Project 1: Finding similar items

Shayakhmetova Ayagoz ID - 29448A

October 27, 2025

1 Introduction

This project implements the MinHash and Locality-Sensitive Hashing algorithm to efficiently detect near-duplicate reviews in the Amazon Books Reviews dataset. The approach reduces computational complexity to approximately $O(n)$ while maintaining high precision. The implementation is based on Chapter 3 of *Mining of Massive Datasets.*

2 Dataset description

2.1 Source and sampling

The Amazon Books Reviews dataset was obtained from Kaggle. The complete dataset contains 3 million book reviews. During the testing phase for selecting the sample size, it was decided that 50,000 observations represent the most optimal choice, since a larger sample was more computationally demanding, while a smaller one made the analysis of results less meaningful and reliable. These 50 000 rows are taken randomly, but with set seed(42), because it allows to work with the same 50 000 rows, for better understanding. Comparing every review against every other review of the whole dataset would require over 1.2 billion comparisons—taking weeks on standard hardware.

2.2 Dataset characteristics

The dataset has following attributes:
- `review/text`: full text content of the review;
- `review/summary`: review title;
- `title`: book title;
- `user_id`: anonymized user identifier;
- `review/score, review/helpfulness`: numerical rating (1-5 stars).

2.3 Dataset characteristics

After sampling dataset exhibited following characteristics:
- total reviews: 50 000;
- unique books: 24 285;
- unique users: 35 350;
- average review length: 143.38 words (before preprocessing).

Rating	Count
1.0	3,369
2.0	2,557
3.0	4,261
4.0	9,677
5.0	30,136

Table 1: Rating distribution

Now that a rough idea of the dataset is taken, the next step is to clean the data if necessary.

3 Preprocessing

Data preprocessing was performed in multiple stages to standardize text format, reduce noise, and prepare the data for similarity detection.

3.1 Text cleaning

The following text normalization steps were applied to all reviews:

1. **HTML entity decoding.** HTML entities (e.g., `&amp;`, `&quot;`) were converted to their Unicode equivalents using Python's `html.unescape()`.
2. **Case normalization.** All text was converted to lowercase to ensure case-insensitive matching.
3. **Character filtering.** Non-alphanumeric characters (except spaces) were removed using regular expressions to eliminate punctuation and special symbols.
4. **Whitespace normalization.** Multiple consecutive spaces were collapsed into single spaces, either leading or trailing whitespace were removed.

3.2 Stop words removal

30 the most popular English words were marked as stop words to reduce dimensionality while preserving semantic content. The stop words' list includes:

- **articles:** the, a, an;
- **conjunctions:** and, or, but;
- **prepositions:** in, on, at, to, for, of, with, by, from;
- **common verbs:** is, was, are, were, been, be;
- **Pronouns:** this, that, it, i, you, he, she, we, they.

3.3 Filtering and deduplication

Two filtering operations were performed:

Length filtering. Reviews with fewer than 5 words (after stop word removal) were removed to ensure sufficient text for meaningful k-shingling with $k = 3$. This threshold guarantees that each retained review can generate at least three 3-word shingles.

Exact deduplication. Reviews with identical cleaned text, book title, and user ID were identified as database duplicates and removed. This step removed 215 reviews. However reviews with identical text but different metadata, like different book or different user, were intentionally retained as these represent legitimate targets for duplicate detection. A total of 1,301 such reviews were identified and preserved for analysis.

After preprocessing, the final working dataset contained 49 744 reviews.

4 Methodology

The near-duplicate detection pipeline consists of four sequential stages: shingling, MinHash signature generation, LSH index construction and candidate verification with Jaccard Similarity.

4.1 Shingling

A shingle size of $k = 3$ was selected to balance specificity and generalization. Larger values of k increase specificity but may miss semantically similar reviews, while smaller values increase false positives.

Example: Consider the review "great book highly recommend everyone" (5 words after preprocessing). The 3-shingle set is:

$$\begin{aligned} S = \{&\text{"great book highly"}, \\ &\text{"book highly recommend"}, \\ &\text{"highly recommend everyone"}\} \end{aligned}$$

This generates $|S| = 5 - 3 + 1 = 3$ shingles. Each review in the dataset was converted to its shingle set representation for subsequent processing. The longest review of the dataset has 3 597 shingles or 3 599 words. The shortest review has 2 shingles or 4 words.

