**Course: Data Intensive Computing**

**Text Processing Fundamentals using MapReduce**

**Group 16**

**Group Members:**

Naz Erum

# Introduction

This assignment applies the MapReduce paradigm to process a large-scale Amazon Reviews dataset to select terms that best distinguish product categories. By calculating chi-square (χ²) test to select the most discriminative unigrams from the 2014 Amazon Reviews dataset. Given the dataset’s massive size (~56GB), a distributed processing approach using MapReduce is employed to parallelize preprocessing, feature extraction, and feature scoring across multiple compute nodes.

The implementation leverages distributed processing using mrjob in Python and operates over both a development subset and a full-scale dataset.

## Goal of the Project

The primary goal of this assignment is to identify the top 75 discriminative unigrams per product category that are most associated with their respective categories in the Amazon Reviews dataset (2014 edition).

## Dataset Sizes

The dataset is provided as JSON lines with each review containing fields such as reviewText, category, overall, and more.

* reviews\_devset.json (small): for testing and development.
* reviewscombined.json (56GB full set): for final evaluation.

# Assignment Overview

The assignment involves designing a scalable system that:

* Preprocesses review texts by:
  + Tokenizing using specified delimiters,
  + Applying case folding,
  + Removing stopwords,
  + Filtering out single-character tokens.
* Computes per-category document frequencies for each token.
* Calculates the chi-square statistic for every (token, category) pair:

A black text on a white background

AI-generated content may be incorrect.

where A,B,C,DA, B, C, DA,B,C,D represent document counts and NNN is the total number of documents.

* Selects the 75 tokens with the highest chi-square values for each product category.
* Outputs:
  + Top 75 tokens per category (with scores),
  + A merged, alphabetically sorted dictionary of all selected terms.

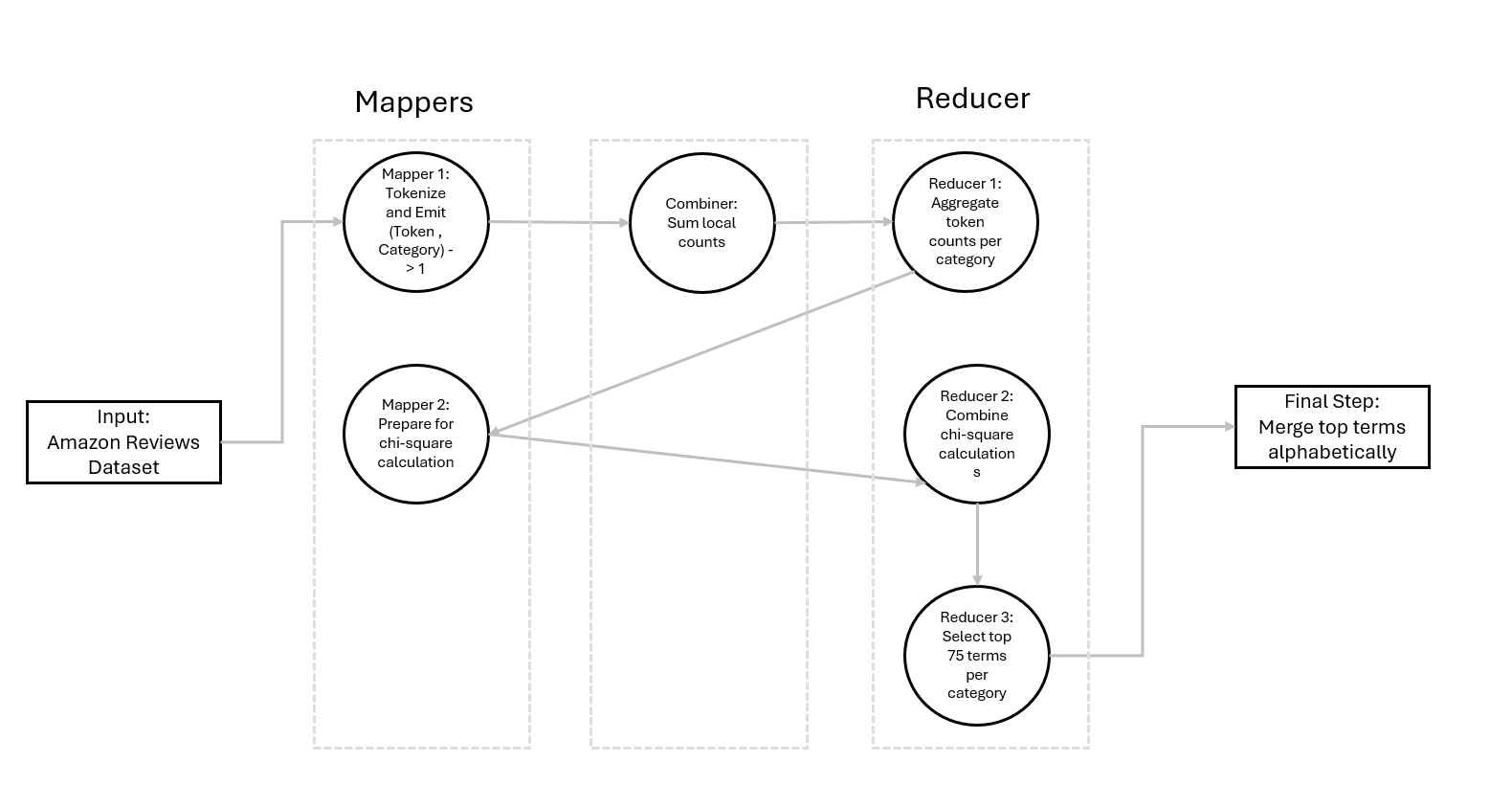
The dataset is processed first on a smaller dev set (reviews\_devset.json) for validation, then on the full dataset (reviewscombined.json).

