Recomendation System: ALS with Pyspark

Yunya Wang: yw4509, Haoxue Li: hl3664, and Gaomin Wu: gw1107

I. BACKGROUND:

User personalization has become prevalent in modern recommender system. Among its various methods, matrix factorization is the most popular and effective technique that characterizes users and items by vectors of latent factors. As it's barely possible for a recommender system to provide relevant and effective recommendations without sufficient data such as past purchases, browsing history, and feedback, big data is the driving force behind recommender systems.

The objective of this project is to create a recommender system with the provided dataset – Goodreads Datasets. In the dataset, we are given explicit feedback data.Each user-book interaction is represented by an interger. An iteger r_{ij} represent the rating given by the $user_i$ to the $book_j$, scaling from one to five.

In this project, we build a latent factor model to decompose the user-item interaction matrix into a product of user factor matrix U and item factor matrix V that consists of embedding for users and items in a common vector space. We use Alternating Least Squares (ALS) to generate the user and item latent matrix. It first holds user matrix fixed and runs gradient descent with item matrix; then it holds item matrix fixed and runs gradient descent with user matrix. ALS is implemented in Apache Spark ML and built for a larges-scale collaborative filtering problems. ALS is doing a pretty good job at solving scalability and sparseness of the Ratings data, and it's simple and scales well to very large datasets.

II. DATA AND METHODOLOGIES:

Data Preprocessing Prior the training, we conducted reprocessing including:
1. Filtered out users with number of interactions less than 10.
2. Remove NA values in book_id,user_id,rating.
3. Convert all data types in the data into integer given our ALS training model could only accept numeric values.

Training Validation Testing Split
1. Select 20% of users to form the validation set. For each validation user, use half of their interactions for training, and the other half should be held out for validation. And the same process is conducted for testing.
2. We also down size our data to 1%,10%,25% for training purposes. As for down-sizing, we down sample by user_id and join their interactions after.

ALS We used the implementation of ALS from Spark machine learning library (pyspark.ml.recommendation). We first fit the ALS model in the training data, and tune the hyper-parameter of ALS on validation dataset. Then we make top 500 book predictions for user in test set for evaluation with different metrics. We also retrieve the latent factors from the model for analysis.

Parallelization As we are training a great amount of models without using the Cross Validation package from pyspark, where parallel programming is used. We decided to implement our own parallization with package multiprocessing.pool. We defined a single task first and created multi-threads based on the number of models (number of combinations of parameters map) and implement parallelization using pool.ima_unordered.

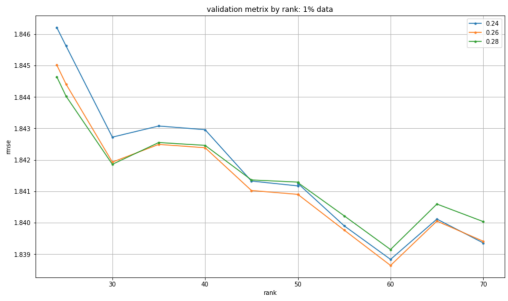
III. PARAMETER TUNING:

In this project, we tune three parameters (rank, regParam and maxIter) with grid search strategy. We use $RMSE$ as the metrics in the process of parameter tuning. As for maxIter, we tried from range 10 to 15 and the $RMSE$ does not vary too much from each other and thus we use default value maxIter of 10 for the rest of tuning.

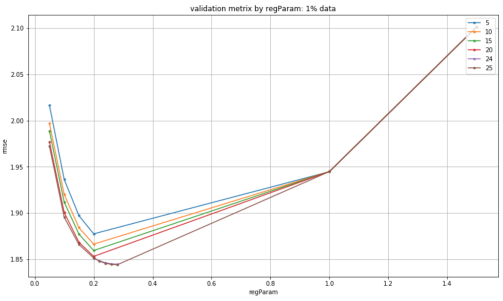
We also start tuning from 1% of the data and gradually increased to 10%, 25% and 100%. The following is the detailed training procedure for 1% of the data.

1. Rank:

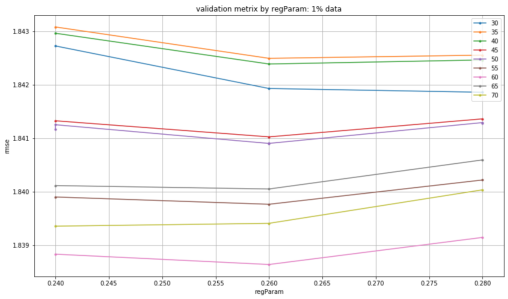
We first search through the combinations of rank (up to 25) and regParam (as shown below); we could see that the RMSE is consistently on a decreasing trend as the rank increases, as the information contains in the latent factors increases. Thus we further increased the rank to 80 and we found the best range for rank at 60.