4.2 MinHash Signatures

MinHash signatures were generated using $m = 128$ permutations, providing a good balance between approximation quality and computational efficiency. The `datasketch` Python library was employed, which implements MinHash using random hash functions rather than explicit permutations for efficiency.

This reduced each review from a variable-size shingle set (of average 94.5 shingles, where min = 2, max = 3 597) to a fixed-size 128-integer signature. This compression represents reduction in representation size for an average review while maintaining similarity estimation accuracy, enabling efficient storage and comparison operations.

4.3 Locality-Sensitive Hashing

An LSH index was constructed with similarity threshold $t = 0.7$ and $m = 128$ permutations. The `datasketch.MinHashLSH` class automatically configures band parameters to optimize the S-curve for the specified threshold.

All 49 744 MinHash signatures were inserted into the LSH index with unique string identifiers. For each review, the LSH index was queried to retrieve candidate reviews (those sharing at least one band). This process generated 1 530 candidate pairs for verification.

4.4 Jaccard Similarity

The Jaccard Similarity coefficient quantifies set similarity and is defined as:

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

where A and B are sets, $|A \cap B|$ denotes the cardinality of their intersection, and $|A \cup B|$ denotes the cardinality of their union. The Jaccard coefficient ranges from 0 (disjoint sets) to 1 (identical sets).

Example: Given two reviews with shingle sets:

$$A = \{\text{“great book highly”}, \text{“book highly recommend”}, \text{“highly recommend everyone”}\}$$
$$B = \{\text{“great book really”}, \text{“book really good”}, \text{“really good read”}\}$$

The intersection is $A \cap B = \emptyset$ (no common 3-word shingles despite sharing individual words), the union is $|A \cup B| = 6$, yielding $J(A, B) = 0/6 = 0$. However, if reviews share the shingle “great book highly”, then $|A \cap B| = 1$ and $J(A, B) = 1/6 \approx 0.167$.

4.5 Candidate verification with Jaccard Similarity

At this point, LSH had identified 1 530 candidate pairs that might be similar based on their MinHash signatures. However, MinHash provides only an approximation — not a guarantee. To ensure the final results were accurate, each candidate pair needed to be verified by calculating the true Jaccard Similarity directly from the original shingle sets.

This verification step goes back to the basics: for two reviews, how many shingles do they share (intersection) compared to how many unique shingles exist between them (union).

4.5.1 How verification works

The verification process was as follows. For each of the 49 744 reviews:

1. The LSH index was queried: ”Which other reviews share buckets with this one?”
2. For each candidate match found (where the candidate’s ID was higher to avoid counting pairs twice):
   - both reviews’ original shingle sets were retrieved;
   - common shingles were counted (intersection);
   - total unique shingles were counted (union);
   - Jaccard Similarity was calculated: common shingles divided by total unique shingles;
   - if this similarity was at least 0.7, the pair was kept as genuinely similar.

This verification revealed an important finding: of the 1 530 candidates suggested by LSH, exactly 764 pairs (50.0%) passed the strict verification test. The other 766 candidates were false positives—pairs that looked similar based on MinHash but weren’t actually similar enough when checked thoroughly. This roughly 50% verification rate is actually ideal, indicating that LSH was well-configured.

5 Experimental Results

5.1 Algorithm Performance Metrics

The near-duplicate detection pipeline achieved the following performance characteristics:

LSH efficiency:

- Total possible comparisons (brute force): 1 193 207 896
- LSH candidates generated: 1 530

Bucket statistics:

- Reviews with candidates: 1 330 (2.67% of dataset)
- Reviews alone in buckets: 48 414 (97.33%)
- Average candidates per review: 0.03
- Maximum candidates for any review: 8

The sparse bucket structure (average 0.03 candidates per review) indicates excellent LSH selectivity, minimizing false positive candidates while maintaining high recall.

Verification effectiveness:

- Candidates verified (Jaccard ≥ 0.7): 764
- Verification rate: 49.9%
- False positives filtered: 766 (50.1%)

The approximately 50% verification rate indicates optimal LSH configuration. Higher verification rates would suggest overly conservative candidate generation, while lower rates would indicate excessive false positives.

