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

Akash Mittal (31909A) — Antonella Convertini (45853A)

July 11, 2025

Abstract

The project explores and investigates the similarity between book reviews using Jaccard Similarity. We 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 we used Spark for distributed processing. Further, we 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 we explore approximate methods like MinHashLSH 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. We 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, we processed random samples of the data but used a seed to keep the outputs reproducible. Further, we used only IDs and review text to find similar pairs. Additionally, we created a unique `review_id` for each review using the fields `'Id'`, `'User_id'`, and `'review/time'`.

3 Preprocessing

- We removed null and very short reviews ¡25 length of characters (it can be configured using `REVIEW_LENGTH` threshold).
- We tokenized and cleaned the review text.
- On the tokenized dataset, we applied lemmatization using NLTK.
- From the structured lemmatized tokens we removed stopwords.
- From the lemmatized tokens, we removed stopwords. These stopwords were imported from NLTK and combined with the following custom stopwords:
  - {`"book", "read", "word", "paragraph", "magazine", "novel", "page", "chapter"`}
- Then, we filtered the tokens by length (`TOKEN_SIZE` = 5), and it removes the review for which we have less than 5 tokens.

4 Methodology

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

1. Token Vectorization

We used Spark's `Tokenizer` and `HashingTF` to convert cleaned and lemmatized tokens into feature vectors.

2. Approximate Jaccard Similarity

We used `MinHashLSH` from Spark MLlib library:

- Trained on hashed token vectors.
- Computed pairwise approximate Jaccard similarities.
- Filtered pairs using a threshold (Jaccard Distance < 0.6), thus Jaccard Similarity ¿= 0.4 (40

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

Metric	**0.5%**	**0.75%**	**1%**	**2%**
Sampled Reviews	15,189	22,779	30,332	60,498
Filtered Sampled Reviews	15,013	22,442	29,808	58,797
Sampling Time (min)	0.02	0.80	0.80	0.81
Filtering Time (min)	0.15	0.11	0.09	0.14
Text Processing Time (min)	0.01	0.01	0.01	0.01
Hashing Time (min)	5.08	5.15	5.32	5.93
LSH Time (min)	0.06	0.10	0.09	0.07
Jaccard Time (min)	3.74	7.69	14.77	50.45
Similar Pairs Found	9	17	30	135

Match Type	Count
Exact Match	4
Near Match	131

5 Results

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

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 "Jaccard Similarity" step shows a steep and non-linear increase in runtime as the sample fraction grows, which is due to the fact the pairwise similarity becomes expensive with increased number of candidate pairs.

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

- We found **135** similar pairs in the **2%** sampled dataset for **¿= 40%** similarity. Further, we observed that many pairs were almost identical even after we 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.
- We counted review pairs with exact (**¿95%**)and near exact matches of the reviews.
- From the review pairs, we calculated the unique book counts to be **199** based on the "bookid".
- Below is a plot for Jaccard Distances Distribution for the Review Pairs :
- Then we calculated that there are **17** 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 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 we analyzed the pairs where bookids are different but reviews are similar (to filter out similar reviews for same books)

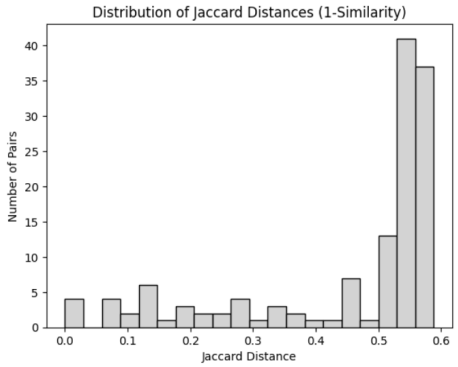

Figure 1: Distribution of Jaccard Distance

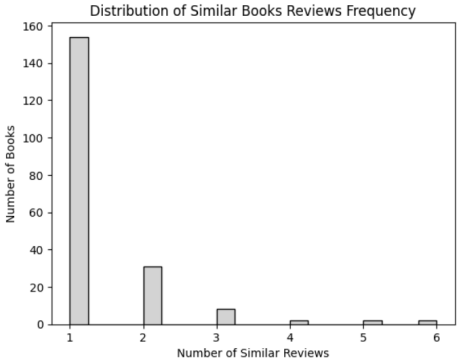

Figure 2: Distribution of Similar Books Reviews Frequency

- The count of review pairs where reviews are similar for different books, is **131** (out of 135) (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, we created a wordcloud for words in reviews column from the **135** similar books:

7 Limitations

- We compute only the approximate similarity from similar candidate pairs using LSH, and didn't compare whole dataset which will cost O(N*N) complexity.
- Since we are using Jaccard Similarity, there is no semantic meaning capture.
- Since, we also removed the stopwords and other common words along with limiting the review length and token length, we are not accounting in short reviews that may be just like 'I loved this book, always recommend", "The book is very nice to read" etc.

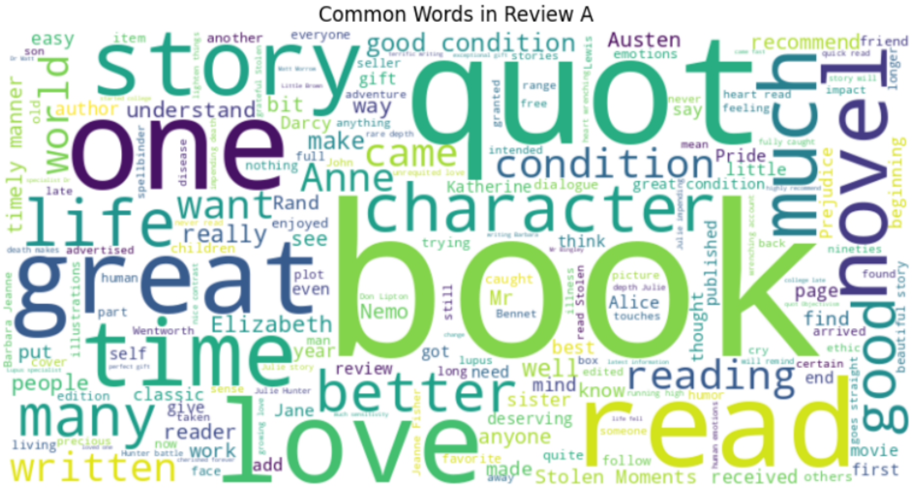

Figure 3: WordCloud

8 Conclusion

This project explains how **MinHashLSH** can be used to efficiently compute review similarities in a large dataset. Further, we also highlight that how computationally 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 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."