Book Recommendation with Goodreads Interactions Histories

Xinmeng Li Yan Li
Center for data science, New York University

Abstract

In this project, our team built a recommender system using the Goodreads dataset. We chose the spark implementation of the alternating least squares (ALS) method and its default values as a baseline model and tried to tune the hyperparameters including rank (dimension) of the latent factors, as well as the regularization parameter lambda to find the best evaluation score on the validation set. We also used UMAP to visualize the high-dimensional item factor distribution in 2D to discover the book genres pattern.

1. Introduction

Collaborative filter models and content-based models are two widely used approaches to build a recommender system. In this project, we train the model by alternating least squares (ALS) and evaluate its performance with the ranking metrics including mAP, NDCG, and precisionAtk. The big utility matrix R can be written as a product of two matrices, one is a user factor matrix, and another is an item factor matrix. The ALS method is to use these two component matrices to recreate the sparse R, in which each row represents all users' ratings for a book and each column represents the book rating histories for a user. The users' ratings are explicit feedback, which helps the model to know if this user likes this item or not. Once we train the model and have a user factor matrix and item factor matrix, we can predict the users' ratings for all books. We can then recommend books based on the ranking of predicted ratings. The spark implementation of ALS allows distributed processing across a cluster and thus improves the computation efficiency.

2. Implementation

Dataset We are provided three .csv files. Goodreads_interactions.csv contains 228,648,342 interactions, each interaction has five features: user_id, book_id, is_read, rating, is_reviewed. For the basic recommender system, we only need columns user_id, book_id, and rating. User_id_map.csv and book_id_map have the mapping of the original user_id/book_id in the dataset downloaded directly from the website, and their corresponding user_id/book_id we used in goodreads_interactions.csv file.

Data preprocessing We noticed if we ignore is_read and is_reviewed, there are a lot of useless ratings in the dataset (rating = 0, meaning this user did not rate this item). After deleting rating = 0 rows, we are left with 104,551,549 interactions, which is about 45.7% of the original dataset. It is difficult for the model to infer user preferences from a few interactions. Therefore, to alleviate this problem, we deleted users who had less than ten interactions.

We found that there are 694,495 unique users and based on users, we split this dataset into three separate datasets: 1. training set, which has all users, but 60% users with all interactions, 40% users with half of their interactions; 2. validation set: 20% users with another half of their interactions; 3. Test set: same as validation set, the rest 20% users with another half of interactions. We choose to split the dataset in this way instead of randomly selecting from all of the interactions to avoid the cold-start problem that a new user in the validation set has no interactions to infer in the training set. We implemented the train-validation-test split by first select 1%, 5% or more users from the shuffled user list, then add indexing to the two small datasets, use the even index rows of these two datasets to form the validation and test set respective-

ly, and concatenate the rest of the rows with odd indices into the training set. Finally, we add indexing to the training, validation, and test set to make the subsequent computation easier and faster.

We saved all three datasets into parquet format to read and process faster in the later steps. For easier test and evaluation, we also build smaller training/validation/test sets using 1% and 5% of the total number of users. Considering that the user who gives rating under three is very likely to be dissatisfied with the item, our goal is to recommend items that have high satisfaction, we make a hypothesis that drop interactions with ratings lower than three will build a better recommender system. Therefore, we also create a dataset containing only interactions with ratings greater than or equal to three with %1 of the original users and evaluate its performance to verify our hypothesis.

Data model We used the spark implementation of ALS method (pyspark.ml.recommendation.ALS[1]) and its default values as a baseline model and tried to tune the hyperparameters. For both of the 1% and 5% dataset, we first do a grid search on the (rank=10,20, lambda=0.01,0.1,1), and then explore more configurations based on the optimal setting in the grid search result. We also attempt to fit the model on the whole dataset. Due to the computation power limit, we only succeed once. Although it is impractical for us to tune the hyperparameter on the full dataset, the performance of the model with the default setting still can provide us some insights.

3. Evaluation

Since our implementation is based on Pyspark, we use the ranking system metrics including Mean Average Precision, Normalized Discounted Cumulative Gain, and Precision at k provided by the pyspark.mllib.evaluation module[2]. We select the optimal hyperparameter setting based on these three scores on the top 500 recommended books for each user in the validation set.

Precision at k $p(k) = \frac{1}{M}\sum_{i=0}^{M-1}\frac{1}{k}\sum_{j=0}^{\min(Q_i,k)-1} rel_{D_i}(R_i(j))$ measures how many of the top k recommended books are actually relevant to users on average across users.

Similar to the p(k), MAP $MAP = \frac{1}{M}\sum_{i=0}^{M-1}\frac{1}{N_i}\sum_{j=0}^{Q_i-1}\frac{rel_{D_i}(R_i(j))}{j+1}$ also measures relevance between all of the recommended books and the ground truth on average across users and books. From the formula we observe that unlike p(k), the MAP takes the order of the recommended books into account.

