Module 2: Working with Text in Python

Course: CS 411 – Text Mining

---

1. Introduction

In Module 1, we explored the nature of text data and why it presents unique challenges for computational analysis. In this module, we move from concepts to practice by learning how to **work directly with text in Python**.

Python provides powerful built-in tools and libraries for handling text. Understanding these tools is essential for text preprocessing, web scraping, API interaction, and natural language processing. This module focuses on **advanced string operations**, **text encoding**, and **normalization**, which form the backbone of all later text mining tasks

2. Text as Data in Python

In Python, text data is represented using the **string (`str`) data type**. A string is a sequence of characters enclosed in quotation marks.

Examples:

```python
text1 = "Text mining is powerful"
text2 = 'Python makes text processing easier'
```

Key properties of Python strings:

- Strings are **immutable** (they cannot be changed in place)
- Strings support indexing and slicing
- Strings can be iterated character by character

Understanding these properties is important for efficient and correct text manipulation.

3. Basic String Operations Review

Before moving to advanced operations, recall common string methods:

- `len()` – length of a string
- `.lower()` / `.upper()` – case conversion
- `.strip()` – remove leading and trailing whitespace
- `.replace()` – substitute substrings
- `.split()` – break text into tokens

Example:

```
text = "  Hello World  "
text = text.strip().lower()
print(text)
```

These operations are frequently used in text cleaning pipelines.

4. Advanced String Operations

Advanced string processing allows us to manipulate text more precisely.

4.1 String Indexing and Slicing

Python strings support indexing:

```
word = "mining"
print(word[0])    # 'm'
print(word[-1])   # 'g'
```

Slicing extracts substrings:

```
print(word[1:4])  # 'ini'
```

4.2 String Formatting

String formatting is useful for reporting and logging text processing results.

Examples:

```
count = 5
print(f"Number of documents: {count}")
```

4.3 Searching Within Strings

Python allows checking for substrings:

```
"data" in "text data analysis"
```

Finding positions:

```
text.find("data")
```

These operations are often used to filter or identify relevant documents.

5. Working with Multiple Text Documents

Text mining typically involves **collections of documents**, not a single string.

Common representations:

- Lists of strings

- Dictionaries mapping IDs to text
- DataFrames with text columns

Example:

```
documents = [
    "Text mining is interesting",
    "Python is widely used",
    "Text data is unstructured"
]
```

Looping through text collections is a core skill in preprocessing.

6. Text Encoding

6.1 What Is Text Encoding?

Text encoding defines how characters are represented as bytes in a computer.

Common encodings:

- ASCII - UTF-8 (most common)
- UTF-16

UTF-8 supports:

- English characters
- Accented characters
- Emojis - Non-Latin script

6.2 Encoding Issues

Encoding problems often appear when:

- Reading text files
- Scraping websites
- Using APIs

Example error:

```
UnicodeDecodeError
```

Solution:

```
open("file.txt", encoding="utf-8")
```

Correct handling of encoding ensures text is read accurately.

7. Text Normalization

Text normalization is the process of converting text into a consistent format.

Common normalization steps include:

- Lowercasing text
- Removing extra whitespace
- Standardizing punctuation
- Handling accented characters

```
Example:
text = "Data   Mining!"
normalized = text.lower().strip()
print(normalized)
```

Normalization improves consistency and model performance.

8. Unicode and Special Characters

Text data may include:

- Accented letters (é, ñ)
- Emojis 😊
- Symbols

Python handles Unicode natively, but preprocessing decisions matter:

- Should emojis be removed?
- Should accented characters be preserved?

These choices depend on the analysis goal.

9. Connection to Web Scraping

According to **Mitchell, Web Scraping with Python (Chapter 8)**:

- Scraped text often contains inconsistent encoding
- HTML entities and special characters must be cleaned

- Text normalization is critical after extraction

This module prepares you for real-world scraped text data.

10. Hands-On Practice Overview

In this module's lab and assignment, you will:

- Manipulate text using Python strings
- Normalize raw text
- Handle encoding issues
- Prepare text for later NLP processing

These skills will be reused throughout the course.

11. Discussion Topics

- Why is text normalization important?
- What problems can occur if encoding is ignored?
- How might normalization choices affect sentiment analysis?

12. Key Takeaways

By the end of this module, you should be able to:

- Confidently manipulate text using Python
- Apply advanced string operations
- Understand and handle text encoding
- Normalize text for downstream analysis

These skills form the technical foundation for text preprocessing and NLP.

13. Looking Ahead

In the next module, we will focus on:

- Regular expressions
- Pattern matching
- Building reusable text cleaning pipelines

Make sure you are comfortable with Python string operations before moving forward.