Finding Similar Reviews

Algorithms for massive data

Daniele Piazza, 45740A*
*Data Science for Economics, Università degli Studi di Milano
daniele.piazza1@studenti.unimi.it

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.

Contents

1 Introduction **2**

2 Data Preparation **2**

3 Algorithms **3**
3.1 Tokenization and Shingling 3
3.2 Hashing 3
3.3 MinHashing 3
3.4 Locality Sensitive Hashing (LSH) 4
3.5 Jaccard Similarity 4
3.6 Pipeline 4

4 Results and Analysis **5**

5 Conclusion **7**

1 Introduction

The aim of this project is to implement a scalable system for identifying pairs of similar book reviews within the Amazon Books Reviews dataset [1]. To effectively manage millions of documents, the pipeline utilizes Apache Spark for distributed processing. The process of similarity detection relies on Locality Sensitive Hashing (LSH) in conjunction with MinHash signatures, a method intended to approximate set similarity in large datasets.

Our methodology begins by preprocessing and tokenizing individual reviews, subsequently transforming them into sets of k shingles, a consecutive word sequences of length k. To manage computational complexity, we hash each shingle into integer representations, creating sparse binary vectors that indicate shingle presence or absence within documents.

These hashed representations undergo transformation into compact MinHash signatures, which maintain probabilistic preservation of Jaccard similarity between original sets. The system then employs MinHash LSH implementation to identify candidate review pairs with high similarity probability by comparing signatures, thereby avoiding computationally expensive exhaustive pairwise comparisons throughout the entire dataset.

The original dataset consists of 3,000,000 book reviews, each containing metadata including reviewer identification, book titles, ratings, summaries, and review content. Our analysis concentrates exclusively on the review text field for similarity calculations, implementing comprehensive preprocessing and normalization procedures to prepare textual content for scalable comparison.

While similarity detection focuses only on processed review text, we subsequently integrate additional metadata fields such as reviewer IDs and book titles to categorize and interpret detected similar review pairs. This approach enables deeper understanding of dataset patterns, including instances where identical users post similar reviews across different books, or where distinct users generate nearly identical reviews for identical or unrelated titles.

2 Data Preparation

An essential preliminary step is the preparation and cleaning of the textual data. Given the structure and content of the Amazon Books Reviews dataset, the analysis concentrates on the `review/text` field, applying a series of preprocessing operations to produce clean and consistent textual data for similarity detection.

- **HTML entity replacement**: Encoded HTML entities (e.g., `&` → &) are replaced with their corresponding characters to recover the intended text.
- **Lowercasing**: All text is converted to lowercase to ensure case-insensitive comparison.
- **Special character removal**: Non alphanumeric characters, such as punctuation and symbols, are removed to retain only letters and digits to reduce ensure uniform input representation.
- **Exact duplicate removal**: Reviews that become identical after the previous preprocessing steps are removed. Since our task involves identifying pairs of similar, but not identical, texts, exact duplicates would trivially score a similarity of 1 and are therefore excluded.
- **Language filtering**: Only reviews written in English are retained, ensuring linguistic consistency and avoiding complications from multilingual content.
- **Stopword removal**: Common English stopwords (e.g., *the*, *and*, *is*, etc.) are removed. Eliminating these high-frequency, low-information terms reduces noise and improves the effectiveness of similarity detection, as meaningful distinctions between documents typically arise from rarer, content-specific terms whose occurrences are concentrated in a subset of documents and correlate with particular topics.
- **Length filtering**: We retain only reviews with a word count between 20 and 200. Extremely short or excessively long reviews are discarded, as they tend to skew similarity measures and increase computational overhead. The distribution of review lengths is shown in Figure 1.

After completing these preprocessing steps, each review consists of a cleaned, lowercase, English text of moderate length, stripped of non essential symbols and stopwords. This standardized text is then ready for tokenization and shingling.

