[1] Project 3
CS 118: Cloud Computing

Assigned: 12/1/25
Due: 12/16/25 at 11:59 PM Medford Time
HW should be completed in your presentation group
Please submit Gradescope in .zip format for both code and results

Contents

1 Introduction

For this project, you will develop a MapReduce application for analyzing word frequency trends in classic novels across different publication years. Specifically, you will build a two-step MapReduce pipeline that:

[1] Created by Zhaoqi(Roy) Zhang 11/30/2025

1. **Ranks words** by overall frequency across all novels
2. **Analyzes temporal trends** showing how top words vary by publication year (similar to Google Ngram Viewer)

You will develop your application as two sets of mappers and reducers using Python 3. Your code will work directly with Project Gutenberg ebook files, cleaning headers/footers detecting publication dates, and analyzing word ranks and year-by-year occurence.

2 Data Set

2.1 Copyright-free Books from Project Gutenberg

The Project Gutenberg(https://www.gutenberg.org/) repository contains over 70,000 free ebooks. For this project, you will work with plain text versions of classic novels. Specifically, you will analyze the top 25 books in the detective fiction category, see details in 2.2.

Dataset Characteristics:

- Plain text files (`.txt` format)
- Each file includes:
    - Header with metadata (title, author, release date)
    - Start marker: `*** START OF THE PROJECT GUTENBERG EBOOK ...`
    - Novel content
    - End marker: `*** END OF THE PROJECT GUTENBERG EBOOK ...`
    - Full license text (footer)

Example File Structure

```
The Project Gutenberg eBook of A Study in Scarlet

Title: A Study in Scarlet
Author: Arthur Conan Doyle
Release date: April 1, 1995 [eBook #244]

*** START OF THE PROJECT GUTENBERG EBOOK A STUDY IN SCARLET ***

[Novel content here...]

*** END OF THE PROJECT GUTENBERG EBOOK A STUDY IN SCARLET ***
```

```
[Full license text...]
```

2.2 List of Books to Analyze

You will analyze the top 25 books in the detective fiction category (https://www.gutenberg.org/ebooks/bookshelf/30):

1. The Adventures of Sherlock Holmes - Arthur Conan Doyle (pg1661)
2. The Works of Edgar Allan Poe — Volume 2 - Edgar Allan Poe (pg2148)
3. The Hound of the Baskervilles - Arthur Conan Doyle (pg2852)
4. A Study in Scarlet - Arthur Conan Doyle (pg244)
5. The Sign of the Four - Arthur Conan Doyle (pg2097)
6. The Works of Edgar Allan Poe — Volume 1 - Edgar Allan Poe (pg2147)
7. The Return of Sherlock Holmes - Arthur Conan Doyle (pg108)
8. The Memoirs of Sherlock Holmes - Arthur Conan Doyle (pg834)
9. ~~The Middle Temple Murder~~ (SKIP - audiobook)
10. The Mysterious Affair at Styles - Agatha Christie (pg863)
11. The Innocence of Father Brown - G. K. Chesterton (pg204)
12. The Moonstone - Wilkie Collins (pg155)
13. The Murder on the Links - Agatha Christie (pg58866)
14. His Last Bow - Arthur Conan Doyle (pg2350)
15. The Return of Sherlock Holmes - Arthur Conan Doyle (pg221)
16. Poirot Investigates - Agatha Christie (pg61262)
17. The Valley of Fear - Arthur Conan Doyle (pg3289)
18. The Wisdom of Father Brown - G. K. Chesterton (pg223)
19. The Secret Adversary - Agatha Christie (pg1155)
20. The Mystery of the Yellow Room - Gaston Leroux (pg2777)
21. The Red House Mystery - A. A. Milne (pg1872)
22. The Man in the Brown Suit - Agatha Christie (pg61168)
23. The Hand in the Dark - Arthur J. Rees (pg32341)
24. Whose Body? - Dorothy L. Sayers (pg58820)
25. The Secret of the Night - Gaston Leroux (pg2830)
26. The Adventure of the Bruce-Partington Plans - Arthur Conan Doyle (Include 30th book to replace #9, and 26th-29th are audiobooks too)

3 Your Task: Two-Step MapReduce Pipeline

2.1 Step 1: Word Ranking

Goal: Extract and count all words across all novels, producing a ranked list of the top 25 words.

