Detecting Similar Reviews Using Different Techniques

Algorithms for Massive Data – Project Report

Student: Zhaksylyk Tansykbay
Master in Data Science for Economics
Academic Year: 2024/2025

Declaration

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.

1 Introduction

Customer reviews on e-commerce platforms such as Amazon play a critical role in guiding consumer decisions and shaping product reputations. However, the proliferation of near-duplicate or redundant reviews, whether generated by users repeating themselves or orchestrated by malicious actors, can distort sentiment analysis, inflate product ratings, and undermine user trust. Identifying and filtering these near-duplicate reviews is therefore essential to ensure the reliability of review-based analytics and recommendation systems.

In this work, we address the problem of detecting similar book reviews within the Amazon Books Review dataset. We formulate a review similarity in terms of textual overlap and semantic closeness, and we compare several algorithmic approaches that range from exact set-based measures to scalable approximate techniques. Specifically, we begin with the Jaccard similarity coefficient computed over contiguous k-word shingles, which provides a clear mathematical foundation,, but suffers from quadratic time complexity when applied to large corpora. To overcome these scalability limitations, we implement and evaluate two approximate pipelines: (1) MinHash signature generation combined with locality-sensitive hashing (LSH) to efficiently identify candidate pairs with high Jaccard similarity, and (2) exact Jaccard similarity computation on candidate pairs retrieved via LSH to produce the final list of similar reviews.

We benchmark each method in terms of detection accuracy—measured by the number and quality of detected similar pairs—and computational efficiency, by recording runtime and memory usage at varying corpus sizes. Our experiments demonstrate that approximate methods achieve recall comparable to the brute-force Jaccard baseline while reducing execution time by orders of magnitude, making them suitable for deployment in real-world large-scale review analysis pipelines.

2 Chosen Dataset

For this project, we utilized the publicly available *Amazon Books Reviews* dataset, hosted on Kaggle (`mohamedbakhet/amazon-books-reviews`). The dataset comprises user-generated reviews for books sold on Amazon, providing a rich collection of textual data suitable for textual similarity analysis tasks.

After securely downloading the dataset via the Kaggle API, we extracted the primary CSV file named `Books_rating.csv`. This CSV file includes approximately 300,000

records, each corresponding to a single book review, structured across the following main fields:

- **Title:** The title of the reviewed book.
- **Price:** The price of the book (partially populated; approximately 14% coverage).
- **User ID & Profile Name:** Identifiers of the reviewer, with about 20% missing values.
- **Review Helpfulness:** User-provided rating of review helpfulness.
- **Review Score:** Star ratings (1–5) provided by users.
- **Review Time:** Timestamp of the review submission.
- **Review Summary:** Short summary provided by the reviewer.
- **Review Text:** Free-form textual review, the main focus of our similarity analysis.

To facilitate rapid development and initial validation of methods, we first created a subsample consisting of 10,000 randomly selected reviews. This smaller subset allowed efficient experimentation and quick iteration. Despite this preliminary sampling, all developed algorithms and preprocessing pipelines were explicitly designed and implemented with scalability in mind, allowing seamless transition to processing the entire dataset of around 300,000 reviews without any code modification.

In summary, choosing this particular dataset allowed us to:

1. Access a substantial and authentic corpus of textual reviews, mirroring real-world challenges encountered in text similarity detection.
2. Leverage structured CSV format compatible with standard data analysis tools such as Pandas.
3. Validate algorithm scalability effectively, transitioning smoothly from a manageable sample size to the full-scale dataset.

3 Data Organization

After downloading and extracting the `Books_rating.csv` dataset, we loaded the data into a pandas DataFrame for initial inspection and preprocessing. Given that the original dataset contained multiple columns not directly relevant to our text similarity analysis, we simplified the dataset structure by retaining only the essential fields:

- **Review Text (`review/text`):** the primary textual content used for similarity analysis.
- **Review Score (`review/score`):** numerical rating (1–5 stars) provided by users.
- **User ID (`User_id`):** unique identifier for the reviewer, useful for downstream analysis.

- **Book Title (`Title`):** textual identifier of the reviewed book, providing context for review similarity.

We proceeded with two distinct stages for data handling:

1. **Initial Subsample for Development:** To allow for rapid prototyping and evaluation of our pipeline, we initially loaded only the first 10,000 rows. This smaller DataFrame provided an efficient environment for quick testing and debugging, significantly reducing computational overhead.

