Recommender System

DS GA-1004 Big Data

Final Project Report

Vishaal Ranjan, Srinivas Rao

1 Group

- Vishaal Ranjan (vr1089)
- Srivinas Rao (svr300)

2 Overview

In this project, we have used the Goodreads Dataset in order to build a recommender system. Finally, we are going to recommend 500 books to every user and assign a rating to every recommended book that shows how likely it is that the user will like the book. We split the original dataset to training, validation and test sets in a 60:20:20 ratio. For training the model, we took 1% of all users as the time required to train for all users was extremely large due to which the prototyping process would have slowed down had we used all the users for training.

Some of the statistics of the data are:
Total user count in Training Set = **832135**
Total user count in Validation Set = **104118**
Total user count in Test Set = **103790**
Range of user_ids in Training set = 0 to 876100
Range of book_ids in Training set = 7 to 2358299
Each of the books are rated on a scale of 1-5 by the users.

3 Data Processing

We have used the file 'goodreads_interactions.csv' as the dataset for implementing our recommender system. The total count of interactions in this datset is: 228648343 (228 million). We made the choice of removing all interactions where the rating = 0 as the range of book ratings is from 1 to 5 (1 being the lowest), so having 0 rating will negatively impact the accuracy of the final predictions. Following this, we filtered out all the users whose total no. of interactions was ¡= 10. An interaction was taken as a row in the csv file for a particular user_id. We downsampled the original dataset to only take 1% of all the users in order to speed up the process of training models and trying out multiple different evaluations. We did this by only selecting the user_id for which user_id%100 = 0.

Following the downsampling, we converted the csv file to a parquet file. We used the randomSplit method and used a random seed (45) to split the data into train, validation and test sets in a 60:20:20 ratio.

Now, it's important for the training set to have each and every user in order to predict the items for a user. In order to achieve this, we ordered the validation set in increasing order of its user_id. We added a new column which contained the sequential ID of the row. From this validation set, we selected all the odd-numbered rows and stored it in a dataframe. After removing the sequential ID column, this dataframe is added to the training set. The even numbered rows will be the new validation set after we remove the sequential ID column. The same process was repeated for the test set as well. After these sequence of steps, the training set contains each and every user_id due to which we will be able to predict books for all the users.

Now, the dataframes of train, validation and test were converted to parquet files.

4 Model and Experiments

4.1 Introduction to ALS

The alternating least squares (ALS) algorithm factorizes a given matrix $\boldsymbol{R}$ into two factors $\boldsymbol{U}$ and $\boldsymbol{V}$ such that $R \approx U^T V$. The unknown row dimension is given as a parameter to the algorithm and is called latent factors. Since matrix factorization can be used in the context of recommendation, the matrices U and V can be called user and item matrix, respectively. The **i**th column of the user matrix is denoted by u_i and the **i**th column of the item matrix is v_i. The matrix R can be called the ratings matrix with $(R)_{i,j} = r_{i,j}$.

$$argmin_{U,V} \sum_{\{i,j|r_{ij}\neq 0\}} (r_{i,j} - u_i^T v_j)^2 + \lambda(\sum_i n_{u_i}||u_i||^2) + \sum_i n_{v_j}||v_j||^2)$$

with being the regularization factor, n_{u_i} being the number of items the user i has rated and n_{v_j} being the number of times the item j has been rated. This regularization scheme to avoid overfitting is called weighted--regularization. Details can be found in the work of Zhou et al..

By fixing one of the matrices U or V, we obtain a quadratic form which can be solved directly. The solution of the modified problem is guaranteed to monotonically decrease the overall cost function. By applying this step alternately to the matrices U and V, we can iteratively improve the matrix factorization.

The matrix R is given in its sparse representation as a tuple of (i,j,r) where i denotes the row index, j the column index and r is the matrix value at position (i,j).

4.2 Hyperparameters in the model

These are the hyperparameters in the model. We have **ONLY** tuned rank and regParam out of these, rest remain to their respective defaults.

- **numBlocks** is the number of blocks the users and items will be partitioned into in order to parallelize computation (defaults to 10).
- **rank** is the number of latent factors in the model (defaults to 10).
- **maxIter** is the maximum number of iterations to run (defaults to 10).
- **regParam** specifies the regularization parameter in ALS (defaults to 1.0).
- **implicitPrefs** specifies whether to use the explicit feedback ALS variant or one adapted for implicit feedback data (defaults to false which means using explicit feedback).
- **alpha** is a parameter applicable to the implicit feedback variant of ALS that governs the baseline confidence in preference observations (defaults to 1.0).

