Similar Amazon Book Reviews

Maedeh Rabiee

Algorithms for Massive Data Project

Data science for Economics(DSE)

Professor Malchiodi

October 2025

University of Milan

Table of Contents

Introduction

This project focuses on detecting similar book reviews within a large textual dataset using Jaccard similarity, MinHash signatures, and Locality-Sensitive Hashing (LSH). The primary objective is to efficiently identify near-duplicate or highly similar reviews among millions of Amazon Book Reviews without performing exhaustive pairwise similarity computations across all documents.

Traditional similarity computation exhibits a quadratic time complexity of $O(n^2)$, rendering it computationally infeasible for large-scale datasets. To address this challenge, MinHashing and LSH techniques are employed to approximate Jaccard similarity and drastically reduce the number of required comparisons while maintaining reliable accuracy.

For implementation, the Datasketch Python library was utilized to perform both MinHash signature generation and LSH indexing. Datasketch offers efficient and well-optimized data structures for probabilistic similarity estimation, enabling scalable processing of large textual datasets.

Furthermore, Datasketch performs exceptionally well within the free tier of Google Colab, which provides limited memory and computational resources. Its efficient design allows the entire pipeline—from shingling to similarity detection—to execute smoothly even in constrained environments, without requiring paid GPU/TPU support. This makes Datasketch a practical and powerful choice for academic experiments and large-scale text similarity detection tasks conducted on cloud-based platforms such as Colab.

Dataset Description

The dataset was obtained from Kaggle:

"Amazon Books Reviews" by Mohamed Bakhet

The dataset used in this project contains approximately 3 million Amazon book reviews, covering 212,404 unique book titles and their corresponding user reviews. It comprises 10 columns, each providing distinct metadata about books and reviewers:

Column Name	Description
Id	Unique book identifier
Title	Title of the book
Price	Price of the book
User_id	Reviewer's unique ID
profileName	Reviewer's name
review/helpfulness	Ratio of helpful votes
review/score	Rating (1–5)
review/time	UNIX timestamp
review/summary	Short summary
review/text	Full text of the review

Overall, the dataset is rich in textual data and provides sufficient variation for natural language analysis. The review/text column served as the primary source for the similarity detection task, as it contains the complete textual content of each review.

Exploratory Data Analysis (EDA)

Before applying the algorithms, I performed an initial Exploratory Data Analysis (EDA) to understand the structure, quality, and potential issues in the dataset.

Missing Values

The table below shows the number and percentage of missing values in each column:

Column	Nulls	Percent (%)
Id	0	0.00%
Title	208	0.01%
Price	2,518,829	83.96%
User_id	561,787	18.73%
ProfileName	561,905	18.73%
Review/Helpfulness	0	0.00%
Review/Score	0	0.00%
Review/Time	0	0.00%
Review/Summary	407	0.01%
Review/Text	8	0.00%
Total	3,643,144	—

Duplicate Detection

A duplicate check revealed:

- 8,774 rows where all columns were identical (0.29% of the dataset).

Data Organization and Pre-processing

- Firstly Duplicated 8,774 rows where all columns were identical were removed to ensure that each review was unique.

For similarity analysis, I retained only the two relevant columns: 'Id' and 'review/text'.
Rows with missing review texts were dropped, leaving 2,991,218 unique reviews, which were saved in the df_clean DataFrame.

Sampling

Due to computational constraints, a 1% random sample (~29,912 reviews) was taken for experimentation and saved as df_sample. This subset was large enough to test the efficiency of LSH while maintaining the dataset's diversity.

Text Normalization

Each review was normalized using the following steps, and the cleaned text was saved in the text norm column:

- Convert text to lowercase
- Remove punctuation and non-alphanumeric characters
- Collapse multiple spaces

Shingling

The next step was word-level shingling, where each review was divided into overlapping groups of three consecutive words (k = 3). We chose k = 3 because it provides a balanced trade-off between recall and precision. Smaller k values (e.g., k=2) increase recall but can merge semantically different texts, while larger k values (k=4–5) increase precision but may miss paraphrases.

