Distributed Computing Final Project

Guangya Wan, Mia Yuan

December 2023 8th

1 Abstract

This project explores the development of a recommendation system for Amazon products using Pyspark and Graph Neural Networks (GNN). The study focuses on the 'Industrial and Scientific' category of Amazon's review dataset [2][3]. The data preprocessing step includes cleaning, text data processing, outlier detection, and focusing on verified reviews. We employed three recommendation system techniques: Collaborative Filtering using Alternating Least Squares (ALS), Content-based Filtering, and a Graphical Neural Network. The methodologies utilized include matrix factorization for ALS and similarity-based algorithms for content-based filtering, and a Light Graph Convolutional Network (LightGCN), which are implemented in either Pyspark or Pytorch. We found that the ALS method with a customized cold start strategy outperformed other techniques in our test set. Results showed a significant improvement in recommendation precision and recall. The study highlights the effectiveness of ALS in handling sparse datasets and providing accurate recommendations compared to other existing methods in our dataset.

2 Data and Methods

2.1 Background of the Dataset

2.1.1 Overview

Our chosen dataset is an updated version of the Amazon review dataset originally released in 2014 [2][3]. It encompasses a comprehensive collection of reviews, product metadata, and relational links such as "also viewed" and "also bought" graphs. The data set is enriched with reviews that include ratings, text, and helpfulness votes, and product metadata that contains descriptions, category information, price, brand, and image features.

2.1.2 Sample Review Structure

A typical review in the dataset is structured as follows:

```
{
  "image": [URL],
  "overall": 5.0,
  "vote": "2",
  "verified": True,
  "reviewTime": "MM DD, YYYY",
  "reviewerID": "ID",
  "asin": "ProductID",
  "style": {
    "Attribute1": "Value1",
    "Attribute2": "Value2"
  },
  "reviewerName": "Name",
  "reviewText": "Text",
  "summary": "Summary",
  "unixReviewTime": UNIXTIME
}
```

Each review includes the reviewer's ID, the product ID (asin), the reviewer's name, vote count, the style dictionary which holds product metadata, the text of the review, overall rating, summary, review time in raw and UNIX formats, and images posted by users.

2.1.3 Metadata Structure

The product metadata is structured as follows:

```
{
  "asin": "ProductID",
  "title": "Title",
  "feature": ["Feature1", "Feature2", ...],
  "description": "Description",
  "price": Price,
  "imageURL": [URL],
  "imageURLHighRes": [HighResURL],
  "also_buy": ["ProductID1", "ProductID2", ...],
  "also_viewed": ["ProductID1", "ProductID2", ...],
  "salesRank": {"Category": Rank},
  "brand": "Brand",
  "categories": [["Category1", "Subcategory1", ...]]
}
```

This includes the product ID, title, features in a bullet-point format, description, price, image URLs, related products, sales rank information, brand name, and a list of categories the product belongs to.

2.1.4 Scope of the Project

For the purpose of this project, we utilized the review and metadata segments from the 'Industrial and Scientific' category. The review dataset used is approximately 202 MB in size with 60,676 rows, and the metadata part is about 80 MB with 167,442 entries.

2.2 Data preprocessing

This dataset contains information about customer reviews, including columns such as 'asin', 'overall', 'reviewText', 'reviewTime', 'reviewerID', 'reviewerName', 'style', 'summary', 'unixReviewTime', 'verified', 'vote', and 'reviewTextlength'. The goal of this data preprocessing report is to ensure data quality, handle missing values, and gain insights through visualization.

2.2.1 Data Cleaning

- Missing Values: Initially, we conducted a comprehensive analysis to outline the quantities and proportions of missing values within each column. Notably, the 'vote' column exhibited a high percentage of nulls, specifically 88.27%, while the 'style' column had 60.67% null values. Given the nature of our model designs, these two variables were deemed unnecessary for further utilization. Consequently, we opted to eliminate both the 'vote' and 'style' columns, alongside the 'image' column. Subsequently, as the overall percentage of missing values in the entire dataframe was found to be minimal, we chose to remove all rows containing null values.

- Outlier Detection: In this part we canceled the entry where the "reviewText_length" is bigger than 1000 or smaller than 10.

2.2.2 Text Data Processing

- Lowercasing: Convert all text to lowercase to ensure uniformity.

- Tokenization: Split the text into individual tokens.

- Removing stop words: Remove common words that don't carry much meaning.

