Recommendation System on Explicit Feedback

DS-GA 1004 Big Data Final Project

Hong Gong
New York University
hg1153@nyu.edu

Yuwei Wang
New York University
yw1854@nyu.edu

Yi Xu
New York University
yx2090@nyu.edu

1 Introduction

The common algorithms of recommendation system includes collaborative filtering, content-based filtering, knowledge-based systems and etc. Collaborative filtering assumes the consistency in human behaviors - People will like similar kind of items they liked in the past. And the system generates recommendations by using rating information on items. It is capable of accurately recommending complex items without requiring an "understanding" of the item itself. In this project, we conduct research on collaborative filtering approach with latent factor decomposition, a matrix factorization approach to decompose user-item interaction information to recommend users' most relevant items, with Alternative Least Square method. We utilize PySpark (version 2.4.0) and implement baseline model on HPC environment. We further implement model on single-machine to compare its performance from HPC settings, and conduct various visualization to detect the hidden correlations between latent factors and other features.

2 Data

2.1 GoodReads Datasets

For the baseline recommendation model construction, the dataset we used is a user-book interaction from *Goodreads* containing over 223M user-book interactions, 876K users and 2.4M books. The dataset has five columns: 'user_id', 'book_id', 'is_read', 'rating' , and 'is_reviewed', where 'is_read', and 'is_reviewed' are binary, and 'rating' is a numerical value ranging from 1 to 5. Each row in the dataset represents one interaction between the user and book.

For the visualization extension, we also used the book genre dataset to further explore the relationship between book genres and ratings.

2.2 Data Processing

Out of the consideration of efficiency, we converted the dataset format from csv to parquet as the first step, then completed the entire data processing procedures. Due to the limit of computing resource, we downsampled dataset into 1%, 5% and 25% apart from the full dataset. The procedures described apply to all sizes of data.

We filtered out all interactions with zero 'rating' since they do not contain information for recommendation. We also only kept users with more than 10 non-zero rating interactions as valid users for data splitting. We randomly split 60% users from unique valid users pool for training set, 20% users for validation set and the rest 20% users for test set. All interactions from 60% training users are kept for the training set, while half interactions from validation users and test users are sent to the training set. In order to achieve this, we fell the parquet dataset back to RDD and used *zipWithIndex* to add the indexes to each interaction and then used *toDF* to get back to a dataframe[1]. Interactions with even indexes are selected to form the training set. And the interactions with odd indexes are saved as validation and test sets correspondingly.

3 Model Implementation

The two main library used was `pyspark.ml`, which contains various recommendation functions based on Spark DataFrame and `pyspark.mllib` which is oriented for RDDs. `pyspark.mllib.evaluation` contains more evaluation metrics including *P@k* and *MAP* which were not available in `pyspark.ml`. Since our original dataset was in data frame format, we adopted `pyspark.ml` to fit recommender model, and used `pyspark.mllib` for evaluation.

For recommendation model fitting, we used Alternating Least Square (ALS) algorithm which was built in PySpark library. It factors the utility matrix into user-to-item matrix U and item-to-user matrix V. The ALS algorithm uncovers the latent factors that explain the observed users to item ratings and tries to find optimal factor weights to minimize the least squares between predicted and actual ratings. It's also designed to run in parallel fashions which significantly contributes to the efficiency of training.

