Amazon Review Analytics & Product Recommendations

Team Members

- Daoyu Li - dl5312
- Dongzhe Fan - df2362
- Penghao Weng - pw1298
- Siqi Wan - sw6195
- Xiaochen Lu - xl3139

1 Introduction

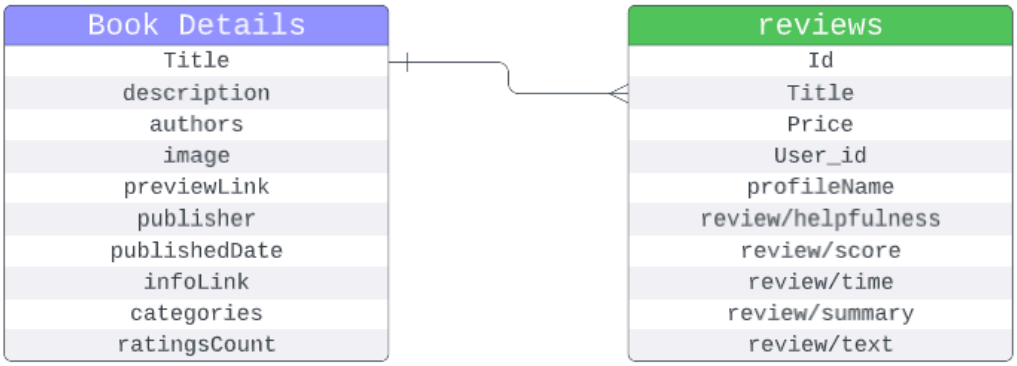
With the development of recent technology, especially large language models, data plays a crucial role in our daily lives, and the power of data cannot be underestimated. At the same time, the pace of people's daily life is getting faster and faster, so few people have enough time to find books they prefer or suitable for them. Meanwhile, the merchants of Amazon struggle with how to recommend the right books to each customer according to their preferences. Therefore, a powerful book recommendation system can help customers or Amazon merchants solve this problem. In the past, due to the limitations of technology, many methods lack the ability to deal with large-scale data, which resulted in recommender systems not being able to accurately discover the preferences of each customer or the potential connections between similar books. Therefore, our project mainly focuses on the following questions:

- How to deal with large-scale data?
- How to train an efficient recommender system that can capture the features of both the customers and the books?

Our project will utilize advanced big data techniques and machine learning algorithms to train a recommender system with a large-scale dataset that can better delineate customer profiles, which can both help people choose the books they are interested in and be a powerful sales tool for merchants on Amazon's website.

Our project will leverage the Amazon Book Reviews dataset, hosted on Kaggle, which encompasses 2.86 GiB of data and spans three million reviews of 212,404 unique books, making it one of the largest publicly accessible collections of its kind. The dataset is structured into two linked tables: 'Book Details' and 'Reviews'. The 'Book Details' table includes fields such as title, description, authors, and publication details, while the 'Reviews' table captures the review ID, title, price, user ID, and detailed review metrics such as helpfulness, score, and text—connected through a foreign key relation with book titles. For clarity, we have attached the object-relational map of the dataset in the image below (sourced from the dataset website on Kaggle). Utilizing collaborative filtering techniques, we will develop a recommendation system to predict user book preferences. The extensive dataset provides a rich base for understanding consumer behaviors and preferences, which is crucial for constructing an effective

recommendation system



2 Technologies

Apache Spark is the main technology adopted in this project. Specifically, we use Pyspark and Spark ML as we write our big data programs in Python.

