Kindle Book Analytics: A Deep Dive into Sales, Ratings, and Trends of 130,000 books

Jun Huang
School of System Design and Intelligent Manufacturing
Southern University of Science and Technology
Shenzhen, China
huangj2021@mail.sustech.edu.cn

***Abstract*—This project presents a comprehensive analysis of the Amazon Kindle Books Dataset 2023, encompassing data for 130,000 Kindle e-books. Utilizing big data techniques for data collection, preprocessing, storage, analysis, and visualization, the project integrates Apache Kafka for efficient data streaming. Innovative approaches in data storage using MongoDB, analysis leveraging techniques like log transformation and CatBoostRegressor, and visualization through AutoViz are highlighted. The research delves into author popularity, the impact of Kindle Unlimited, the influence of 'Best Seller' and 'Editors' Pick' tags, publication trends, genre-specific sales performance, and the development of an AI recommendation model. This research offers significant insights into the evolving landscape of digital reading, marketing strategies, and consumer preferences in e-publishing.**

***Index Terms*—Big data, E-Book Analysis, MongoDB, Data Visualisation, E-Publishing Marketing Strategies**

I. Introduction of the Dataset

It is a collaboration project. This part is written by Xizhe Hao (haoxz2020@mail.sustech.edu.cn), intentionally hidden here by Huang Jun.

II. Data Storage

We employed the MongoDB as our data storage tool. The decision to utilize MongoDB for data storage in this project was driven by its suitability for managing the specific needs of the Kindle Books Dataset. MongoDB's document-oriented structure is adept at handling fuzzy search queries, a frequent requirement given the dataset's nature. This NoSQL database offers a more dynamic approach to data storage and retrieval compared to traditional relational databases like MySQL, particularly beneficial for datasets with varied and complex search demands.

The use of MongoDB in this project was primarily for importing processed CSV data into a database, with a focus on efficient data retrieval and management. This choice also facilitated easier integration with a Vue3-based front-end, enhancing the user interface and interaction. Despite the project's short duration of two weeks, MongoDB's flexible schema and straightforward implementation enabled us to effectively manage the dataset, providing a reliable foundation for basic data storage and retrieval functionalities, essential for this scale of project.

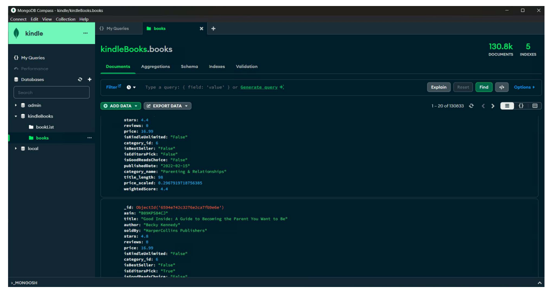


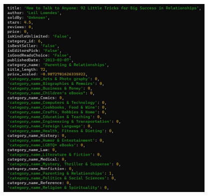
Fig. 1: Our MongoDB Compass

Fig. 2: Using MongoDB Shell Tools to import

A. Data import to MongoDB Atlas

For the data upload process in our project, we utilized the 'mongoimport' tool, a part of the MongoDB command-line utilities, to upload the processed CSV data directly to MongoDB Atlas, the cloud-based database service. This method was chosen for its simplicity and effectiveness, allowing for a straightforward transfer of data from local storage to the cloud.

At the same time, our preprocessed data has additional columns as shown in the fig. 3a for visual analysis. This kind of redundant column does not need to be saved after being stored in the database. On the contrary, it will increase the running time and additional storage cost when we retrieve it. So, we tried to use the pymongo library in python to connect to the database and perform the deletion operation. As shown in the figure 3b, we use field.startswith() func to collect all the columns to be deleted and unset them by update.