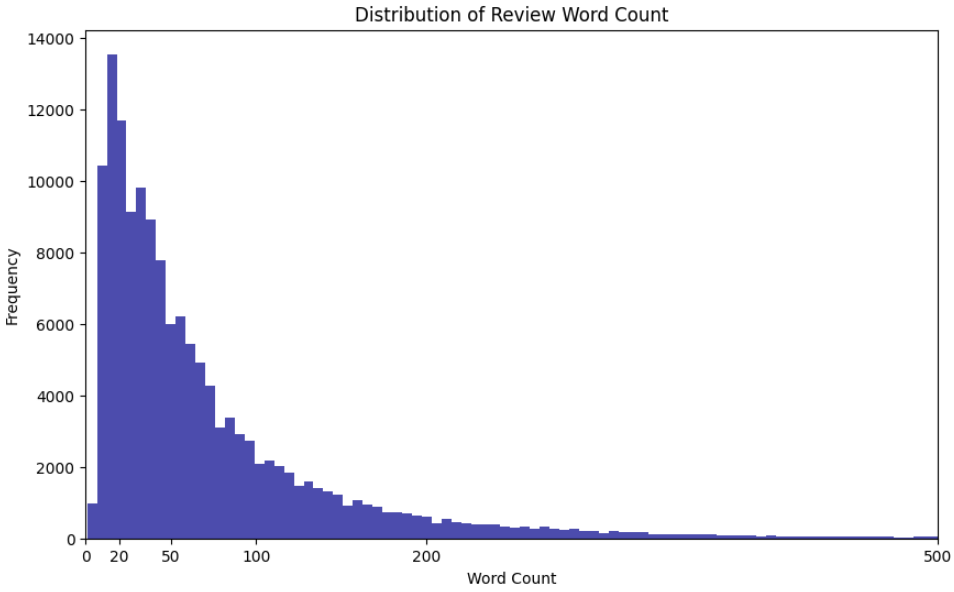

Figure 1: Histogram of review word counts. Most reviews are relatively short, with a long tail. Reviews containing 20–200 words are retained for analysis.

3 Algorithms

This section describes our scalable pipeline for identifying similar review pairs. The process transforms review text into token sets, groups these into shingles, converts shingles to numerical values through hashing, builds MinHashLSH structures for efficient candidate pair identification, and measures similarity using Jaccard similarity to quantify hashed shingle set overlap.

3.1 Tokenization and Shingling

After preprocessing, each review is split into individual tokens (words). These tokens are then transformed into a set of k shingles, which are contiguous sequences of k consecutive words extracted from the text. Since shingles are treated as sets, duplicate occurrences within the same document are ignored, focusing solely on unique content patterns.

Selecting appropriate k values proves critical in this process. Small k values increase probability of common shingles appearing across unrelated documents, artificially inflating similarity scores. Conversely, larger k values create more discriminative representations but may overlook partial similarities, particularly in shorter texts. [2]

In this implementation, we set $k = 3$, which offers a good trade-off between sensitivity to local word order and resistance to superficial text changes.

3.2 Hashing

To efficiently represent shingles and enable scalable comparison, we map each unique shingle to integer values through hash functions. These function distribute input keys uniformly across a predefined number of buckets, minimizing the risk of collisions and ensuring an approximately random distribution. [2]

In this project, the `HashingTF` [3] method is used, which hashes each shingle into an integer within a fixed-dimensional vector space, producing a sparse binary vector indicating the presence or absence of each shingle. This approach reduces memory requirements and enables fast set operations via integer arithmetic.

3.3 MinHashing

Since comparing high dimensional binary vectors directly is computationally expensive, we adopt `MinHashLSH` [4] to generate compact signatures that approximate Jaccard similarity between sets. This technique applies n independent hash functions $h_1, h_2, \ldots, h_n$ to the indices of the non zero elements in each hashed shingle vector and records the minimum hash value obtained for each function:

$$h_{\min}(A) = \min h(i) : i \in A$$

The resulting signature is a vector of n integers summarizing each document's content. The key property of MinHash is that the probability of two sets having the same value for a given hash function is equal to their Jaccard similarity. This allows for accurate estimation of set similarity using compact signatures instead of full dimensional vectors. [2]

In our pipeline, we use $n = 30$ hash functions, balancing estimation accuracy and computational overhead.

