NLP2: Mastering Text Handling: Types of Sentences and Preprocessing Steps in NLP

Saket Kumar

5 min read · Sep 3, 2024

Types of Data in NLP

In Natural Language Processing (NLP), the text we work with can vary widely in form and complexity. Different types of sentences and text structures pose unique challenges that need to be addressed during preprocessing. Below are some common types of text data you may encounter, along with the specific challenges they present:

1. HTML Tags and Markup

Example: `<p>This is a paragraph of text that contains <strong>bold</strong> and <em>italic</em> elements.</p>`

Challenge: Handling text with embedded HTML or XML tags requires stripping out the tags or correctly parsing the content.

2. Case Sensitivity

Example: `The CEO announced a new product today. the ceo later explained the details in a press release.`

Challenge: Inconsistent use of uppercase and lowercase letters may affect entity recognition or sentiment analysis.

3. Mentions and Handles

Example: `Great work on the project, @User123! Looking forward to the next steps. #TeamWork`

Challenge: Mentions, handles, and hashtags are common in social media text and need to be processed appropriately, often by removing or analyzing them separately.

4. URLs and Hyperlinks

Example: `Check out this article: https://www.example.com/blog/nlp-intro`

Challenge: URLs may need to be removed, replaced with a placeholder, or analyzed for link text.

5. Emojis and Special Characters

Example: `I love this product! 😍👍 #Amazing`

Challenge: Emojis and special characters can carry significant sentiment or meaning and may require special handling.

6. Spelling Errors and Typos

Example: `I recieved the pacakge yestarday, but it was dammaged.`

Challenge: Misspelled words and typos can interfere with text analysis and need correction through spell-checking or context-based correction.

7. Abbreviations and Acronyms

Example: `The company's ROI has been increasing since Q1 FY2023.`

Challenge: Understanding abbreviations and acronyms requires either expansion or domain-specific knowledge.

8. Ambiguity and Sarcasm

Example: `Oh, great, another delay. Just what we needed. 🙄`

Challenge: Sarcasm and ambiguous statements can be difficult for NLP models to interpret correctly.

9. Complex Sentences with Multiple Clauses

Example: `Although the meeting started late, it was productive and we managed to cover all the key points before the end.`

Challenge: Complex sentences with multiple clauses require the model to understand the relationships between different parts of the sentence.

10. Text with Noise (e.g., Ads, Disclaimers)

Example: `Special offer: Buy now and get 20% off! *Terms and conditions apply.`

Challenge: Noisy text, such as advertisements or disclaimers, can dilute the main content and may need to be filtered out.

Text Preprocessing: The Essential Steps for NLP Success

Before diving into Natural Language Processing (NLP) tasks, it's crucial to clean and preprocess your text data to ensure that your models perform effectively. Text data often comes in raw and unstructured forms, containing noise, inconsistencies, and various elements that may hinder accurate analysis. To address these challenges, here are 11 essential steps for text preprocessing, along with how each step can be implemented:

1. **Text Lowercasing**
   Convert all text to lowercase to ensure uniformity, making the processing case-insensitive.
   How to do it:
   In Python, you can use:

```
text = text.lower()
```

2. Remove HTML Tags

Strip out any HTML or XML tags to avoid processing non-text elements embedded in the content.

How to do it:

Using the `BeautifulSoup` library:

```
from bs4 import BeautifulSoup
text = BeautifulSoup(text, "html.parser").get_text()
```

3. Remove URLs

Identify and remove URLs from the text to eliminate irrelevant links.

How to do it:

Using regular expressions:

```
import re
text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
```

4. Remove Punctuation

Eliminate punctuation marks, which generally do not contribute meaningfully to text analysis.

How to do it:

Using the `string` module:

```
import string
text = text.translate(str.maketrans('', '', string.punctuation))
```

5. Tokenization

Split the text into individual words or tokens, forming the basic units for analysis.

How to do it:

Using the `nltk` library:

```
from nltk.tokenize import word_tokenize
tokens = word_tokenize(text)
```

6. Remove Stopwords

Filter out common stopwords (e.g., “and,” “the,” “is”) that do not add significant meaning to the text.

How to do it:

Using the `nltk` library:

```
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
tokens = [word for word in tokens if word not in stop_words]
```

7. Handle Special Characters and Numbers

Remove or normalize special characters, numbers, and symbols as they may not be relevant in certain analyses.