- Sentiment Analysis: In our data preprocessing pipeline for the Amazon purchase review dataset, we incorporated sentiment analysis to gain insights into the emotional tone of customer reviews. Leveraging PySpark and the Natural Language Toolkit (NLTK), we developed a sentiment analysis function that assigned a numeric sentiment score to each review. This score, captured in the newly created "sentiment_score" column, quantifies the overall sentiment expressed in the review text. The sentiment analysis process involved the utilization of the VADER (Valence Aware Dictionary and sEntiment Reasoner) sentiment intensity analyzer, which

provided a compound score reflecting the positivity or negativity of the text. Positive scores indicate positive sentiments, negative scores indicate negative sentiments, and values close to zero suggest a more neutral tone. By integrating sentiment analysis into our data preparation, we now have a valuable feature that encapsulates the emotional context of customer reviews, setting the stage for deeper analysis and potentially enhancing the performance of future models.

2.2.3 Feature Engineering

- Create new features "reviewText_length" variable: First we introduced the "reviewText_length" column to quantify the length of each review. This new feature offers insights into the review's verbosity and conciseness, enabling us to explore correlations with other variables.
- Convert categorical variables into numerical representations using techniques like one-hot encoding or label encoding.

2.2.4 Exploratory Data Analysis (EDA)

- Piechart of overall review.[1]
- Number of reviews for top 100 items.[2]
- Distribution of Ratings ('overall'): visualized the distribution of ratings using a histogram; identified the most common rating values.[3]
- Distribution of reviewText_Length.[4][5]
- Wordcloud of users'reviews.[6]

2.3 Introduction to Collaborative Filtering ALS

Collaborative Filtering is a type of recommendation system that makes predictions about a user's interests by collecting preferences from many users (collaborating). It assumes that if a user has the same opinion as another user on an item, they will share similar preferences on other items. ALS is a matrix factorization technique used in collaborative filtering. The idea is to factorize the user-item interaction matrix into two lower-rank matrices: one representing users and the other representing items. The factorized matrices are then used to approximate the original matrix, capturing latent features that explain user-item interactions.

2.4 Code Implementation of Collaborative Filtering ALS

In our implementation, we utilized PySpark's ALS class, configuring parameters such as rank, max iterations, and regularization parameters to optimize the model's performance. Additionally, we performed hyperparameter tuning using

grid search and cross-validation to identify the best combination of parameters, enhancing the accuracy of our recommendations. For the cold start problem, we first calculated the popularity of each item, and for those users who only appeared in the test dataset, we recommended the first ten most popular items to them.

2.5 Introduction to Content-Based Filtering

Content-based filtering is a recommendation system approach that focuses on item attributes. It recommends items similar to those a user likes, based on their previous actions or explicit feedback. This method involves creating item profiles, which are based on attributes like text descriptions, categories, or metadata, and user profiles, which are developed from the user's interactions with these items.

2.6 Content-Based Filtering Implementation Overview

Our project implemented a content-based filtering system using PySpark, focusing on industrial and scientific product metadata. This process involved data preparation (cleaning, handling null values), feature engineering (vectorizing and normalizing price data), and creating user and item profiles (based on brand and price). We calculated the similarity between profiles to generate recommendations, excluding items already reviewed. The system's performance was evaluated using metrics like precision, recall, and F1 score, demonstrating its adaptability and scalability in product recommendations.

2.7 Introduction to LightGCN

LightGCN, introduced as a simplified version of Graph Convolutional Networks (GCNs), is specifically designed for the task of recommendation systems [1]. It simplifies the traditional GCN model by removing feature transformation and non-linear activation, focusing instead on the core idea of neighborhood aggregation for user-item interaction. This model operates by efficiently learning user and item embeddings through the propagation of information across the user-item interaction graph. LightGCN has demonstrated notable improvements in recommendation tasks by effectively capturing the collaborative filtering effect inherent in the user-item graph structure [1]. Its simplicity and efficiency make it a compelling choice for large-scale recommendation systems.

2.8 LightGCN Implementation Overview

In this project, we utilized LightGCN as a third approach for our recommendation system, leveraging Pytorch and Pytorch Geometric for implementation. The process encompassed data preparation, feature engineering, and defining the LightGCN model with custom functions. Key steps included cleaning and

transforming product metadata, generating user and item profiles, and developing the model with specialized functions for data handling, loss computation, and performance evaluation. The model's effectiveness was gauged using metrics like precision, recall, and F1 score, showcasing LightGCN's adeptness at providing scalable and personalized recommendations in a specific product domain.

3 Results

3.1 Metrics

In our study, we employed several metrics to evaluate the performance of our recommendation systems. These include:

- **Root Mean Square Error (RMSE)**: Used to measure the accuracy of predictions made by the model (ALS only, since the other models does not support regression output), calculated both for training and test datasets.
- **Precision@10**: This metric assesses the accuracy of the top 10 recommendations given by the model.
- **Recall@10**: Evaluates how many of the actual relevant items are captured in the top 10 recommendations.
- **F-1 Score@10**: A harmonic mean of precision and recall at 10, providing a balance between the two metrics for the top 10 recommendations.

