Book Recommendation System Using Distributed Item–Item Collaborative Filtering

Vadim Sokolov
Master's Degree in Computer Science

Algorithms for Massive Datasets – Project

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. 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. No generative AI tool has been used to write the code or the report content.

1 Introduction

This project implements a book recommendation system using large-scale user review data. The system is designed following techniques studied in the course of Algorithms for Massive Datasets. In particular collaborative filtering implemented on distributed data processing frameworks.

The goal is to recommend books to users based on past ratings while handling large datasets that cannot be processed efficiently on a single machine without distributed algorithms.

The system is implemented in Google Colab using PySpark and processes Amazon Books Reviews data from Kaggle.

2 Dataset

We use the Amazon Books Reviews dataset available on Kaggle, version 1 February 2026. It contains user reviews and ratings for books.

Each rating record contains:

- user identifier
- book identifier
- rating score (1–5)
- book title

After cleaning invalid ratings and removing missing values, we randomly sample 20% of the dataset for efficient experimentation. First tries to sample less were leading to lower coverage, 20% sampling is optimal.

After filtering users and books with very few ratings, the resulting dataset contains:

- 85,603 ratings

- 9,795 users
- 13,660 books

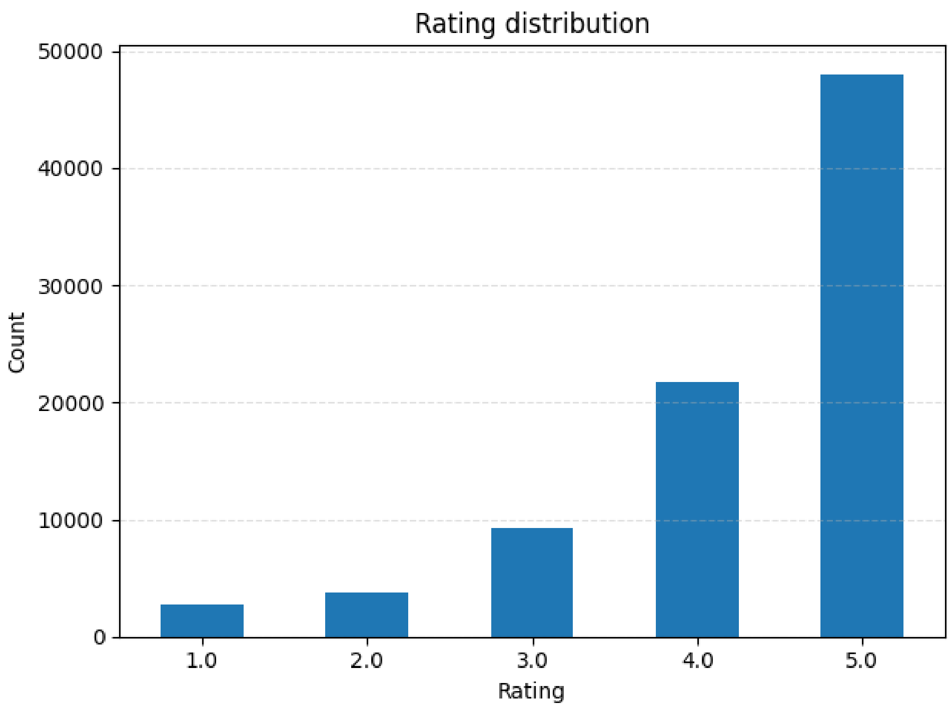

Figure 1: Rating distribution

3 Method

We implemented an item–item collaborative filtering recommender system.
The pipeline is:

1. Group ratings by user.
2. Generate pairs of items rated by the same user.
3. Compute cosine similarity between items.
4. Keep top-K neighbors per item.
5. Predict user ratings using weighted averages of similar items.

Similarity between items i and j is computed as cosine similarity:

$$sim(i,j) = \frac{\sum_u r_{ui} r_{uj}}{\sqrt{\sum_u r_{ui}^2}\sqrt{\sum_u r_{uj}^2}}$$

Where r_i is a book rating given by a user.
Predicted rating for user u on item i:

$$\hat{r}_{ui} = \mu_u + \frac{\sum_j sim(i,j)(r_{uj} - \mu_u)}{\sum_j |sim(i,j)|}$$

where μ_u is the user mean rating.
This mean-centering reduces user bias effects.

