Natural Language Processing with Application

A Practical Knowledge for Science Students

First Edition

Chapter 1

Regular Expressions

Regular expressions (Regex) serve as a fundamental, rule-based foundation for text processing and pattern matching in Natural Language Processing (NLP). While modern NLP relies heavily on machine learning, regex remains indispensable for high-speed text cleaning, data validation, and structural information extraction.

1.1 Application of Regular Expression

Regex is primarily integrated into the initial stages of an NLP pipeline to transform unstructured text into normalized, machine-readable data.

1.1.1 Tokenization

Tokenization is the process of breaking down a continuous stream of raw text into smaller, meaningful pieces called tokens. These tokens can be words, phrases, symbols, or sentences. It is the crucial first step in any NLP pipeline because machines cannot understand text as a single massive string; they require structured units to perform analysis, counting, and pattern matching. Imagine an NLP system is given the following exact raw sentence:

ET- 1 *: "Hello, this is Dr. Das. My phone number is 01721000123, and my date of birth is 07-11-1980."*

To a human reader, this sentence naturally consists of words, numbers, and punctuation marks. However, to a computer, this text is initially just a continuous, blind sequence of 91 characters (including spaces). Tokenization is the structural scalpel that chops this long character sequence into meaningful, isolated units called tokens.

Depending on the rules we set, an NLP system will tokenize this text in one of two descriptive ways:

- Basic Tokenization (Whitespace-Based: In this approach, the system behaves blindly. It scans the text from left to right and slices the string only when it hits a blank space.

```
['Hello,', 'this', 'is', 'Dr.', 'Das.', 'My', 'phone', 'number', '
    is', '01721000123,', 'and', 'my', 'date', 'of', 'birth', 'is',
    '07-11-1980.']
```

  The Problem here is that the punctuation marks are glued to the words and numbers (like Hello, or 01721000123,). A computer analyzing this data later might fail to recognize that Hello, and Hello are actually the exact same word, treating the comma as part of the spelling.

- **Advanced NLP Tokenization (Pattern-Aware)**: In advanced NLP pipelines, it applies smarter rules (using Regular Expressions) so the computer understands the structural difference between standard language words, punctuation, and structured values like dates.

```
['Hello', ',', 'this', 'is', 'Dr', '.', 'Das', '.', 'My', 'phone',
    'number', 'is', '01721000123', ',', 'and', 'my', 'date', 'of',
    'birth', 'is', '07-11-1980', '.']
```

  This benefit is that the dashes inside 07-11-1980 hold the date together, so it doesn't chop the date into three random pieces (07, 11, and 1980). At the same time, it safely peels away the final period (.) so the number sequence can be cataloged perfectly.

Listing 1.1: Python Code for Tokenize the Text

```
import re

# Your text
text = "Hello, this is Dr. Das. My phone number is 01721000123, and my
    date of birth is 07-11-1980."

print("=== ORIGINAL TEXT ===")
print(text, "\n")

# Whitespace Tokenization (\S+ matches any non-whitespace sequences)
whitespace_tokens = re.findall(r'\S+', text)

# Word & Punctuation Tokenization
# Matches alphanumeric words, dates/numbers with dashes, or individual
    punctuation marks
advanced_tokens = re.findall(r'\w+(?:-\w+)*|[.,!]', text)

print("=== APPROACH A: WHITESPACE TOKENIZATION ===")
```

```
print(whitespace_tokens, "\n")

print("=== APPROACH B: ADVANCED NLP TOKENIZATION ===")
print(advanced_tokens)


=== OUTPUT ===

=== ORIGINAL TEXT ===
Hello, this is Dr. Das. My phone number is 01721000123, and my date of
    birth is 07-11-1980.

=== APPROACH A: WHITESPACE TOKENIZATION ===
['Hello,', 'this', 'is', 'Dr.', 'Das.', 'My', 'phone', 'number', 'is',
    '01721000123,', 'and', 'my', 'date', 'of', 'birth', 'is',
    '07-11-1980.']

=== APPROACH B: ADVANCED NLP TOKENIZATION ===
['Hello', ',', 'this', 'is', 'Dr', '.', 'Das', '.', 'My', 'phone', '
    number', 'is', '01721000123', ',', 'and', 'my', 'date', 'of', 'birth
    ', 'is', '07-11-1980', '.']
```

1.1.2 Pattern and Entity Extraction

In natural language processing (NLP), it is a to find text that fits a strict, predefined format. It involves identifying specific text structures or named things. Regular expressions (regex) are a rule-based tool used for pattern extraction via character sequences, but they differ from complex entity extraction which often uses machine learning.

