Massive Data Analysis - Similarity in Book Reviews - Using MinHashLSH and Jaccard Algorithm

September 10, 2025

Abstract

The project explores and investigates the similarity between book reviews using Jaccard Similarity. I have implemented MinHash Locality Sensitive Hashing (LSH) on a different sampled subset of Amazon book reviews and made the code scalable to run on large dataset. The objective is to identify similar reviews efficiently at scale, so I used Spark for distributed processing. Further, I focussed on similar reviews by filtering out various stop words and filler text.

1 Introduction

Finding similar reviews in a large corpus is a fundamental problem in NLP and recommender systems. Traditional Jaccard similarity becomes computationally expensive for large datasets, so I explored approximate methods like LSH with minhashing to scale similarity detection. Spark is used for its ability to handle large datasets in parallel.

2 Dataset

The dataset used is **Amazon Book Reviews**, containing 3 million reviews. I used the `Books_rating.csv` file which includes fields such as:

- `'Id'`
- `'Title'`
- `'Price'`
- `'User`$_i d'$`'profileName'`
- `'review/helpfulness'`
- `'review/score'`
- `'review/time'`
- `'review/summary'`
- `'review/text'`

Due to computation constraints, I processed random samples of the data but used a seed to keep the outputs reproducible. Further, I used only IDs and review text to find similar pairs. Additionally, I created a unique `review_id` for each review using the fields `'Id'`, `'User_id'`, and `'review/time'`.

3 Preprocessing

- I removed null and very short reviews less than length of 50 characters (it can be configured using `REVIEW_LENGTH` threshold).
- I tokenized and cleaned the review text.
- On the tokenized dataset, I applied lemmatization using NLTK.
- From the structured lemmatized tokens I removed stopwords.
- From the lemmatized tokens, I removed stopwords. These stopwords were imported from NLTK and combined with the following custom stopwords:
  - {`"book", "books", "read", "reads", "reading", "word", 'words', "paragraph", "paragraphs", "magazine", "magazines", "novel", "novels", "page", "pages", "chapter", "chapters"`}
- Then, I filtered the tokens by length (`TOKEN_SIZE` = 15), and it removes the review for which I have less than 15 tokens.

4 Methodology

I used the following steps on the lemmatized tokens to compute similarity between the reviews:

1. Token Vectorization

I used the MD5 hash function and further processing to encode the tokens into integers.

2. Hash Functions Generation And Computing Minhash Signatures

Then I generated 100 random functions (can be configured using $\texttt{NUM}_H ASHESasperrequirement).Thefunctionsareth$

`Then I used custom lsh banding technique on the min hash signatures to identify candidate pairs. Further, i restricted the candidate pairs bucket list to be with less than 100 candidates in each bucket in output, just to keep the code simple and not exhaust the memory with very large pairwise comparisons. Finally the pairs were presented with a similarity threshold` $> 0.2$`), i.e.` 20

5 Results

The similarity computation was done for a different sampled fractions of the dataset. The results and insights from each are as follows :

Table 1: Performance metrics and similarity results across different dataset sample sizes.

Metric	**1%**	**2%**
Sampled Reviews	30,332	60,498
Filtered Sampled Reviews	29,555	58,322
Sampling Time (secs)	80.65	47.52
Filtering Time (secs)	16.58	12.19
Lemmatized reviews	24,113	47,597
Text Processing Time (secs)	356.89	381.64
Time for token hashing (secs)	0.19	0.14
Time for MinHash (secs)	0.16	0.12
Time for LSH banding (secs)	0.10	0.13
Candidate pairs generated	893	3544
Time for candidate pair generation (secs)	686.46	729.88
Time for Jaccard similarity (secs)	0.35	0.40
Final similar pairs found	10	60
Completed LSH pipeline (min)	27.43	35.04

From the above analysis, we see that the processing steps are run with consistent time limits for different steps, thus they are scaling well with the increased dataset. However, the steps for lemmatization (text-processing) and generating candidate pairs even is taking larger times, doesn't have a steep increase when we move from 1% to 2% of the sample dataset.

6 Analysis of Similar Pairs found in 2% sampled fraction of Dataset

- I found **60** similar pairs in the **2%** sampled dataset for **¿= 20%** similarity. Further, I observed that some pairs were almost identical even after I removed the duplicates. This is due to the fact of various typos in text, some additional filler words in reviews and slight variation of reviews posted under different books etc.
- I counted review pairs with exact (**¿95%**)and near exact matches of the reviews.

Table 2: Counts of exact and near-exact matches in the 2% sampled dataset.

Match Type	Count
Exact Match	4
Near Match	56

- From the review pairs, I calculated the unique book counts to be **106** based on the "bookid".
- Below is a plot for Jaccard Similarity Distribution for the Review Pairs :
- Then I calculated that there are **15** book review pairs with almost identical reviews but different book IDs.
  - These different book ids with the same reviews introduces suspicion and also points out to plagiarism in reviews. It also points out that

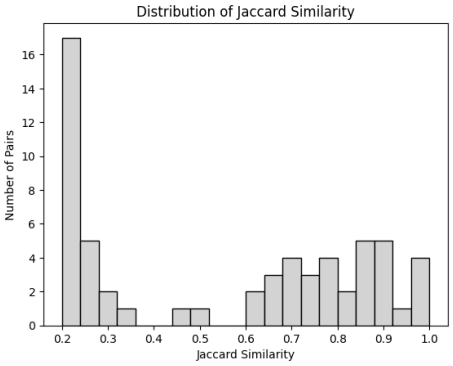

Figure 1: Enter Caption

Figure 2: Distribution of Jaccard Similarity

there can be different bookid for same book which are present in the dataset. And the reveiws just show redundant data (maybe sourced from different platforms), if not plagiarised. So I analyzed the pairs where bookids are different but reviews are similar (to filter out similar reviews for same books)

- The count of review pairs where reviews are similar for different books, is **56** (out of 60) (based on book ids only). Thus, almost all review pairs have different books (or can have same book if it is mentioned with different id for different reasons, e.g. version of book) .
- The below plot shows the distribution of similar books reviews frequency:
- Finally, I created a wordcloud for words in reviews column from the **60** similar books:

7 Limitations

- I computed similarity from similar candidate pairs using LSH and that too with buckets with candidate pairs less than 100 items, and didn't compare whole dataset which will cost O(N*N) complexity.

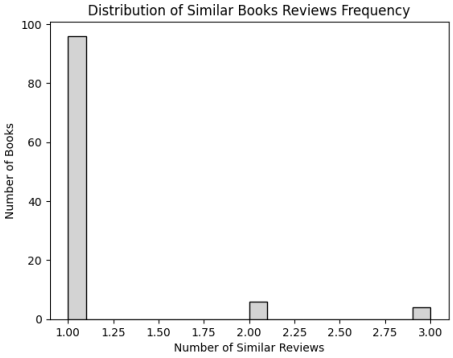

Figure 3: Enter Caption

- Since I am using Jaccard Similarity, there is no semantic meaning captured.
- Since, I also removed the stopwords and other common words along with limiting the review length and token length, I am not accounting in short reviews that may be just like 'I loved this book, always recommend", "The book is very nice to read" etc.

8 Conclusion

This project explains how **Locally Sensitive Min-Hashing** can be used to efficiently compute review similarities in a large dataset. Further, I also highlighted that how computational the algorithms become when running on massive datasets, which calls for optimized algorithms that pulls in a trade-off between precision and recall of the desired objective (similarity analysis in our case).

9 Declaration

"We declare that this material, which we now submit for assessment, is entirely our our 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."