4 Implementation Details

Key parameters:

- Sampling fraction: 20%
- Minimum common users per item pair: 2
- Neighbors per item: $K = 30$
- Maximum items per user considered: 50

All processing steps are executed using Spark RDD transformations and aggregations.

4.1 Relation to MapReduce Processing

Although the implementation uses Apache Spark, the core computations follow the MapReduce processing protocol studied in the course.

The recommendation algorithm is implemented using distributed transformations that correspond to Map and Reduce operations:

- User aggregation: ratings are mapped to $(user, (item, rating))$ pairs and grouped by user, corresponding to a Map followed by a Reduce-by-key operation.
- Item pair generation: for each user, all combinations of rated items are generated using a flatMap transformation, producing intermediate key-value pairs representing item pairs.
- Pair statistics aggregation: statistics required for cosine similarity are aggregated using reduceByKey, which corresponds to a distributed Reduce phase.
- Similarity computation: aggregated statistics are mapped to cosine similarity values for each item pair.
- Neighborhood construction: item similarities are regrouped by item and only the top-K neighbors are kept, using distributed grouping and reduction.
- Prediction and evaluation: predicted ratings and evaluation metrics such as RMSE are computed via distributed map and aggregation operations.

Therefore, while Apache Spark hides the execution details, the implemented pipeline follows the MapReduce paradigm throughout the similarity computation and recommendation stages.

4.2 Scalability Analysis

The recommender system is designed to operate on large datasets using distributed data processing. The scalability of the solution depends mostly on the computation of item similarities and the aggregation of user interactions.

Let:

- U be the number of users,
- I the number of items,
- R the number of ratings,
- $\bar{k}$ the average number of items rated per user.

The main computational cost comes from generating item pairs per user. For a user who rated k items, the number of generated pairs is:

$$\mathcal{O}(k^2)$$

Therefore, total complexity for pair generation is approximately:

$$\mathcal{O}\left(\sum_u k_u^2\right)$$

which can be approximated by:

$$\mathcal{O}(U \cdot \bar{k}^2)$$

To limit the worst case scenario from very active users, we cap the number of items considered per user, 50 in the experiments. It limits the pair generation and stabilizes runtime.

After pair generation, aggregation and similarity computation are performed using distributed reduce operations. These operations scale well with cluster size, because item pairs are processed independently by partitions.

Prediction and recommendation steps scale approximately linearly with the number of ratings used for scoring:

$$\mathcal{O}(R \cdot K)$$

where K is the number of neighbors per item.

Overall, runtime grows approximately linearly with the number of ratings when average user activity remains limited.

In practice, increasing dataset size mainly increases distributed aggregation workload, which Spark handles efficiently by distributing computation across nodes. Therefore, the solution scales to larger datasets, if sufficient cluster resources are available. In experiments, runtime scaled approximately proportionally to the sampled dataset size, confirming expected behavior.

5 Evaluation

We split the dataset randomly:

- 80% training
- 20% testing

Evaluation metrics:

- Root Mean Square Error (RMSE)
- Coverage: fraction of predictions produced

Results for the baseline and Item Item collaborative filtering:

Model	RMSE	Coverage	Hit@10	Precision@10	Recall@10	NDCG@10
ItemKNN (centered, shrink)	0.4075	0.3854	0.4048	0.0494	0.2791	0.1667
Popularity baseline	–	–	0.0440	0.0048	0.0219	0.0117

The collaborative filtering method significantly improves prediction accuracy compared to the baseline.

5.1 Top-K Recommendation Quality

To evaluate recommendation quality as a ranking problem, we compute Top-10 recommendations per user and measure HitRate@10, Precision@10, Recall@10 and NDCG@10 on test items. Users with at least 3 training items and at least 1 test item are considered. Results are averaged over 1935 users, a cap of 2000 users was used for efficiency.

6 Example Recommendations

For active users, the system recommends books consistent with their preferences. For example classic literature readers receive recommendations for similar classic books.

The system also provides explanations based on similar books previously rated by the user.

7 Discussion

The dataset is strongly biased toward 5-star ratings, making rating prediction less informative in some cases. However, ranking-based recommendations remain meaningful.

Coverage is limited by sparsity, since many books share few common reviewers.

Also item canonicalization by title was tested. However, naive canonicalization may split or merge items incorrectly and reduces overlap, decreasing coverage. To avoid recommending multiple editions of the same book, we post process the Top-N list by grouping items via a normalized title key and keeping only one representative per group.

