Recommender Systems using Latent Factor Models: A Case Study with Goodreads Books Dataset

Kuan-Lin Liu
Center of Data Science
New York University
New York, United States
kll482@nyu.edu

Duo Jiang
Center of Data Science
New York University
New York, United States
dj1057@nyu.edu

Zichange Ye
Center of Data Science
New York University
New York, United States
zy1545@nyu.edu

TABLE I: Dataset Schema

Column	Type	Description
user_id	Integer	An index represents the user
book_id	Integer	An index represents the book
is_read	Integer	Whether the user reads the book
rating	Integer	The rating of the book given by the user
is_reviewed	Integer	Whether the user provides a review

***Abstract*—In this project, we utilized collaborative filtering to build a recommender system on the Goodreads dataset. In particular, we used Alternating Least Square (ALS) and Spark to train our models, and attained RMSE of 1.6908, PrecisionAt of 0.770 on the test set. Furthermore, we implemented fast searching that increased the speed of the query of 10 users from 86 seconds to 0.33 seconds. We also explored the visualization of the learned space in this paper.**

***Index Terms*—Latent Factor Model, Alternative Least Square, PySpark, Goodreads**

I. Implementation

The description of the implementation is divided into two parts: (1) data preprocessing; (2) modeling. All work is done as Spark jobs on on Dumbo, the Hadoop cluster at New York University as the original dataset is large (4.1GB). Besides, we do fast-queries and build Latent Factor Model in PySpark, which is the Python API for Spark. The whole data science process is generally separated into three stages from data pre-processing to model evaluation. The code for this project is available online[1].

A. Data Preparation

The [3] [4] provides the goodreads_interactions dataset in a csv format. The schema and the content of the dataset is shown in Table I and II. We first transformed the dataset into parquet formats, as we expect that the Parquet format helps to improve the efficiency of Spark SQL queries and compression.

Downsampling allows fast prototyping and less sparse dataset. Our first sampling decision was to sample 1%, 10% and 100% from the entire data, as smaller data allowed testing on local machines. We saved these samples on the HDFS for future queries. Our second decision was to focus on the users with at least 500 interactions, which is the default given by the project instruction. Users with less items are considered to have low interactions in the user-item matrix, thus are less likely to be informative for the Latent Factor Model. By filtering out user-item pair with less than 500 interactions, we kept those users who have more interactions on the Goodreads website and decreased the sparsity of the data.

B. Modeling with Latent Factor Model

[2] introduces the latent factor model which is a model-based collaborative filtering with matrix factorization techniques. [1] [5] and Spark's ML library uses the alternative least squares (ALS) method to find the latent factors. We splitted the training, validation, and testing set as following. 60%, 20%, and 20% of all user-items interaction pairs are assigned to training, validation, and testing sets respectively. Then, for each user in the validation and testing set, we chose half of their interactions, and moved them to the training set (namely, *stratified* sampling). As a result, the training set will cover all `user_id`, which is a requirement for the Latent Factor Models[2]. In the latent factor model, there are two hyper-parameters, `rank` and λ `regularization`, which basically control the complexity of the model. We use the validation set to find the best hyper-parameter configuration and then obtain the evaluation result from the test set.

II. Evaluation Metrics

We used precision at k (PrecisionAt) and root mean square error (RMSE) which respectively represent the ranking metrics and the regression metrics.

We chose precision at k because in deployment, only the performance of the top-ranked recommended items would be taken into account. Therefore, after fitting the model on the data set, we sorted the `book_id` column by the rating for each user and kept the top 500 books. The equation of PrecisionAt is shown below:

$$p(k) = \frac{1}{M} \sum_{i=0}^{M-1} \frac{1}{k} \sum_{j=0}^{min(|D|,k)-1} rel_{D_i}(R_i(j)) \qquad (1)$$

[1]https://github.com/GaryLKL/Goodreads-Recommendation-Systems

[2]We also implemented and tested a cross-validation to tune the hyper-parameters on 1% on our dataset. The testing took approximately sixteen minutes for a four-fold cross validation, and gave a warning of the shortage of the memory. We decided to not to proceed this strategy on the entire dataset due to concerns about computational resources.

TABLE II: A Preview of Goodreads_interactions.csv

user_id	book_id	is_read	rating	is_reviewed
424628	148	1	5	0
429176	148	0	0	0
122969	148	1	4	0
131460	148	0	0	0
280618	148	0	0	0

, where M is the number of the total users. D is a set of N ground truth documents. k is the number of top books. R is a list of recommended documents. $rel_D(r)$ is 1 when $r \in D$ and 0, otherwise.

Besides, we used the RMSE instead of MSE to evaluate the regression fit because it is on the same scale of the scores and thus more interpretable (Dua et al. 2016). The equation of RMSE is:

$$RMSE = \sqrt{\frac{\sum_{i=0}^{N-1}(y_i - \hat{y_i})^2}{N}} \tag{2}$$

N is the number of books for each user. y_i is the rating score and $\hat{y_i}$ is the predicted rating score.