NDCG at k $NDCG(k) = \frac{1}{M}\sum_{i=0}^{M-1}\frac{1}{IDCG(D_i,k)}\sum_{j=0}^{n-1}\frac{rel_{D_i}(R_i(j))}{\log(j+2)}$ measures the relevance between the first k recommended books and the truly relevant books. Unlike p(k), the order of recommendation matters in NDCG.

4. Results

Validation Score

1% DATA				5% DATA		
(rank,lambda)	MAP	NDCG	precision	MAP	NDCG	precision
(10,0.01)	2.358E-05	7.693E-04	2.318E-04	2.681E-06	1.412E-04	3.801E-05
(10,0.1)	1.001E-05	4.867E-04	1.123E-04	1.386E-06	9.606E-05	2.419E-05
(10,1)	6.878E-06	2.287E-04	4.464E-05			

(20,0.01)	4.882E-05	2.287E-03	8.323E-04	1.271E-05	4.088E-04	1.117E-04
(20,0.1)	3.493E-05	8.381E-04	1.987E-04	5.016E-06	2.416E-04	5.328E-05
(20,1)	3.801E-06	1.759E-04	4.176E-05			
(50,0.01)	3.547E-04	9.611E-03	3.070E-03	8.671E-05	2.155E-03	6.698E-04
(100,0.01)	9.791E-04	2.052E-02	5.485E-03			
(200,0.01)	2.502E-03	4.119E-02	8.858E-03			
(100,0.01) rating>=3	1.327E-03	2.112E-02	4.571E-03			
(100,0.01) with max_iter=20	2.373E-03	3.763E-02	8.102E-03			
(200,0.01) with max_iter=20	5.556E-03	6.108E-02	1.150E-02			

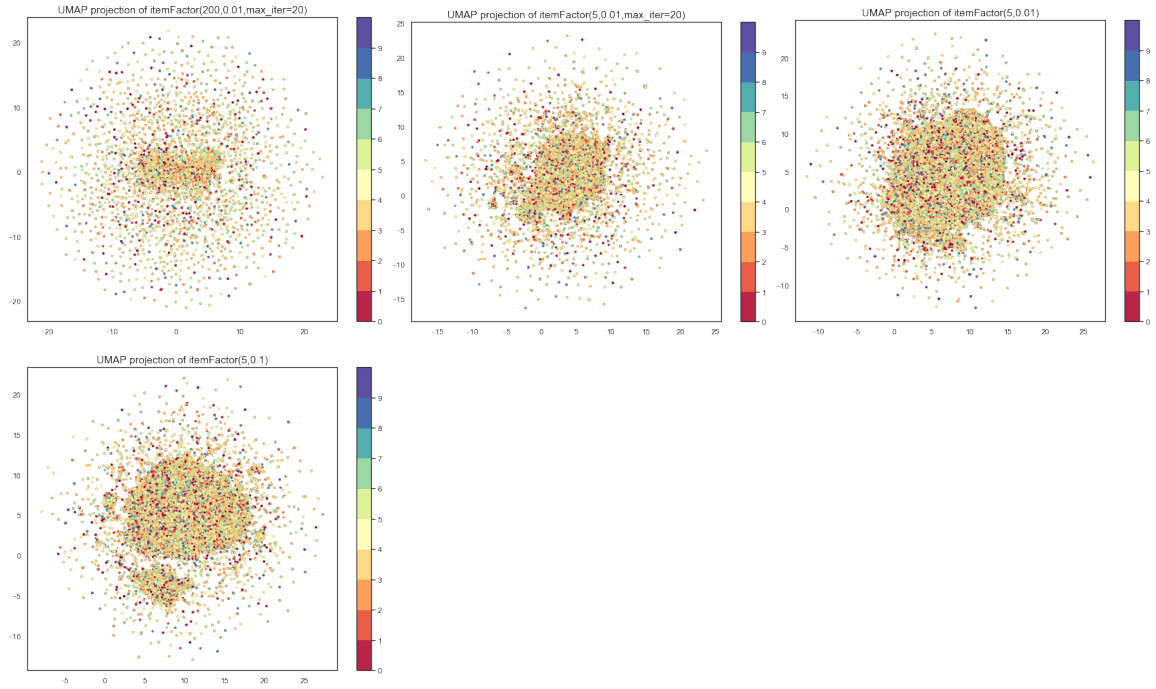
We first do a grid search with (rank=10,20, lambda=0.01,0.1,1) on the 1% dataset and observe that the performance improves as rank increases or lambda decreases. Consequently, with a relatively small lambda, we increase the rank dimension. It turns out that we reach the memory limit of dumbo when the rank is greater than 200. Considering that the convergence might slow down with a large rank and small lambda, we increase the max_iter to 20 for (rank=100,200, lambda=0.01). It turns out that as iteration maximum increases, the three scores increase, which is the same as our expectation. Consequently, the optimal setting is rank=200, lambda=0.01, max_iter = 20 for 1% of the data. As mentioned in the data preprocessing section, we also run the model with (100,0.01) on the 1% dataset containing solely the interactions with ratings greater than or equal to 3. It turns out that all of the three scores are very close no matter whether we filter out the low rating interactions or not. Since we are not using the implicit preference, we did not tune the alpha parameter.

