UNIVERSITÀ DEGLI STUDI DI MILANO

AMD Project 1 - Finding Similar Items

AUTHOR

Michele Coaro - 41208A

Abstract

This project implements a scalable similarity detection system, designed to work on the Amazon Books Review dataset. The main challenge lies in the task of identifying similar items, which becomes computationally prohibitive when dealing with millions of reviews, due to the time complexity being $O(n^2)$. The primary objective is to develop an efficient algorithm that can detect pairs of book reviews with high textual similarity, specifically those sharing at least 80% of their content (Jaccard similarity ≥ 0.8). Another potential issue in implementation is scalability, which has been addressed through the use of the library Spark. This approach reduces the computational complexity from quadratic to nearly linear time while maintaining high accuracy in finding similar pairs. The project revolved around building a pipeline that includes tokenization, stop word removal, and binary term frequency vectorization, before applying the LSH algorithm. The system operates on a subsample of 50,000 reviews for rapid iteration and testing, while maintaining the ability to scale to the full dataset without code modifications.

Contents

1 Introduction

This project implements a scalable duplicate detection system to tackle the fundamental challenge of similarity in datasets, a cornerstone of modern data processing. The need is underscored by the impossibility to apply a more traditional pairwise comparison due to the size of the dataset and its quadratic complexity. The proposed solution leverages MinHash signatures combined with a text processing pipeline to achieve near-linear time complexity and good recall on the duplicate reviews injected as a test, while highliting the importance of the tradeoff between recall and precision . The devised system employs a multi-stage pipeline that transforms raw review text through tokenization, stop word removal, and binary TF (presence/absence) vectorization before applying LSH for approximate similarity joins.

The remainder of this report is organized as follows:

- **Section 2** describes the dataset characteristics and preprocessing steps taken, including data quality checks and the handling of null or empty reviews.
- **Section 3** details the algorithmic approach, covering the text processing pipeline, LSH implementation with MinHash, and the rationale behind key parameter choices.
- **Section 4** outlines the experimental methodology, including the evaluation framework with synthetic duplicate injection and the metrics used to assess performance.
- **Section 5** presents the results, analyzing both the detection performance on augmented data and examples of discovered similar pairs in the original dataset.
- **Section 6** discusses the scalability characteristics of the solution and its behavior when transitioning from subsampled to full dataset processing.
- **Section 7** concludes with key findings, limitations of the approach, and potential extensions for improved similarity detection.

The primary objective was, as previously stated, to craft a solution that balances computational efficiency with results accuracy, as well as learning how to properly develop a working pipeline. PySpark's distributed computing capabilities were a key factor in the realization of this project, resulting in a solution which demonstrates how a modern big data approach can effectively address classical similarity search problems at scale through the combination of distributed processing and probabilistic algorithms.

2 Dataset and Preprocessing

The Amazon Books Review dataset was obtained via the Kaggle API and consists of the`Books_rating.csv` file containing review metadata and text content. Initial exploration revealed several data quality issues requiring preprocessing before similarity detection could be performed effectively.

The preprocessing pipeline implemented the following steps:

1. **Column Standardization**: Renamed columns containing special characters (e.g., `review/text` to `review_text`) for Spark compatibility
2. **Data Filtering**: Removed reviews with null or empty text content, eliminating them from the dataset. In a following step, rows that hash to zero features have been dropped as well
3. **Feature Selection**: Retained only review ID and text columns, reducing memory overhead
4. **Subsampling**: Implemented configurable sampling of 50,000 reviews for development, with ability to process the full dataset

Text preprocessing happened within the Spark pipeline, specifically transforming raw reviews through tokenization using regex patterns, removing English stop words (through Spark's standardized list), and finally by creating a binary term frequency vector. The binary encoding approach (presence/absence rather than counts) has been chosen to optimize Jaccard similarity computation. Additionally, reviews that became empty after the text processing phase have been filtered out, to ensure meaningfulness and density of the dataset.

To sum up, the preprocessing approach helped creating a clean and suitable dataset for large-scale similarity detection, as well as giving the option of working on a smaller subsample for faster computation.

3 Algorithmic Approach

The whole algorithm implementation employs Locality-Sensitive Hashing (LSH) with MinHash signatures to reduce the complexity problem from $O(n^2)$ to approximately $O(n)$ (but not quite, just a subquadratical area that approaches the lower bound) complexity while maintaining high accuracy for Jaccard similarity detection.

3.1 Text Processing Pipeline

A four-stage Spark ML pipeline was created, transforming raw text into hash signatures suitable for similarity detection:

1. **Regex Tokenization**: Splits review text on non-word characters (pattern: `\W+`), converting each review into a list of tokens
2. **Stop Word Removal**: Filters out common English words using Spark's default stop word list, retaining only content-bearing terms
3. **Binary Hashing TF**: Converts token lists into sparse binary vectors of dimension 2^{20} (1,048,576), where each dimension indicates term presence
4. **MinHash LSH**: Generates compact signatures for efficient similarity search using 5 independent hash tables

The choice of binary term frequency vectors (rather than count-based) aligns perfectly with Jaccard similarity, which measures the ratio of intersection to union of sets, caring only about term presence rather than frequency.

