Finding Similar Book Reviews

Scalable Book Review Similarity System

Algorithms for Massive Data

Master in Data Science for Economics

Academic Year 2024/25

Viola Awor
Università degli Studi di Milano

October 24, 2025

1 Introduction

This report presents the work for a scalable system designed to identify similar book reviews from the large-scale Amazon Books Review dataset. The system applies techniques from the field of Algorithms for Massive Data to compute textual similarity efficiently, even when handling millions of records.

The main focus is on using **Locality Sensitive Hashing (LSH)** with **MinHashing** to approximate **Jaccard Similarity** among textual reviews. This approach avoids the computational explosion of comparing every possible pair of documents (an $O(N^2)$ operation) while maintaining strong accuracy in detecting near-duplicate or semantically similar reviews.

The report outlines the data acquisition, preprocessing pipeline, vocabulary construction, similarity computation methods, scalability strategies, and final results of the implementation.

2 Dataset and Data Selection

The dataset used for this project is the *Amazon Books Review* dataset, publicly available on both Kaggle and Data.world. The dataset contains millions of user reviews for books, including fields such as review text, rating, and book metadata.

For scalability testing, the dataset was loaded into an **Apache Spark DataFrame**. Spark was chosen for its distributed data processing architecture, which allows efficient computation over very large datasets. Since the full dataset contains over 3 million records, a 5% sample (approximately 150,000 entries) was used for experimentation and testing.

Sampling allowed for manageable development and debugging without compromising representativeness. The system was designed so that the sampling ratio can be easily adjusted to larger scales in future tests.

3 Data Organization

The reviews were organized into a Spark DataFrame, where each row represents a single review. This format is ideal for distributed processing and enables the use of Spark MLlib's feature engineering and similarity libraries.

Each review was represented by the following key columns:

- `review/text`: The raw textual review content.
- `tokens`: List of words after tokenization.

- `tokens_clean`: Tokens after filtering and cleaning steps.

This structure ensures flexibility and traceability throughout the preprocessing and feature engineering stages.

4 Preprocessing Techniques

The preprocessing phase focused on cleaning and preparing the textual data to improve the quality of similarity computation.

4.1 Tokenization

The text was tokenized using Spark's `RegexTokenizer`, which splits the text into tokens based on a regular expression pattern (`\W`). This ensures that punctuation and non-alphanumeric characters are treated as delimiters.

4.2 Stop Word Removal

Common English stop words such as "the," "is," and "and" were removed using Spark's `StopWordsRemover`. These high-frequency words carry little semantic weight and can distort similarity measurements.

4.3 Number Removal

A custom User-Defined Function (UDF) was applied to remove numerical tokens. Numbers were excluded because they do not contribute meaningfully to the textual similarity between book reviews, especially when review scores are already captured in separate columns.

4.4 High-Frequency Token Filtering

To improve the discriminative power of the vocabulary, very frequent tokens (appearing in more than 5% of the sampled reviews) were removed. These tokens were treated as *custom stop words*, as they tend to be too common across documents to provide useful distinction.

Examples include words such as "book" or "read," which are contextually expected in nearly all reviews. This filtering step helps focus on words that truly differentiate content.

5 Vocabulary Creation and Filtering

After cleaning, the vocabulary was constructed using Spark's `CountVectorizer`. This method builds a vocabulary based on token frequencies across all documents.

Two parameters were particularly important:

- `vocabSize = 5000`: Limits the vocabulary to the 5,000 most frequent terms.
- `minDF = 10`: Ensures that only tokens appearing in at least 10 documents are retained.

This process reduces dimensionality and ensures that only meaningful words are represented in the final feature vectors.

The `CountVectorizer` output was used to transform each review into a vector representation suitable for similarity analysis.

6 Hashing and Similarity Calculation

After vocabulary construction, each review's cleaned tokens were transformed into numerical vectors using Spark's `HashingTF`. This technique implements the "hashing trick" to efficiently map tokens to fixed-length feature vectors without maintaining a full vocabulary dictionary.

The vectorization parameters were as follows:

- `numFeatures = 8192`: The dimensionality of the hashed feature space.

Once feature vectors were obtained, **MinHashLSH** (a Locality Sensitive Hashing implementation) was used to perform approximate similarity computation. MinHashLSH hashes similar feature vectors into the same "buckets," drastically reducing the number of pairwise comparisons required.

The key parameters were:

- `numHashTables = 10`: The number of hash tables used.
- `sim_threshold = 0.05`: The maximum allowed Jaccard Distance between reviews to be considered similar.

An approximate similarity join was performed using `MinHashLSHModel.approxSimilarityJoin`, producing pairs of reviews that had Jaccard Distance less than or equal to 0.05. Duplicate or null token sets were filtered out before the join to ensure consistency and efficiency.

7 Scalability Considerations

The scalability of the system is ensured by leveraging Apache Spark's distributed computing framework. Spark allows tasks such as tokenization, stopword removal, and hashing to be parallelized across multiple nodes. This enables the system to handle datasets that exceed the memory capacity of a single machine.

Locality Sensitive Hashing (LSH) contributes significantly to scalability by replacing the exhaustive pairwise similarity computation ($O(N^2)$) with a more efficient bucket-based approach. Instead of comparing all review pairs, only items hashed into the same or nearby buckets are considered, dramatically reducing computational cost.

This design allows the system to scale horizontally—additional machines can be added to the Spark cluster to process larger subsets of the dataset in parallel.

8 Results

The system's output is a Spark DataFrame containing pairs of similar reviews based on the Jaccard Distance. The final result includes:

- The IDs and titles of the paired books.
- Their corresponding review texts.
- The computed Jaccard Distance between their token sets.

The pairs are sorted in ascending order of Jaccard Distance, meaning that the most similar pairs appear at the top. The top 10 most similar pairs are displayed using the `show()` function.

These results effectively demonstrate the system's ability to detect overlapping or near-duplicate reviews, which is potentially useful for identifying redundant entries or correlated opinions on similar books.

9 Conclusion

This project demonstrates a scalable and efficient approach to identifying similar book reviews using Apache Spark and Locality Sensitive Hashing. Through careful preprocessing, token filtering, and distributed computation, it achieves robust results even when applied to large-scale data.

The combination of `HashingTF` and `MinHashLSH` proved effective for approximate similarity computations while maintaining computational feasibility. The methodology can easily scale to the full dataset or be integrated into broader applications, such as:

- Detecting duplicate or bot-generated reviews.
- Grouping related book entries.
- Enhancing recommendation systems.

Future improvements could involve experimenting with other similarity metrics (such as cosine similarity), optimizing hash table parameters, or applying lemmatization using libraries like SpaCy for more linguistically informed preprocessing.

10 References

Rajaraman, Anand, and Jeffrey Ullman (2014). *Mining of Massive Datasets*. Available online at: `http://infolab.stanford.edu/~ullman/mmds/book.pdf`

Amazon Books Review Dataset: `https://www.kaggle.com/datasets/mohamedbakhet/amazon-books-reviews`

Spark MLlib Documentation: `https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.feature.HashingTF.html`

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/our 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.