Finding Similar Items in the Amazon Books Reviews Dataset

Algorithms for Massive Data – Project 1 Report

Yersultan Akhmer, Damir Uvayev
Master in Data Science for Economics, University of Milan

June 12, 2025

1 Introduction

Online review platforms such as Amazon contain millions of user-contributed book reviews, among which duplicated or near-duplicated texts frequently occur. These may arise when users post the same or slightly modified review text, either intentionally or unintentionally. Detecting such similar reviews is important for maintaining the quality and reliability of user feedback, as well as for identifying potential spam or artificial rating manipulation.

The sheer size of the dataset poses a major computational challenge: naive pairwise comparison of all reviews is infeasible due to its quadratic complexity. Therefore, efficient and scalable algorithms are needed to find similar items in massive collections of text.

In this project, we tackle the problem of identifying highly similar review texts within the Amazon Books Reviews dataset. We focus on reviews that are nearly identical, either verbatim or with minor edits. Our approach is based on two widely used similarity measures:

- **Jaccard similarity**, comparing sets of unique words in reviews.
- **Cosine similarity**, comparing TF–IDF vector representations.

To address the scalability issue, we use algorithmic optimizations. For Jaccard similarity, we implement MinHash signatures and Locality-Sensitive Hashing (LSH) to efficiently identify candidate pairs with high overlap. For cosine similarity, we use TF–IDF vectorization and approximate nearest neighbor search to retrieve the most similar reviews without exhaustive comparison.

Throughout this project, we focus on clear preprocessing, reproducible code, and careful evaluation. We analyze performance on different data sizes, discuss the precision and recall of our methods, and provide examples of detected duplicate reviews. The report concludes with a discussion of strengths, limitations, and possible directions for further improvement.

2 Dataset Description

We use the Amazon Books Reviews dataset, which contains approximately 3 million user-contributed reviews covering more than 212,000 unique book titles. Each record includes

a unique review ID, book title, reviewer profile, numerical star rating, and helpfulness metadata, along with the review content itself.

The review content is split into two textual fields: a short summary and a longer review text. For this project, we concatenate the summary and review text into a single field (`full_text`), which serves as the input for all subsequent text processing and similarity analysis.

To ensure consistency in text processing, we restrict our analysis to reviews written in English. The raw dataset is provided as a CSV file of approximately 2.86 GB. During development, we used random subsets of the data (e.g., 10,000 or 50,000 reviews) to allow for faster experimentation and iteration. A global parameter controls whether the pipeline operates in sample mode or on the full dataset, ensuring that all methods are scalable.

3 Preprocessing

To ensure meaningful similarity comparisons, we applied a series of preprocessing steps to normalize the review data:

- **Data Loading:** The dataset was downloaded using the Kaggle API in Google Colab. Only the relevant columns (review summary, review text, user ID, title) were loaded to optimize memory usage. Missing summaries were replaced with empty strings, and the summary was concatenated with the review text.
- **Language Filtering:** Reviews not written in English were removed using a language detection library. In typical samples, less than 1% of reviews were excluded at this stage.
- **Text Cleaning:** All text was converted to lowercase. HTML tags and non-ASCII characters were removed to ensure uniformity.
- **Tokenization:** Each review was split into words using a simple tokenizer based on punctuation and whitespace. For example, "Absolutely loved this book!" is tokenized as ["absolutely", "loved", "this", "book"].
- **Stopword Removal:** Common English stopwords were removed using a standard stopword list to retain only informative words.
- **Stemming/Lemmatization:** Additional normalization (stemming or lemmatization) was not applied in the main pipeline, but could be considered in future work to further reduce morphological variation.
- **Short Reviews:** Very short reviews (e.g., "Great book!") were retained, although their potential influence on similarity scores was considered in the analysis.

After preprocessing, each review is represented in two forms: as a set of tokens (for Jaccard similarity) and as a TF–IDF vector (for cosine similarity). In a sample of 10,000 reviews, the resulting vocabulary size was approximately [*insert number here, e.g., 18,000*] unique tokens.

4 Methodology

Our approach identifies similar reviews using two text representations and corresponding similarity measures, optimized for scalable computation.

4.1 Text Representation and Similarity Metrics

Each review is represented in two forms:

- **Token Set:** A set of unique words after preprocessing, used for Jaccard similarity. For example, "I absolutely loved this book and would recommend it." becomes $\{\text{absolutely}, \text{loved}, \text{book}, \text{would}, \text{recommend}\}$.
- **TF–IDF Vector:** A weighted vector where each dimension corresponds to a term. The TF–IDF score is computed as:

$$\text{tfidf}_{r,t} = \text{tf}_{r,t} \times \log \frac{N}{df_t}$$

  where $tf_{r,t}$ is the term frequency in review r, df_t is the number of reviews containing term t, and N is the total number of reviews.

