Big Data Final Project Report

Goodreads Datasets Recommender

Implementation of implicit matrix factorization to generate recommendations

Team Members:
Meenakshi Jhalani (mgj265)
Rajat Ravindrakumar Bapuri (rrb398)
Shiva Sanketh Ramagiri Mathad (srm714)
GitHub Repository: https://github.com/nyu-big-data/final-project-nyu-big-data.git

1. Introduction

With the recent rise of web services like Spotify, Netflix, Amazon, etc., it is imperative to implement recommender systems on their websites, which are algorithms aimed at predicting user's interests and needs based on their past behaviors and feedback, and as a result helping boost sales. For this project, we implemented collaborative filtering with implicit matrix factorization from Spark's ALS module on Goodreads dataset to suggest book recommendations for its users. In addition, we explore some extensions to help with performance of our model. We also discuss some of the complications we faced during the implementation of our model and the techniques we used to moderate these issues.

2. Data Splitting and sub-samplings

The raw dataset has about 228 million rows of data of the format <user_id,book_id,is_read,rating,is_reviewed>. But all the rows in the dataset very not valuable to the training, so we cleaned the dataset as follows:

- Step 1: Remove the rows where is_read or is_reviewed or rating was zero, which means the user has not read or reviewed the book.
- Step 2: Remove the users whose occurrence in the data from step one was less than 10 times.
- Step 3: Collect the distinct user_ids from the filtered interactions of step 2. Randomly split all these unique user_ids in the 60-20-20 (Train-Validation-Test) ratio.
- Step 4: Join the user_id splits from the third step with the resulting interactions in the second step, this divides user interactions into three parts.
- Step 5: Now take 20% of the validation set user interactions from the fourth step and group the set by user_id, order it by book_id and assign sequential row numbers to each group. From each group take the odd numbered rows and concatenate with training interactions set and update the validations interactions set with the remaining odd numbered rows.
- Step 6: Repeat step 5 for test interactions set.

As a result 100% of the user_ids are in the training set, 20% of user_ids are in the validation set and remaining 20% in the test set or in terms of interactions there are 80% of interactions in training set, 10% of interactions in validation and remaining 10% in test set.

3. **Training, Evaluation and Results**
   Training was done using the ALS model from MLlib library and 20% of the data. For hyperparameter tuning, we chose the values as mentioned in the table below. Validation was done on RMSE and mAP metrics as they better evaluate the model on the complete valuation data. The best model with RMSE of 0.8924, mAP=0.9216 (Rank: 12, RegParam: 0.1) and this was used for testing. For testing the result, we calculated both RMSE and "Precision at k" values.

Hyper params		**RegParam**			
		0.01	**0.05**	**0.1**	**0.3**
Rank	**8**	RMSE=1.1862 mAP=0.8624	RMSE= 1.0334 mAP=0.9008	RMSE=0.9752 mAP=0.8865	RMSE=0.9958 mAP=0.8987
	12	RMSE=1.1211 mAP=0.8863	RMSE=0.9248 mAP=0.9187	RMSE=0.8924 mAP=0.9216	RMSE=1.0483 mAP=0.8641
	16	RMSE=1.1452 mAP=0.8902	RMSE=1.0267 mAP=0.8934	RMSE=0.9088 mAP=0.9124	RMSE=1.0674 mAP=0.8679

Test Results: Rank: 12, RegParam: 0.1
RMSE: 0.8981, mAP=0.9281, Precision at k(500): 0.002

4. **Extensions**

a) ***Comparison to single-machine implementations***
Here we compare the parallel ALS model with a single-machine implementation model from the library "lightfm". We primarily focus on accuracy and time taken for model fitting as part of our comparison.

Data preparation for lightfm:
Most of the data splitting remains the same for our basic recommender model and lightfm model except for creation of subset of data and parsing of input as a sparse matrix.

1. Step 1 and Step 2 of basic recommender system split is followed

2. Only 10% distinct user_ids from the filtered interactions are considered. This is done as single machine implementation takes an enormous amount of time for execution. Randomly split all these 10% unique user_ids in the 60-20-20 (Train-Validation-Test) ratio.
3. Step 4, 5, 6 from basic recommender system split is followed creating around 1.2 million training samples.
4. All the unique book_ids from Training data is extracted and used to filter out the book ids from Validation and Testing which are not present in Training data. This was done as the sparse matrix cannot have unseen data in the resulting validation matrix and this approach is likely to give better results than including unseen book ids from Validation/ Testing in training with mean value. This data is used in the MLlib ALS model.
5. We create a sparse matrix of train-validation and train-testing data and this is used in the lightfm model.

Both the models are compared with the same data and same parameters. Both validation and testing data are used for testing purposes. Below are the results of comparison. (Lightfm model was run locally as there was an issue with dumbo lightfm installation)

Evaluation

Hyperparameter values	model	Precision at k K = 100	Model fit time	Score prediction for Evaluation time.
Rank: 12 RegParam: 0.01 Max Iters: 15	lightfm	0.00033	32.23 sec	915.48 sec
	ALS	0.00025	9.64 sec	3.22 sec
Rank: 12 RegParam: 0.1 Max Iters: 15	lightfm	0.00038	29.07 sec	1043.47 sec
	ALS	0.00023	7.24 sec	3.45 sec

b) Exploration/Visualization

Data Visualization is extremely important to gain insights into distribution of certain variables and to discover potential correlations between variables. However, it is very challenging to get visualizations of high-dimensional datasets, like ours, which have large numbers of variables. A common approach to such a problem is to execute dimensionality reduction whilst retaining as much information as possible. The technique we used to help us reduce dimensions in our dataset is t-Distributed Stochastic Neighbor Embedding (t-SNE).