Then, Stopwords (such as *the*, *and*, *is*) were removed to reduce noise.

Example:

Text: "I really enjoyed reading this book"
→ Shingles (k=3): {"really enjoyed reading", "enjoyed reading book"}

Each document was therefore represented as a set of shingles, capturing local word patterns and phrase structure. This representation was saved in the shingles column.

MinHash Signatures

Since directly comparing large sets is computationally expensive, I applied **MinHashing** to create compact representations of each review's shingles. MinHash simulates random permutations to produce short signatures, where the probability that two signatures match at a given position equals their Jaccard similarity.

I used 128 permutations per document to balance accuracy and memory efficiency. The resulting signatures were stored in the minhash column.

Locality-Sensitive Hashing (LSH)

To avoid comparing every pair of the 29,912 reviews, Locality-Sensitive Hashing (LSH) was applied on top of the MinHash signatures. LSH divides each signature into bands and hashes them into buckets. Reviews that fall into the same bucket in at least one band are considered candidates for being similar.

This method drastically reduced the number of required pairwise comparisons while maintaining high recall for near-duplicate reviews.

LSH Parameters and Setup

The following parameters were used for Locality-Sensitive Hashing (LSH):

- **Threshold (t):** 0.6 (minimum Jaccard similarity)
- **Bands (b):** 18
- **Rows per band (r):** 7 $\Rightarrow$ ($18 \times 7 \approx 128$ hash values)

A threshold of **0.6** was chosen as it provides a balanced trade-off between recall and precision.
A lower threshold (e.g., 0.5) would increase the number of candidate pairs but also the verification cost, whereas higher thresholds would risk missing potential duplicates.

For **LSH index construction**, a unique identifier was generated for each review to enable efficient indexing.
Specifically, the identifier was created by concatenating the book_id with the corresponding row index (in the format book_id_row_index). This composite key ensured that each review remained distinct while maintaining its association with the original book information. The resulting identifier was stored in the unique_key column.

To ensure accuracy, self-pairs and duplicate pairs were removed prior to verification.

This process yielded the following results:

- **Candidate pairs:** 331
- **Total possible pairs:** 29,912 $\times$ 29,911 / 2 = 447,348,916
- **Reduction:** 99.9999% fewer comparisons

These results clearly demonstrate how scalable and efficient the LSH approach is for massive datasets.

Similarity Computation and Verification

The Jaccard similarity function was applied to compute the exact similarity between the shingles of each candidate pair.
This function measures the ratio of the intersection to the union of two sets, defined as:

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

Only pairs with a Jaccard similarity above 0.6 were retained as verified duplicates or near-duplicates

Results

- 331 verified similar pairs
- Mean similarity: 0.994
- Median similarity: 1.000
- Min similarity:0.667

These metrics confirm that the LSH candidate selection was highly precise, with nearly all shortlisted pairs exhibiting strong semantic or literal overlap.

Similarity Distribution

The table below summarizes the distribution of similarity scores among the 331 verified pairs:

Range	Count	Percentage (%)
0.60 – 0.65	0	0.0%
0.65 – 0.70	1	0.3%
0.70 – 0.75	3	0.9%
0.75 – 0.80	0	0.0%
0.80 – 0.85	2	0.6%
0.85 – 0.90	0	0.0%
0.90 – 0.95	5	1.5%
0.95 – 1.00	320	96.7%
Total	331	100.0%

Similarity Evaluation

The resulting similarity distribution (Table 1) revealed that 96.7% of all pairs had a similarity score between 0.95 and 1.00, confirming that nearly all detected pairs were true near-duplicates or identical reviews.