4.3 Evaluation metrics used

- **RMSE:** We use the Root Mean Squared Error as our **regression metric** evaluator. The error is calculated on the predictions of the ratings. The RMSE is calculated as usual: $\text{RMSE} = \sqrt{\frac{\sum_{i=0}^{N-1}(\mathbf{y}_i - \hat{\mathbf{y}}_i)^2}{N}}$ where $\hat{\mathbf{y}}_i$ is the predicted rating and $\mathbf{y}_i$ is the actual rating.

- **MAP accuracy:** MAP is a measure of how many of the recommended books are in the set of true relevant books, where the order of the recommendations is taken into account (i.e. penalty for highly relevant documents is higher). Mean average precision for a set of queries is the mean of the average precision scores for each query. It is calculated as :

$$\text{MAP} = \frac{\sum_{q=1}^{Q} \text{AveP(q)}}{Q}$$

where Q is the number of queries.

4.4 Hyperparameter tuning

As said earlier, we have tuned the hyperparameters rank and regParam in this project. Surprisingly, we got better results for **higher** ranks than the usual low ranks (See below). We chose regParam = 0.09 and rank = 50 as it was giving satisfactory results, not the best but even higher ranks would take more computation time so we chose this. The hyperparameter tuning is as shown below.

Rank: 10	Lambda: 0.070000	RMSE Validation: 0.626367	Test Loss: 0.882552
Rank: 20	Lambda: 0.070000	RMSE Validation: 0.592292	Test Loss: 0.880362
Rank: 30	Lambda: 0.070000	RMSE Validation: 0.578202	Test Loss: 0.883002
Rank: 40	Lambda: 0.070000	RMSE Validation: 0.568223	Test Loss: 0.878958
Rank: 50	Lambda: 0.070000	RMSE Validation: 0.564548	Test Loss: 0.878885
Rank: 60	Lambda: 0.070000	RMSE Validation: 0.560909	Test Loss: 0.879229
Rank: 70	Lambda: 0.070000	RMSE Validation: 0.558267	Test Loss: 0.878512
Rank: 80	Lambda: 0.070000	RMSE Validation: 0.555825	Test Loss: 0.876558
Rank: 90	Lambda: 0.070000	RMSE Validation: 0.553673	Test Loss: 0.877509
Rank: 10	Lambda: 0.080000	RMSE Validation: 0.628299	Test Loss: 0.873465
Rank: 20	Lambda: 0.080000	RMSE Validation: 0.595566	Test Loss: 0.868628
Rank: 30	Lambda: 0.080000	RMSE Validation: 0.581853	Test Loss: 0.869649
Rank: 40	Lambda: 0.080000	RMSE Validation: 0.572872	Test Loss: 0.865821
Rank: 50	Lambda: 0.080000	RMSE Validation: 0.569419	Test Loss: 0.865804
Rank: 60	Lambda: 0.080000	RMSE Validation: 0.566027	Test Loss: 0.865449
Rank: 70	Lambda: 0.080000	RMSE Validation: 0.564151	Test Loss: 0.865140
Rank: 80	Lambda: 0.080000	RMSE Validation: 0.562010	Test Loss: 0.863694
Rank: 90	Lambda: 0.080000	RMSE Validation: 0.559713	Test Loss: 0.864030
Rank: 10	Lambda: 0.090000	RMSE Validation: 0.631399	Test Loss: 0.866694
Rank: 20	Lambda: 0.090000	RMSE Validation: 0.600641	Test Loss: 0.860134
Rank: 30	Lambda: 0.090000	RMSE Validation: 0.587577	Test Loss: 0.860215
Rank: 40	Lambda: 0.090000	RMSE Validation: 0.579864	Test Loss: 0.856681
Rank: 50	Lambda: 0.090000	RMSE Validation: 0.576497	Test Loss: 0.856687
Rank: 60	Lambda: 0.090000	RMSE Validation: 0.573314	Test Loss: 0.856108
Rank: 70	Lambda: 0.090000	RMSE Validation: 0.572202	Test Loss: 0.855969
Rank: 80	Lambda: 0.090000	RMSE Validation: 0.570172	Test Loss: 0.854976
Rank: 90	Lambda: 0.090000	RMSE Validation: 0.568014	Test Loss: 0.854879
Rank: 10	Lambda: 0.100000	RMSE Validation: 0.635490	Test Loss: 0.861649
Rank: 20	Lambda: 0.100000	RMSE Validation: 0.607107	Test Loss: 0.854325
Rank: 30	Lambda: 0.100000	RMSE Validation: 0.594894	Test Loss: 0.853740
Rank: 40	Lambda: 0.100000	RMSE Validation: 0.588392	Test Loss: 0.850647
Rank: 50	Lambda: 0.100000	RMSE Validation: 0.585140	Test Loss: 0.850592