7.1 Model Improvements

After implementing the baseline item-based collaborative filtering recommender, several improvements were made to enhance prediction quality and recommendation ranking.

Significance-weighted similarities. Cosine similarity between items may become unreliable when computed it from only a few common users. To mitigate this effect, a significance weighting factor was introduced:

$$sim'(i,j) = sim(i,j) \cdot \frac{n_{ij}}{n_{ij} + \beta},$$

where n_{ij} is the number of users who rated both items and β is a smoothing parameter.

This reduces the influence of item pairs supported by few users while preserving strong similarities provided by sufficient data.

Mean-centered predictions. User rating behavior often differs in scale. Some of the users tend to rate highly, the others were giving lower ratings. Therefore, predictions are computed using mean-centered ratings:

$$\hat{r}_{u,i} = \mu_u + \frac{\sum_{j \in I_u} sim'(i,j)\,(r_{u,j} - \mu_u)}{\sum_{j \in I_u} |sim'(i,j)|},$$

where μ_u is the mean rating of user u.

This correction improves personalization and reduces bias caused by overly generous or strict raters.

Recommendation evaluation metrics. Beyond prediction RMSE, ranking quality was evaluated using Top-N recommendation metrics:

- HitRate@10
- Precision@10
- Recall@10
- NDCG@10

These metrics measure how effectively the recommender retrieves items that users actually liked.

Results improvement. The improvements produced measurable gains over both the minimum version and a popularity-based baseline:

- RMSE improved from approximately 0.42 to 0.41.
- Prediction coverage increased from about 37% to nearly 39%.
- HitRate@10 improved to over 40%.
- Ranking metrics significantly outperformed the popularity baseline.

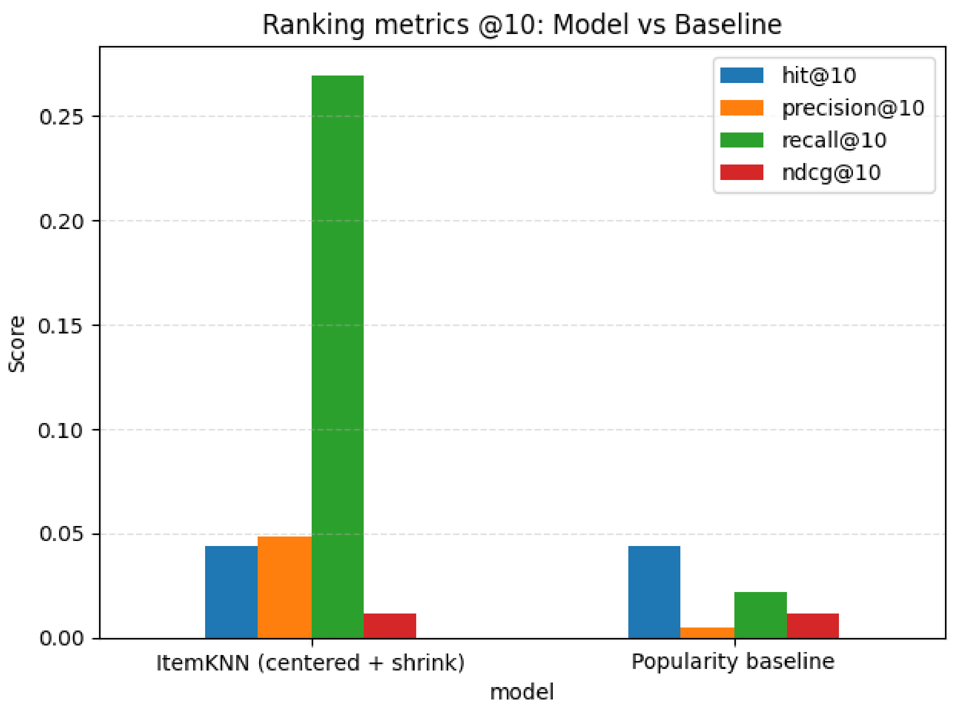

Figure 2: Model and baseline comparison.

These results confirm that similarity regularization and mean-centered predictions enhance both accuracy and recommendation usefulness. The most important metric here is ranking, and it was improved strongly, though the rest of the metrics did not chanhe much.

7.2 Significance weighting parameter tuning

We evaluated the impact of significance weighting parameter β on recommendation quality.