3.4 Locality Sensitive Hashing (LSH)

To efficiently identify pairs of documents with high similarity, we employ Locality Sensitive Hashing (LSH) on the MinHash signatures. The signature matrix is divided into b bands of r rows each. Within each band, the corresponding segment of each signature vector is hashed to a bucket. Document pairs whose signatures match in at least one band are considered candidate pairs.

This banding strategy exploits the fact that highly similar documents are more likely to agree in several consecutive hash values. The probability that a pair becomes a candidate depends on their Jaccard similarity and follows an S-shaped curve, where pairs above a certain threshold have a high chance of being considered candidates, while dissimilar pairs are unlikely to collide. [2] Band and row numbers control trade-offs between false positives and false negatives. Higher band numbers increase sensitivity, while fewer bands raise similarity thresholds, reducing candidate numbers.

In our implementation, we use Spark's `approxSimilarityJoin` to efficiently retrieve candidate pairs whose estimated similarity exceeds a chosen threshold.

3.5 Jaccard Similarity

The similarity between sets A and B is measured using the **Jaccard similarity**, defined as:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

This metric represents the ratio of the number of common elements to the total number of unique elements in both sets. While exact computation requires pairwise set intersection and union, which is computationally infeasible for large datasets, the MinHashLSH pipeline efficiently approximates this value, preserving a high probability of detecting near-duplicate documents while avoiding unnecessary comparisons between unrelated ones. [2]

3.6 Pipeline

A visual example of our complete processing pipeline is provided in Figure 2, demonstrating how sample Amazon book reviews transform through each step: HTML entity replacement, special character removal, lowercasing, stopword removal, tokenization, shingling, hashing, and MinHash signature generation.

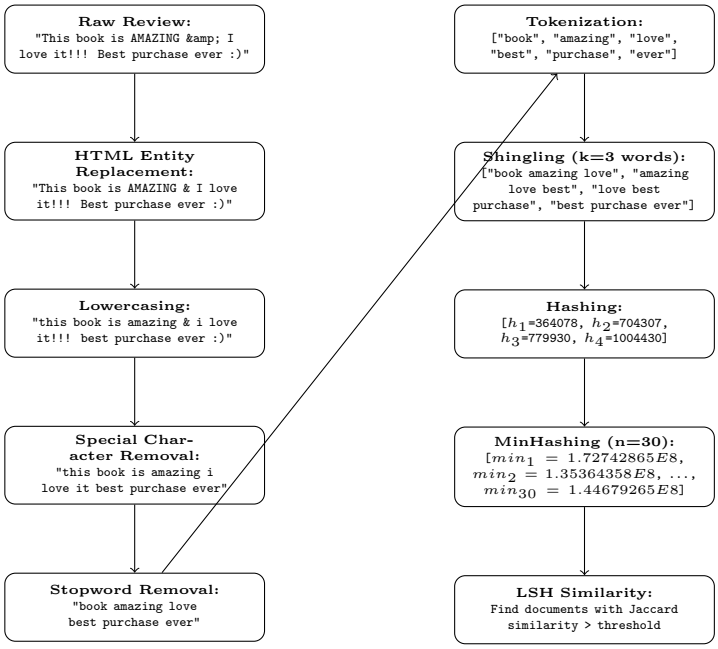

Figure 2: Complete text preprocessing pipeline showing transformation of sample review through all processing steps

4 Results and Analysis

We tested our pipeline on a 5% sample of the full dataset for efficiency purposes, but can be adjusted this via Boolean `SAMPLE` and `SAMPLE_FRACTION` variables.