4 Evaluation Metrics

We have tried to adopt all common evaluation metrics for recommender systems, in order to present an overall performance measurement. Here is a brief list and interpretation of metrics used: [2]

[1] Adapted code from https://towardsdatascience.com/adding-sequential-ids-to-a-spark-dataframe-fa0df5566ff6

[2] For detailed explanation, check: https://spark.apache.org/docs/latest/mllib-evaluation-metrics.htmlranking-systems

- **RMSE**: Rooted mean square error between predicted and ground truth rating, of all possible user-item interactions.
- **Precision at k**: Number of the top k recommended items which are in the set of true relevant item sets, averaged across all users (the order of recommendation is not relevant).
- **MAP**: Mean Average Precision, number of all recommended items which are in the set of true relevant item sets, averaged across all users. Penalty is assigned according with order.
- **NDCG**: Normalized Discounted Cumulative Gain, number of the top k recommended items which are in the set of true relevant item sets, averaged across all users, and taken into account of orders

In addition to metrics above, we also recorded time for fitting model to evaluate efficiency. We report RMSE for all recommendation outcomes. For ranking metrics, we filtered top k = 500 recommended items for evaluation.

5 Results

5.1 Hyperparameter tuning

We used grid search to tune 3 hyperparameters: max iterations, regularization parameter, and ranks of latent factor matrices. For evaluation metrics, RMSE function is within the same `pyspark.ml`; other ranking metrics needs to be transformed into a top 500 dictionary before computing by `pyspark.ml.evaluation`. The script of transferring into dictionaries was written by our own and turned out not performing efficiently. Due to limited computing capacity, the ranking metrics was only measured on 1% dataset. For other sizes of data sets (5%, 25%, 100%) we used RMSE for hyperparameter tuning.

5.1.1 ***Ranking Metrics on 1% dataset*** Throughout our exploration, we find out that the scores of all three ranking metrics will improve along with the increase of the rank for latent factor. We extensively tune the rank parameter for different models, and settled on the best score and runtime tradeoff at rank = 160. With a rank bigger than 160, the ranking metrics improves slowly and randomly, while the runtime would explode and appear obviously not feasible for any bigger datasets. Ranking metrics score also improves with a smaller regularization parameter, but a smaller regularization parameter (from 0.01 to 0.001) will hugely increase the runtime (1364s to 7322s). As for maximum iteration, the change of maximum iteration does not have too much impact on the ranking scores.

We plot the performance along with hyperparameters in figure 1 below for clearer demonstration:

Table 1: Tuned hyperparameters on Ranking Metrics

Dataset	Rank	Max Iter	Reg param
1%	160	10	0.01

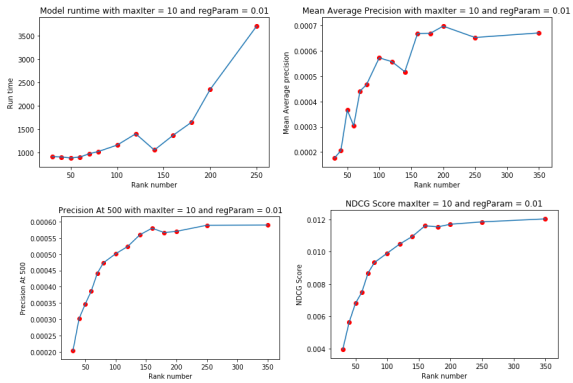

Figure 1: MAP, Precision at k, NDCGs and time on various hyperparameters, tuned on 1% dataset

5.1.2 ***RMSE and Time on larger datasets*** Here we present a sample hyperparameter tuning on 5% dataset as follows. The scores indicate a local minimum of RMSE with changing of ranks and regularization parameters (see the left graph in figure 2 and 4). For maximum iteration, the scores decreases monotonically, but shows a pattern of converging (check the left graph of figure 3). Thus, we settled hyperparameters on the local minimum of ranks and reg parameter, and the best trade-off point between RMSE and time for maximum iteration.