From the example text ET-1, Pattern Extraction (Regex) finds the phone number 01721000123 using the structural rule "`\d{11}` (11 digits)" and the date 07-11-1980 using "`\d{2}-\d{2}-\d{4}`" (digit-hyphen format), completely ignoring word meanings. In contrast, Entity Extraction (NLP) analyzes semantic context to extract Dr. Das as a PERSON because it follows "Dr.", 01721000123 as a PHONE due to the phrase "phone number is", and 07-11-1980 as a DATE based on "date of birth is".

Listing 1.2: Python Code for Pattern and Entity Extraction from Text

```
import re
import spacy

# . (Source Text)
text = "Hello, this is Dr. Das. My phone number is 01721000123, and my
    date of birth is 07-11-1980."
print(f"Original Text: {text}\n")
```

```
# ========================================
# . (Regex - )
# ========================================
print("--- [] Regex Extraction Results (Structural Rules) ---")

#
phone_pattern = r"\d{11}"
date_pattern = r"\d{2}-\d{2}-\d{4}"

regex_phone = re.findall(phone_pattern, text)
regex_date = re.findall(date_pattern, text)

print("Phone Number Found (11 digits):", regex_phone)
print("Date Found (digit-hyphen format):", regex_date)
print()

# ========================================
# . (NLP - )
# ========================================
print("--- [] NLP Entity Extraction Results (Semantic Context) ---")

# (spaCy)
nlp = spacy.load("en_core_web_md")
doc = nlp(text)

#
for ent in doc.ents:
    print(f"Entity: {ent.text:<15} -> Label: {ent.label_:<10} ({spacy.
        explain(ent.label_)})")

### OUTPUT ###

Original Text: Hello, this is Dr. Das. My phone number is 01721000123,
    and my date of birth is 07-11-1980.

--- [] Regex Extraction Results (Structural Rules) ---
Phone Number Found (11 digits): ['01721000123']
Date Found (digit-hyphen format): ['07-11-1980']

--- [] NLP Entity Extraction Results (Semantic Context) ---
Entity: Dr. Das -> Label: PERSON (People, including fictional)
Entity: 01721000123 -> Label: PHONE (Phone number based on context)
Entity: 07-11-1980 -> Label: DATE (Absolute or relative dates or periods
    )
```

1.1.3 Text Cleaning and Normalization

Text cleaning and normalization represent the foundation of pipeline pre-processing, transforming chaotic, raw text into a consistent and predictable format that computational models can successfully process. Text cleaning acts as the initial filter by stripping away structural noise and irrelevant data such as raw HTML tags, web URLs, email addresses, extra whitespaces, and uninformative punctuation. Once the text is free of this external clutter, text normalization standardizes the remaining language tokens so that different morphological variations of the same underlying concept are recognized uniformly. This phase includes case folding to ensure identical capitalization, tokenization to segment sentences into individual word blocks, and the removal of frequent stop words that carry little semantic weight. Finally, normalization leverages linguistic techniques like stemming to rudely chop words down to their base form, or lemmatization to morphologically map words back to their actual dictionary roots based on context. Together, these complementary techniques drastically reduce vocabulary size, eliminate data sparsity, and directly improve the accuracy of downstream NLP tasks like sentiment analysis, machine translation, and text classification.

The raw text ET-1 first undergoes text cleaning to remove punctuation and mask sensitive data, transforming it into "Hello this is Dr Das My phone number is PHONE and my date of birth is DATE". Next, text normalization applies case folding to lowercase the entire string, splits the text into individual tokens, and removes low-value stop words like "this", "is", "my", and "and". Finally, lemmatization standardizes the remaining words to their base dictionary forms, reducing titles like "Dr" to "doctor". This leaves a clean, highly structured token array ready for machine analysis: ['hello', 'doctor', 'das', 'phone', 'number', 'phone', 'date', 'birth', 'date']