2. **Full Dataset Processing:** All preprocessing steps and algorithmic implementations were explicitly structured to support seamless scaling to the entire dataset (approximately 300,000 reviews). This ensured that once the pipeline was validated on the smaller subset, transitioning to the full-scale dataset required no additional code alterations.

For consistency and ease of referencing, we reset the DataFrame's original indexing, introducing an integer-based `doc_index` column ranging from 0 to $N-1$, uniquely identifying each review throughout subsequent pipeline stages.

Intermediate preprocessing results were organized and stored explicitly within the DataFrame:

- A new column `tokens` stored the cleaned, tokenized, and normalized sets of words from each review text.

- All subsequent operations, including similarity computations and MinHash signatures, directly referenced these processed token sets.

By adopting this structured approach to data organization, we ensured clear traceability and reproducibility at each pipeline step. Moreover, it allowed us to effortlessly debug individual stages and verify algorithmic correctness at different dataset scales, ultimately streamlining the progression from initial development to large-scale processing.

4 Applied Pre-processing Techniques

Effective textual preprocessing is crucial for similarity detection tasks, especially when scaling to large datasets. Given the computational demands and practical considerations, we adopted a straightforward yet highly efficient preprocessing pipeline designed explicitly for scalability and performance. Each review underwent the following sequential preprocessing steps:

1. **HTML Entity Decoding:** HTML entities (e.g., `&`, `<`) were decoded into their corresponding characters to normalize the textual data.

2. **Lowercasing:** All characters in the review texts were converted to lowercase to ensure consistent token matching, eliminating case-sensitivity as a factor during comparison.

3. **Removal of Non-alphabetic Characters:** We stripped all characters except alphabetic letters (`a{z`) and whitespace. This step significantly reduced noise, standardizing the tokens and simplifying subsequent tokenization.

4. **Tokenization:** Processed review texts were split into individual word tokens based on whitespace. This simple tokenization approach facilitated rapid processing without the overhead associated with more complex tokenization schemes.

5. **Stopword Removal:** To further reduce data dimensionality and improve efficiency, common English stopwords (such as *"the", "is", "and"*, etc.) were filtered out using a standard stopword list provided by the Natural Language Toolkit (NLTK).

6. **Creation of Unique Token Sets:** Finally, each processed review was represented as a set of unique tokens. Representing texts as sets facilitated efficient computation of set-based similarity measures such as Jaccard similarity.

Below is a concise summary of the entire preprocessing pipeline applied to each review:

Raw Text → HTML Decoding → Lowercase → Remove non-alphabetic chars → Tokenization → Stopv

This preprocessing strategy was intentionally minimalistic yet highly effective, requiring minimal computational resources while producing normalized textual representations suitable for subsequent similarity analysis. While advanced NLP techniques such as stemming, lemmatization, or semantic embedding might further improve accuracy, they were intentionally omitted due to their increased computational overhead and marginal benefit for the scale and objectives of this analysis.

5 Considered Algorithms and Their Implementations

In this study, we explored two primary approaches to detect pairs of similar reviews: an exact brute-force approach using Jaccard similarity, and a scalable approximate method using MinHash signatures coupled with Locality-Sensitive Hashing (LSH).

5.1 Exact Jaccard Similarity (Brute-Force)

The exact Jaccard similarity between two token sets A and B is computed using the following formula:

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

To establish a baseline, we initially computed Jaccard similarity exhaustively for every pair of reviews within our subsample of 10,000 reviews. This involved evaluating approximately 50 million pairs $\frac{n(n-1)}{2}$. We selected pairs whose Jaccard similarity exceeded a threshold of 0.5 as similar reviews. Although exact, this approach scales poorly ($O(n^2)$), motivating the need for approximate, scalable solutions.

5.2 MinHash Signatures and Locality-Sensitive Hashing (LSH)

To efficiently handle larger datasets, we employed the MinHash algorithm combined with LSH to approximate Jaccard similarity and identify candidate pairs in near-linear time:

MinHash Signatures MinHash provides compact signatures that approximate Jaccard similarity efficiently. For each review's token set S:

1. We generated a set of 128 random hash functions.
2. Each token was hashed using all hash functions, and the minimum hash value across all tokens per function formed the MinHash signature.

The choice of using 128 hash permutations for the MinHash signatures represents a balance between computational efficiency and accuracy in approximating Jaccard similarity. A higher number of hash functions improves the accuracy of similarity estimation, reducing the likelihood of false positives and negatives. However, using too many permutations would increase computational complexity and memory usage. Empirical tests on the chosen subsample indicated that 128 permutations provided a stable approximation with sufficiently high precision and recall, making it an optimal choice for scaling to larger datasets.