Since our raw data contains 3 million reviews of 212,404 unique books, we need a distributed computing tool to speed up the computation of large-scale data, and we have seen many of them throughout the semester, such as Hadoop, Spark, and Dask. We finally chose Spark for the following reasons:

- **Familiarity**: Since none of us had prior experience with big data technologies, choosing a tool that is relatively easy to understand and become comfortable with was a sensible approach.
- **Accessibility**: Running distributed computing tasks in the cloud can be costly, but we already had a JupyterLab environment set up for us. Despite some initial crashes, we were eventually able to reproduce our analytics on a medium-sized local subset of the data.
- **Rich APIs and Community**: As part of the Apache ecosystem, PySpark offers a wide range of APIs for data transformation, aggregation, and summarization. It also benefits from a strong community and ecosystem, providing numerous plugins, extensions, extra functionalities, and abundant learning materials, all of which greatly facilitate working with big data.

3 Methodology

3.1 Data Preprocessing

Data preprocessing is the cornerstone of scalability and efficiency. For the Amazon Book Reviews dataset, it misses 19% of the book categories. To build a solid book recommendation system, we first fill in the missing categories by the similarity between different books.

Roughly evaluating the similarity between books by their raw text is subjective and time-consuming. It also lacks a scientific evaluation system for assessing the similarity of raw texts. Hence, we encode the titles and descriptions of books into latent space. In this way, we can easily calculate the Cosine Similarity between each latent variable. Specifically, we utilize the Word2Vec model to encode both the book's title and description into text embeddings. We calculated the weighted cosine similarity between title and description embeddings, and for each book with a missing category, we selected the categories of the top 10 most similar books based on similarity scores. The category with the highest frequency among these was assigned as the category for the book. To calculate the Cosine Similarity, we utilize Pytorch, then we wrap the calculate function as a Pyspark UDF (User-Defined Function) to adapt it to the PySpark framework.

3.2 Alternating Least Squares (ALS)

To generate personalized book recommendations, we choose Collaborative filtering as our approach. It is based on the assumption that people who agreed in the past will agree in the future, and that they will like similar kinds of items as they liked in the past. The system predicts the rating and generates recommendations using only information about rating history for different users or items.

We employed the Alternating Least Squares (ALS) algorithm, a collaborative filtering technique provided by SparkML. ALS is particularly suitable for handling large-scale, sparse datasets such as the Amazon Book Reviews dataset, where user ratings are only available for a small fraction of the user-book pairs.

3.2.1 Data Preprocessing for ALS

To ensure the dataset was ready for ALS training, we performed the following preprocessing steps:

- **Filtering Users with Insufficient Records:** Retained only users with 10 or more records(reviews).
- **User and Book Indexing:** Encoded raw User IDs and Book IDs into numerical indices to align with the input requirements of the ALS model.
- **Data Integrity:** Removed missing or invalid entries (e.g., NaN values) to maintain data integrity.
- **Train-Test Split:** The dataset was split into training (80%) and testing (20%) sets to evaluate the model's generalization capability.

This preprocessing ensures that the ALS model can effectively learn latent factors representing user preferences and book features.

3.2.2 ALS Model Training

The ALS model was trained on the preprocessed data using explicit user feedback of ratings. We tuned some key hyperparameters to optimize performance:

- Rank: Set to 100 to capture a higher-dimensional latent representation of the sparse dataset.
- Regularization Parameter (regParam): Set to 0.1 to balance model complexity and prevent overfitting.
- Max Iterations: Set to 10 to ensure convergence.
- Cold Start Strategy: Configured to drop NaN predictions to avoid issues when recommending books to users not seen during training.

These parameters were chosen based on the sparsity and scale of the dataset to improve the recommendation quality and computational efficiency.

3.2.3 Evaluation of ALS

The trained ALS model was evaluated on the test set using the Root Mean Squared Error (RMSE), a standard metric for explicit feedback recommendation systems. RMSE measures the difference between the predicted and actual ratings and is a quantitative measure of our model's accuracy.
We obtained an RMSE score of 0.8175, which demonstrates the model's ability to make accurate predictions and highlights its effectiveness in capturing user-book interactions.