3.2 LSH Configuration and Parameters

The MinHash LSH implementation uses the following key parameters:

- **Hash Tables**: 5 independent tables
- **Hash Buckets**: 2^{20} dimensions should provide sufficient space to minimize hash collisions while remaining computationally feasible
- **Similarity Threshold**: Jaccard distance ≤ 0.2 (equivalent to $\geq 80\%$ similarity) captures near-duplicate content

4 Experimental Methodology

To evaluate the effectiveness of the LSH-based similarity detection system, an experiment was designed using synthetic duplicate injection. This approach provides ground truth data for measuring precision and recall, essential metrics for assessing the algorithm's performance in a probabilistic setting.

4.1 Synthetic Duplicate Generation

The evaluation framework creates known duplicate pairs by:

1. Randomly selecting 250 reviews from the original dataset using a fixed seed for reproducibility
2. Creating exact copies of these reviews with modified IDs (prefixed with `"dup_"`)
3. Adding to the original dataset these synthetic duplicates

Adding 250 duplicates, i.e., entries with Jaccard distance equal to 0.0 provides reliable ground truth for evaluation. The algorithm should be able to identify exact duplicates with ease.

4.2 Performance Metrics

The evaluation computes two key metrics:

- **Precision**: $\frac{\text{True Positives}}{\text{Total Predicted Pairs}}$ â measures the fraction of detected pairs that are actual duplicates
- **Recall**: $\frac{\text{True Positives}}{\text{Total Ground Truth Pairs}}$ â measures the fraction of actual duplicates that were detected

High recall is particularly important for this application, as missing duplicate content (false negatives) is generally more problematic than occasional false positives, which can be filtered through additional verification if needed.

4.3 Validation Approach

Beyond quantitative metrics, the implementation includes qualitative validation by joining detected similar pairs with their original text content. This allows manual inspection of results to verify that:

- Detected pairs exhibit genuine textual similarity
- The Jaccard distance correlates with human perception of similarity

- Different types of duplicates (exact copies, minor variations, template reviews) are properly identified

This dual approach of quantitative metrics on synthetic data combined with qualitative analysis on real data provides comprehensive validation of the similarity detection system's effectiveness.

5 Results

The similarity detection system was evaluated on both the augmented dataset (with synthetic duplicates) and the original review data. Results demonstrate the effectiveness of the LSH approach for identifying duplicate content at scale.

5.1 Synthetic Duplicate Detection

On the augmented dataset containing 250 injected exact duplicates, the system achieved:

- **Recall**: 1.000 (250/250 duplicates detected)
- **Precision**: 0.774 (193/250 true positives)

The high recall indicates that the LSH algorithm successfully identifies exact duplicates, with only 2 pairs missed due to the probabilistic nature of the hashing. The precision value reflects that the system also discovered legitimate similar pairs beyond the synthetic duplicates, as well as false positives, which imply that further filtering steps might be needed .

6 Conclusions

In this project a scalable similarity detection system for the Amazon Books Review dataset was implemented through Apache Spark and Locality-Sensitive Hashing. The solution demonstrates how modern distributed computing frameworks combined with probabilistic algorithms can effectively address the computational challenges of finding similar items in massive text collections.

6.1 Key Achievements

The implemented system achieves several important objectives:

- **Computational Efficiency**: Reduces computational complexity, which could have been as high as $O(n^2)$ through implementation of LSH
- **Effective Detection**: Successfully identifies synthetic test duplicates while discovering numerous naturally occurring similar review pairs in the dataset

The evaluation results demonstrate the system's effectiveness: while maintaining high recall on known duplicates, the algorithm also discovered a significant number of naturally similar review pairs in the dataset (evidenced by the precision metrics). This indicates the system successfully identifies not only exact duplicates but also reviews sharing substantial content overlap - a valuable capability for real-world applications such as spam detection and review quality analysis. The preprocessing pipeline effectively handles data quality issues, while the binary term frequency vectors and multi-table LSH approach provide an optimal balance between detection capability and computational efficiency.

6.2 Limitations and possible improvements

Although the project has been designed with the clear intent of acting as a teaching agent in the context of algorithms used to treat massive bodies of data, the results, however satysfiying, leave room for improvement in various key aspects:

1. **Semantic Similarity of Reviews**: Current approach captures lexical overlap but misses semantically similar reviews with different vocabulary, opening up the possibility of integrating a NLP pipeline that compares beyond the simple lexicon of choice
2. **Parameter Optimization**: Systematic tuning of hash tables and bucket sizes could improve the precision-recall trade-off, whereas in this approach the parameters have mostly been chosen after (extensive) empirical testing
3. **In-depth search**: Due to the nature of the dataset, it would be possible to implement algorithms that perform more detailed searches based on user entries. This however seems to stray from the initial assignment of the project

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work, and including any code produced using generative AI systems. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.