These metrics were crucial in determining the effectiveness of our recommendation systems and guiding our choice of the most suitable model.

3.2 Collaborative Filtering ALS results

- Base Model: Precision@10 = 0.004285, Recall@10 = 0.04261, F-1@10 = 0.007784, RMSE (test) = 3.3120
- Best Model: Precision@10 = 0.004483, Recall@10 = 0.04449, F-1@10 = 0.008140, RMSE (test) = 3.4915

3.3 Content-based Filtering results

- Precision@10 = 0.002069
- Recall@10 = 0.04235
- F-1@10 = 0.003808

3.4 GNN results

- Precision@10 = 0.001250
- Recall@10 = 0.01207
- F-1@10 = 0.002255

4 Conclusions

4.1 Findings

Model	Precision@10	Recall@10	F-1@10	RMSE (test)
Collaborative Filtering ALS (Base)	0.004285	0.04261	0.007784	3.4915
Collaborative Filtering ALS (Best)	0.004483	0.04449	0.008140	3.3120
Content-based Filtering	0.002069	0.04235	0.003808	N/A
GNN	0.001250	0.01207	0.002255	N/A

Table 1: Summary of Results for Recommendation Models

Our study successfully implemented a recommendation system using Pyspark and Pytorch. The Alternating Least Squares (ALS) approach, particularly with a customized cold start strategy, proved to be the most effective. This method demonstrated high precision and recall, outperforming both content-based filtering and LightGCN approaches in our test dataset.

4.2 Limitations

The primary limitations of our study included limited data usage, the simplicity of the chosen architectures, and the prevalence of cold starts in the test set. These aspects may have influenced the overall performance and generalizability of our findings.

4.3 Future Work

For future enhancements, we plan to expand the scope of our algorithm to include a larger dataset, incorporate missing data imputation techniques, explore more sophisticated architectures to include side features besides user-item interaction. Additional sensitivity analysis will also be a focus.

References

[1] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. *LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation.* arXiv:2002.02126, 2020.

[2] Chenwei Cai, Ruining He, Julian McAuley. *SPMC: Socially-aware personalized Markov chains for sparse sequential recommendation.* International Joint Conference on Artificial Intelligence (IJCAI), 2017.

[3] Tong Zhao, Julian McAuley, Irwin King. *Improving latent factor models via personalized feature projection for one-class recommendation.* Conference on Information and Knowledge Management (CIKM), 2015.