Applying the pipeline to our sample yields numerous similar review pairs. Most of these are near duplicates or re-posted content. We categorized candidate pairs based on whether they share the same user IDs and/or book titles. The fraction of reviews in each category can be seen in Figure 3.

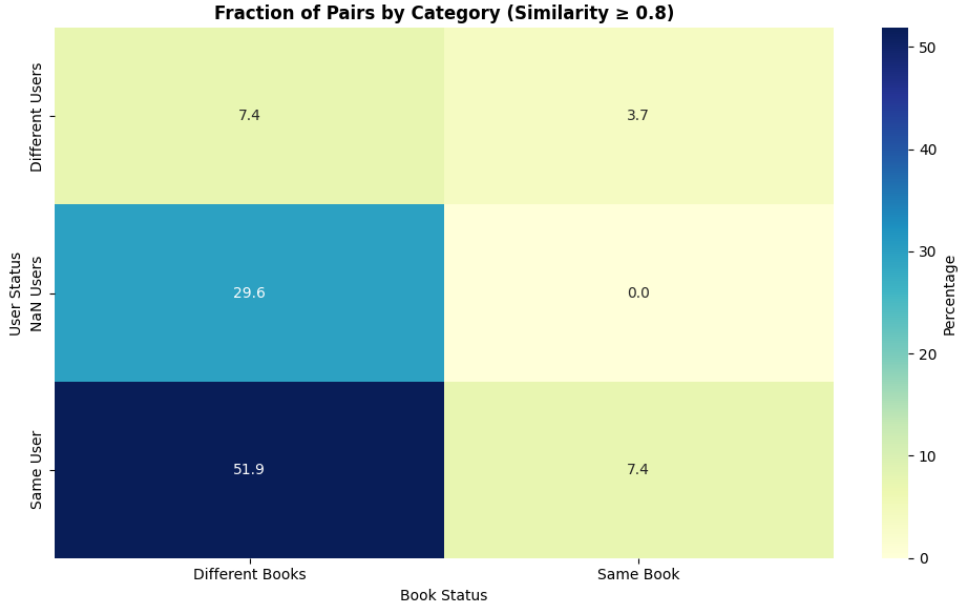

Figure 3: Matrix showing fraction of users for each category with Jaccard Similarity ≥ 0.8

- **Same user & different title**: The majority of flagged pairs (51.9%) involve the same user posting highly similar reviews under different book listings. This behavior is common when sellers list multiple editions of the same book under separate product entries. For instance, we identified

reviews for *"1984"* replicated under *"George Orwell 1984"*, where the content was nearly identical, with minor modifications such as altering the final phrase from *"Winston smith1984"* to *"Winston Smith"*.

- **Anonymous user & different title**: A substantial portion (29.6%) of pairs involve reviews without a registered user IDs, posted under different book titles. Many of these reviews share similar phrasing and often display patterns typical of spam or automated content. For example, we identified a cluster of reviews from anonymous or unregistered users that start by mentioning one book but quickly switch to promoting another title: *"Stolen Moments by Barbara Jeanne Fisher"*. This cluster appears in the top-left corner of Figure 4.