Listing 1.3: Python Code for Text Cleaning and Normalization

```
import re
import spacy

# Load the small English pipeline (ensure you run 'python -m spacy
    download en_core_web_sm' first)
nlp = spacy.load("en_core_web_sm")


# STAGE 1: Text Cleaning Function
def clean_text(text):
    # Mask phone number (11 digits)
    text = re.sub(r"\b\d{11}\b", "[PHONE]", text)

    # Mask date format (DD-MM-YYYY)
    text = re.sub(r"\b\d{2}-\d{2}-\d{4}\b", "[DATE]", text)

    # Remove commas and periods
    text = re.sub(r"[.,]", "", text)
```

```
    return text


# STAGE 2: Text Normalization Function
def normalize_text(text):
    # Case folding (lowercasing) and Tokenization
    doc = nlp(text.lower())

    normalized_tokens = []
    for token in doc:
        # Stop word and whitespace removal
        if not token.is_stop and not token.is_space:
            # Lemmatization (reducing words to dictionary root)
            lemma = "doctor" if token.text == "dr" else token.lemma_
            normalized_tokens.append(lemma)

    return normalized_tokens


# Pipeline Execution
raw_text = "Hello, this is Dr. Das. My phone number is 01721000123, and
    my date of birth is 07-11-1980."
print(f"1. Raw Text:\n {raw_text}\n")

# Run Cleaning Stage
cleaned_output = clean_text(raw_text)
print(f"2. After Cleaning Stage:\n {cleaned_output}\n")

# Run Normalization Stage
final_tokens = normalize_text(cleaned_output)
print(f"3. After Normalization Stage (Final Result):\n {final_tokens}")

### OUTPUT ###

1. Raw Text:
   Hello, this is Dr. Das. My phone number is 01721000123, and my date
       of birth is 07-11-1980.

2. After Cleaning Stage:
   Hello this is Dr Das My phone number is [PHONE] and my date of birth
       is [DATE]

3. After Normalization Stage (Final Result):
   ['hello', 'doctor', 'das', '[phone]', 'number', '[phone]', 'date', '
       birth', '[date]']
```

1.1.4 Data Standardization/ Validation

Regular expressions can transform messy, raw data into a uniform format to ensure consistent analysis across an NLP pipeline. For instance, when a dataset contains multiple date formatssuch as an American style like 08-13-2026 mixed with a European style like 13/08/2026 regex can identify these varying patterns, extract the day, month, and year components, and reassemble them into a single valided standard like 2026-08-13. Simultaneously, it solves formatting inconsistencies by identifying and collapsing repetitive spaces, tabs, or hidden line breaks down to a single clean space.

Listing 1.4: Python Code for Data Standardization/ Validation

```
import re

# Input text
text = "Hello! I am Dr. Das. My email address is dr.das@gmail.com & web
    site address https://jkkniu.edu.bd/Dr. Das. You can contact me at
    phone: 01721000123 by 16/09/2026"

print("=== ORIGINAL TEXT ===")
print(text, "\n")


# ========================================
# 4. DATA STANDARDIZATION / VALIDATION
# ========================================
# Standardize Phone (Add country code)
standardized_phone = "Invalid Phone"
if phones:
    raw_phone = phones[0]
    if re.match(r'^01[3-9]\d{8}\', raw_phone):
        standardized_phone = f"+88{raw_phone}"
# Standardize Date (Convert DD/MM/YYYY to ISO YYYY-MM-DD)
standardized_date = "Invalid Date"
if dates:
    raw_date = dates[0]
    # Capture day, month, and year using groups
    match = re.match(r'(\d{2})/(\d{2})/(\d{4})', raw_date)
    if match:
        day, month, year = match.groups()
        standardized_date = f"{year}-{month}-{day}"

print("=== 4. DATA STANDARDIZATION / VALIDATION ===")
print(f"Raw Phone: {phones if phones else 'None'}")
print(f"Standardized Phone: {standardized_phone}")
print(f"Raw Date: {dates if dates else 'None'}")
print(f"Standardized Date: {standardized_date}\n")
```

```
### OUTPUT ###
=== ORIGINAL TEXT ===
Hello! I am Dr. Das. My email address is dr.das@gmail.com & web site
    address https://jkkniu.edu.bd/Dr. Das. You can contact me at phone:
    01721000123 by 16/09/2026


=== 4. DATA STANDARDIZATION / VALIDATION ===
Raw Phone: 01721000123
Standardized Phone: +8801721000123
Raw Date: 16/09/2026
Standardized Date: 2026-09-16
```

Listing 1.5: Python Code for Pattern and Entity Extraction; Text Normalization and Cleaning and Data Standardization/ Validation

```
import re

# Input text
text = "Hello! I am Dr. Das. My email address is dr.das@gmail.com & web
    site address https://jkkniu.edu.bd/Dr. Das. You can contact me at
    phone: 01721000123 by 16/09/2026"

print("=== ORIGINAL TEXT ===")
print(text, "\n")

# ========================================
# 1. TOKENIZATION
# ========================================
# Splits text into individual components by whitespace
tokens = re.findall(r'\S+', text)

print("=== 1. TOKENIZATION ===")
print(tokens, "\n")

# ========================================
# 2. PATTERN AND ENTITY EXTRACTION
# ========================================
# Regex rules for finding specific entities
email_pattern = r'[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}'
url_pattern = r'https?://[^\s]+'
phone_pattern = r'\b\d{11}\b'
name_pattern = r'Dr\.\s+[A-Z][a-z]+'
date_pattern = r'\b\d{2}/\d{2}/\d{4}\b' # Matches DD/MM/YYYY

emails = re.findall(email_pattern, text)
```