III. Evaluation Results

We discussed the choices of evaluation metrics, PrecisionAt and RMSE, and hyper-parameter tuning with respect to 1%, 10%, and all of the users. A wide range of ranks and λ regularization are selected for grid search and model tuning.

A. Rank

- A list of ranks, 5, 10, 20, 30, and 50 are chosen.

B. λ Regularization

- A list of λ value, 0.001, 0.01, 0,1, and 1, are chosen to train the ALS model.

C. Prediction Result

TABLE III: Evaluation Results.

Data	Rank	λ	Eval. Metrics	Values on Vali.	Values on Test
1%	50	0.1	PrecisionAt	Near 0.68	0.7125
1%	50	0.1	RMSE	[1.8, 2.5]	1.8402
10%	50	0.1	PrecisionAt	Near 0.763	0.770
10%	50	0.1	RMSE	[1.70, 2.11]	1.6908

- *1% of the dataset & PrecisionAt:* The validation results of PrecisionAt are around 0.68 and very close to each other. The best number of rank is 50 and the λ term is 0.1. The PrecisionAt result is 0.7125.
- *1% of the dataset & RMSE:* The validation results of RMSE are in a range from 1.8 to 2.5. The best number of rank is 50 and the λ term is 0.1. The RMSE result is 1.8402.
- *10% of the dataset & PrecisionAt:*
- *10% of the dataset & RMSE:*

TABLE IV: The Comparison of Query Time for 10 users, by Approaches

Methods	Time (Seconds)
Naive Approach	86.10
Spark Approach	197.42
Annoy Approach	0.33

IV. Extensions

1) Fast-search: Query time is crucial in building a recommender system. Therefore, we compared query time of three methods to associate 500 books with each user.

- **Naive Approach.** Bring user latent factors and item latent factors into memory of driver machine and use heapq(priority queue) to find the top 500 books for each user.
- **Spark Approach.** Since computing dot product between user latent factor and item latent factor is parallelizable, the second method take advantages of spark to do the computation.
- **Annoy Approach.** For each user, the best books to recommend are the books that are the nearest neighbors of this user. Therefore, we can build a spatial data structure in the space of all book latent factors after which we can easily find the most nearest neighbors of user at query time. The Annoy library[3] did a good job at implementing it and it reduced the query time a lot.

Comparison of query time can be seen in table IV. The reduction of time achieved by Annoy Library is incredible since it reduced the query time for 10 users from tens of seconds using naive approach to less than a second, a ten-fold improvement in speed. However, it is surprising that spark does not improve query time which can be proven by the fact that naive approach is faster than Spark Approach. This is possibly due to the fact that in spark approach we used a full sorting algorithm which spark may not help a lot while in naive approach we used a priority queue which prevents us from sorting the whole result.

A. Exploration

To visualize and better understand what kind of information item vectors encode, we used T-SNE to project the item factors into 2 dimensions. There can be many kinds information that is encoded, we mainly want to see if we can find some structure of ratings and genre in the item factors.

1) Since T-SNE can be slow if dimension is high, we first used PCA to project to 50 dimensions from original 100 rank. This is followed by applying T-SNE to all items with truncated average rating. The result is in figure 1. Although books with different average ratings overlap with each other in this reduced dimension, it is clear that, in the plot, we can find an axis along which the average rating of books increase from 1 to 5 especially

[3]https://github.com/spotify/annoy

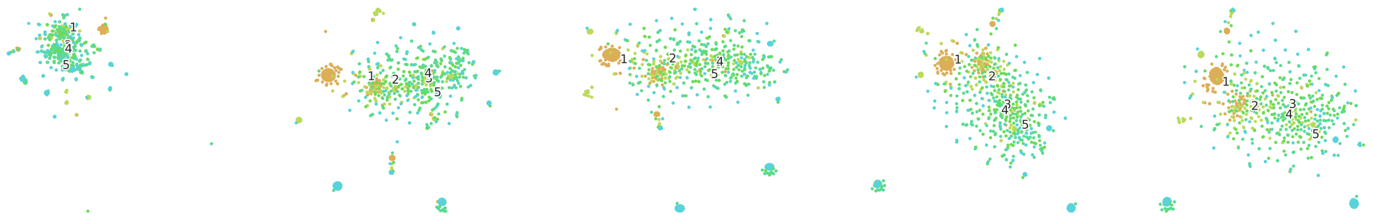

Fig. 1: T-SNE Plots of item factors colored by average ratings and with perplexity 5,15,25,35,45 from left to right