5.2 Similarity distribution

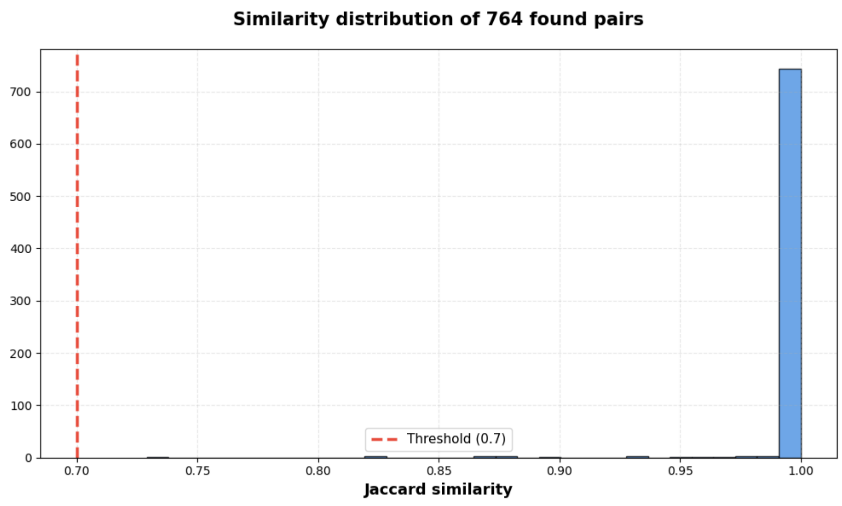

Figure 1: Similarity Distribution histogram showing 764 pairs

Figure 1 shows the distribution of Jaccard similarity scores for all 764 detected pairs. The distribution exhibits extreme right-skew, with approximately 740 pairs (96.9%) clustered at similarity 1.0 (perfect duplicates). This indicates that users predominantly copy reviews exactly rather than paraphrasing.

Category breakdown:

- Near-Duplicates (≥ 0.9): 755 pairs
- High Similarity (0.8–0.9): 8 pairs
- Substantial Similarity (0.7–0.8): 1 pair

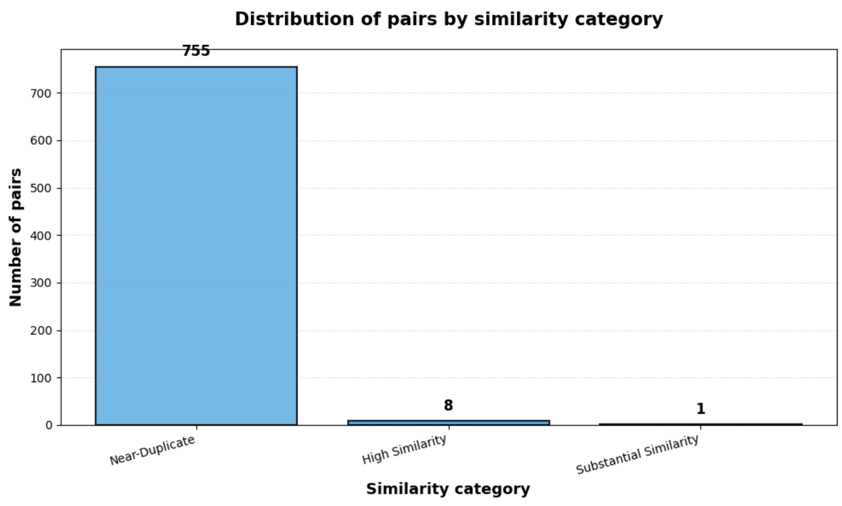

Figure 2: Category Distribution bar chart

The overwhelming dominance of near-duplicates, shown in Figure 2, validates the conservative threshold choice and suggests that moderate similarity (0.7–0.8 range) is rare in this corpus.

