Project 1: Finding similar items

Algorithm for Massive Data

July 10, 2025

Abstract

This project presents a robust similarity detection system for book reviews, leveraging Locality-Sensitive Hashing (LSH) with MinHash. Designed to handle large-scale text data, the system efficiently identifies duplicate and near-duplicate reviews in the Amazon Books dataset by combining distributed computing with Apache Spark and a streamlined preprocessing pipeline. The pipeline includes text normalization, tokenization, stopword removal, stemming, and memory-efficient feature extraction using HashingTF.

To enable rigorous evaluation, synthetic duplicates are introduced into the dataset, allowing precise measurement of detection performance. Experiments across varying similarity thresholds highlight the system's ability to balance precision and recall effectively. The solution demonstrates strong accuracy, efficient processing time, and clear scalability, making it suitable for deployment in real-world review platforms.

[0]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/us or any other person for assessment on this or any other course of study.

Contents

1 Introduction

The exponential growth of user-generated content on e-commerce platforms has created unprecedented challenges in maintaining data quality and integrity. Book review platforms, in particular, face issues with duplicate reviews that can arise from technical glitches, deliberate spam, or users posting similar content across multiple products. This project addresses these challenges by developing a sophisticated similarity detection system specifically designed for book reviews.

The primary motivation behind this work stems from the need to:

- Maintain the integrity of review systems by identifying and potentially removing duplicate content
- Improve user experience by ensuring diverse and authentic review content
- Provide scalable solutions that can handle millions of reviews in production environments
- Develop techniques that balance computational efficiency with detection accuracy

This project implements a complete end-to-end solution that processes raw review text, extracts meaningful features, and identifies similar reviews using state-of-the-art algorithms. The system leverages distributed computing paradigms to ensure scalability while maintaining high accuracy in duplicate detection.

2 Dataset Description and Analysis

2.1 Dataset Overview

The project is based on the Amazon Books Reviews dataset, a comprehensive collection of customer reviews from Amazon's book marketplace available through Kaggle. This dataset represents real-world challenges in review processing, including varying review lengths, diverse writing styles, and natural language complexities.

2.2 Data Selection Strategy

From the complete dataset, the project focuses on specific attributes essential for similarity detection:

- `review/text`: The primary textual content of customer reviews, containing the raw opinions and feedback
- `review_id`: A system-generated monotonically increasing identifier assigned to each review for tracking purposes

The selection process applies rigorous filtering criteria to ensure data quality:

- Reviews must be non-null and non-empty to avoid processing errors
- Minimum length requirement of 10 characters to exclude trivial reviews
- Post-preprocessing requirement of at least 3 unique words to ensure meaningful content

2.3 Dataset Characteristics

The experimental dataset consists of 10000 carefully selected reviews that represent a diverse range of:

- Review lengths: from short opinions to detailed analyses
- Writing styles: from casual to formal academic critiques
- Content types: plot summaries, character analyses, personal opinions, and recommendations
- Language complexity: varying vocabulary richness and sentence structures

This sample size balances computational efficiency with statistical significance, allowing for meaningful evaluation while maintaining reasonable processing times.

3 Preprocessing Techniques

3.1 Text Cleaning and Normalization

The preprocessing phase implements sophisticated text cleaning mechanisms that prepare reviews for analysis while preserving semantic content. The cleaning process addresses common issues in user-generated content:

- Consistent capitalization through lowercase conversion

- Removal of punctuation, numbers, and special characters
- Normalization of whitespace to single spaces
- Elimination of leading and trailing whitespace

This multi-step process ensures uniformity across all reviews while maintaining the core semantic content necessary for similarity detection.

3.2 Tokenization Strategy

The tokenization process employs a sophisticated regex-based approach that goes beyond simple space splitting. The system uses Spark's RegexTokenizer with careful pattern matching to handle various text formats and ensure consistent token extraction across diverse writing styles.

3.3 Linguistic Processing

3.3.1 Stopword Removal

Comprehensive stopword removal is conducted using Spark's built-in English stopword list as part of the project's preprocessing pipeline, which includes common articles, prepositions, conjunctions, auxiliary verbs, and pronouns. Additionally, the system filters words with length 2 characters, removing abbreviations and fragments that don't contribute to semantic meaning.

3.3.2 Stemming Implementation

The Snowball Stemmer provides sophisticated morphological analysis, handling various morphological variations including:

- Plural forms (books → book)
- Verb conjugations (running → run)
- Comparative forms (better → better, biggest → big)
- Derived forms (quickly → quick)

The stemming process includes additional validation to ensure stems are meaningful (length greater than 1) and contribute to the similarity detection process.

4 Data Organization and Pipeline Architecture

4.1 Distributed Processing Framework

The proposed framework employs Apache Spark as its core processing engine, leveraging its distributed computing capabilities to handle large-scale data efficiently. The architecture consists of several interconnected components that work together to process reviews from raw text to similarity scores.