This high-precision outcome demonstrates the effectiveness of the chosen parameters — 3-word shingles, 128 MinHash permutations, and an LSH threshold of 0.6 — in filtering out dissimilar reviews while successfully retaining genuine duplicates.

All results were exported into multiple files for transparency and further analysis:

- similar_pairs.csv – a compact version containing review identifiers and similarity scores.

- similar_pairs_FULL_TEXT.csv – includes the complete review texts for each pair.
- similar_pairs_FULL_REPORT.txt – a detailed, human-readable summary of all matched pairs.
- summary_statistics.txt – provides overall statistics and the similarity distribution.

Together, these outputs formed a complete, scalable pipeline capable of detecting and verifying near-duplicate reviews within the Amazon Books Reviews dataset.

Example of Similar Pairs

Example (Similarity = 1.000)

PAIR #1

Similarity Score: 1.000

Key 1: B000GLN7HQ_18665

Key 2: B000HWEE7Q_22204

REVIEW 1 (COMPLETE TEXT):

This would be a good starting place for someone unfamiliar with Christie's works. It is one of the Hercule Poirot detective stories; I hadn't read any of them before and therefore can say that it doesn't seem to have a negative effect reading this one out of sequence. (It did make me want to read more of the series though!)The story is set in a small English village where gossip is the favored pasttime and nothing much of significance ever really happens; however, the village inhabitants are stunned when two mysteries unfold at once: one involving blackmail and a suicide, and the other involving murder. Hercule Poirot, who has just moved to the village to retire, is irresistibly pulled into the intrigue and starts to investigate. He is aided by the village doctor, Dr. Sheppard, who narrates the tale.It's been a long time since I was completely absorbed in a mystery, and this delivered that deliciously maddening desire to have to know-right now-- who done it? It's difficult to put it down once started, and the ending was nothing short of nail-biting shock. Christie is a master at keeping the reader guessing and then delivering an emotionally stunning wrap-up of her tale. This one kept me on the edge of my seat, and I can't wait to read more of her novels.

REVIEW 2 (COMPLETE TEXT):

(Identical to Review 1)

PAIR #331

Similarity Score: 0.667

Key 1: 0460112872_19140

Key 2: B000L28LS0_15176

REVIEW 1 (COMPLETE TEXT):

Easy to get on my Kindle and to transport it everywhere. Rapid download to all my connected electronics. And you cannot go wrong with so little money for it. Love those classics.

REVIEW 2 (COMPLETE TEXT):

Easy to get on my Kindle and to transport it everywhere. Rapid download to all my connected electronics. And you cannot go wrong with free.

Conclusion

In this project, I built a scalable end-to-end pipeline to detect duplicate and near-duplicate reviews within the Amazon Books Reviews dataset, which contains approximately three million entries.

The approach is simple yet effective. Each review was converted into 3-word shingles (k=3), then compressed using MinHash with 128 random permutations, and finally indexed with Locality-Sensitive Hashing (LSH) using a Jaccard similarity threshold of 0.6 to efficiently shortlist candidate pairs.

When tested on a 1% random sample (29,912 reviews), the total number of possible review pairs (about 4.47×10) was reduced to just 331 candidates by the LSH step. After applying the exact Jaccard similarity, all 331 pairs were confirmed to have similarity values above 0.6, proving that the method achieved high accuracy while drastically reducing computational cost.

The entire pipeline was implemented using the Datasketch Python library, which provides simple yet powerful tools for MinHashing and LSH. Moreover, it runs smoothly on Google Colab's free environment, making it an ideal solution for large-scale text processing without requiring high-end hardware.

Overall, this project demonstrated that MinHashing and LSH, when combined with the Datasketch library, provide a fast, memory-efficient, and reliable approach for solving large-scale text similarity problems.

References

1. Rajaraman, A., & Ullman, J. D. (2011). *Mining of Massive Datasets*. Cambridge University Press.
2. Datasketch Python Library: https://ekzhu.github.io/datasketch

Declaration by author

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. 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.