How to do it:

Using regular expressions:

```
text = re.sub(r'[^A-Za-z\s]', '', text)  # Removes anything that's not a letter
```

8. Spelling Correction

Correct misspelled words to improve the accuracy of the analysis.

How to do it:

Using the `TextBlob` library:

```
from textblob import TextBlob
text = str(TextBlob(text).correct())
```

9. Stemming and Lemmatization

Reduce words to their root or base form to ensure consistency (e.g., "running" → "run").

How to do it:

Using the `nltk` library for stemming:

```
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()
tokens = [stemmer.stem(word) for word in tokens]
```

Or lemmatization:

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
tokens = [lemmatizer.lemmatize(word) for word in tokens]
```

Original word: Running, Better and Studies

After Stemming: Run, Better and Studi.

After Lemmatization: Run, Good and Study.

10. Handling Contractions

Expand contractions to their full forms (e.g., “can’t” → “cannot”) to avoid ambiguity.

How to do it:

Using the `contractions` library:

```
import contractions
text = contractions.fix(text)
```

11. Text Normalization

Standardize text by removing extra spaces, handling accents, or normalizing slang to ensure uniformity across the dataset.

How to do it:

Remove extra spaces:

```
text = ' '.join(text.split())
```

Handling accents using `unidecode`:

```
from unidecode import unidecode
text = unidecode(text)
```

Here are some more lesser known:

12. Removing HTML Entities

HTML entities like `&`, `<`, and `>` can appear in text, especially when scraped from the web.

How to do it:

Use the `html` library in Python:

```
import html
text = html.unescape(text)
```

13. Handling Repeated Characters

Text from informal sources like social media may contain repeated characters (e.g., “loooove”). Reducing these to a single character can help.

How to do it:

Using regular expressions:

```
text = re.sub(r'(.)\1+', r'\1', text)
```

14. Removing Non-ASCII Characters

If your text data is in English, you might want to remove non-ASCII characters to avoid processing issues.

How to do it:

Using `unidecode` or by filtering:

```
text = text.encode("ascii", "ignore").decode()
```

15. Handling Negations

Negations can flip the meaning of a sentence (e.g., “not good”). Expanding them to “not_good” can help retain meaning.

How to do it:

Using custom rules:

```
text = re.sub(r'\b(not)\s+(\w+)', r'\1_\2', text)
```

16. Sentence Segmentation

If you are working with multi-sentence inputs, breaking text into individual sentences can improve analysis.

How to do it:

Using the `nltk` library:

```
from nltk.tokenize import sent_tokenize
sentences = sent_tokenize(text)
```

17. Handling Synonyms and Antonyms

Replacing synonyms with a common term or expanding antonyms can standardize the text.

How to do it:

Using the `WordNet` corpus in `nltk`:

```
from nltk.corpus import wordnet
# Example function to find synonyms
synonyms = wordnet.synsets("happy")
```

18. Removing Noise Words and Phrases

Some words or phrases may be irrelevant to your analysis (e.g., “click here”). Removing these noise elements can clean the data.

How to do it:

Using custom stopword lists:

```
noise_words = ["click here", "subscribe", "terms and conditions"]
tokens = [word for word in tokens if word not in noise_words]
```

19. Removing Outliers in Text Length

Texts that are unusually short or long compared to the rest of your dataset may be outliers and could be filtered out.

How to do it:

Setting length thresholds:

```
if len(tokens) < min_len or len(tokens) > max_len:
    # Handle outliers
```

20. Language Detection and Filtering

If your dataset contains multiple languages, filtering out non-target languages can improve analysis consistency.

How to do it:

Using the `langdetect` library:

```
from langdetect import detect
if detect(text) != 'en':
    # Handle non-English text
```

In Part 2, we cover different sentence types in NLP and essential text preprocessing steps. Learn how to handle various text structures and refine data through various techniques . Stay tuned for Part 3, where we'll dive deeper into advanced NLP techniques and applications.

NLP | Nlp Courses | Data Science | Text Preprocessing | Interview

Written by Saket Kumar

Edit profile

10 Followers

Seasoned mechanical engineer looking to get into Data Scientist role.

More from Saket Kumar

Saket Kumar

Saket Kumar