Finding similar items project

Nazar Liubas

June 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.

Contents

1 Introduction

This project aims to develop and test efficient data mining algorithms in order to detect pairs of similar book reviews among the large dataset and output pairs that exhibit high similarity based on a chosen metric.

To measure similarity between reviews, Jaccard similarity is used as the key metric. It compares sets of words extracted from reviews and evaluates their overlap. By transforming each review into a set of tokens and comparing defined candidate pairs, the system identifies those with a sufficiently high Jaccard index, indicating strong similarity.

In order to speed-up the computatio nprocess, algorithms MinHash and Locality-Sensitive Hashing are applied.

The implementation is designed to scale with large datasets and thus benefits from distributed computing capabilities using Apache Spark. This allows for efficient review processing, making the approach suitable for large amounts of data.

The implementation can be found on GitHub repository [4].

2 Dataset and data preprocessing

For this assignment, the Amazon Books Reviews dataset was used [1]. Since the goal was to implement a detector of pairs of similar book reviews, a column 'review/text' of the file 'Books_ratings.csv' was used as the principal column for analysis. In order to identify the rows uniquely, an 'Id' column was generated as concatenation of columns 'Id' (the id of book reviewed), 'User_id' (the id of user who rate the book) and 'review/time' (time of given the review). All other columns were deleted for the purpose of efficient memory use.

The entire data set is large (3,000,000 lines), so only subsets of the first N lines were used for the analysis. N used was between 50 and 1,000 for the algorithm correctness test, and 100,000 lines for the final execution of algorithm.

To deal with text data in more efficient way, the following preprocessing techniques were applied, as inspired by [3]:

1. Removal of double quotes
2. Splitting the document by non-word characters to get the tokens
3. Removal of stopwords, i.e., the most common words in language, which don't convey much information

In order to benefit from parallel computing, distributed storage and scalability, data was organized as Spark Resilient Distributed Dataset during the algorithms execution.

3 Algorithms applied

The algorithms implemented in this project are based on the work of Leskovec, Rajaraman, and Ullman [2].

To compare documents, the Jaccard similarity was used as the primary metric. For two sets S and T, their Jaccard similarity is defined as: $|S \cap T|/|S \cup T|$.

To efficiently approximate Jaccard similarity for large sets, MinHash was developed. MinHash is a technique for estimating the Jaccard similarity by comparing compact

signatures derived from the minimum values of multiple hash functions applied to the set elements. For each document, a MinHash signature was generated using the SHA-1 hash functions from Python's hashlib library, with hash values truncated to six digits. Details on selecting the optimal number of hash functions are provided in Section 4.1.

To avoid the computations of exhaustive pairwise comparisons, Locality-Sensitive Hashing (LSH) was used to identify candidate pairs. LSH is a technique that hashes similar items into the same buckets with high probability, thereby enabling fast approximate nearest neighbor search. The method requires setting parameters: the number of bands B and the number of rows R in each band, such that B*R=N, where N is the number of hash functions. In this implementation, B=20 and R=4, yielding a similarity threshold of approximately 0.5 for considering two documents as similar. LSH was implemented in code using the SHA-1 hash function (truncated into 10-digit hash values), analogous to the MinHash implementation.

Finally, all candidate pairs identified by LSH were filtered to retain only those with an estimated Jaccard similarity greater than 0.5.

4 Experiments

4.1 N for the minhash signatures

In order to select the proper parameter N, length of minhash signatures, an experiment was conducted on 50 rows of data. First, Jaccard similarity was calculated on actual tokens of each combination of lines and then on created signatures of documents. The values of N that were tried are: (10, 30, 80, 100, 200, 400, 1000).

Then, the Mean Absolute Error (MAE) was found between the calculated similarity on actual tokens and on signatures. The formula for MAE is:

$$\text{MAE}(y_i, \hat{y}_i) = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|$$

The results of the experiment are presented in Figure 1. As shown, the most significant reduction in error occurs at N=80, showing that this is a good choice for the number of hash functions in the MinHash algorithm. Moreover, given that the maximum possible Jaccard similarity is 1.0, and the observed error falls below 0.015, the performance of the approximation can be considered satisfying.

Taking this into account, and also noticing in Figure 2 that the execution time is linearly dependent from the number of hash functions, it was decided to proceed with N=80. The time here is execution of Jaccard similarity calculation for each pair among 50 lines of the dataset.

4.2 Scalability

The system scales effectively with the size of the data due to its implementation with the Apache Spark library, which parallelizes operations across partitions. Additionally, caching, approximate similarity search via MinHash, candidate pairs identification via LSH and configurable subsampling make it possible to process subsets of different scale without redesign.

