Assignment 2: Text Analysis and Rule-based Pluralization

Course: Computational Linguistics - 1

Deadline: February 8th, 2025 — 23:59

1 General Instructions

1. The assignment must be implemented in Python.
2. Only the re (regular expressions) library should be used for text processing.
3. Submitted assignment must be your original work. Please do not copy from any source.
4. Implementation of BPE tokenizer should be from scratch without using any existing tokenizer libraries.
5. A single .zip file needs to be uploaded to the course portal.
6. Your grade will depend on implementation correctness, code clarity, and pattern coverage.

2 Task 1: English Word Pluralization

Implement a rule-based system using regular expressions to handle English word pluralization. First, identify examples for each category (e.g., adding -s). Then, create a simple regex for each category. Finally, combine all the patterns into a single regex.:

1. Basic Pluralization Rules
   (a) Regular plural forms (adding -s)
   (b) Words ending in -s, -sh, -ch, -x, -z (adding -es)
   (c) Words ending in consonant + y (change y to -ies)
   (d) Words ending in vowel + y (adding -s)
2. Advanced Pluralization Rules
   (a) Words ending in -f or -fe (change to -ves)
   (b) Words ending in -o (adding -es or -s)
   (c) Irregular plurals (through dictionary lookup)

   Note: Handle capitalization preservation (CAT → CATS, iPhone → iPhones)
3. Special Cases (Include atleast 5, include the list in README):
   (a) Compound words (e.g., ”mother-in-law” → ”mothers-in-law”)

(b) Words that are same in singular and plural

(c) Latin/Greek origin words (e.g., criterion → criteria)

(d) Handle exceptions to general rules

Note: You are required to make a list of relevant words based on categories on your own.

2.1 Example Cases and Expected Output

```
# Basic Cases
cat → cats
box → boxes
baby → babies
toy → toys

# F/Fe endings
leaf → leaves
knife → knives
roof → roofs (exception)

# O endings
potato → potatoes
photo → photos
piano → pianos

# Irregular forms
child → children
mouse → mice
person → people

# Compound words
mother-in-law → mothers-in-law
passer-by → passers-by

# Same in singular and plural
sheep → sheep
species → species
aircraft → aircraft

# Special cases
phenomenon → phenomena
cactus → cacti
analysis → analyses
```

2.2 Example implementation structure

```
def make_plural(word):
    # Handle irregular plurals
    if word.lower() in irregular_plurals:
        return preserve_caps(word, irregular_plurals[word.lower()])

    # Handle same singular/plural
    if word.lower() in unchanged_plurals:
        return word

    # Apply regex patterns in order
    for pattern, replacement in plural_rules:
        if re.search(pattern, word):
            return re.sub(pattern, replacement, word)

    # Default case
    return word + 's'
```

3 Task 2: Zipf's Law Analysis

Analyze and visualize Zipf's Law in provided corpus:

1. Data Processing:
    (a) Calculate word frequencies
    (b) Rank words by frequency
    (c) Compute log-log values for plotting
2. Analysis:
    (a) Plot word frequency vs rank (log-log scale)
    (b) Calculate and plot the theoretical Zipf's curve
3. Documentation:
    (a) Explain deviations from theoretical Zipf's Law
    (b) Document observations about high and low-frequency words

3.1 Example Usage and Expected Output

```
# Zipf's Law Analysis
frequencies = get_word_frequencies(corpus)
plot_zipf_distribution(frequencies)
# Expected: Log-log plot showing power law distribution
```

4 Task 3: Byte-Pair Encoding (BPE) Tokenizer

Implement a BPE tokenizer from scratch:

1. Basic Implementation:
    (a) Initialize vocabulary with character-level tokens

   (b) Implement frequency counting of token pairs
   (c) Create merge rules based on frequencies
   (d) Apply merge rules to tokenize text

2. Advanced Features:
   (a) Support for different vocabulary sizes
   (b) Handle special tokens (UNK to handle OOV-Out-of-Vocabulary issues, PAD, etc.)

3. Testing and Validation:
   (a) Test segmentation output of few sentences
   (b) Compare results with different vocabulary sizes

4.1 Example Usage and Expected Output

```
# BPE Tokenizer
tokenizer = BPETokenizer(vocab_size=1000)
tokenizer.train(corpus)
tokens = tokenizer.encode("Hello world!")
# Expected: List of subword tokens
```

5 Dataset

Use the text dataset from here for tasks 2 and 3.

6 Submission Requirements

Submit a zip file named `<roll_number>_assignment2.zip` containing:

1. Source Code:
   (a) pluralizer.py
   (b) zipf_analysis.py
   (c) bpe_tokenizer.py

2. Documentation:
   (a) `README.md` with implementation details and assumptions
   (b) List of implemented rules and their priority order
   (c) Analysis of edge cases and limitations

3. Test Results:
   (a) Test cases and their outputs
   (b) Error analysis for failed cases
   (c) Coverage statistics for different rule types

7 Resources

1. Python Regular Expressions Documentation:
   `https://docs.python.org/3/library/re.html`

2. English Pluralization Rules:
   `https://en.wikipedia.org/wiki/English_plurals`

3. Zip's Law:
   `https://en.wikipedia.org/wiki/Zipf%27s_law`

4. Regular Expression Testing Tool:
   `https://regex101.com`

5. Original BPE Paper:
   `https://arxiv.org/abs/1508.07909`