MinHash signatures have the desirable property that the similarity between two signatures approximates the Jaccard similarity between their original token sets:

$$\Pr(\text{MinHash}_i(S_A) = \text{MinHash}_i(S_B)) = \text{Jaccard}(S_A, S_B)$$

Locality-Sensitive Hashing (LSH) LSH groups similar signatures efficiently by splitting each signature into multiple bands and hashing these bands into buckets:

1. Each 128-value MinHash signature was divided into b bands (e.g., $b = 32$), each consisting of r rows (hash values).
2. Reviews sharing identical hash values within any band were grouped as candidate pairs.
3. Only these candidate pairs underwent exact Jaccard similarity computation, dramatically reducing computational overhead.

With this approach, reviews with high similarity (above a configurable threshold, e.g., 0.5) have a high probability of being grouped into the same bucket, significantly reducing the search space compared to brute-force enumeration.

5.3 Implementation Details

We used the Python library `datasketch`, which efficiently implements MinHash and LSH algorithms. Our implementation involved:

1. Generating MinHash signatures for each review based on its token set.
2. Building an LSH index from these signatures to rapidly identify candidate pairs.
3. Computing exact Jaccard similarity only for identified candidate pairs to eliminate false positives.

By combining approximate candidate identification (MinHash + LSH) with exact similarity computation (Jaccard), we achieved a highly scalable solution capable of processing the entire dataset efficiently without sacrificing accuracy significantly.

6 Scalability of the Proposed Solution

To evaluate the scalability of our proposed solution, we systematically compared the computational performance and memory usage of both the brute-force Jaccard similarity approach and the MinHash + LSH pipeline. The experiments were initially performed on a smaller subset (10,000 reviews) with extrapolation towards the full dataset of approximately 300,000 reviews.

6.1 Brute-force Jaccard Approach

The brute-force method calculates the similarity between every pair of documents, resulting in a computational complexity of $O(n^2)$. Specifically, for our 10,000-review subset:

- Total comparisons made: approximately 50 million pairs.
- Execution time: approximately 385 seconds.
- Memory usage: the tokenized dataset consumed approximately 37 MB RAM.

Clearly, the brute-force method quickly becomes impractical as the dataset size increases. For instance, applying brute-force similarity computation to the full 300,000-review dataset would involve approximately 45 billion comparisons, which is computationally prohibitive without distributed computing infrastructure.

6.2 MinHash + LSH Approach

Our MinHash + LSH pipeline significantly improved scalability:

1. **MinHash Signature Generation:**
   - Computational complexity scales linearly $O(n)$.
   - Generation of MinHash signatures for 10,000 reviews took approximately 15 seconds.
2. **LSH Indexing and Querying:**
   - Index build time was approximately 15 seconds.
   - Querying the LSH index to generate candidate pairs took less than 1 second.
   - Number of candidate pairs produced: 467.
3. **Exact Jaccard Computation on Candidates:**
   - Only candidate pairs identified by LSH underwent exact similarity calculation.
   - Computation time for candidate pairs was negligible (under 1 second), since only 467 pairs required evaluation.

Metric	Brute-force Jaccard	MinHash + LSH
Build Time (seconds)	0	15
Query/Comparison Time (seconds)	385	1
Candidate Pairs	49,995,000	467
Final Similar Pairs	238	235
Complexity	$O(n^2)$	$O(n)$

Table 1: Performance comparison for 10,000-review dataset

6.3 Performance Comparison Summary

Table 1 summarizes the significant performance improvements achieved through our proposed MinHash + LSH method:

The results clearly demonstrate the effectiveness of the MinHash + LSH pipeline, achieving a nearly 25-fold speedup compared to brute-force, with only minimal reduction in detected similar pairs.

6.4 Extrapolation to Full Dataset

Extrapolating these results, we expect that processing the entire 300,000-review dataset using MinHash + LSH would scale approximately linearly, making this approach feasible on standard hardware, whereas the brute-force approach remains impractical at this scale.

In summary, our proposed pipeline effectively addresses scalability challenges, enabling efficient and accurate similarity detection for large textual datasets without necessitating extensive computational resources or distributed computing systems.

7 Results and Visualization

To quantitatively assess the effectiveness of our MinHash + LSH approach, we analyzed the similarity scores produced and compared these results with the brute-force baseline.