Mapper (step1_mapper.py):

- Reads Project Gutenberg files from stdin
- Strips headers and footers automatically
- Extracts words from novel content only
- Filters stop words (the, a, an, is, was, etc.)
- Cleans words (lowercase, remove punctuation, minimum length 3)
- Outputs: `word\t1` for each word occurrence

Reducer (step1_reducer.py):

- Aggregates word counts
- Keep one word's count in memory at a time, so no need for a hash table
- Sorts by frequency (descending)
- Outputs: `word\tcount` (e.g., `holmes\t887`)

2.2 Step 2: Year-Based Analysis

Goal: For the top 25 words in step 1, show frequency distribution across publication years (not the release year, which refers to when the book was added to Project Gutenberg). See 3.3 for details of publication years.

Mapper (step2_mapper.py):

- Reads Project Gutenberg files from stdin
- Strips headers and footers automatically
- Detects publication year from title/author
- Only processes words in the top 25 list from Step 1
- Outputs: `word\tyear\t1` for each occurrence

Reducer (step2_reducer.py):

- Aggregates counts by word-year combination
- Keep one (word, year)'s count in memory at a time, so no need for a hash table
- Generates formatted output with ASCII bar charts
- Shows temporal trends for each word

3 Implementation Specifications

3.1 Data Preparation in Both Steps

Mappers in both steps should handle:

1. Header Detection: Identify and skip all lines before `*** START OF...` marker
2. Footer Detection: Stop processing at `*** END OF...` marker
3. Table of Contents Removal: Detect and skip TOC sections

3.2 Word Filtering in Step 1

Stop Words (NLTK English Corpus):
Use NLTK's built-in English stopwords list for filtering. This provides a standardized, comprehensive list of common words to exclude.

You can set it up with:

```
pip install nltk
python3 -c "import nltk; nltk.download('stopwords')"
```

Usage in Mapper:

```
from nltk.corpus import stopwords

# Load English stopwords
STOP_WORDS = set(stopwords.words('english'))
```

NLTK's English stopwords include:

- Articles: the, a, an
- Common verbs: is, was, are, been, have, had, do, does, did
- Pronouns: i, you, he, she, it, they, we, me, him, her
- Prepositions: of, to, in, for, on, with, at, by, from
- Conjunctions: and, or, but, if, because
- Modal verbs: will, would, can, could, should, may, might
- Common adverbs: now, then, here, there, very, too, so
- And more (~179 words total)

Here are some additional filters/steps we ask to add:

- Minimum length: 3 characters
- Must be alphabetic only (no digits, punctuation)
- Case-insensitive (all converted to lowercase)

3.3 Publication Year in Step 2

Sadly, Project Gutenberg book doesn't include the actual publication year in downloaded text. You can put the following list we searched for the top 25 books for the Detective Fiction collection in your mappers:

```
KNOWN_DATES = {
    # Arthur Conan Doyle - Sherlock Holmes
    'study in scarlet': 1887,
    'sign of the four': 1890,
    'adventures of sherlock holmes': 1892,
    'memoirs of sherlock holmes': 1894,
    'hound of the baskervilles': 1902,
    'return of sherlock holmes': 1905,
    'valley of fear': 1915,
    'his last bow': 1917,
    'bruce-partington plans': 1908,

    # Edgar Allan Poe
    'edgar allan poe': 1845,
    'works of edgar allan poe': 1845,

    # Agatha Christie
    'mysterious affair at styles': 1920,
    'murder on the links': 1923,
    'poirot investigates': 1924,
    'secret adversary': 1922,
    'man in the brown suit': 1924,

    # G. K. Chesterton - Father Brown
    'innocence of father brown': 1911,
    'wisdom of father brown': 1914,

    # Wilkie Collins
    'moonstone': 1868,

    # Gaston Leroux
    'mystery of the yellow room': 1907,
    'secret of the night': 1914,

    # A. A. Milne
    'red house mystery': 1922,
    # Dorothy L. Sayers
    'whose body': 1923,

    # Arthur J. Rees
    'hand in the dark': 1920,
}
```

You may need to add entries for any books not already in the database.