We then do a grid search with (rank=10,20, lambda=0.01,0.1) on the 5% dataset. Within our expectation, the scores increase as rank increases or lambda decreases. According to this pattern, we scale up the rank and we run out of memory when it reaches 50. Therefore, the best model configuration for 5% of the data is rank=50 and lambda = 0.01. Due to the computation resources limit, we can only fit the whole dataset on the model once. For the 100% of the dataset, with the setting (10,0.1), we get 5.122E-06, 1.337E-05, 1.606E-06 for MAP, NDCG and precision respectively.

Test Score The optimal setting on 1% data is (200, 0.01) with max_iter=20. The scores on the test set are MAP: 0.00583, NDCG: 0.0607 precision at 500: 0.0111**.** The optimal setting on 5% data is (50, 0.01). The scores on the test set are MAP: 0.00497 NDCG: 0.0369, precision at 500: 0.0156. For 5% of data, it is impractical for us to explore a higher rank. We run out of memory for the test score calculation on the full dataset.

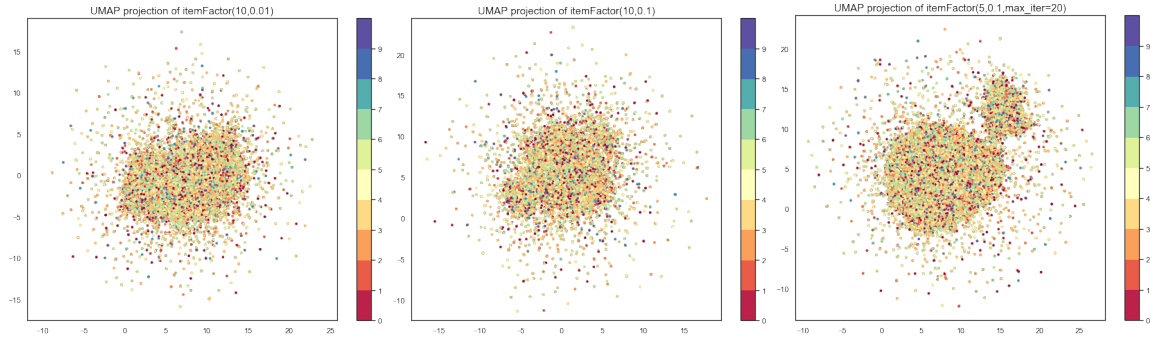
5. Extension

Since tSNE does not scale, preserve global data structure, or work with high-dimensional data directly, we decide to use UMAP for the visualization, which has higher computation efficiency and consumes less memory. We use the genre extracted from the fuzzy genre dataset[3] to build a pair (book_id, genre). For the book attached to multiple genres, we select the genre with the maximum count.



(a) (b) (c)

(d)



(e) (f) (g)

We first visualize the UMAP projection of the item factor of the model trained on 1% data with the optimal setting(200,0.01,max_iter=20), and observe that the points are scattered (Figure a). We wonder if the rank dimension will matter, so we change the rank to 5 and revisualize. The cluster in the middle becomes larger but there are many dots scattered around (Figure b). This might because the insufficient data is not representative of the genre distribution, so we train the model on 10% of the data and do a grid search on rank=(5,10) and lambda = (0.01,0.1) (Figure c-f). We observe that when the rank is small and lambda is large, there are two clusters (Figure d). We wonder if the model has converged, so we rerun the model with maximum iteration 20 and get two separate clusters (Figure g). Due to the computation limit, we are not able to experiment with more data. Clusters in the above graphs are all composed of multiple-color dots, which indicates that the genres are blended into one another. We expect to have better plots in 3D, because we definitely lose more information in a 2D plot and harder to see patterns. In the original fuzzy dataset, a large number of books are associated with multiple labels. Although we use the label that most people use, actually, books are not strictly to one genre, which will probably cause the distribution of each class not distinct from one another.

6. Contribution
   We brainstorm and debug through zoom and write all of the code together.

Appendix

[1] pyspark.ml.recommendation module, website:
https://spark.apache.org/docs/latest/api/python/pyspark.ml.html#module-pyspark.ml.recommendation
[2] ranking system matrix evaluation from spark, website:
https://spark.apache.org/docs/latest/mllib-evaluation-metrics.html#ranking-systems
[3] the supplementary meta-data of books downloaded from UCSD website and two papers:
https://sites.google.com/eng.ucsd.edu/ucsdbookgraph/books?authuser=0

[4] Mengting Wan and Julian McAuley. 2018. Item recommendation on monotonic behavior chains. In Proceedings of the 12th ACM Conference on Recommender Systems (RecSys '18). Association for Computing Machinery, New York, NY, USA, 86–94. DOI:https://doi.org/10.1145/3240323.3240369

[5] Misra, R., Nakashole, N., & McAuley, J. Fine-Grained Spoiler Detection from Large-Scale Review Corpora. ACL Anthology. https://www.aclweb.org/anthology/P19-1248. Accessed 12 May 2020