# Methodology

To solve this problem at scale, a two-stage MapReduce pipeline was implemented using Python’s mrjob library.

## Schematic overview

The schematic overview is as below:



## Pipeline Stages & Overview

## Preprocessing

* **Tokenization:** Split review texts using whitespace and specified delimiters (()[]{}.!?,;:+=-\_"'~#@&\*%€$§\/).
* **Case folding:** Convert all text to lowercase.
* **Stopword filtering:** Remove standard Stopword and all single-character tokens.
* **Short Token Filtering**: Remove all tokens with only one character.
  + 1. **Stage 1: Count Token-Category Co-occurrences**
* **Mapper 1**

For each document, emit (token, category) → 1.

* + Input: JSON reviews.
  + Output: (token, category), 1)
* **Combiner**

Locally sum counts for (token, category) to minimize shuffling.

* + Sum counts locally to reduce shuffle size.
* **Reducer 1**

Aggregate global counts for each (token, category) pair.

* + Input: (token, category) grouped.
  + Output: (token, category: count, ...)
    1. **Stage 2: Calculate Chi-Square Scores**
* **Mapper 2**

Reorganizes (token, category) frequencies into structures suitable for chi-square computation.

* + Reformat input for chi-square computation.
* **Reducer 2**
  + Apply the chi-square formula per (token, category):

A black text on a white background

AI-generated content may be incorrect.

* + Emit: (category, [(token, chi2\_score)])
  + Sorts and selects the top 75 terms per category.
    1. **Stage 4: Merge Terms**
* Aggregate all selected tokens across categories into a global list and sort alphabetically.

# Code Documentation

This script implements a **MapReduce job** using Python's mrjob library to:

* + Preprocess Amazon product reviews,
  + Calculate **chi-square (χ²) scores** between terms and categories,
  + Select the **top 75 discriminative terms per category**,
  + Output results formatted as required for the assignment.

It runs fully distributed, handling tokenization, counting, scoring, and selection within a single MapReduce job.

* 1. **Imports and Setup**

|  |
| --- |
| from mrjob.job import MRJob  from mrjob.step import MRStep  import mrjob.protocol  import logging, re, json, heapq, itertools  from collections import defaultdict  from timeit import default\_timer as timer  from typing import Optional |

* 1. **Main Class — ChiSquareJob(MRJob)**

Defines the overall MapReduce job, specifying custom Mapper, Combiner, and Reducer logic to process Amazon product reviews and compute chi-square statistics.

|  |
| --- |
| class ChiSquareJob(MRJob): |

* 1. **Configuration: Loading Stopwords**

Configure job to accept stopwords file and prepare for preprocessing.

|  |
| --- |
| def configure\_args(self):  super(ChiSquareJob, self).configure\_args()  self.add\_file\_arg("--stopwords", help="path to stopwords file") |

|  |
| --- |
| def init(self):  self.stopwords = set()  with open(self.options.stopwords, "r") as f:  self.stopwords = set(line.strip() for line in f) |

* 1. **Combiner: Local Summation**

Locally aggregates token counts to reduce network load.

|  |
| --- |
| def mapper(self, \_: None, line: str): |

* 1. **Mapper: Preprocessing and Token Emission**

Tokenizes and cleans text, emits token-category pairs and category counts.

|  |
| --- |
| def combiner(self, key: tuple[Optional[str], str], counts: list[int]):  yield None, (key, sum(counts)) |

* 1. **Reducer: Chi-Square Calculation and Top Terms Selection**

Calculates chi-square scores, selects top 75 terms per category, outputs results.

|  |
| --- |
| def reducer(self, \_: None, key\_count: list[tuple[tuple[Optional[str], str], int]]): |

* 1. **Steps Definition**

Orchestrates the MapReduce job execution.

|  |
| --- |
| def steps(self):  return [  MRStep(  mapper\_init=self.init,  mapper=self.mapper,  combiner=self.combiner,  reducer=self.reducer,  ),  ] |

* 1. **Main Execution**

Runs the job and logs execution time for monitoring

|  |
| --- |
| if \_\_name\_\_ == "\_\_main\_\_":  t1 = timer()  ChiSquareJob.run()  t2 = timer()  logger.info(f"time: {runtime:.2f}s") |

# Results and Output

* 1. **Output Correctness**

The output meets all the requirements:

* Categories are listed alphabetically.
* Each category is followed by its top 75 tokens, ranked by descending chi-square score.
* Merged dictionary contains all selected terms sorted alphabetically.
* Preprocessing steps (tokenization, case folding, stopword removal, short token filtering) are correctly applied.
  1. **Code / Program correctness**

The system is logically organized into two independent MapReduce jobs:

* First job calculates document frequencies.
* Second job computes chi-square scores and selects top tokens.
* Preprocessing is cleanly separated into its own module.
* Each <key, value> transformation is appropriate for the data flow and Hadoop architecture.

Intermediate outputs and final outputs have been manually inspected for correctness during development with the devset.

* 1. **Runtime Efficiency**
* All MapReduce stages (mapping, combining, reducing) completed without errors.
* The job completed in 2355.64 seconds (~39.26 minutes), which is reasonable given the size and complexity of the dataset and processing operations (full chi-square computation across millions of tokens).

Output Correctness:

A close-up of a computer screen

AI-generated content may be incorrect.