CS 5130 Lab 6: Book Recommendation Engine Report

Teoman Kaman

November 28, 2025

1 Memory Optimization

We saved memory by loading only the columns we need and choosing the best data types (like using `int32` for numbers and `string` for text).

Metric	Before Optimization	After Optimization
Memory Usage	36.62 MB	4.77 MB
Reduction	31.85 MB (87.0%)	

Table 1: Memory usage comparison.

2 Performance Analysis

Calculating the similarity matrix takes the most time. The time it takes grows quickly as we add more books (complexity of $O(n^2)$).

- **Matrix Computation Time**: About 3.4 seconds.
- **Recommendation Time**: Average of 0.0139 seconds per search.
- **Scalability**: The current method works well for this dataset. However, if we had many more books, it would be too slow and use too much memory. For larger datasets, we would need different methods.

3 Algorithm Comparison

We compared three ways to recommend books:

1. **Content-Based Filtering**:
   - **Strengths**: Finds books with similar genres or authors. It works even if we don't know what the user likes yet.
   - **Weaknesses**: Can be too specific. It might not show popular or high-quality books.
2. **Popularity-Based**:
   - **Strengths**: Simple to understand. Good for new users because it shows books that most people like.
   - **Weaknesses**: Not personalized. It shows the same ”best-sellers” to everyone.
3. **Hybrid Approach**:
   - **Strengths**: Combines the best parts of the other two. It finds similar books but also considers quality.
   - **Weaknesses**: Harder to set up because you have to decide how much importance to give to each part.

4 Design Decisions

4.1 Popularity Score Calculation

We calculated popularity by combining how much people engaged with the book (ratings and reviews) and how much they liked it (average rating):

$$Engagement = 0.7 \times \log(1 + num\_ratings) + 0.3 \times \log(1 + num\_reviews)$$

$$Popularity = Engagement \times AverageRating$$

We used a log function to make sure books with millions of ratings didn't completely dominate the score.

4.2 Content-Based Weighting

We created a "feature vector" for each book using:

- **Genres**: We marked which genres a book belongs to.
- **Numbers**: We used the page count and average rating.

We used "cosine similarity" to measure how similar two books are based on these features.

4.3 Hybrid Combination

The hybrid method combines the scores like this:

$$Score = 0.7 \times ContentScore + 0.3 \times NormalizedPopularityScore$$

This gives more importance to similarity but uses popularity to help choose the best books among similar ones.

5 Recommendations Quality

We tested the system with "1984" and "Harry Potter".

- **1984**: It correctly recommended other classic dystopian books like "Animal Farm" and "Brave New World".
- **Harry Potter**: It recommended other books in the series and similar fantasy books like "The Hunger Games".

I also fixed a problem where the same book would appear multiple times. Now, the list is clean and unique.