5.3 User behavior analysis

Analysis of the relationship between users and books, shown in Figure 3, in similar pairs revealed distinct patterns:

- Same User, Different Book: 643 pairs
- Different User, Different Book: 109 pairs
- Different User, Same Book: 6 pairs
- Same User, Same Book: 6 pairs

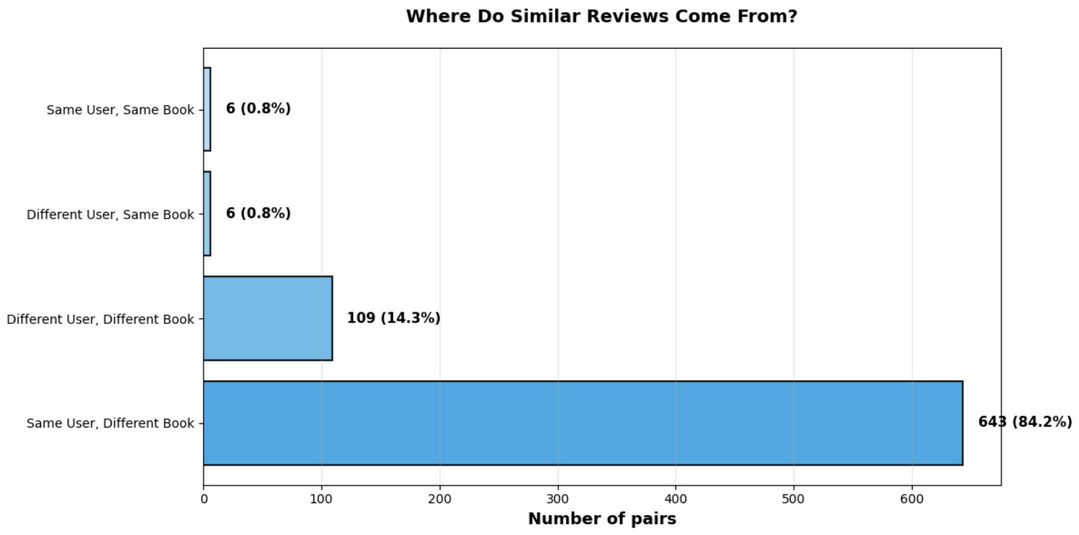

Figure 3: User/Book Relationship horizontal bar chart

The dominant pattern involves users reusing their own review text across different books. This represents template-based reviewing behavior rather than plagiarism. Users write one comprehensive review and apply it to multiple books (often in a series, by the same author, or of similar genre).

The low incidence of cross-user plagiarism is encouraging, suggesting effective platform moderation or organic reviewer behavior. Generic template reviews (different users, different books) likely arise from very short, simple reviews ("great book, highly recommend") that naturally converge to similar phrasing.

5.4 Book similarity network

Figure 4 presents a similarity matrix for the top 12 books appearing most frequently in similar pairs.

The dominant clustering pattern involves different editions of classic literature. For example, Wuthering Heights, Pride and Prejudice, The Hobbit, where all pairs show perfect similarity (1.0000).

Users copy their reviews verbatim across different editions (e.g., Penguin Classics vs. Dover Thrift) of the same work. This behavior is rational: if a user enjoyed the content, their opinion applies equally to all editions.

Books with empty rows (e.g., Little Women, Of Mice and Men) appear in the top 12 by frequency but have no connections to other top-12 books. Their similar pairs involve less popular books that didn't make the top-12 cutoff.