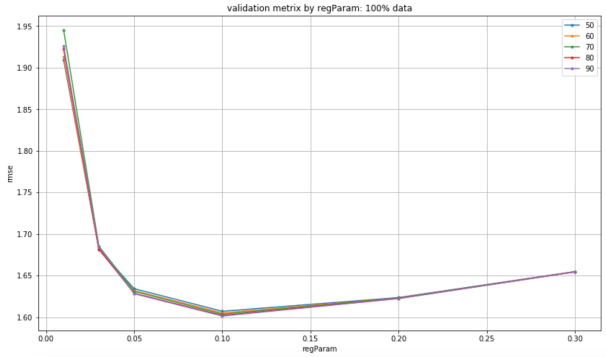
2. RegParam:
We first tune around range from 0.05 to 0.3 and find the best range at round 0.2 to 0.3.



Then we further fine tune it around range 0.2 to 0.3 and land at 0.26.



After we used the similar training procedure for 10%,25% and 100% of data. Only difference is that, we start tuning rank from 50 as we have already learnt that rank smaller than that is not going to provide additional power. The following is the metric graph by rank and regParam.



And the best parameter map is at rank of 90 and regParam at 0.1.

The following is the summary table for RMSE of the best model we have obtained from the training procedure for different size of data. We could see that, as we increase the size of the data, the RMSE keep increasing as we are having more training power with more observable data history.

	1%	10%	25%	100%
Best RMSE From the Validation Data	1.839	1.708	1.665	1.601

IV. EVALUATION:

For evaluation, we focus on MAP (Mean Average Precision) and Precision at top 500.

Suppose our recommendation system deals with a set of M users. Each user (u_i) having a set of N_i ground truth relevant documents: $U = u_0, u_1, ..., u_{M1}$. And from our recommendation system, we have a list of Q_i recommended documents, in order of decreasing relevance: $Ri = [r_0, r_1, ..., r_{Q_i 1}]$.We define a function which, provided a recommended document and a set of ground truth relevant documents, returns a relevance score for the recommended document. $rel_D(r) = 1$if $r \in D$ else 0.

Precision at k Precision at k is a measure of how many of the top k recommended documents are in the set of true relevant documents averaged across all users[1].

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

Note that in this metric, the order of the recommendations is not taken into account.

MAP(Mean Average Precision) MAP is a measure of how many of the recommended documents are in the set of true relevant documents.

$$p(k) = \frac{1}{M} \sum_{i=0}^{M-1} \frac{1}{N} \sum_{j=0}^{Q_i-1} \frac{rel_{D_i}(R_i(j))}{j+1} \qquad (2)$$

Compared with precision at top k, MAP takes the order of the recommendations into account. In this way, we impose high penalty on highly relevant items not being recommended with high relevance.

The following tables are the summary of MAP and

Precision for our test and validation from the best model trained at different size of data.

	1%	10%	25%	100%
Validation: MAP at 500	0.002071	0.002192	0.002222	0.002290
Test: MAP at 500	0.002070	0.002188	0.002236	0.002290

	1%	10%	25%	100%
Validation: Precision at 500	0.022858	0.023211	0.023527	0.024016
Test: Precision at 500	0.02224	0.02316	0.02332	0.02409

V. EXTENSION 1: COLD START

The default setting for ALS in pyspark.ml and pyspark.mllib library is to drop any rows in the DataFrame of predictions that contain NaN values. However, this approach can only recommend items that show in the utility matrix before, while as it can't recommend books that does not appear before.

Dataset we take 25% data from the original data as the whole data for cold start extension. We split whole data into training data and test data based on 8:2 random splitting. Then for a subset of hold out items, we split the training data set into two parts: actual training data (80%) and holdout (20%) data based on items. This means there is no intersection of books between training and hold out data.

Supplementary data we use book genres initial data (*goooreads_book_genres_initial.json.gz*) for supplementary book data. There are 10 basic genres for books: children, comics and graphic fantasy and paranormal, fiction, history and historical-fiction and biography, mystery and thriller and crime, non-fiction, poetry, romance, young-adult.

KNN model We use k nearest neighbors model to get nearest neighbors for each book in the hold out set based on their distance with the books in training set. Here we use Euclidean distance and use pyspark BucketedRandomProjectionLSH package. Instead of calculating all cross-join distances, we get hash set (number of hash-tables: 5) for each book and then use projection LSH to get their similarity. And instead of restricting exact number of neighbors, we set a threshold 0.8 for the Euclidean distance as similar.

In this way, we find that every item in hold out set has at least 1 nearest neighbors and at most 1080 nearest neighbors.

