CS 7319 Software Architecture

Homework 2: Sentiment Analysis in Two Architectures

Instructor: Dr. Isaac Chow
Fall 2025

Goal: Implement the same functionality using two different software architectures to experience architectural trade-offs directly.

Part A is a single-program design in any language.
Part B is a MapReduce-style pipeline (as in the provided notebook).

1. Learning Outcomes

- Model a problem using two architectural styles and explain the differences.
- Implement a simple keyword-based sentiment analyzer.
- Compare code structure, data flow, complexity, and scalability implications.

2. Problem Statement (common to both parts)

Given a text dataset (one post per line), decide overall mood using keywords:

• Positive keywords: happy, excited, thrilled, love

• Negative keywords: sad, depressed, angry, upset

Classification rules:
• Positive: contains ≥1 positive keyword and no negative keyword
• Negative: contains ≥1 negative keyword and no positive keyword
• Mixed: contains both positive and negative keywords
• Neutral: otherwise

Data

Provided datasets in the data/ folder:
• data/sample_us_posts.txt (starter)
• data/sample_us_posts_mixed.txt (mixed sentiments)

• data/your_name_posts.txt – (your own posts)

Part A — Single-Program Architecture (language of your choice)

Architectural intent: A simple, single-process program that reads the dataset, scans each line, classifies it, and aggregates totals.

Requirements:
1) Choose any language (e.g., Python, Java, C#, …).

2) Read input from a text file (one post per line).
3) Tokenize case-insensitively (letters/apostrophes are enough).
4) Classify each line using the rules above.
5) Aggregate totals per category and print a summary:
  • Counts for Positive/Negative/Mixed/Neutral
  • Verdict: “Happier” (Positive>Negative), “Sadder” (Negative>Positive), or “Tied”
6) CLI usage example:
  sentiment_prog <path-to-textfile>
7) Output example:
  Positive=123 Negative=95 Mixed=17 Neutral=64
  Verdict: Happier
8) (Optional) Produce a simple bar chart.

What to submit for Part A:

1. Screen shots showing that your program is working as intended
2. Brief Description of your architecture, language, and how to build/run
3. Source file(s)

Part B — MapReduce-Style Architecture (as in the existing notebook)

Architectural intent: Model the problem as Map → Shuffle/Group → Reduce to simulate data-parallel processing.

Requirements:
1) Map: implement map(line) → [(label, 1)] emitting one pair per line (Positive/Negative/Mixed/Neutral).
2) Shuffle/Group: group all values by label.
3) Reduce: sum values per label to get totals.
4) Config knobs (recommended): USE_COMBINER=True/False; NUM_REDUCERS if you simulate partitions.
5) Outputs: same summary and verdict as Part A; if using the notebook, include two charts (counts; Positive vs Negative %).
6) Use both datasets (run once per dataset).

What to submit for Part B:

1. Screen shots showing that your program/notebook is working as intended
2. Brief Description of your architecture, explaining where Map, Shuffle, and Reduce occur in your code

3. Completed notebook (or equivalent code)

Comparison & Reflection (Bonus)

Write ½–1 page comparing the two architectures:
• Structure & responsibilities (mapper vs single pass)
• Complexity (which code is simpler? easier to evolve?)
• Scalability & performance (how would each handle 100,000 × data?)

Step-by-Step Instructions

Part A (single program):
1) Pick a language you're comfortable with.
2) Implement reading → tokenizing → classification → tallying.
3) Run on all datasets.
4) Print counts and verdict; save output (copy/paste or screenshot).
5) (Optional) Generate a bar chart.

Part B (MapReduce-style):
• Using the provided Jupyter Notebook:
  1) Launch Jupyter and open sentiment_mapreduce.ipynb.
  2) (Optional) Use the Dataset Selector to choose a dataset.
  3) Run cells top to bottom.
  4) Record totals and the verdict.
  5) Screenshot the two charts.
• Or implement Map/Shuffle/Reduce in your own language and print the same outputs.