by removing suffixes. For example, "running," "runner," and "runs" are reduced to "run."
- Example: "running" → "run"

- **Lemmatization:**

- Similar to stemming but more advanced, lemmatization reduces words to their lemma or dictionary form, considering the context and grammatical role.
- Example: "running" → "run" (verb), "better" → "good" (adjective)

- **Handling Contractions:**

- Expands contractions to their full forms to maintain consistency in text.
- Example: "don't" → "do not"

- **Normalization of Variants:**

- Converts different textual variants of the same entity to a standard form.
- Example: "USA," "U.S.," and "United States" → "United States"

- **Removing Extra Whitespace:**

- Trims excessive whitespace from the text to avoid inconsistencies.
- Example: "This is a test." → "This is a test."

Purpose of Text Normalization

- **Consistency:** Ensures uniformity in the text data, which is critical for reliable analysis and model training.
- **Accuracy:** Helps in improving the accuracy of NLP models by reducing variability in the text data.
- **Efficiency:** Simplifies text processing by focusing on the core content and reducing noise.

Summary

Text normalization is a preprocessing step in NLP that involves converting text into a consistent and standardized format. It includes operations such as lowercasing, punctuation removal, tokenization, and lemmatization, among others. By normalizing text, you prepare it for more effective and accurate analysis and modeling.

Implementation of normalization and tokenization

```
dictionary = {
    "can't": "cannot",
    "won't": "will not",
    "they're": "they are",
    "I'm": "I am",
    "it's": "it is",
    "U.S.A.": "USA" # Add more contractions as needed
}
import re
def varda(a,dictionary):

    #expand contractions
    for i,j in dictionary.items():
        if i in a:
            a=re.sub(i,j,a)
```

```
#remove punctuations
t=re.sub(r'\W',' ',a)
#remove extra white space
t=re.sub(r'\s+',' ',t)
#lower case
t=t.lower()
#tokenization
t=t.split()

return t
```

Output:

```
a="Ok, so “literature with a capital L” , I can't WAIT to go U.S.A. it's , I'm"
print(varda(a,dictionary))

['ok', 'so', 'literature', 'with', 'a', 'capital', 'l', 'i', 'cannot', 'wait', 'to', 'go', 'usa', 'it', 'is', 'i', 'am']
```

Thank You