Latent Representation Mapping We train the recommendation system using training set and get the item matrix and the user matrix. Thus we can get the latent representation for each item in training set.

We use the latent representation from training model to calculate the latent representation for each hold out book. For each holdout book, we get their nearest neighbors, and calculate the average latent features from these neighbors.

	book_id	user_id	rating
0	1174955	262554	2.456925
1	173455	583601	4.063086
2	140678	264845	1.765560
3	93681	412372	1.930299
4	857871	646726	1.758787
5	13429	754276	2.998549
6	69621	214423	3.283679
7	6361	517672	1.539619
8	29105	173458	3.574426

FIG. 1. KNN mapping examples

New model We generate a new item matrix using the rating of hold out set from KNN model. We multiply the union of this new item matrix and original item matrix and the user item matrix to get the final utility matrix. From this utility matrix, we use ALS (rank:10, numIterations:10, reg: 0.1) to get the final test result rmse 2.47. The full ALS model using training set and hold out set has the test result rmse 1.78.

Examples We dive into some examples. From table1, we can see that most books in hold out set have a reasonable rating compared with predicted rating. However, we can still see that some books have a large gap between true ratings and predicted rating. Predicting rating merely based on genres will lead to some errors since books from the same genres will still have various ratings. Particularly, for non-fiction books, there are very large differences between our prediction and true ratings. In the future, we may take other information about books, i.e. publisher, publication date, into consideration.

book id	True ratings	Predicted rating	Main genres
46743	2,3	3.01	fiction and fantasy
71792	4	3.14	mystery and fiction
1588090	2	1.87	fiction
173455	4	4.25	crime and fiction
788525	5	1.58	non-fiction

TABLE I. Caption

VI. EXTENSION 2: LATENT FACTOR VISUALIZATION

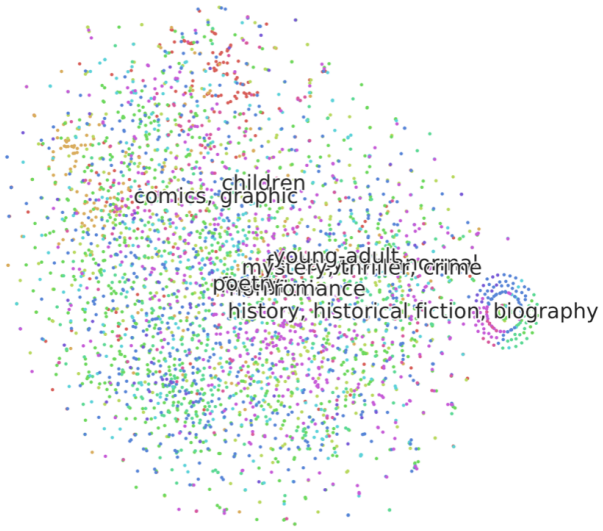


FIG. 2. Book latent factors of 90 dimension reduced to two dimension by t-SNE for visualization. Different colors of the data points means the corresponding genre of the book. The label of genre is placed at the median of all the books in the genre.

In this section, we tend to analyse the meaning of latent factor representation learned by ALS model. In the dataset we are using, we have extra information about books, but not users. So in this section, we would focus on analysis of book latent factors. Here, we choose the ALS model with parameter: rank=90, regParam=0.1 and maxIter=10. As the book latent factors are learned from user-books interactions. Small distance of book latent factors would indicate similar ratings/co-occurrence for an given user. If we assume that users tend to have similar interactions in similar genres, we can see different cluster of genre in the latent factor space.

Implementation For visualization in the latent factor space, we use t-SNE from Scikit Learn (sklearn.manifold.TSNE). t-SNE is a tool to visualize high-dimensional data. It converts similarities between data points to joint probabilities and tries to minimize the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data.

We use book genres mentioned above (*goodreads_book_genres_initial.json.gz*) as supplementary book data to explain the cluster of latent factor in the space. And as one book can be in several genre, here we only consider the main genre (genre with the highest score in the data.) For visualization purpose and computation limit, we use 40000 (1%) books to plot the latent factor space.

The result is shown in the Fig.2. There is no clear clustering of genre as we expected in the model. A possible reason is that we are using only (1%) of the data. However, genres of $children, comics-graphic$ have a boundary with the other genres. This can be explained by the difference of target user in this two cluster, the children and the older user.

VII. CONTRIBUTION AND COLLABORATION:

Yunya Wang: Baseline Model, Extension 1, Report
Haoxue Li: Baseline Model, Extension 1, Report
Gaomin Wu: Baseline Model, Extension 2, Report

---

[1] The definition is from Spark document: https://spark.apache.org/docs/latest/mllib-evaluation-metrics.htmlranking-systems.