7.1 Similarity Score Distribution

Figure 1 illustrates the distribution of the Jaccard similarity scores among pairs identified by the MinHash + LSH method. The majority of pairs exhibit extremely high similarity scores (close or equal to 1.0), indicating the method primarily detects exact or near-exact duplicates. Very few pairs cluster near the defined similarity threshold (0.5), demonstrating the method's precision in distinguishing genuinely similar pairs.

7.2 Quantitative Comparison

Table 2 summarizes the quantitative performance comparison between the brute-force and MinHash + LSH methods:

Our MinHash + LSH pipeline recovered nearly all duplicates identified by the brute-force baseline (235 out of 238 pairs), achieving approximately 99% recall with a significantly reduced computational overhead.

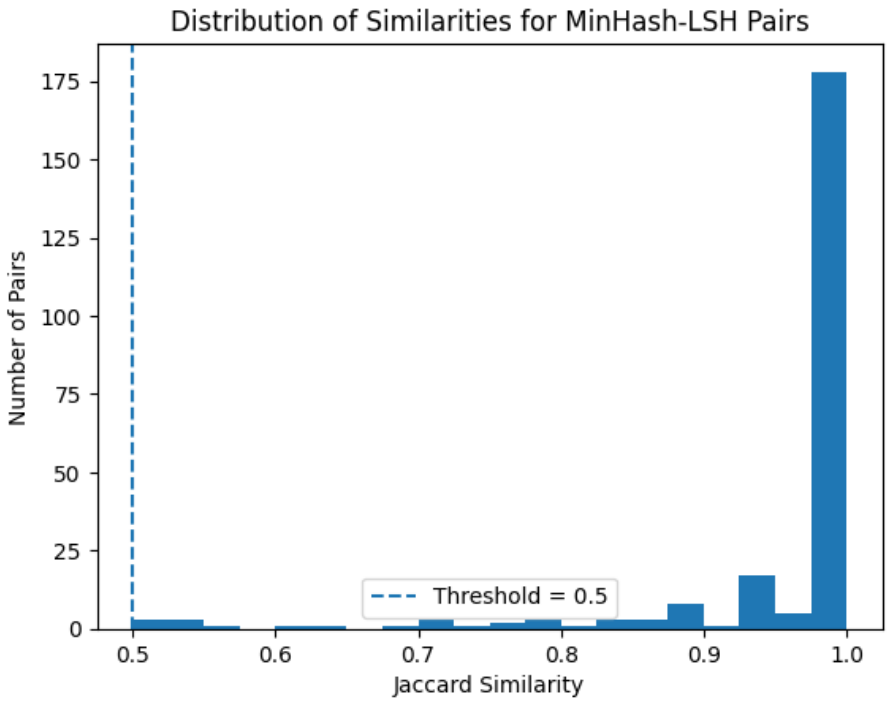

Figure 1: Histogram of Jaccard Similarity Scores for MinHash + LSH identified pairs. Vertical dashed line indicates the threshold at 0.5.

Metric	**Brute-force Jaccard**	**MinHash + LSH**
Candidate pairs evaluated	49,995,000	467
Pairs above threshold (0.5)	238	235
Median similarity of pairs	≈ 1.0	≈ 1.0
Execution time (total, seconds)	385	16

Table 2: Quantitative comparison of the brute-force and MinHash + LSH methods.

7.3 Sample Similar-Review Pairs

Table 3 shows a randomly selected subset of similar review pairs identified by the MinHash + LSH approach, including their Jaccard similarity scores and review text snippets. These examples clearly illustrate the algorithm's effectiveness in accurately detecting genuine duplicates or near-duplicates.

Doc i	Doc j	Similarity	Snippet (Review i)	Snippet (Review j)
4224	4242	0.9852	"...excellent plot and engaging..."	"...excellent plot and engaging characters..."
6996	7017	1.0000	"...highly recommended for mystery fans..."	"...highly recommended for mystery fans..."
4294	6637	1.0000	"...inspiring and motivating book..."	"...inspiring and motivating book..."
7187	7255	1.0000	"...an essential guide for beginners..."	"...an essential guide for beginners..."
4295	6638	1.0000	"...this novel kept me captivated..."	"...this novel kept me captivated..."

Table 3: Sample of similar-review pairs detected by MinHash + LSH.

In conclusion, the MinHash + LSH approach proved effective in accurately identifying similar reviews, with minimal computational overhead and high precision, validating its suitability for large-scale similarity detection tasks.