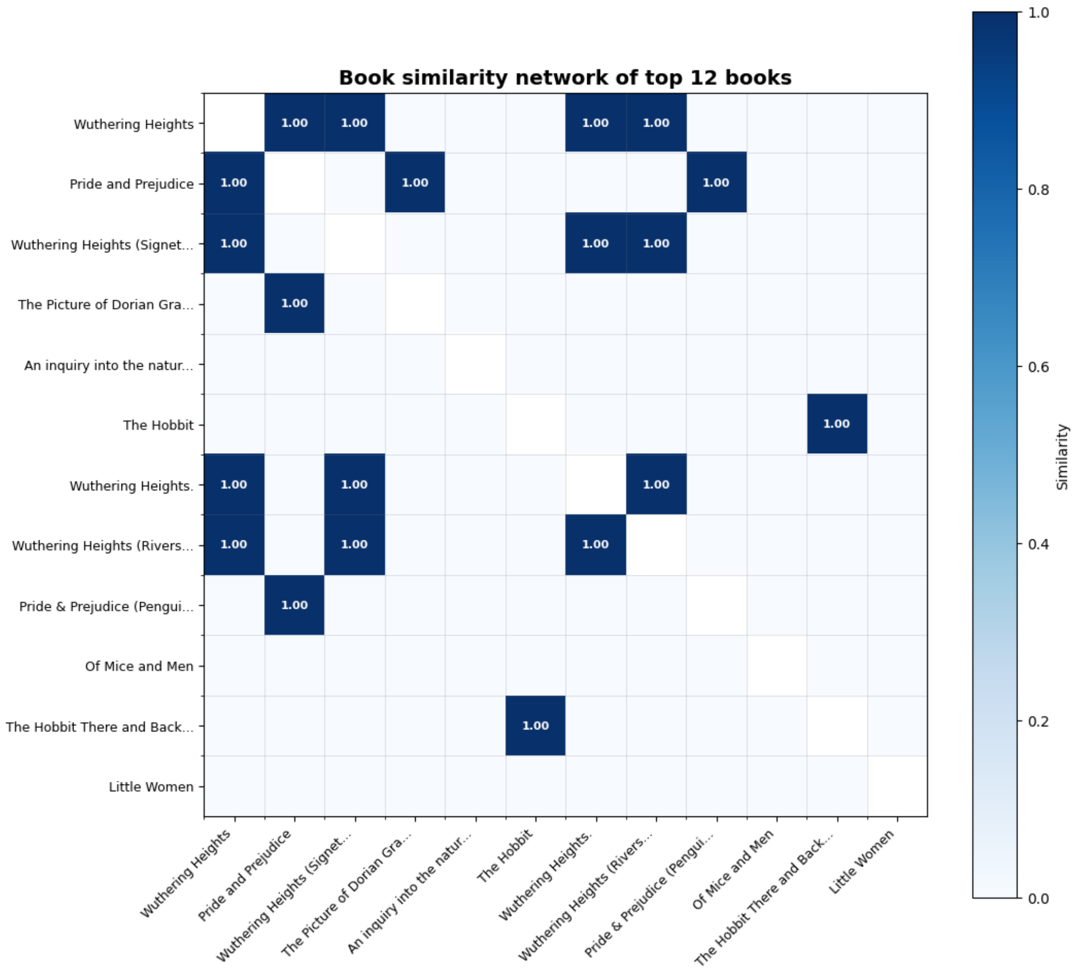

Figure 4: Book Similarity Matrix heatmap

5.5 Quality assessment

Manual inspection of the top 20 pairs revealed 100% high-quality matches with similarity ≥ 0.8. The highest-ranked pair exhibited perfect similarity (Jaccard = 1.0000), representing identical review text for "The Hobbit" posted by the same user for different editions.

> **Review 1 & 2 (Jaccard = 1.0000):** "I first read The Hobbit when I was in 7th grade, it was required reading. The difference between it and other required readings for..."
>
> **Relationship:** Same User, Different Book
>
> **Books:** The Hobbit (original) and The Hobbit There and Back Again

This example validates algorithm correctness: truly identical reviews were successfully detected.

6 Conclusion

This project successfully demonstrated that MinHash and Locality-Sensitive Hashing can efficiently detect near-duplicate reviews in large datasets. Working with 49 744 Amazon book reviews, the algorithm identified 764 similar pairs while examining only 1 530 candidates.

The results revealed interesting patterns in how people write reviews. The overwhelming majority (84.2%) of similar pairs involved users copying their own reviews across different books, typically for different editions of the same work or books in a series. This template-based reviewing behavior, while perhaps not ideal for readers seeking fresh perspectives, represents legitimate content reuse rather than plagiarism. Cross-user copying was remarkably rare (only 0.8% of pairs), suggesting that Amazon's review system is working well or that most reviewers naturally write original content.

The quality of detected duplicates was excellent = 98.8% had similarity scores above 0.9, meaning nearly perfect matches. The algorithm's verification rate indicated optimal LSH configuration: it was neither too permissive (wasting time on obvious non-matches) nor too conservative (potentially missing real duplicates). Manual inspection of the top-ranked pairs confirmed that all were genuine duplicates, validating the technical approach.

Overall, the implementation demonstrated that MinHash, LSH and Jaccard Similarity are powerful tools for large-scale similarity detection. The algorithm worked efficiently, produced accurate results, and gave meaningful insights into user behavior on the Amazon Books Reviews dataset.