3.2.4 Enhancing Recommendations with Hybrid Filtering

To further improve the relevance of recommendations, we combined the ALS results with user preferences for book categories. By calculating the average rating for categories, we identified each user's "top categories" and filtered the recommendations to match these preferences. This hybrid approach balances collaborative filtering with content-based personalization, ensuring the recommendations align with both user behavior and content similarity.

4 Problem Solving in Action

During the course of the project, our team faced several obstacles that required extra effort. Below, we outline the major issues encountered and how we managed to resolve them.

Java Out of Memory Error

One of the initial challenges we faced was a **Java out of memory error** that occurred during PySpark job execution. This error was caused by insufficient memory allocated to the JVM for code execution and processing large datasets. In our case, the PySpark default configurations were not optimized for the size of the dataset we were working with.

Solution:
To resolve this, we adjusted the Spark configuration settings, specifically increasing the executor memory and driver memory using parameters like spark.executor.memory and spark.driver.memory. These

adjustments allowed the Spark jobs to process the data (for scaling up, we might need to further increase the memory capacity).

Handling Complex Data Structures in CSV Columns

Our dataset presented another set of challenges due to its structure. Certain columns, such as the **authors** and **categories** in the Books table, were stored as string representations of arrays (e.g., "['author1', 'author2']"). When read into PySpark as a CSV, these were interpreted as plain strings rather than lists, making further processing cumbersome.

Additionally, the **description** column contained a long paragraph of text, often including commas, which confused the CSV reader despite the field being enclosed in double quotes (""). This led to incorrect parsing of the data.

Solution:

1. For columns like authors and categories, we implemented a transformation step that parsed these strings into proper array structures using PySpark's functions.split and functions.regexp_replace.
2. For the description column, we applied preprocessing mechanisms to isolate the column from the rest of the CSV and manually resolve the parsing issues (by discarding the commas in the description paragraph since the embedding is token-based anyways).

By applying these solutions, we ensured the data was properly ingested and transformed, allowing us to proceed with our analysis effectively.

Sparsity of Data

The original dataset is sparse because many users have very few book reviews. Such sparsity is common in big data scenarios but is difficult for collaborative filtering. Although algorithms like ALS are able to handle sparse data because it is based on matrix factorization, sufficient user-item interactions can help ALS model to learn more meaningful latent factors and have a higher score of accuracy. Additionally, processing such a sparse dataset in a distributed environment like PySpark requires careful consideration of scalability and memory usage.

Solution:

To fix this issue, we implemented a filtering step in PySpark to remove users with insufficient interactions. The steps were as follows:

1. Counting User Records: The number of interactions per user was calculated using groupBy and agg to create a record count for each user.
2. Filtering Sparse Users: Users with fewer than 10 records were filtered out (record_count >= 10), ensuring only users with sufficient data remained.
3. Inner Join with Filtered Data: The filtered user list was joined back with the original dataset to retain only valid records.

The PySpark implementation ensures efficient and distributed processing of the large dataset. After applying the filtering step, the dataset size was reduced from 3000000 to 819551 records, significantly increasing data density and improving the ALS model's ability to generate accurate recommendations.

5 Project Summary

In this project, we build a solid recommendation system on the Amazon Book Reviews dataset. We implement our framework based on Apache Spark to scale up for large-scale data. We preprocess the raw dataset by filling missing categories. We conduct Word2vec model to encode the book's raw text such as the book title and book description into latent space. After that, we calculate the cosine similarity of each word embedding to fill the missing categories. With preprocessed data, we conduct an ALS model to build our recommendation system.

In the future, we plan to align LLMs to our recommendation systems. With the great power of the LLM's in-context learning ability, we can fill the missing categories in a more effective and meaningful way. Besides, by designing proper prompts, we can use an LLM (e.g., GPT-4) to generate dense representations of items by providing their descriptions or metadata and select the best items or rank the item according to the user's preference.

Acknowledgement

We thank Professor Juan Rodriguez for teaching us all the big data tools used in this project.