3.4 Starter code

We provide a skeleton of Python code for both the mapper and the reducer to help folks get started quickly. Note that this is a universal skeleton, but you might need to add extra steps for different MapReduce steps.

Mapper

```
import sys
import re
import string
from nltk.corpus import stopwords

STOP_WORDS = set(stopwords.words('english'))

for line in sys.stdin:
      line = line.rstrip()

      # Check for start of content marker
      if '*** START OF' in line:

      # Check for end of content marker
      if '*** END OF' in line:

      # Check for Table of Content

      # Process content only
      if in_content:
          # Extract words from lines
          words = re.findall(r'\b[\w\']+\b', line)

          for word in words:
          # Clean word: remove punctuation, lowercase
```

```
        # Filter: minimum length 3, alphabetic only, not a stop word
        if len(word) >= 3 and word.isalpha() and word not in STOP_WORDS:
```

Reducer

```
    word_counts = {}

    for line in sys.stdin:
        line = line.strip()

        # Parse input: word\tcount

        # Aggregate counts
        if word in word_counts:
        else:

    # Sort by count (descending) and output

    # Print result
    for word, count in sorted_words:
        print(f"{word}\t{count}")
```

4 Run MapReduce Steps

Step 1: Overall word ranking

```
cat data/*.txt > results/combined_input.txt

python3 code/step1_mapper.py < results/combined_input.txt >
results/step1_mapped.txt

# Use LC_ALL=C prefix to ensure consistent sorting behavior across
different machines & systems
LC_ALL=C sort results/step1_mapped.txt > results/step1_sorted.txt

python3 code/step1_reducer.py < results/step1_sorted.txt >
results/step1_reduced.txt

head -100 results/step1_reduced.txt > results/step1_topwords.txt
```

Step 2: Year-based analysis

```
export TOP_WORDS_FILE=results/step1_topwords.txt

python3 code/step2_mapper.py < results/combined_input.txt >
results/step2_mapped.txt

# Use LC_ALL=C prefix to ensure consistent sorting behavior across
different machines & systems
LC_ALL=C sort results/step2_mapped.txt > results/step2_sorted.txt

python3 code/step2_reducer.py < results/step2_sorted.txt >
results/step2_analysis.tx
```

5 Submission & Grades

5.1 Deliverables & Grades

You need to submit your implementation and output in separate Gradescope assignments.

For implementation, be sure to include the following files

- `step1_mapper.py`
- `step1_reducer.py`
- `step2_mapper.py`
- `step2_reducer.py`

And for output results submission:

- `step1_reduced.txt` - All words ranked by frequency (from 25 books), in case there is inconsistent sorting behavior in your machine
- `step1_topwords.txt` - Top 25 words
- `step2_analysis.txt` - Year-based analysis showing trends for top 25 words

5.2 Expected Output Format

Step 1 - Word Rankings:

```
upon    2074
man     1032
holmes  887
little  703
```

Just one word per line, and the count is separated by spaces. The results must be sorted

Step 2 - Year Analysis:

```
Word: holmes (Total: 887)
  1887:   156
  1890:   203
  1892:   445
  1902:    83

Word: mystery (Total: 89)
  1845:    12
  1887:    18
  1890:    31
  1892:    28

Word: detective (Total: 156)
  1887:    45
  1892:    67
  1902:    32
  1920:    12
```

Each word's year-by-year occurrence must start with `Word:` and be followed by the word and total count. Each year's occurrence count is separated by `:` and spaces between the year and count.

Also, the demos above are just for format, not the actual result.

5.3 Grades

In total, upper bound of project 3 grade is 100 points — 50 for code submission, 50 for correctness.
The implementation submission will not be tested/graded — it earns you full points (50 points) as long as it is submitted.
The output would be graded against the solution using the autograder. Each correct result for the top 25 words in step 1 is worth 1 point, and each correct result for top 25 words' year-by-year occurrence in step 2 is worth 1 point. Result correctness earns you 50 points.