(a) Columns

(b) Delete columns

Fig. 3

Fig. 4: Return all the 130733 items to the client

B. Fuzzy Search

III. Fuzzy Search Implementation and Iteration

In our project, the fuzzy search feature was iteratively developed and refined to enhance the user experience. Initially, our implementation allowed users to perform fuzzy searches on the 'title' and 'author' fields of the books dataset. However, we noticed that such a search would often return an overwhelming number of results, as shown in the figure **??**. To address this, we introduced several iterations to improve the search functionality.

Our first major iteration involved sorting the search results by the book's star ratings. Although this method prioritized highly-rated books, it did not necessarily reflect the relevance of the search query to the results. Recognizing this, we then shifted our focus to implementing a relevance-based sorting algorithm. We employed a regular expression (regex) query, which matched the search terms with the 'title' and 'author' fields. The relevance of each book was determined by the frequency of the search terms appearing in these fields.

To further refine the search results, we introduced a combined scoring system. This system integrated both the relevance score and a weighted score, which considered factors such as whether the book was a 'Best Seller' or an 'Editors' Pick'. The combined score allowed us to sort the results in a way that balanced both the popularity and relevance of the books to the search query.

The final iteration of our fuzzy search feature included pagination to handle large volumes of data efficiently. We added query parameters for page number and limit to control the number of results displayed per page. This not only improved the performance of our search feature but also enhanced the user experience by providing a more manageable set of search results.

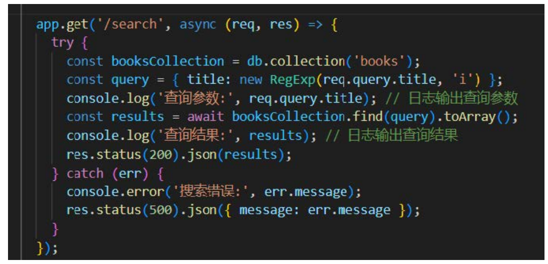


Fig. 5: Regular expression (regex) search codes

Overall, the iterative development of our fuzzy search feature played a crucial role in achieving an efficient and user-friendly search experience in our project.

1) Relevance Algorithm Details: The relevance algorithm in our project is designed to quantify the pertinence of each book in relation to a user's search query, particularly focusing on the 'title' and 'author' fields. This is achieved through the implementation of a frequency-based scoring mechanism.

While the relevance score for each book is calculated based on the frequency of query term matches within the 'title' field, normalized by the length of the title. This is represented by the following equation:

$$\text{relevanceScore} = \frac{\text{frequency of regexQuery in title}}{\text{length of title}} \quad (1)$$

Where the frequency is determined by the number of matches found using the regexQuery in the book's title. This approach ensures that books with a higher concentration of query terms in their titles are ranked higher, aligning the search results more closely with the user's intent.

2) Pipeline Application and Indexing: The MongoDB aggregation pipeline in our project plays a vital role in optimizing the search operation. We designed a concise pipeline to streamline data processing, enhancing performance especially for large datasets. The pipeline consists of stages like 'match', 'addFields', and 'sort', each tailored to refine the search results efficiently.

Pseudo-code for the pipeline:

- Match Stage: Filter documents using regexQuery on 'title' and 'author'.
- Add Fields: Compute relevanceScore based on query match frequency.
- Sort: Order documents by combined relevance and weighted scores.

This pipeline structure allows for the processing of data in a sequential and efficient manner. The 'match' stage, enhanced by indexing on key fields, significantly reduces the data volume for subsequent stages, leading to faster query execution times.

Indexing 'title' and 'author' fields ensures rapid access during the 'match' stage, enhancing the overall speed of the pipeline. This optimization is crucial for handling complex searches within the extensive dataset.

3) Efficient Pagination with Pipeline: The integration of pagination in our search functionality significantly enhanced performance by dividing the search and data retrieval process into manageable segments. Pagination is crucial in our project as it not only improves the user experience by providing a structured and organized view of the results but also optimizes the server's load, especially when dealing with large datasets.