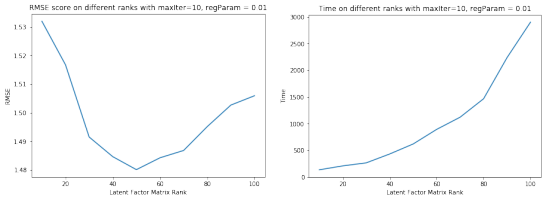

Figure 2: RMSE and Time on various ranks, with Reg=0.01 and maxIter = 10

Table 2: Tuned hyperparameters to use on various data sets

Dataset	Rank	Max Iter	Reg param
5%	50	10	0.1
25%	34	10	0.1
100%	20	10	0.1

Larger sizes of datasets also show similar patterns of performance, so we skipped performance analysis for larger

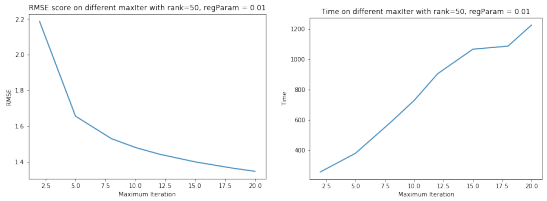

Figure 3: RMSE and Time on various maxIters, with Reg=0.01 and rank = 50

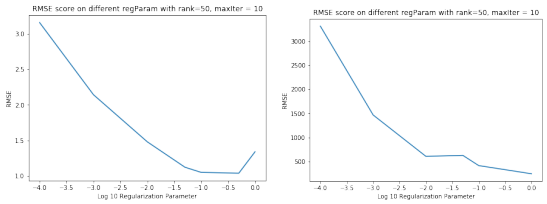

Figure 4: RMSE and Time on various maxIters, with Reg=0.01 and rank = 50

sets to avoid duplication [3]. The full set of optimal hyperparameters used for all sizes of datasets is listed in Table 2.

5.2 Model performance

The validation and test score, using the optimal hyperparameters given above, is recorded in Table 2.

Table 3: Validation and Test Score on all sizes of data

Dataset	Validation RMSE	Validation Time	Test RMSE	Test Time
1%	1.89606	89.85521	1.90396	101.4693
5%	1.48014	620.47685	1.48467	631.5624
25%	0.89479	704.32922	0.89602	902.6629
100%	0.82806	1083.9011	0.82818	962.57043

6 Single Machine Implementation

In this section, we introduce `lightfm` package for recommendation system implementation in Python and compare our experiment results. We also utilized `pandas` and `scipy` in intermediate steps for data structure transformation.

6.1 Implementation with LightFM package

`lightfm` [4] is a package designed for various types of recommender systems. It covers functions for both implicit and explicit feedback models, and can be applied to collaborative filtering models, hybrid models as well as cold-start recommendations.

In this project we only explored the collaborative filtering part of the `lightfm` package. A summary of implementation steps is listed as follows:

- Input dataframe of interactions, convert to initial utility matrix with `pandas.pivot_table`, then convert to coordinate format with `scipy.sparse.crs_matrix` which is the acceptable format for fitting `lightfm` models.
- Training and test split with built-in function `lightfm.cross_validation` [5]. We set split ratio as 0.8/0.2 and split into train and test set, corresponding with data splitting method for Spark ALS model.
- Fit dataset into a lightfm instance, with corresponding hyperparameters selected from Section 5. We conducted experiments on both Weighted Approximate Rank Pairwise Loss and Bayesian Personalised Ranking for better performance comparison.
- Get test score of precision at k = 500. The total time consumed for fitting and testing model was also recorded in a combination of last two steps.

The complete Python script of lightfm implementation is provided within the same repository.

6.2 Performance comparison with Spark's ALS model

We ran lightfm model (both warp and bpr) on 1% dataset, with the same hyperparameter combinations as in Spark ALS. Some instant observations are:

- Spark ALS and LightFM produces roughly same magnitude of ranking metrics and evaluation time, although in different patterns. There's only minor difference between lightfm warp and bpr methods.
- The optimal hyperparameter combinations for Spark ALS and LightFM do not agree. This may result from different splitting methodologies of original dataset.

Comparisons between lightfm and Spark ALS, on precision at k and time, over ranks and regularization parameter are shown in figure 5.

Comparisons between lightfm warp and bpr models, on precision at k and time, over ranks and regularization parameter are shown in figure 6.

