Natural Language Processing using Python Programming

Notebook 02.1: Text Preprocessing Fundamentals

Part of the comprehensive learning series: Natural Language Processing using Python Programming

Learning Objectives:

- Master essential text cleaning and normalization techniques
- Understand the difference between stemming and lemmatization
- Implement stopword removal and punctuation cleaning
- Build a comprehensive text preprocessing pipeline
- Apply best practices for preparing text data for machine learning

---

- Text preprocessing is the essential process of cleaning and normalizing raw text data into a standard, workable format.
- Without this step, NLP models treat 'Running', 'ran', and 'runs' as three separate, unrelated words, leading to poor analysis.
- This notebook covers the core normalization techniques.

1. Setting up: Libraries and Sample Text

- We will use `NLTK` for its wide range of stemming and stopword resources, and `SpaCy` for its industrial-strength lemmatizer.

In [1]:
```
# Import necessary libraries
import nltk
import spacy
import re # Regular Expressions for cleaning

# Ensure NLTK resources are available
nltk.download('stopwords', quiet=True)
nltk.download('wordnet', quiet=True)
nltk.download('punkt', quiet=True)

# Load SpaCy model
nlp = spacy.load('en_core_web_sm')
```

In [2]:
```
# Example raw text
raw_text = "The quick brown foxes are running, and they aren't stopping! " \
"They've ran 100 miles, but are still better than the competitors."
```

```
print(f"Raw Text: {raw_text}")
```

```
Raw Text: The quick brown foxes are running, and they aren't stopping! They've ran
100 miles, but are still better than the competitors.
```

---

2. Cleaning and Normalization

2.1 Lowercasing and Punctuation Removal

- **Lowercasing** reduces the vocabulary size by treating 'The' and 'the' as the same word.
- **Punctuation and Special Character Removal** eliminate noise that rarely carries semantic value in a statistical model.

In [3]:
```
# Lowercasing
text_lower = raw_text.lower()
print(f"Lowercased: {text_lower}")
```

```
Lowercased: the quick brown foxes are running, and they aren't stopping! they've ra
n 100 miles, but are still better than the competitors.
```

In [4]:
```
# Punctuation and Special Character Removal (using regex)
# [^a-z\s]: matches anything that is NOT a lowercase letter or whitespace
text_clean = re.sub(r'[^a-z\s]', '', text_lower)
print(f"Punctuation Removed: {text_clean}")
```

```
Punctuation Removed: the quick brown foxes are running and they arent stopping they
ve ran  miles but are still better than the competitors
```

2.2 Stopword Removal

- **Stopwords** are extremely common words (like 'a', 'is', 'the', 'and') that contribute little to the semantic meaning of a document.
- Removing them reduces the size of the feature space and speeds up training.
- We must first tokenize the clean text before removing stopwords.

In [5]:
```
# Import NLTK's stopword list and tokenizer
# Corpus is a collection of texts
# Tokenization is the process of splitting text into individual words or tokens
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

# Get the standard English stopword list
stop_words = set(stopwords.words('english'))

# 1. Tokenize the clean text
tokens = word_tokenize(text_clean)

# 2. Filter out the stopwords
filtered_tokens = [w for w in tokens if not w in stop_words]
```

```
print(f"Stopwords removed: {filtered_tokens}")
print(f"Reduction in size: {len(tokens)} -> {len(filtered_tokens)}")
```

```
Stopwords removed: ['quick', 'brown', 'foxes', 'running', 'arent', 'stopping', 'the
yve', 'ran', 'miles', 'still', 'better', 'competitors']
Reduction in size: 20 -> 12
```

---

3. Stemming vs. Lemmatization

- Both methods aim to reduce inflected words (words with prefixes/suffixes) to a base form, but they differ significantly in their approach and resulting quality.

3.1 Stemming (NLTK PorterStemmer)

- **Stemming** is a heuristic process that chops off the ends of words, often resulting in a root that is **not a true word**.
- It is fast but crude.

In [6]:
```
# Import NLTK's PorterStemmer for stemming
# PorterStemmer is a common stemming algorithm
from nltk.stem import PorterStemmer

# Create the stemmer object
stemmer = PorterStemmer()
words_to_stem = ['running', 'runs', 'ran', 'better', 'studies', 'studying', 'organ

stemmed_words = [stemmer.stem(w) for w in words_to_stem]

print("Word       -> Stem")
print("------------------")
for original, stemmed in zip(words_to_stem, stemmed_words):
    print(f"{original:<10} -> {stemmed}")

# Note: 'better' -> 'better', 'studies' -> 'studi' (not a real word!)
```

```
Word       -> Stem
------------------
running    -> run
runs       -> run
ran        -> ran
better     -> better
studies    -> studi
studying   -> studi
organization -> organ
```

3.2 Lemmatization (SpaCy)

- **Lemmatization** is a morphological analysis process that reduces a word to its **lemma** (dictionary form).
- It requires knowledge of the word's `Part-of-Speech (POS)` tag to be accurate and is generally preferred for high-quality NLP tasks.

