Big Data Collocation Extraction Engine

alimoha, Mohamed Ali, 214298184
mohamedt, Mohamed Taha, 324990480

This project is a distributed text-mining engine built on **Hadoop MapReduce**. Its goal is to automatically extract **Collocations** from the Google 2-Grams dataset.

The system processes massive datasets of 1-grams and 2-grams to calculate the **Log-Likelihood Ratio (LLR)** for every bigram, surfacing the most significant linguistic pairings.

How to Run:

run command : `java -jar target/runner-1.0.jar <jarS3> <unigramS3> <bigramS3> <outBaseS3> <reducers> [instanceCount] [useCombiner] [localOutDir] [localLogsDir]`

Argument Reference:

- `<jarS3>`: S3 path to your compiled `dsp2` JAR file.
- `<unigramS3>`: S3 path to the 1-gram input file.
- `<bigramS3>`: S3 path to the 2-gram input file.
- `<outBaseS3>`: Base S3 path for job outputs.
- `<reducers>`: Number of reducers to use.
- `[instanceCount]`: (Optional) Number of EC2 instances (nodes) for the cluster.
- `[useCombiner]`: (Optional) `1` to enable Combiner, `0` to disable.
- `[localOutDir]`: (Optional) Local directory to download the final output to.
- `[localLogsDir]`: (Optional) Local directory to download logs to.

Implementation Details:

The Algorithm: Log Likelihood Ratio (LLR)

The engine determines if two words w1, w2 are a collocation by comparing two hypotheses:
1.H1 (Independence): The occurrence of w2 is independent of w1.
2. H2 (Dependence): The occurrence of w2 depends on the presence of w1.

We calculate the LLR score. A high score indicates a strong collocation.

MapReduce Architecture (4-Step Pipeline)

Job 1: Process 2-grams (Filtering & Aggregation)

- **Goal:** Clean raw data and aggregate counts for bigrams.
- **Input:** Raw 2-gram dataset.

- **Mapper:**
    - Parses lines (`ngram`, `year`, `count`).
    - Converts years to decades (e.g., 1995 → 1990).
    - **Cleaning:** Tokenizes text, keeps only English/Hebrew letters, enforces length >= 2.
    - **Stopwords:** Filters out common noise words using `StopWords.java`.
- **Reducer:** Aggregates counts for identical keys.
- **Output:** `(decade, w1, w2) -> c12`.

Job 2: Join with Unigrams c1

- **Goal:** Attach the count of the first word (c1) to every bigram.
- **Mapper 1 (From Job 1):** Emits `(decade, w1, w2) -> c12`.
- **Mapper 2 (From 1-grams):** Emits `(decade, w1, *) -> c1`.
    - *Note:* The special character `*` is used to ensure the unigram count arrives at the Reducer alongside the bigrams.
- **Partitioner:** Groups keys by `(decade, w1)`.
- **Reducer:**
    - Receives the unigram count (c1) first (due to `*` sorting).
    - Attaches c1 to every bigram associated with w1.
- **Output:** `(decade, w1, w2[*]) -> "c12,c1"[“c1”]`

Job 3: Join with Unigrams (c2) & LLR Calculation

- **Goal:** Attach the count of the second word (c2) and calculate the final score.
- **Mapper (From Job 2):** if the output if from the bigram (second word is not *), Swaps the key to group by the *second* word: `(decade, w2, w1) -> "c12,c1"`.
- **Partitioner:** Groups keys by `(decade, w2)`.
- **Reducer:**
    - Receives c2 (via the `*` marker).
    - Calculates LLR using `likelihoodRatio.get(c1, c2, c12, N)`.
- **Output:** `(decade, w1, w2) -> score`.

Job 4: Sort & Top 100

- **Goal:** Sort collocations by score per decade and output the top 100.
- **Mapper:** Emits `(decade, score) -> "w1,w2"`.
- **Comparators:**
    - **Partitioner:** Groups by `decade` (so all years go to the same Reducer).
    - **Grouping Comparator:** Sorts by `score` (Descending).
- **Reducer:** Iterates through the sorted values and outputs the first 100 pairs for each decade.
- **Output:** Final Text File.

Design Notes

- **AWS Integration:** The project uses standard Hadoop `Text` and `LongWritable` types to ensure compatibility with Amazon EMR.
- **Scalability:** The design uses a "Reduce-Side Join" approach (or "Map-Side" if 1-grams fit in memory) to handle datasets that exceed the RAM of a single machine.