Similarity Detection in Amazon Book Reviews

Melany Yohana Gomez Herrada
Student ID: 39351A
Data Science for Economics (DSE)
Course: Algorithms for Massive Data

11 settembre 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.

Introduction

The goal of this project is to implement a system that identifies pairs of similar book reviews from a dataset. The dataset, obtained from Kaggle, contains book reviews in CSV format. The task consists in encoding the textual content of the reviews and comparing them using a similarity metric.

While various methods exist, this project adopts an approach based on the **Jaccard similarity** as the measure of review similarity, approximated via the MinHashLSH algorithm in PySpark. The method focuses on the `review/text` column of the dataset, comparing reviews based on overlapping tokens and shingled phrases. Reviews with a high degree of overlap are flagged as similar. Moreover, the reviews were divided by sentiment (positive and negative) to ensure semantic consistency in the comparisons. To scale the computation to thousands of reviews, the system uses the `MinHashLSH` algorithm provided by PySpark's machine learning library.

This report outlines the implementation details, preprocessing steps, evaluation strategy, and results obtained using this approach.

Implementation

2. Data Loading and Sampling

The dataset file `Books_rating.csv` is loaded into a Spark DataFrame with automatic schema inference. To reduce computational overhead, a **5% random sample** is extracted using a fixed seed to ensure reproducibility. This sampling yields approximately **150,000 reviews**, offering a representative subset of the full dataset while maintaining scalability for distributed processing.

The following three columns are retained:

- `Id` — unique identifier of the review.
- `review/score` — numerical rating, later used to infer sentiment.
- `review/text` — the content of the review to be analyzed.

3. Text Cleaning and Tokenization

Each review undergoes several preprocessing steps to normalize the textual data:

1. HTML entities are decoded, and text is converted to lowercase.
2. Special characters, numbers, and punctuation are removed.
3. Tokenization is performed using Spark's `RegexTokenizer`, which splits the text into words.
4. Common English stopwords are removed using `StopWordsRemover`.

To eliminate overly short or long reviews, only reviews containing between **5 and 200 tokens** are retained. Additionally, a new column for **sentiment** is introduced: reviews with a score strictly greater than 3 are labeled as `positive`, while the rest are labeled as `negative`. This separation enables more semantically consistent comparisons, especially when detecting similarity within reviews of the same polarity.

4. Shingling and Feature Extraction

To better capture local word order and context, each review is converted into a set of **shingles** specifically, sequences of three consecutive tokens (3-grams). For example, the sentence "the book was great" would produce the shingles "the book was", "book was great", etc.

The set of shingles for each review is then transformed into a binary feature vector using `HashingTF`, which maps the string-based 3-grams to a fixed-length sparse vector space. This representation is well-suited for approximate similarity computation.

5. Similarity Detection with MinHashLSH

To identify similar reviews efficiently, the system leverages `MinHashLSH`, a hashing (LSH) technique that provides a scalable approximation of the **Jaccard similarity** between sets of shingles.

Given two sets of shingles A and B, the Jaccard similarity is defined as:

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

which measures the proportion of overlap between the two sets. Computing $J(A, B)$ exactly for all pairs in a large dataset is computationally expensive, so MinHash is used to approximate this value efficiently.

In MinHash, each set is mapped to a compact signature vector:

$$\text{MinHash}_k(S) = [\min h_1(S), \min h_2(S), \ldots, \min h_k(S)],$$

where $h_1, \ldots, h_k$ are independent hash functions. The probability that two sets A and B have the same MinHash value under a given hash function is equal to their true Jaccard similarity:

$$\mathbb{P}[\min h(A) = \min h(B)] = J(A, B).$$

Using these signatures, Spark's `MinHashLSH` estimates the similarity between reviews in sublinear time. A threshold of $\theta = 0.5$ is applied to retrieve pairs of reviews with significant overlap in content.

To avoid symmetric duplicates, only pairs where `reviewA.Id` > `reviewB.Id` are retained.

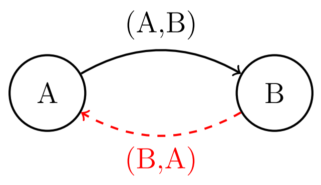

Figura 1: Example of symmetric duplicates. The system keeps only one (e.g., (A, B)) by requiring `reviewA.Id > reviewB.Id`.

Although the condition `reviewA.Id > reviewB.Id` removes symmetric duplicates, in practice the same review text may appear under different book identifiers (for example, multiple editions of the same title). This can cause what looks like duplicate output pairs in the results.

To avoid this, we applied a final deduplication step using `dropDuplicates(["docA", "docB"])`, ensuring that each unique pair of reviews appears only once in the output.

Furthermore, a secondary filter is applied to isolate **moderately similar pairs**:

$$0.5 \leq \hat{J}(A, B) \leq 0.95,$$

which are most useful for qualitative inspection and downstream analysis such as detecting duplicates, spam patterns, or repetitive user behavior.

6. Results Summary

To maintain semantic consistency, similarity detection is performed separately for `positive` and `negative` reviews. The table below summarizes the number of similar pairs found in each group:

Review Sentiment	**All Similar Pairs**	**Similarity 0.5–0.95**
Positive	3761	103
Negative	1161	14

7. Examples of Output

For both positive and negative reviews, the system displays examples of highly similar and moderately similar pairs, ranked by Jaccard similarity score.

To illustrate the meaning of moderately similar reviews (similarity between 0.8 and 0.95), we report below two representative cases extracted from the output:

- **Pair A (similarity** $\approx$ 0.94**)**: Both reviews discuss Emily Brontë. The texts are almost identical, but one review contains the phrase "or Frankenstein" while the other does not. This small lexical difference prevents them from being exact duplicates but still results in very high overlap.
- **Pair B (similarity** $\approx$ 0.83**)**: Two reviews of *Little Women* differ only by a minimal wording change: one describes the story as a "diary of their lives", while the other uses "recounting of their lives". Such minor variations are captured by the similarity detector as moderately similar pairs.

These examples demonstrate how the system detects reviews that are not exact duplicates but differ only by subtle wording or phrasing changes, confirming the usefulness of approximate similarity detection in practice.

These results are useful for:

- Detecting duplicate or near-duplicate reviews submitted by different users.
- Identifying potential spam or templated reviews that follow similar linguistic patterns.
- Uncovering repetitive behavior by users across multiple reviews.

This output provides valuable insight into the structure and redundancy of user-generated content and can serve as a foundation for tasks such as deduplication, spam detection, or content summarization.

Conclusion

The system built in this project was able to detect pairs of similar reviews in the Amazon Books dataset using Jaccard similarity with MinHashLSH. The method worked well in finding both exact duplicates and near-duplicates, where reviews were slightly rephrased but still very close. Using 3-gram shingles proved to be a good compromise, since smaller values gave too much noise and larger values missed useful overlaps. The approach is limited because it only works on surface text overlap, so it cannot detect deeper semantic similarity or paraphrases.

Even with these limits, the method proved to be scalable and efficient. In the future, it could be improved by adding language filtering, using metadata like reviewer IDs and book titles, or testing more advanced models that capture meaning beyond word overlap.

References

[1] Jure Leskovec, Anand Rajaraman, and Jeffrey D. Ullman. *Mining of Massive Datasets.* Cambridge University Press, 3rd edition, 2020.

[2] Sam Chahine. *NLP: How to Find the Most Distinctive Words Comparing Two Different Types of Texts.*

[3] Apache Spark Documentation. *MinHashLSH — PySpark MLlib.*