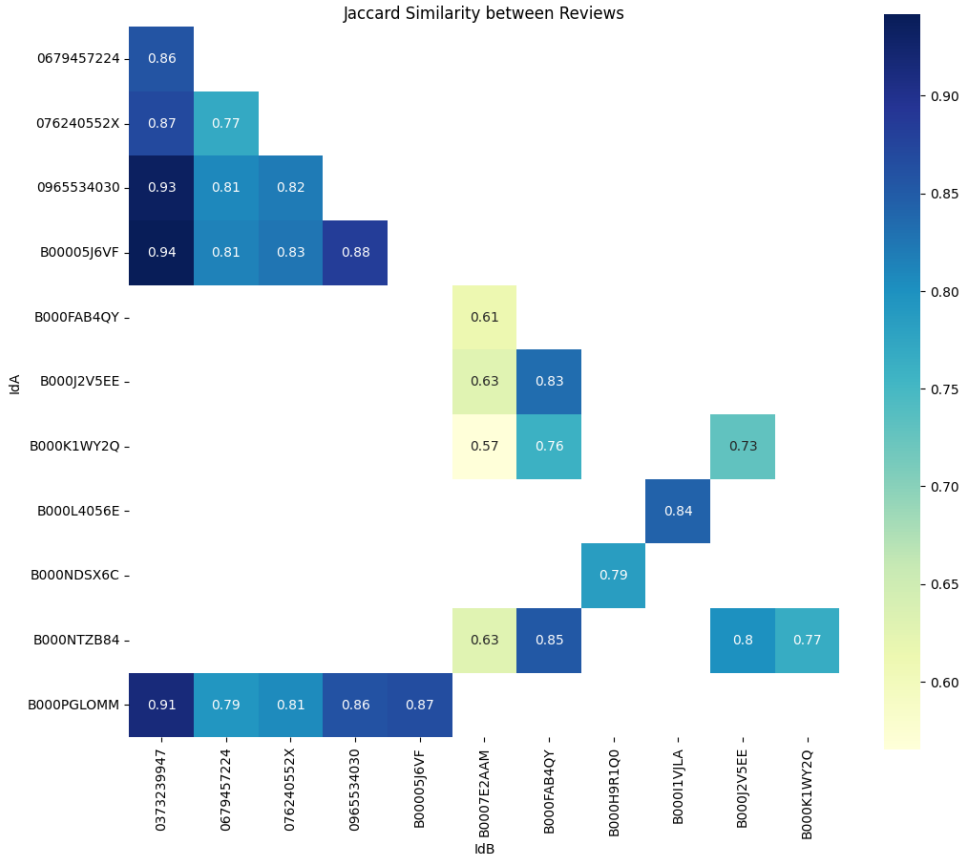

Figure 4: Matrix showing pairwise similarity scores between reviews, with distinct cluster of highly similar anonymous reviews in top-left corner

- **Different user & different title**: A smaller fraction (7.4%) of flagged pairs consist of completely different users reviewing different books with highly similar text. These are the most likely candidates for spam or generic reviews.
- **Same user & same title**: Only a limited number of pairs (7.4%) fall into this category, where the same user posted near-duplicate reviews for the same book. Since exact duplicates had already been removed during preprocessing, these cases typically involve minor wording variations or reposted content.
- **Different user & same title**: A minimal proportion (3.7%) of similar pairs involve different users reviewing the same book with almost identical text. These instances might include generic or potential cases of copy-paste behavior.

Illustrative matrices of similarity scores between books and users are shown in Figures 5.