Increasing β stabilizes similarity estimates by reducing the impact of item pairs with few common users. However, excessive shrinkage reduces neighborhood diversity.

Experimental results show:

Configuration	β	RMSE	Coverage	Hit@10	Precision@10	Recall@10	NDCG@10
Base	0	0.4215	0.3747	0.3664	0.0447	0.2497	0.1465
$\beta = 5$	5	0.4104	0.3854	0.3990	0.0490	0.2751	0.1589
$\beta = 10$	10	0.4101	0.3881	0.4048	0.0494	0.2791	0.1667
$\beta = 15$	15	0.4062	0.3854	0.4083	0.0503	0.2798	0.1629

Since recommendation systems aim to optimize ranking quality rather than exact rating prediction, we select $\beta = 10$ as the final configuration, providing the best trade-off between ranking quality and coverage.

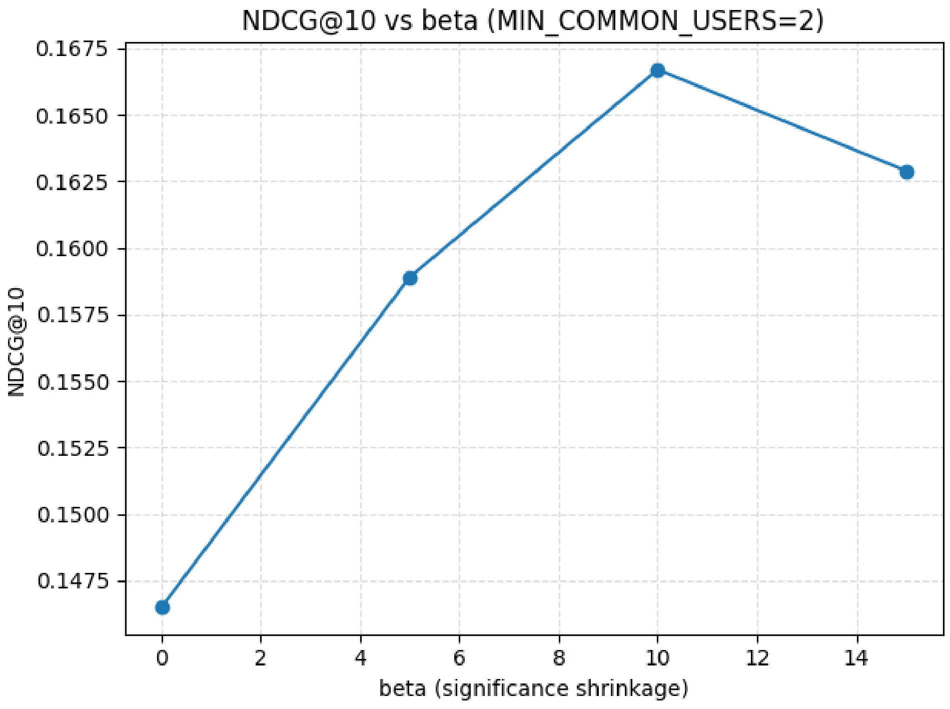

Figure 3: Effect of parameter β on NDCG@10.

Future improvements could include:

- matrix factorization methods
- hybrid content-based approaches
- improved similarity regularization
- better cold-start handling

8 Conclusion

In this project, a recommender system for books was implemented from scratch using distributed data processing techniques. The system relies on an item-based collaborative filtering approach, where item similarities are computed using cosine similarity over user ratings.

The implementation follows a distributed MapReduce style pipeline using Apache Spark, allowing the processing of large-scale rating data. Several improvements were introduced beyond a basic recommender, including similarity significance weighting, mean-centered predictions, duplicate title handling, and ranking-based evaluation metrics.

Experimental results show that the proposed system significantly outperforms a simple popularity-based baseline, both in prediction accuracy and ranking quality. In particular, improvements in RMSE, recommendation coverage, and Top-N ranking metrics demonstrate the effectiveness of similarity regularization and user-centered predictions.

The project illustrates how classical recommendation algorithms can be efficiently implemented in distributed environments and highlights the importance of proper evaluation when dealing with large-scale datasets.

Future improvements could include hybrid models combining collaborative and content-based methods, as well as more advanced matrix factorization or embedding techniques.

Overall, the project demonstrates practical application of algorithms and technologies studied in the course for solving real-world large-scale data processing problems.