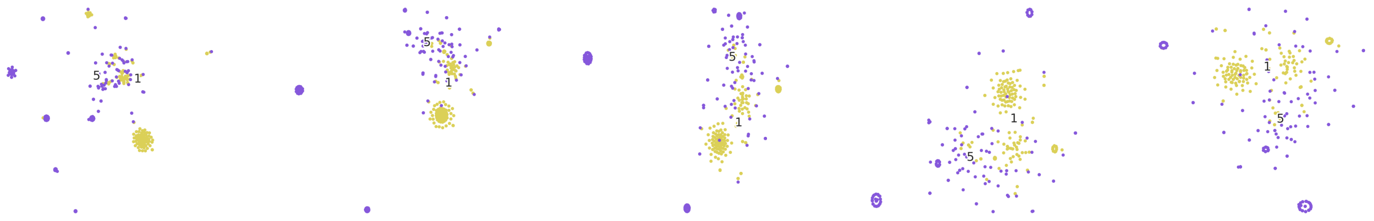

Fig. 2: T-SNE Plots of item factors colored by rating 1, 5 and with perplexity 5,15,25,35,45 from left to right

in plots with perplexities 35 and 45. To further see if books with different quality(ratings) are separated in this reduced space, we plot only books with average rating 1 or 5 using T-SNE in figure 2. Although there is no single line separating books with rating 1 and 5, we do think there is some structure about the rating that is captured by this 2 dimensional graph. Also, it seems that books with rating 5 are more diverse in these graphs compared with books with rating 1. One possible reason for that is there are more ratings of 5 than ratings of 1 which means that the cluster for ratings of 1 is inherently more diverse.

2) The second feature we are interested is genre and its effects in the reduced dimensions. The way we extracted genre is by taking the most frequent genre in gooreads-book-genres-initial.json.gz from UCSD Book Graph[4]. We failed to find any structure when we plot all genres in a single two dimensional figure. Therefore, we then tried to plot two features at a time in the figure and tried to find structure. In some of the pair of features, there is no clear pattern we can find as in figure 3. In other plots, more clear pattern can be found as in figure 4. A natural problem arises: is the pattern in the plot statistically important or not. Given that we have so many pairwise plots, we are suspicious because it can be a consequence of luck. In a word, we refuse to make conclusion about whether these two dimensional graphs encode genre information.

[4]https://sites.google.com/eng.ucsd.edu/ucsdbookgraph/home

V. Future Work

1) Due to the memory limitation of the cluster, we are not able to test a great range of hyper-parameters on the full dataset. We expect to revise our codes to improve the modeling efficiency so that we can tune on all of the users in the near future.
2) Although T-SNE has been widely used in visualization of recommendation systems, other dimension reduction techniques deserve to be explored in the future work.

VI. Contributions

Kuan-Lin Liu finished the basic framework of the recommender system. Duo Jiang is responsible for debugging the system, implementing the extensions, and write up. Zichang Ye worked on the models training, the extensions and write up.

References

[1] Yehuda Koren. Factorization meets the neighborhood: A multifaceted collaborative filtering model. pages 426–434, 08 2008.

[2] Claude Sammut and Geoffrey I. Webb, editors. *Latent Factor Models and Matrix Factorizations*, pages 571–571. Springer US, Boston, MA, 2010.

[3] Mengting Wan and Julian J. McAuley. Item recommendation on monotonic behavior chains. In Sole Pera, Michael D. Ekstrand, Xavier Amatriain, and John O'Donovan, editors, *Proceedings of the 12th ACM Conference on Recommender Systems, RecSys 2018, Vancouver, BC, Canada, October 2-7, 2018*, pages 86–94. ACM, 2018.

[4] Mengting Wan, Rishabh Misra, Ndapa Nakashole, and Julian J. McAuley. Fine-grained spoiler detection from large-scale review corpora. In Anna Korhonen, David R. Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 2605–2610. Association for Computational Linguistics, 2019.

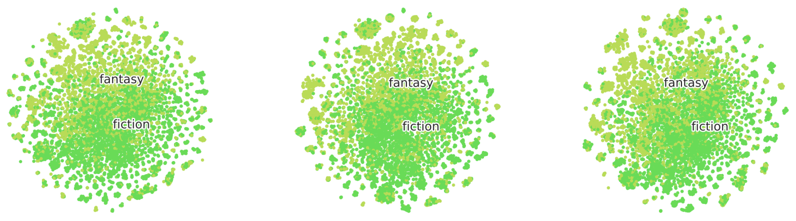

Fig. 3: T-SNE Plots of item factors colored genres fantasy and fiction with perplexity 20, 30, 40 from left to right

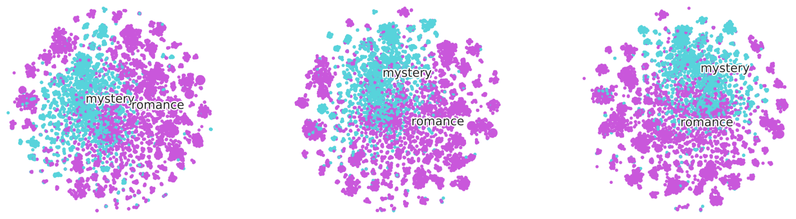

Fig. 4: T-SNE Plots of item factors colored genres mystery and romance with perplexity 20, 30, 40 from left to right

[5] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. Large-scale parallel collaborative filtering for the netflix prize. In *Proc. 4th Int'l Conf. Algorithmic Aspects in Information and Management, LNCS 5034*, pages 337–348. Springer, 2008.