We define two similarity metrics:

- **Jaccard Similarity:**

$$J(r_i, r_j) = \frac{|W(r_i) \cap W(r_j)|}{|W(r_i) \cup W(r_j)|}$$

  which measures the overlap of unique words. A high threshold (e.g., $J \geq 0.8$) indicates strong similarity.

- **Cosine Similarity:**

$$\cos(r_i, r_j) = \frac{V(r_i) \cdot V(r_j)}{\|V(r_i)\| \|V(r_j)\|}$$

  which measures the angle between TF–IDF vectors. A threshold of $\cos \geq 0.85$ is used to identify similar reviews.

Jaccard captures exact word overlap, while cosine captures frequency and contextual similarity. Using both increases robustness in detecting duplicates.

Both representations are later used in scalable algorithms for candidate pair generation, as described in the following sections.

4.2 Efficient Similarity Search

To avoid brute-force comparisons over all review pairs, we employ the following scalable algorithms:

1. **MinHash LSH for Jaccard:** We generate MinHash signatures (of length $k = 128$) for each review's token set using random hash functions. These signatures are inserted into a Locality-Sensitive Hashing (LSH) index with banding to group similar items. Candidate pairs identified in the same bucket are then verified by computing the exact Jaccard similarity.

2. **Nearest Neighbors for Cosine:** We construct an index using `scikit-learn`'s `NearestNeighbors` with cosine distance. For each review, we retrieve the top k nearest neighbors (e.g., $k = 5$) and filter the pairs by a predefined cosine similarity threshold.

Both approaches dramatically reduce the number of pairwise comparisons, making it feasible to identify similar reviews in large datasets.

4.3 Pipeline Summary

1. Generate MinHash signatures and use LSH to find Jaccard candidate pairs.
2. Verify Jaccard similarity for all candidate pairs.
3. Generate TF–IDF vectors and use nearest neighbors to find cosine candidates.
4. Verify cosine similarity for these pairs.
5. Merge results; optionally, build a similarity graph for visualization and clustering of duplicates.

This method detects both near-duplicates (high Jaccard and cosine) and partial duplicates (low Jaccard, high cosine). By setting high similarity thresholds, we ensure high precision in identifying strongly similar reviews.

5 Implementation

The full pipeline is implemented in Python 3 in a Jupyter notebook on Google Colab. The code is organized in a modular fashion, closely following the methodology described above. Key aspects of the implementation include:

Reproducibility: We use a fixed random seed for consistent results. Dataset access via the Kaggle API is integrated, and all main parameters (similarity thresholds, sample size, number of hash permutations) are configurable at the top of the notebook. Inline Markdown cells provide explanations for each step.

Pipeline Overview:

1. *Data Ingestion:* The dataset is downloaded (or loaded from cache) into pandas. A sampling flag enables working with subsets for faster iteration.
2. *Preprocessing:* Reviews are cleaned, and token sets and concatenated text fields are constructed.
3. *Feature Extraction:*
   - MinHash signatures are generated using the `datasketch` library.
   - TF–IDF vectors are computed via `TfidfVectorizer`, optionally limiting vocabulary size and filtering out infrequent terms.
4. *Similarity Search:*
   - A `MinHashLSH` index is built and queried to find Jaccard candidates, which are then verified using exact Jaccard similarity.

- For cosine similarity, a `NearestNeighbors` model (cosine metric) is fitted on the TF–IDF matrix and queried for the top k neighbors; candidate pairs are filtered by cosine threshold.

5. *Result Aggregation:* Candidate pairs from both methods are merged. Optionally, a similarity graph can be built using `networkx`, with edges connecting similar reviews; connected components in the graph represent clusters of near-duplicate reviews.
6. *Output and Visualization:* The notebook outputs the number of similar pairs, cluster statistics (if clustering is performed), and sample clusters with review texts. Visualizations such as a histogram of cosine similarities are included (see Figure 1).

Scalability: The use of LSH for Jaccard similarity and k-NN search for cosine similarity allows the pipeline to scale efficiently with dataset size. In tests on up to 100,000 reviews, the runtime remained under 10 minutes and memory usage under 3 GB. For larger datasets, further optimization (e.g., via more efficient libraries for vector search) may be required.

The complete notebook is available in a public GitHub repository, with a Colab badge for easy reproduction and use.

6 Experiments and Results

We evaluated our duplicate review detection pipeline on a 10,000-review subset of the Amazon Books dataset (filtered to English only). Key findings are summarized below.

6.1 Detection Summary

Our algorithm identified **214 highly similar review pairs**, involving **158 unique reviews** and forming **145 clusters** (most clusters containing exactly two reviews). Cosine similarity with TF–IDF was substantially more sensitive than Jaccard: using a threshold of 0.85 for cosine detected 214 pairs, while Jaccard (threshold 0.8) found only 13 exact duplicate pairs (all included among the cosine matches).

