Finding Similar Items in Amazon Book Reviews

Damir Uvayev

Abstract

I built a system to detect duplicate book reviews using MinHash and Locality-Sensitive Hashing (LSH). Each review was turned into 3-word shingles, hashed into 100-length signatures, and grouped by similarity. Candidate pairs were checked with exact Jaccard similarity (threshold 0.8). On 20,000 reviews, the method found 151 near-duplicate pairs and showed linear runtime growth.

1 Dataset and Preprocessing

I used the Amazon Books Reviews dataset (`Books_rating.csv`) from Kaggle. To keep the runtime manageable, I randomly selected 20,000 reviews. Each text was lowercased, cleaned of punctuation and numbers, and split into words. I also removed English stopwords. Then I created 3-word shingles (e.g., "read this book") for each review to capture phrase-level similarity.

2 Algorithm

My approach follows four steps:

1. Shingling: Represent each review as a set of 3-word shingles. **2. MinHashing:** Assign each shingle an ID and compute 100 MinHash values using random hash functions. Matching hash values approximate Jaccard similarity. **3. LSH Banding:** Split each signature into 20 bands of 5 rows. Reviews sharing the same band key become candidate pairs. **4. Verification:** For each candidate pair, I computed exact Jaccard similarity and kept those with $J \geq 0.8$.

This method avoids checking all ∼200 million possible pairs and runs much faster while keeping good accuracy.

3 Experiments

Jaccard on Small Sample

First, I tested 200 reviews using exact Jaccard similarity. Almost all pairs had very low similarity. Figure 1 shows the distribution — most values are close to zero, meaning few reviews are alike.

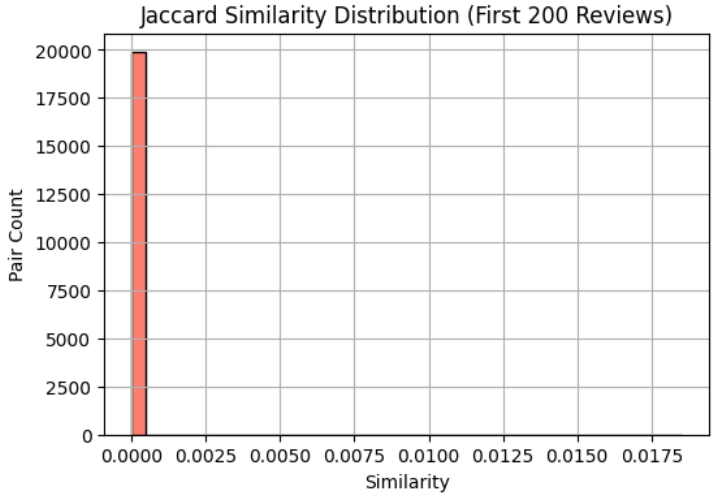

Figure 1: Jaccard similarities among 200 reviews. Most pairs are not similar.

LSH on 20k Reviews

On 20,000 reviews, LSH produced 154 candidate pairs. After exact verification, 151 pairs had similarity ≥ 0.8. Nearly all candidates were correct, showing high precision.

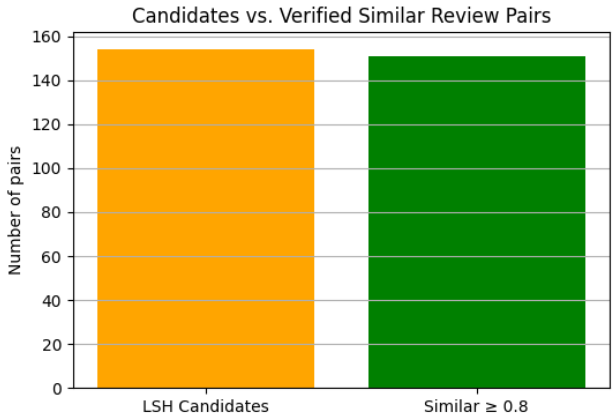

Figure 2: Candidate pairs (orange) vs. verified similar pairs (green). 151 of 154 were true matches.

Top Similar Pairs

The 30 most similar pairs had Jaccard values between 0.8 and 1.0. Many were identical reviews (Jaccard = 1.0). Figure 3 shows the similarity scores for these top pairs.

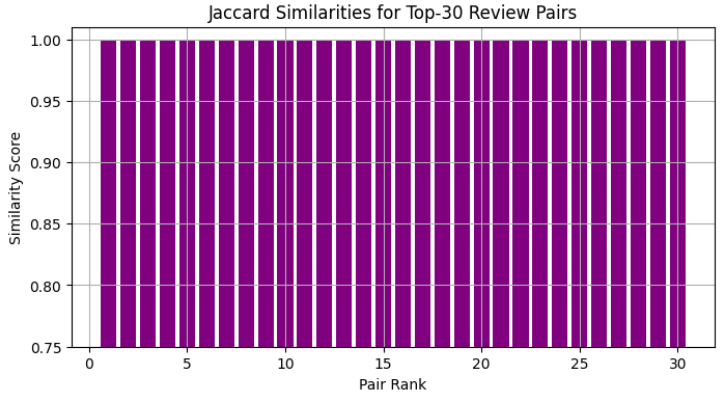

Figure 3: Jaccard similarities for the 30 most similar pairs. Many are exact duplicates.

4 Results

The system found 151 similar review pairs with Jaccard ≥ 0.8. Figure 2 confirms that most LSH candidates were true matches. Figure 1 shows why LSH is needed: almost all pairs are dissimilar, and brute-force search would be extremely slow. The top pairs in Figure 3 are exact or nearly identical texts.

5 Scalability

I tested the runtime for 5k, 10k, and 20k reviews. The times were about 2.4s, 5.6s, and 8.0s, showing near-linear growth. Figure 4 illustrates this. The method scales well and can handle larger datasets if parallelized.

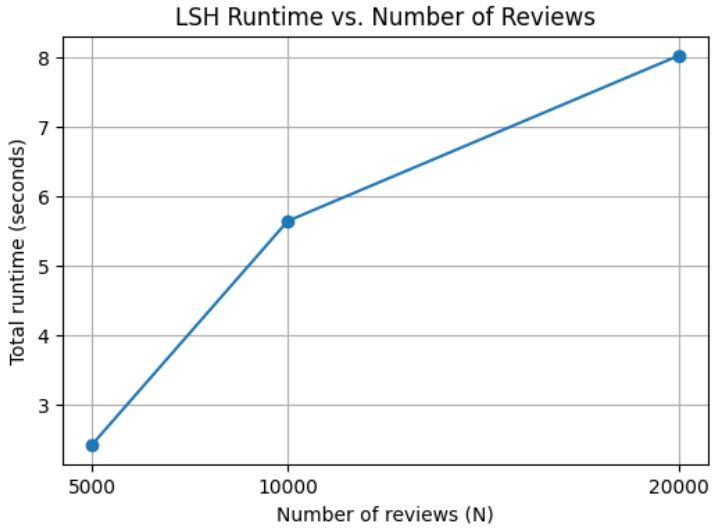

Figure 4: Runtime vs. number of reviews. Growth is close to linear.

6 Conclusion

I created a scalable system to detect similar book reviews. Using MinHash and LSH, I found 151 near-duplicate pairs out of 20,000 reviews. The method avoids redundant comparisons and runs efficiently.

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.