Finally, it's worth mentioning that lightfm has a limitation for large scales of datasets. The transforming step to pandas pivot table and scipy sparse cooMatrix both contain dimension restrictions. Also lightfm only contains limited built-in evaluation metrics. These factors may also impact choice of tools to use for other systems.

[3] For more details, check RMSE_MODEL_RESULT in github repo

[4] For more details, check: https://github.com/lyst/lightfm

[5] Note that lightfm requires training and test utility matrix to be of same dimensions, i.e. same lists of user and item sets, in order to perform evaluations, so we didn't follow the original splitting methods for Spark ALS, but randomly split the sparse utility matrix generated from step 1

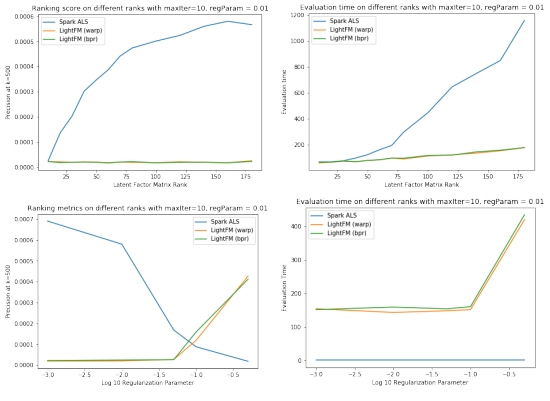

Figure 5: Comparisons between lightfm and ALS

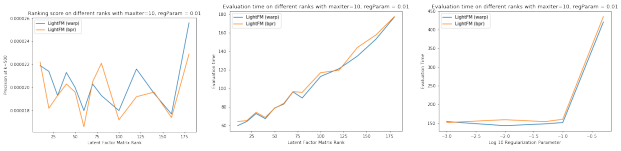

Figure 6: Comparisons between lightfm warp and bpr models

7 Visualization

We extended our project to visualize the potential patterns may existing in the item and user latent factors. More specifically, we exploited the latent factors from the optimal models with rank = 160, maximum iteration = 10 and regularization parameter = 0.01 on 1% dataset. we performed PCA, t-SNE and UMAP on our item and user latent factors to reduce dimensions and conduct visualizations.

7.1 Rating VS Book Latent Factors

The item latent factors extracted from model include IDs and latent vectors, so we transformed them into the corresponding latent matrix with 160 latent features. We calculated the ceiling average rating for each book and append them to the item latent matrix by matching 'book_id'. We performed Principal Component Analysis on item latent matrix so that each item can be visualized in 2D and 3D. And we color books based on their ceiling average rating. The result of PCA shown that the first PC contains 12.4% information of book, the second and third PC explains 0.96% and 0.74% variations. The graphs are shown in figure 7. In the 2D scatter plot, from left to right along the first PC, the dot color changes from red to purple, showing the increased ceiling ratings. It proves that the first PC contains major information about book rating. And the same pattern can also be observed in 3D plot.

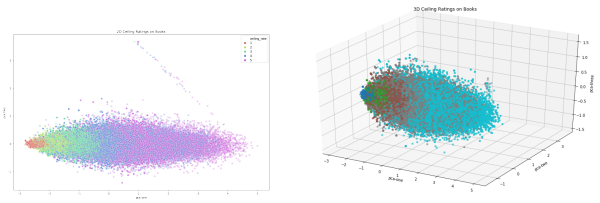

Figure 7: Book Rating in item latent factors

In addition, we also tried t-SNE to represent this relationship (see graph 8). But this time, the orientation of book distribution changes. Unlike PCA, rating decreases from 5 to 1 along the t-SNE two axis, indicating t-SNE two dominantly controls book ratings.