We were interested in seeing if our model was able to recognize genres as an implicit feature. Hence,we extracted the model features, joined it with the genres table and created a plot of the product matrix features - reduced to dimension size 2 with labels of genres.

Since t-SNE scales quadratically in the number of objects, it is computationally very expensive and learning becomes drastically slow. Hence, we decided to reduce our sample size to only 1500 interactions to reduce the burden on our machine in terms of memory and time.

We were not able to complete the further processing tables for book_id and book_genres as there were memory issues in Dumbo. But we did export the product features, randomly sampled the features, saved it to a json file on Dumbo. We downloaded the file and performed t-SNE on the features and plotted them on our local machines. Figure 1 is the plot. We can clearly see that there are clusters being formed in the figure. We think we could have plotted each genre with different color provided computations were successful on Dumbo.

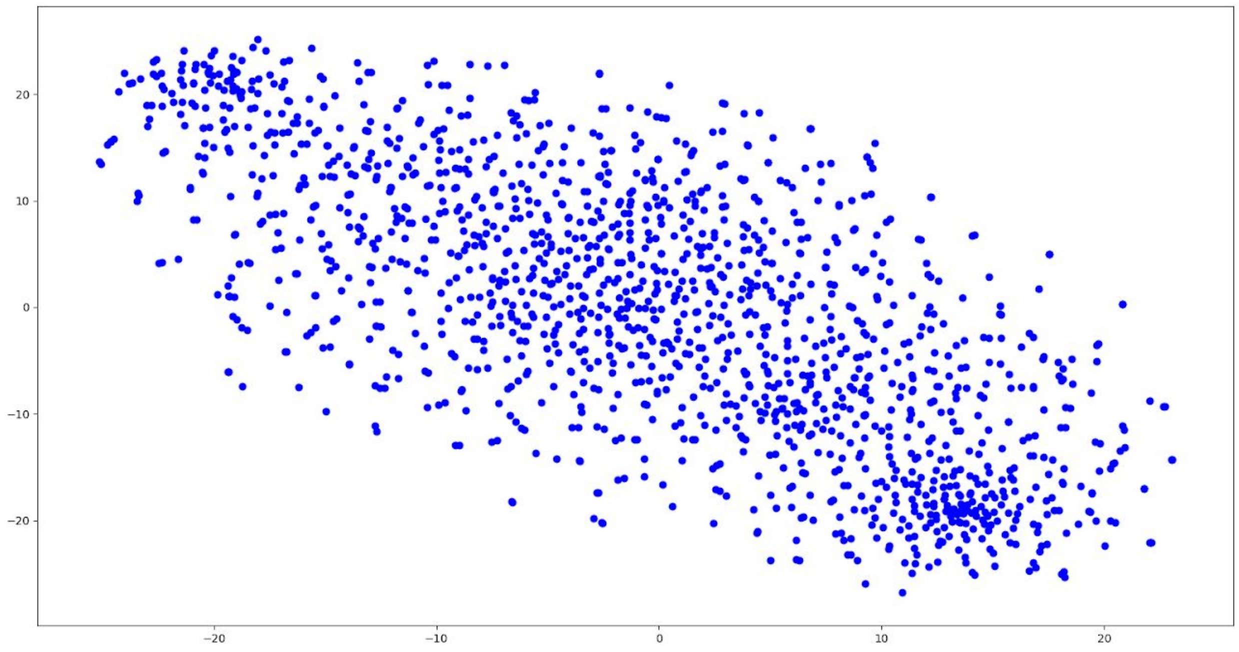

Figure1. Product Features Visualization

We think that a model with ideal Rank and RegParam can produce discernible clusters, but this was not possible because of limited processing powers on our local machines and high traffic on Dumbo.

5. Complications

The first complication we faced was during data preprocessing, which is to decide which rows of data to discard, for example data had rows with is_read=0 and is_reviewed=1 (which we could argue that a user has reviewed a book without reading?). Secondly, for working with a subsampled dataset, we had to keep only approximately 10% of the data, for this we could not just eliminate 90% of the data directly as this would mean losing data disproportionately. To counter this, we used 10% of unique users from the filtered data, thus keeping all interactions of these users. Lastly, while executing our pipeline, many jobs were killed due to substantial burden on the cluster the entire time, which prevented us from implementing our pipeline in a single step. We faced many run-time issues , memory issues and high waiting times. The lengthy running time of our jobs were mostly due to memory issues given the size of the data and slow performance of RDD based methods. We also faced network issues where we would lose connection to the server and were forced to implement the process all over again.

6. Contributions

We had a meeting initially to discuss individual understanding of the project and clear some questions. We implemented data splitting together by sharing screens on zoom, as it was the trickiest and difficult part of the project. After that we split the training, evaluation, extensions part. Training implementation of the model and saving it to HDFS was implemented by Shiva, evaluation and writing the features to file by Rajat. Visualization extension by Meenakshi. Lightfm extension was implemented by Shiva and Rajat together. Code cleaning, review and report template by Meenakshi. Rest of the report was equally contributed by all three of us.

7. Conclusion

We are focusing on three metrics RMSE, mAP, precision @ k. Depending on the number of book recommendations we have to provide to each user we should concentrate on different metrics, for example, if we want our model to predict well on all the data, we have to tune the parameters such that RMSE reduces and mAP increases. On the other hand if we want better recommendations for top K books, we should tune hyperparameters such that precision at K increases. Given the limited processing power on Dumbo we could not tune parameters for better precision at K. We intend to continue to improve the model by tuning the hyperparameters on complete data and explore more extensions which might lead to better results.