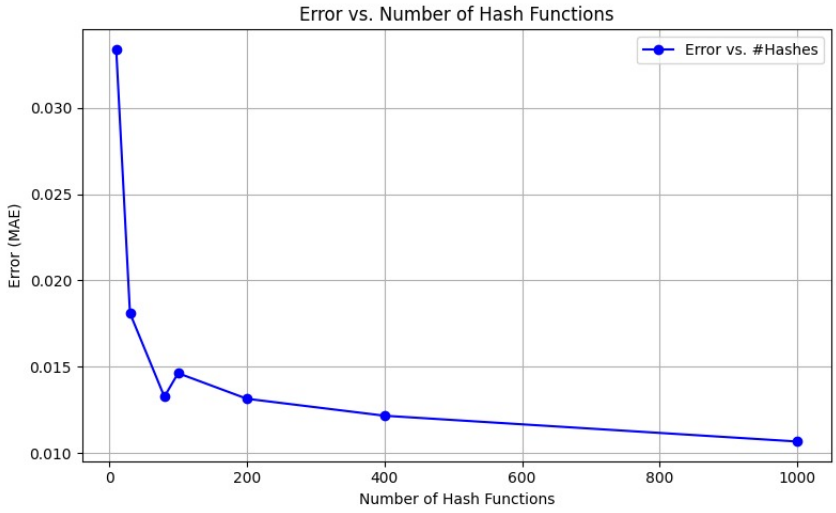

Figure 1: Mean Absolute Error depending on number of hash functions N

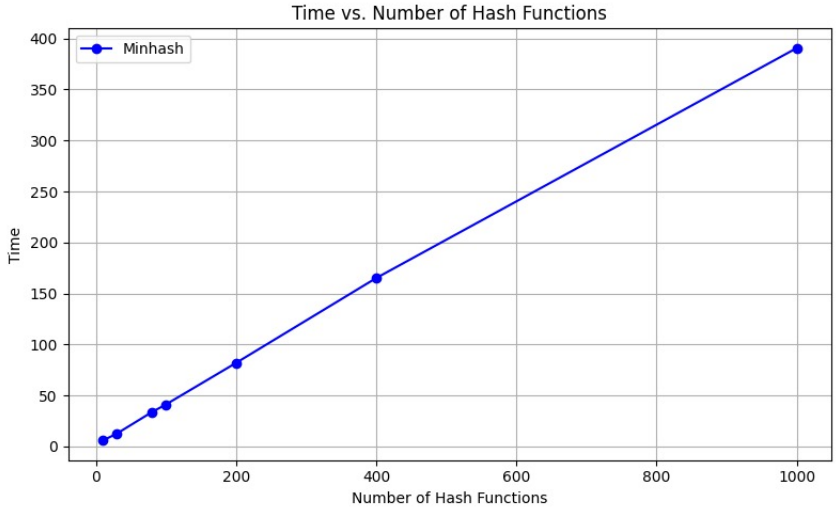

Figure 2: Execution time depending on number of hash functions N

The final algorithm was run on subsets of data of different sizes (500, 1000, 3000, 10000, 20000, 50000]). The Figure 3 shows linear relation between size and execution time, which shows significant improvement compared to expected quadratic complexity of 'brute force' pairwise comparison approach.

4.3 Final execution

The final execution of the algorithms was performed on a subset of 100,000 records and required approximately 20 minutes to complete on Google Colab.

Two reviews were considered similar if their Jaccard similarity exceeded a threshold of 0.5. Based on this criterion, a total of 9,092 similar pairs were identified and saved to the file 'found_pairs.csv'.

The distribution of the similarity scores among the detected pairs is illustrated in Figure 4. Notably, a significant portion of these pairs exhibit a similarity score of 1.0, indicating a high occurrence of duplicate reviews. Upon manual inspection, it was ob-