4.2 Pipeline Stages

The data processing pipeline implements a sequential transformation approach with the following detailed stages:

1. **Data Ingestion**: The system reads CSV files using Spark's built-in CSV reader with automatic schema inference, handling various data types and potential formatting issues
2. **Initial Filtering**: Removes malformed records, empty reviews, and reviews that don't meet minimum quality criteria
3. **Text Preprocessing**: Applies the comprehensive cleaning, tokenization, and linguistic processing described in the previous section
4. **Feature Generation**: Transforms processed text into numerical vectors using the Hashing Trick, creating sparse representations suitable for similarity computation
5. **Similarity Detection**: Applies MinHash LSH to identify candidate pairs and compute similarity scores

4.3 Data Flow Management

The pipeline implements intelligent data flow management through:

- Lazy evaluation to optimize query execution
- Strategic caching of intermediate results to avoid recomputation
- Partition management to ensure balanced workload distribution
- Memory-efficient transformations that minimize data shuffling

5 Algorithmic Implementation

5.1 Feature Extraction using HashingTF

Feature extraction is carried out using the Hashing Trick (also known as feature hashing) for efficient feature extraction. This technique provides several critical advantages for large-scale text processing:

5.1.1 Mathematical Foundation

The hashing function maps words to indices in a fixed-size vector:

$$h : \text{word} \rightarrow \{0, 1, ..., n-1\} \tag{1}$$

where $n = 5000$ represents the chosen vocabulary size. The hash function ensures uniform distribution of words across the vector space while maintaining deterministic mapping.

5.1.2 Trade-offs with Alternative Approaches

The choice of HashingTF over alternatives reflects careful consideration:

- **vs. TF-IDF**: While TF-IDF provides better feature weighting through inverse document frequency, it requires maintaining a vocabulary dictionary and computing global statistics, limiting scalability
- **vs. Word Embeddings**: Dense embeddings (Word2Vec, GloVe) capture semantic relationships but require pre-trained models and higher memory usage
- **HashingTF Benefits**: Constant memory usage, no vocabulary maintenance, and streaming compatibility make it ideal for large-scale applications
- **HashingTF Limitations**: Potential hash collisions and lack of semantic understanding

5.2 Locality-Sensitive Hashing with MinHash

5.2.1 Theoretical Background

MinHash provides a probabilistic method for estimating Jaccard similarity between sets. For two sets A and B, the Jaccard similarity is defined as:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{2}$$

The MinHash algorithm approximates this similarity using multiple hash functions:

$$\Pr[h_{\min}(A) = h_{\min}(B)] = J(A, B) \tag{3}$$

where $h_{\min}(S) = \min_{x \in S} h(x)$ represents the minimum hash value for set S.

5.2.2 LSH Implementation Architecture

The implementation uses a banding technique to reduce the number of comparisons. The system divides MinHash signatures into bands, where documents with identical bands become candidates for similarity. The number of hash tables (3) controls the precision-recall trade-off, with each table using different hash functions for independent estimates.

6 Experimental Design and Methodology

6.1 Evaluation Framework

The experimental design implements a controlled evaluation methodology using synthetic duplicates. This approach is critical as it provides precise ground truth for measuring detection accuracy - without known duplicates, it would be impossible to calculate precision and recall accurately.

6.1.1 Ground Truth Generation

The evaluation process creates artificial duplicates through:

1. Random selection of 25 source reviews using a fixed seed for reproducibility
2. Creation of exact duplicates with modified identifiers (prefixed with "dup_")
3. Integration of duplicates into the original dataset
4. Maintenance of mapping between original and duplicate pairs

6.1.2 Evaluation Metrics

The proposed approach utilizes standard information retrieval metrics:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{4}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{5}$$

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{6}$$

6.2 Parameter Configuration

Table 1: Experimental Configuration Summary

Parameter	**Value**
Sample Size	10000 reviews
Artificial Duplicates	25
Vocabulary Size	5000
LSH Hash Tables	3
Similarity Thresholds	[0.3, 0.5, 0.7]
Random Seed	42

7 Results and Analysis

7.1 Performance Metrics Across Thresholds

The system evaluation reveals significant variations in performance based on similarity threshold selection:

Table 2: Performance Results Summary

Threshold	**Precision**	**Recall**	**F1-Score**	**Pairs Found**
0.3	0.008	0.960	0.016	3125
0.5	0.490	0.960	0.649	49
0.7	0.846	0.880	0.863	26

7.2 Threshold Impact Analysis

The results demonstrate a clear trade-off between precision and recall:

- **Low threshold (0.3)**: High recall but impractical due to excessive false positives
- **Medium threshold (0.5)**: Balanced performance suitable for semi-automated workflows
- **High threshold (0.7)**: Optimal F1-score with high precision for automated systems

7.3 Performance Characteristics

The system demonstrates consistent efficiency:

- Total processing time: 25-28 seconds for 10000 reviews
- Average throughput: 350-400 reviews per second
- Linear scaling with dataset size

8 Conclusion

This project successfully implements a comprehensive and scalable similarity detection system for book reviews using advanced hashing techniques. The combination of thorough text preprocessing, efficient feature extraction through the hashing trick, and Locality-Sensitive Hashing with MinHash creates a robust solution for duplicate detection at scale.

Key achievements include:

- High accuracy with F1-score of 0.863 using optimal configuration
- Scalable architecture processing 10000 reviews in under 30 seconds
- Configurable precision-recall trade-offs for different use cases
- Robust preprocessing handling real-world text variations
- Rigorous evaluation methodology using synthetic duplicates

The project provides a solid foundation for production deployment while identifying clear paths for future enhancement. The modular architecture facilitates component improvements without system redesign, and the evaluation framework enables systematic performance assessment of future modifications.