A Appendix

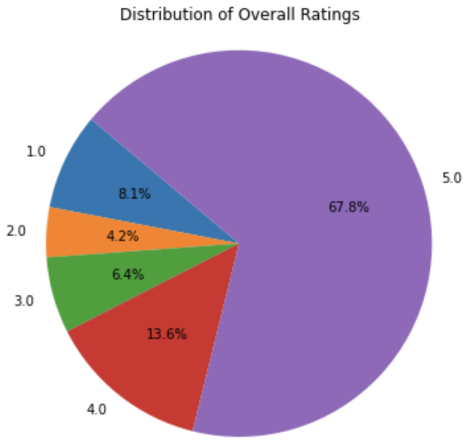

Figure 1: Piechart of ratings

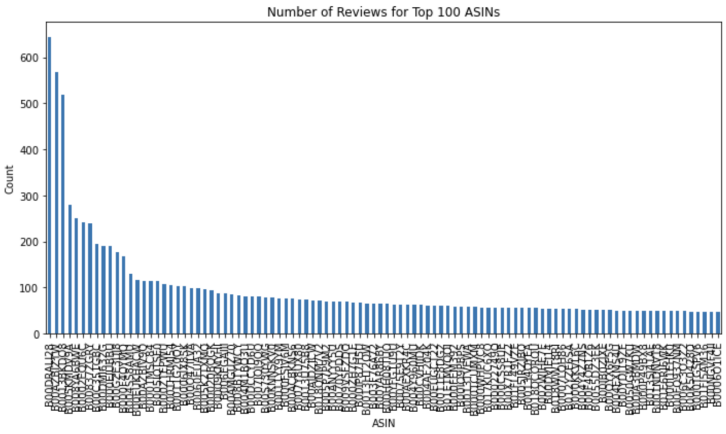

Figure 2: Reviews numbers of items

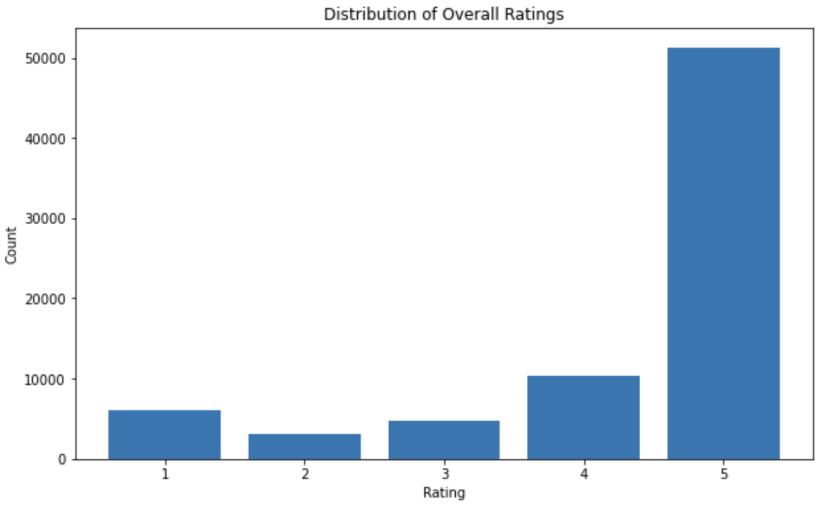

Figure 3: Bar chart of ratings

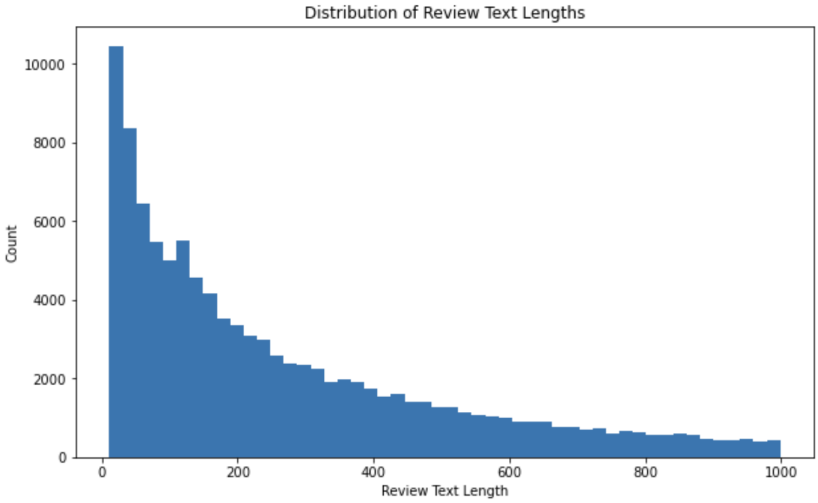

Figure 4: Text length of unverified users

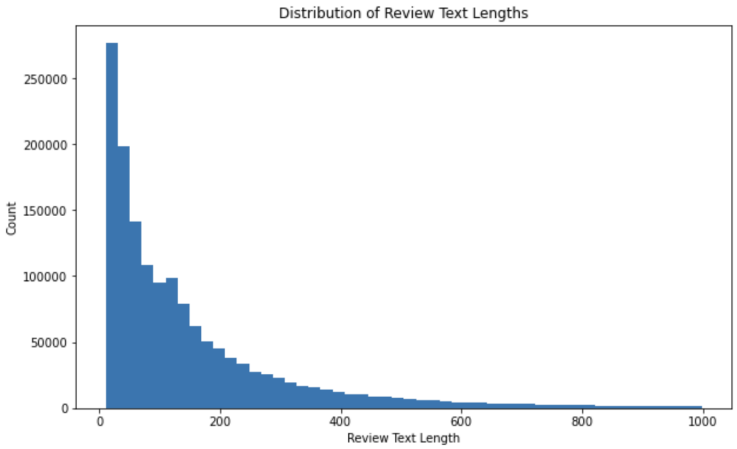

Figure 5: Text length of verified users

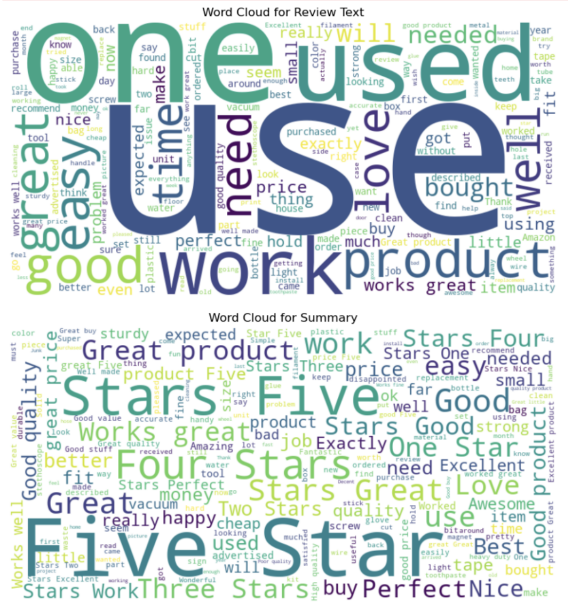

Figure 6: Wordclouds of text data