In [7]:
```python
# SpaCy handles tokenization, POS tagging, and lemmatization in one go via the Doc
doc = nlp(raw_text)

print("Token    | POS Tag | Lemma")
print("---------|---------|--------")
for token in doc:
    # token.lemma_ provides the base form
    if not token.is_punct and not token.is_space:
        print(f"{token.text:<8} | {token.pos_:<7} | {token.lemma_}")

# Observation: 'running' -> 'run', 'aren't' -> 'be', 'ran' -> 'run', 'better' ->
# SpaCy provides dictionary-accurate base forms like 'well' for 'better', unlike s
```

```
Token    | POS Tag | Lemma
---------|---------|--------
The      | DET     | the
quick    | ADJ     | quick
brown    | ADJ     | brown
foxes    | NOUN    | fox
are      | AUX     | be
running  | VERB    | run
and      | CCONJ   | and
they     | PRON    | they
are      | AUX     | be
n't      | PART    | not
stopping | VERB    | stop
They     | PRON    | they
've      | AUX     | have
ran      | VERB    | run
100      | NUM     | 100
miles    | NOUN    | mile
but      | CCONJ   | but
are      | AUX     | be
still    | ADV     | still
better   | ADJ     | well
than     | ADP     | than
the      | DET     | the
competitors | NOUN    | competitor
```

3.3 Comparison and Best Practice

Feature	Stemming (NLTK)	Lemmatization (SpaCy)
Method	Rule-based (chopping)	Dictionary-based (morphological analysis)
Result	May not be a real word ('studi')	Always a real word ('study')
Speed	Faster	Slower (requires POS tagging)
Best for	Quick retrieval/search, high-volume/low-accuracy tasks	Machine Learning, Deep Learning, high-accuracy tasks

> **Data Scientist's Tip:** For most modern, high-quality NLP projects (classification, sentiment), **Lemmatization (using SpaCy)** is the preferred method.

4. Full Preprocessing Pipeline (Function)

- Let's combine these steps into a reusable function to apply to real data later.
- We'll prioritize the quality of SpaCy's lemmatization.

In [8]:
```
def custom_text_cleaner(text):
    """Performs lowercasing, stopword removal, and lemmatization."""

    # 1. Lowercase and remove non-alphanumeric (except spaces) for cleaning
    text_lower = text.lower()
    text_clean = re.sub(r'[^a-z\s]', '', text_lower)

    # 2. Process with SpaCy for Lemmatization and Stopword filtering
    doc = nlp(text_clean)

    # 3. Use SpaCy's internal stopword list, which is efficient
    tokens = [token.lemma_ for token in doc
              if not token.is_stop and
              not token.is_punct and
              not token.is_space]

    return " ".join(tokens)

# Test the full pipeline on a new sentence
new_sentence = "I am studying data science because the opportunities are incredibl
cleaned_text = custom_text_cleaner(new_sentence)

print(f"Original: {new_sentence}")
print(f"\nCleaned:  {cleaned_text}")
```

```
Original: I am studying data science because the opportunities are incredibly appea
ling.

Cleaned:  study datum science opportunity incredibly appealing
```

5. Summary and Next Steps

- We have established the core techniques for text normalization: lowercasing, noise reduction, stopword removal, and the difference between stemming and the superior lemmatization.
- These steps are essential before text can be vectorized for machine learning.
- In the next notebook (2.2), we will focus on **Tokenization**, the very first step in the pipeline, and how `NLTK` and `SpaCy` handle complex cases like contractions and sentence boundaries.

Key Takeaways

- **Text Normalization:** We mastered essential preprocessing techniques including lowercasing, punctuation removal, and stopword filtering.

- **Stemming vs Lemmatization:** We learned the crucial difference between rule-based stemming and dictionary-based lemmatization, with SpaCy's lemmatization being the preferred approach for quality NLP tasks.
- **Pipeline Development:** We built a comprehensive, reusable text preprocessing function combining all techniques for real-world application.

Next Notebook Preview

- Now that we have clean, normalized text, the next step is to dive deeper into **tokenization techniques**.
- We will explore **advanced tokenization methods**, handling contractions, sentence boundaries, and complex text structures using both NLTK and SpaCy.

About This Project

This notebook is part of the **Natural Language Processing using Python Programming for Beginners** repository - a comprehensive, beginner-friendly guide for mastering NLP using Python, NLTK, and SpaCy.

Repository: `NLP`

Author

Prakash Ukhalkar

GitHub prakash-ukhalkar