Our pagination implementation works as follows:

- Calculate the number of documents to skip (*skip*) based on the current page number (*page*) and the desired number of results per page (*limit*).
- Apply the skip and limit operations within the aggregation pipeline to control the flow of data.

The key advantage of this approach is that pagination is coupled with the sorting mechanism within the same pipeline. This ensures that sorting is applied to the entire dataset before the pagination logic takes effect, resulting in a consistent and accurate order of results across pages. Specifically, the 'sort' stage arranges documents based on their combined relevance and weighted scores, and subsequently, the 'skip' and 'limit' stages slice the sorted data into paginated segments.

By incorporating pagination directly within the aggregation pipeline, we achieved an efficient balance between data processing speed and user experience. This method proves particularly effective for our extensive dataset, ensuring that users receive prompt responses while maintaining the integrity and order of the search results.

4) Implementation of Weighted Scoring: In our project, we introduced an additional column, 'weightedScore', in our database to facilitate an enhanced sorting mechanism for the search results. This weighted score is calculated by aggregating various factors that could influence a book's overall appeal, including its star rating and specific tags like 'Best Seller', 'Editors' Pick', and 'Good Reads Choice'. The calculation of the weighted score is as follows:

$$\begin{aligned} \text{weightedScore} = &\text{isEditorsPick} + \text{isBestSeller} \\ &+ \text{stars} + \text{isGoodReadsChoice} \end{aligned} \tag{2}$$

Where 'isBestSeller', 'isEditorsPick', and 'isGoodReadsChoice' contribute additional points to the score based on their boolean values, indicating the presence of these respective tags. Specifically, being a 'Best Seller' adds 3 points, an 'Editors' Pick' adds 1.5 points, and a 'Good Reads Choice' adds another 1.5 points.

This weighted scoring system is integrated into our search algorithm, allowing us to sort books not only by their relevance to the search query but also by their perceived quality and popularity. And the improvement of search relevance is shown in figure 8.

Finally, we built a front-end search platform based on vue3. Please see the attachment for the source code and demonstration video. This front end includes all the functions mentioned above, and can implement fuzzy search, custom sorting, paging, viewing details and other functions.

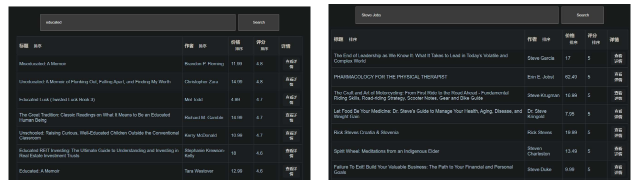


Fig. 6: Only sorting by stars

Fig. 7: Only sorting by relevance

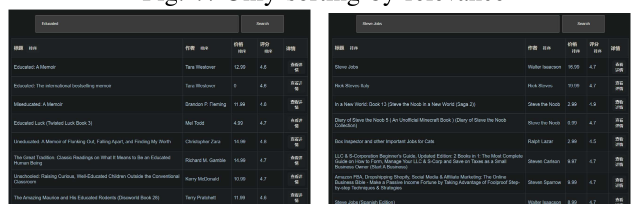


Fig. 8: Sorting by our final methods

IV. Data Analysis

It is a collaboration project. This part is written by Yuanyao Chen (12011226@mail.sustech.edu.cn), intentionally hidden here by Huang Jun.

V. Data Visualization

It is a collaboration project. This part is written by Yuanyao Chen (12011226@mail.sustech.edu.cn), intentionally hidden here by Huang Jun.

Acknowledgment

We extend our deepest gratitude to Assistant Professor Zheng Xiaochen, whose profound academic guidance and incisive critiques have been pivotal in shaping our research.

We also express our heartfelt thanks to our Teaching Assistant, Li Zifan. Your unwavering support and insightful feedback have been invaluable throughout the development of our project.

References

[1] https://www.kaggle.com/datasets/asaniczka/amazon-kindle-books-dataset-2023-130k-books/data