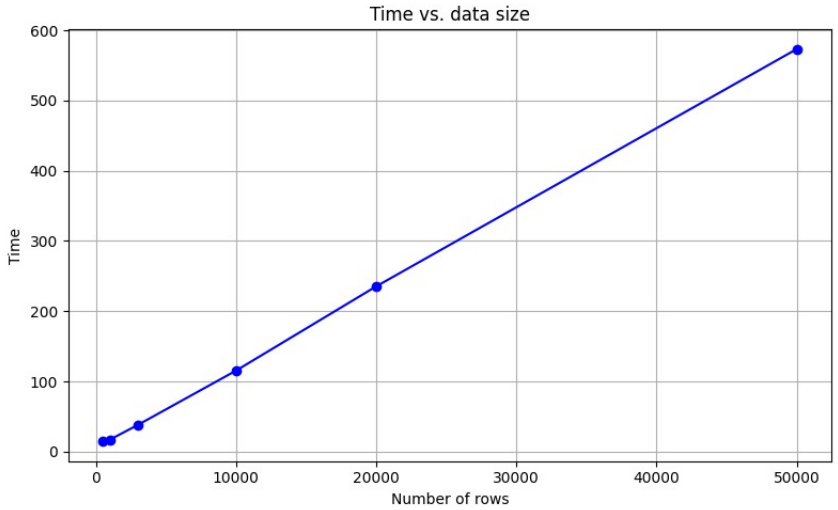

Figure 3: Final algorithm execution time depending on data size

served that while the text content was identical, the synthetic identifiers differed—likely due to how the original dataset was constructed. For example, the same book might be assigned different IDs, or an identical review could have been posted at different times.

Furthermore, some reviews were found to be nearly identical but not exactly the same, with only 'time' difference in identifier. These values likely represent edits of the same original comment over time.

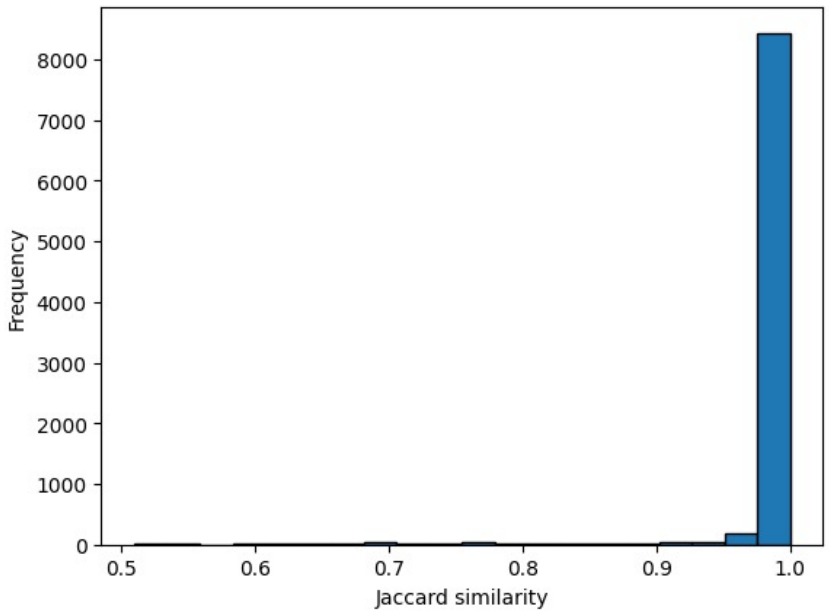

Figure 4: Histogram of similarities between found pairs

Some short examples of pairs are:

Example 1:

- *"just a great book. one of the best i have ever read"*
- *"The best book ever read but one of hardest"*

- Jaccard similarity: **0.62**

Example 2:

- *"The Hobbit is the best book i have ever read!"*
- *"This book might have been the best book I have ever read.Very adventerous.BEST BOOK EVER!"*
- Jaccard similarity: **0.54**

5 Conclusions

The final implementation successfully detected pairs of similar reviews using MinHash and Locality-Sensitive Hashing (LSH), optimized for scalability through Apache Spark.

A total of 9,092 pairs with Jaccard similarity greater than 0.5 were found among 100,000 lines and saved into 'found_pairs.csv'. The histogram in Figure 4 reveals that a large number of these pairs have a similarity score of exactly 1.0, showing a presence of duplicate reviews. But also a number of highly similar but not identical reviews were found.

To improve the current approach, one could try using other similarity methods such as sentence embeddings (e.g., BERT), which could detect more nuanced similarities.

References

[1] Mohamed Bakhet. *Amazon Books Reviews*. `https://www.kaggle.com/datasets/mohamedbakhet/amazon-books-reviews/data`. Accessed via Kaggle. 2020.

[2] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. "Finding Similar Items". In: *Mining of Massive Datasets*. Cambridge University Press, 2014, pp. 73–134.

[3] Dario Malchiodi. *Entity Resolution*. `https://malchiodi.di.unimi.it/archive/entity-resolution/entity-resolution.pdf`. Lecture notes, University of Milan. 2015.

[4] Liubas N. *Finding Similar Items*. `https://github.com/liubas3171/Finding-similar-items`. Accessed: 2025-06-09. 2025.