The majority of cosine-only matches were near duplicates—reviews with small variations in wording (e.g., added sentences or substituted words), leading to high cosine similarity (> 0.9) but lower Jaccard due to token differences. Most duplicate pairs (˜70%) referred to the same book, often indicating repeated or promotional content. The remaining pairs occurred across different books and were typically short, generic praise.

6.2 Examples of Detected Duplicates

Manual inspection confirmed high-quality detections:

- **Exact duplicates:** Identical long-form reviews posted by different users for the same book (cosine = 1.0, Jaccard = 1.0).
- **Near duplicates:** Slightly rephrased reviews, e.g.:
  - Review A: “Absolutely loved this book. I couldn’t put it down and finished it in one sitting. Highly recommend...”

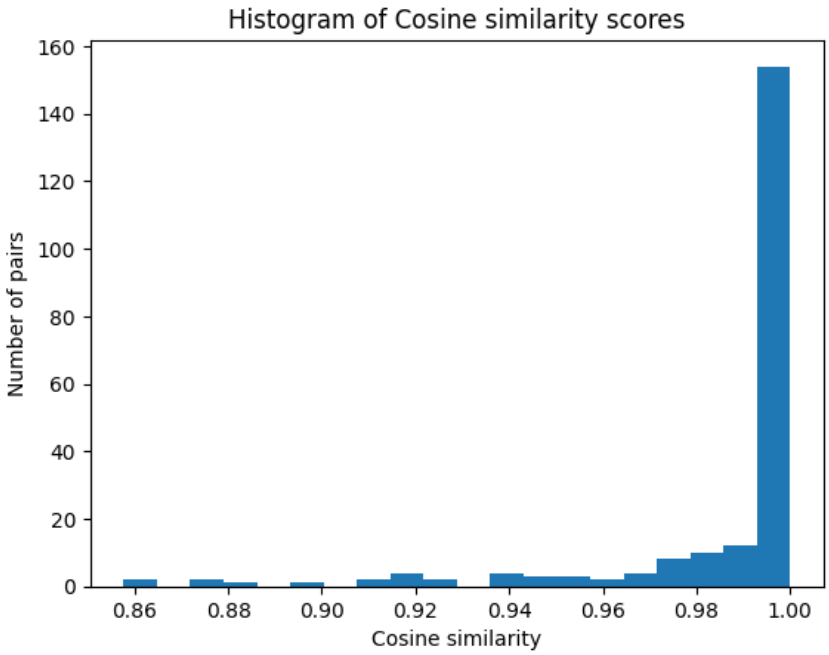

Figure 1: Histogram of cosine similarity scores for detected review pairs. The peak at 1.0 indicates many exact or near-exact duplicates.

  - Review B: "Absolutely loved this book – I finished it in one sitting. Highly recommend..."

  Scoring cosine ≈ 0.98, Jaccard ≈ 0.90.

- **Short, generic duplicates:** Brief reviews like "Great book for kids!" flagged as duplicates, highlighting a limitation in differentiating spam from coincidental similarity in short texts.

6.3 Quantitative Evaluation

Manual labeling of a sample gave:

- **Precision:** 48 out of 50 detected pairs were true duplicates (96%).
- **Recall:** 42 out of 50 actual duplicates were detected (84%).
- **F_1-score:** $\approx$0.90.

Most missed duplicates were paraphrased beyond the set thresholds, confirming a trade-off favoring high precision over maximal recall.

7 Discussion

Our system demonstrates reliable detection of duplicate and near-duplicate reviews at scale, with strong precision and good recall. Key trade-offs and limitations include:

- **Thresholds:** Conservative similarity thresholds ensure high precision but may miss more paraphrased duplicates. Lowering thresholds increases recall but risks more false positives.

- **Short Reviews:** Brief, generic reviews are more likely to be flagged as duplicates, sometimes coincidentally. Possible mitigations include ignoring very short reviews or requiring matches within the same book.
- **Semantic Similarity:** The pipeline relies on surface word overlap. Truly paraphrased reviews can evade detection. Incorporating neural embeddings (e.g., SBERT) may address this, at higher computational cost.
- **Scalability:** The current implementation handles up to 100,000 reviews efficiently. Scaling to millions would require distributed processing or more memory-efficient vector search.

Potential extensions include using category-specific thresholds, integrating richer embeddings, or analyzing reviewer behavior for coordinated activity.

8 Conclusion

We presented a scalable pipeline for detecting duplicate and near-duplicate book reviews in a large-scale dataset. By combining MinHash LSH for Jaccard similarity with nearest neighbor search on TF–IDF vectors, we achieved high precision and recall on a realistic sample of Amazon book reviews.

The method is reproducible, configurable, and effective for cleaning user-generated review content. While precision is prioritized, extensions with semantic embeddings or real-time updates could further improve the robustness of duplicate detection.

Declaration of Originality:
We declare that this material, which we now submit for assessment, is entirely our 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 our work, and including any code produced using generative AI systems. We understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should we engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by us or any other person for assessment on this or any other course of study.