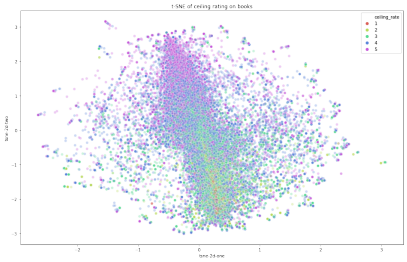

Figure 8: t-SNE of book latent factors

Furthermore, we used UMAP to visualize the relationship between book latent factor and rating ceiling statistics. It turns out that UMAP also exhibits the same patterns among the books with different rating ceilings (see figure 9). It is very clear in the center of the map that there is a gradual change of colors. It seems that for UMAP, with a higher minimal distance, meaning that we do not allow points to pack too close to each other, the clustering pattern is more clear.

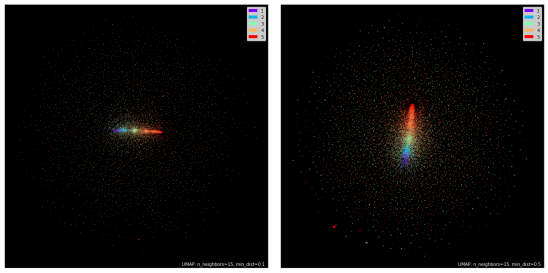

Figure 9: UMAP for book ceiling rating in item latent factors

7.2 Book Genre VS Book Latent Factors

We tried to discover hidden book genre information in item latent factors. Unfortunately, by observing figure 10, neither

PCA nor UMAP displays any obvious patterns for genre information since the points are distributed randomly.

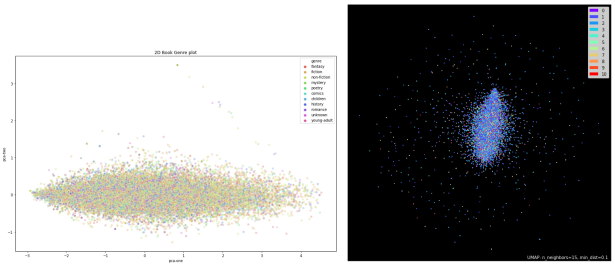

Figure 10: Book genre in item latent factors

7.3 Rating VS User Latent factors

Aside from item latent factor, we explored the relationship between user and rating on user latent factor since we assume people who agree in the past will agree in the future. We implemented PCA on user latent matrix, and colored each users by their ratings. The variances explained by the first, the second and the third PCs are 0.01193%, 0.00847%, and 0.00839%, which are much smaller comparing the PCA results of item matrix. The 2D and 3D representation are shown in graph 11. The first PC contains information about the relationship between ratings and users. And we also performed t-SNE on user latent matrix. The result is shown in the graph 11, which is even worse than PCA representations.

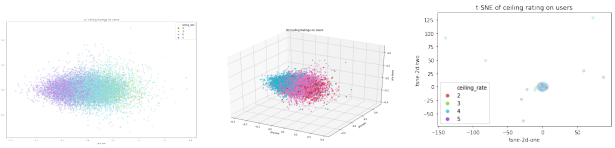

Figure 11: Users latent factors

We concluded from our visualizations that item latent factors do contain certain amount of rating information, but there is no clear relationship or information for books' genres. As for the user latent factors, it keeps some patterns between users and ratings, but less obvious than item latent factors. Part of the reason is that its principal components only explains a very small amount of total variance. Ideally, we should find out a clearer pattern in user latent factor, such as its correlation to user ratings and user preferred genre. Future analysis for ALS latent factor can be conducted by cross-comparing different models with different latent factor ranks or examining the correlation between latent factors and other user/item features.

8 Contribution and collaborations

- **Data processing and splitting**: Yi Xu
- **Baseline model and tuning**: Hong Gong, Yi Xu, Yuwei Wang
- **Single Machine Implementation:** Yuwei Wang
- **EDA / Latent Factor Exploration:** Yi Xu, Hong Gong
- **Final Report:** Yi Xu, Hong Gong, Yuwei Wang