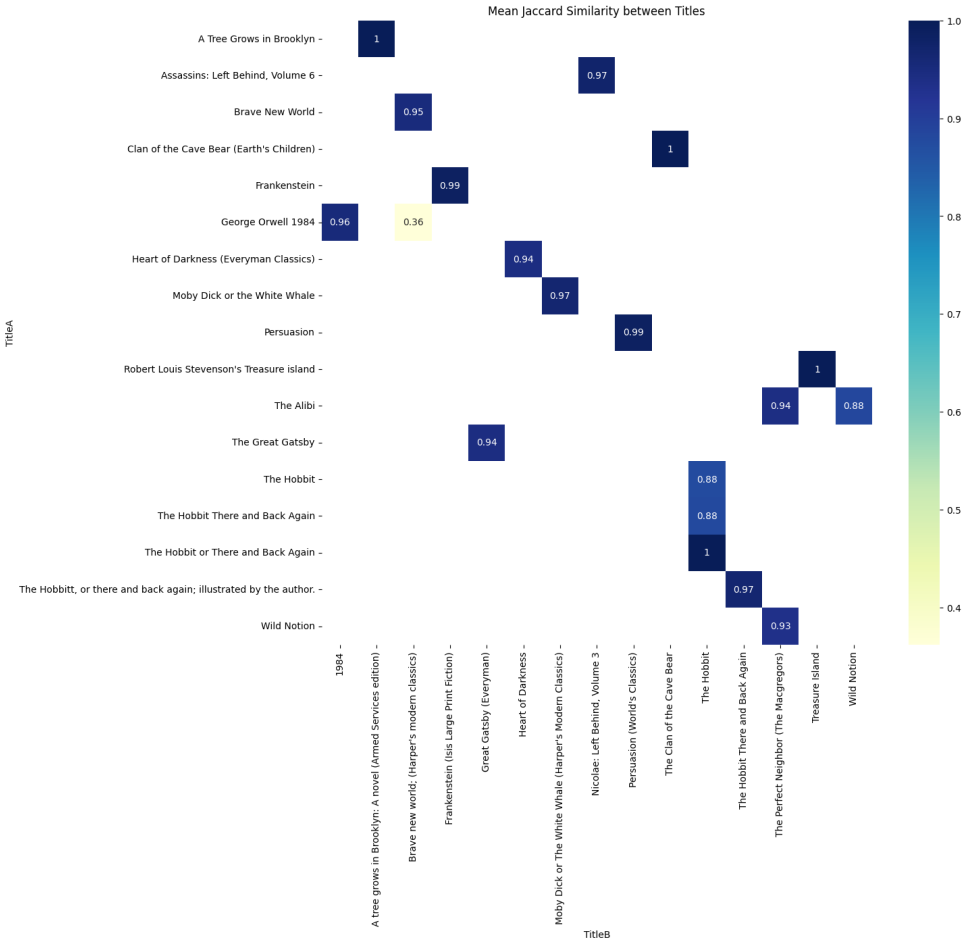

Figure 5: Matrix showing similarity between books, with noticeable books having similar titles suggesting different editions

5 Conclusion

This project showcased the efficacy of the MinHash-LSH approach in detecting near duplicate book reviews within large dataset. By employing text preprocessing, shingling, hashing, and MinHash signatures, coupled with Spark's distributed LSH implementation, the pipeline efficiently reduced the review corpus to a manageable set of highly similar candidate pairs, eliminating the need for exhaustive pairwise comparisons.

The results validated the reliability of this approach: the majority of the identified pairs were nearly identical reviews linked to different editions of the same book, or multiple submissions from the same user. Distinct reviews with varying phrasing were typically excluded, suggesting that the selected Jaccard similarity threshold (set at 0.8) struck an appropriate balance between sensitivity and precision.

A considerable number of near duplicate reviews originated from Amazon's multi edition product listings, which often led users to submit identical reviews under different book entries. Although these instances are innocuous, the pipeline also uncovered a smaller group of questionable reviews: nearly identical texts associated with unrelated book titles, frequently submitted by anonymous or unregistered users. These occurrences likely indicate spam, promotional material, or automatically generated reviews.

In conclusion, the MinHashLSH pipeline proved to be a scalable and effective solution for approximate similarity detection in massive text datasets. It successfully filtered redundant content, highlighted meaningful duplicate patterns, and exposed potential spam cases, all while maintaining computational efficiency through distributed processing.

References

[1] Mohamed Bekheet. *Amazon Books Reviews*. 2022. URL: https://www.kaggle.com/datasets/mohamedbakhet/amazon-books-reviews.

[2] Anand Rajaraman Jure Leskovec and Jeffrey D. Ullman. *Mining of Massive Datasets*. Cambridge University Press, 2014.

[3] Spark. *HashingTF*. URL: https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.feature.HashingTF.html#pyspark.ml.feature.HashingTF.numFeatures.

[4] Spark. *MinHashLSH*. URL: https://spark-apache-org.translate.goog/docs/latest/api/python/reference/api/pyspark.ml.feature.MinHashLSH.html?_x_tr_sl=en&_x_tr_tl=it&_x_tr_hl=it&_x_tr_pto=sc.