4.5 Results

We get RMSE ≈ 1 when we evaluate the ALS model on the test set. We get MAP ≈ 0.02. The results are as shown below. In our code, we have also included the option of getting the top 500 recommendations filtered by a particular user_id. It is also possible to get each of the book recommendations on a separate row along with its predicted rating.

```
Rank: 50        Lambda: 0.090000        RMSE Validation: 1.043121       Test Los
s: 1.046779
Predictions of Ratings on the Test Data:
+-------+-------+------+----------+
|user_id|book_id|rating|prediction|
+-------+-------+------+----------+
| 375800|    148|     3|       4.0|
| 161400|    148|     4|       4.0|
|  14700|    471|     3|       2.0|
| 346500|    833|     4|       4.0|
|  56900|    833|     3|       3.0|
| 144000|    833|     3|       4.0|
| 373000|    833|     4|       4.0|
| 226800|    833|     3|       3.0|
| 256400|    833|     3|       3.0|
| 265100|    833|     5|       4.0|
| 569000|    833|     3|       3.0|
| 326100|    833|     5|       4.0|
|  20100|    833|     4|       4.0|
|  78500|    833|     3|       4.0|
| 382100|    833|     5|       4.0|
| 705700|    833|     4|       4.0|
| 316100|    833|     3|       4.0|
| 128000|    833|     5|       5.0|
|  17300|    833|     5|       4.0|
| 161200|    833|     1|       2.0|
+-------+-------+------+----------+
only showing top 20 rows
```

```
+-------+--------------------+
|user_id|     recommendations|
+-------+--------------------+
|   4900|[[498638, 5.75356...|
|   5300|[[259582, 5.18324...|
|   9900|[[971623, 5.44093...|
|  18800|[[1008545, 4.7916...|
|  21700|[[1311478, 5.3997...|
|  78400|[[214867, 5.73131...|
|  81900|[[259582, 6.56315...|
|  85100|[[259582, 4.74263...|
| 100800|[[259582, 6.25777...|
| 109800|[[502735, 5.41735...|
+-------+--------------------+
only showing top 10 rows
```

5 Extension

For the extension, we have chosen to tackle the Cold Start Problem in which we can give recommendations to users which are not part of the training set. For this purpose, we have used the book genres dataset in addition to the original interactions file. For this extension, we retrained the model and performed the hyperparameter tuning again. After tuning the hyperparameters, regParam = 0.14 and rank = 10 give fairly low RMSE values. For the baseline model, we ensured that each and every user has representation in the training set. For this extension, we did not include the users from validation and test set in the training set due to which there will be some users that have no interactions in training set. Following this, we evaluated its performance in comparison to the baseline model.

6 Contribution of Team Members

Vishaal: Filtered out users with interactions less than 10, converted csv files to parquet, downsampled the data to 1% of users, split data to train, test and validation, added all the users from validation and test set to training set, trained the ALS model using default parameters, evaluated RMSE and MAP values.

Srinivas: Performed hyperparameter tuning and evaluated the ALS on tuned parameters on the test data on the cluster to obtain the results in 4.4 and 4.5. Worked on the cold-start extension.

7 References

[1] https://spark.apache.org/docs/latest/api/python/pyspark.ml.htmlmodule-pyspark.ml.recommendation

[2] https://spark.apache.org/docs/2.2.0/mllib-evaluation-metrics.html

[3] Y. Zhou, ”Large-Scale Parallel Collaborative Filtering for the Netflix Prize”, Proc. 4th Int’l Conf. Algorithmic Aspects in Information and Management, pp. 337-348, 2008.

[4] https://spark.apache.org

[5] https://ci.apache.org