

# Movie Recommender System on MovieLens Dataset

DS-GA 1004 Big Data Final Project

Mohit Kewlani  
New York University  
mmk9369@nyu.edu

Sidharth Shyamsukha  
New York University  
ss14885@nyu.edu

## 1. Introduction

Collaborative filtering, content-based filtering, knowledge-based systems, and other algorithms are common recommendation system algorithms. Collaborative filtering presupposes that human behavior is consistent: people will like items that they liked previously. In addition, the system offers suggestions based on item ratings. It is capable of accurately proposing complex products without needing that the item be "understood." We employ the Alternative Least Square method to conduct research on collaborative filtering with latent factor decomposition, a matrix factorization methodology to decompose user-item interaction information to recommend users' most relevant items. We utilize PySpark (version 3.0.1) and implement baseline model on HPC environment. We compare the performance of our ALS model with the baseline popularity model. Further We implement model on single-machine, LightFM to compare its performance from ALS model.

## 2. Data

### 2.1 Movielens Dataset

This dataset contains a set of movie ratings from the MovieLens website, a movie recommendation service. This dataset was collected and maintained by [GroupLens](#), a research group at the University of Minnesota. There are 5 versions included: "25m", "latest-small", "100k", "1m", "20m". In all datasets, the movies data and ratings data are joined on "movieId". The 25m dataset, latest-small dataset, and 20m dataset contain only movie data and rating data. The 1m dataset and 100k dataset contain demographic data in addition to movie and rating data. We will be using the "latest-small" and "latest" version for our project.

### 2.2 Data Pre-Processing

The data was first split into 2 parts to separate the data and prevent overlapping in validation and test sets. Then the

first part was split randomly in the ratio 0.8 and 0.2 into train and test. The second part was also split randomly in the ratio 0.8 and 0.2 but named train and val. Next train data frames from both the halves were merged again giving us a train, test and validate sets. Hence ensuring that some train has some user interaction from both test and val sets, to give better results. And splitting up the data using the above way ensured that no user in test and val overlap and it cannot now know any values from val beforehand. The same splitting technique was used for the larger dataset.

## 3. Model Implementation

The two main library used was **pyspark.ml**, which contains various recommendation functions based on Spark DataFrame and **pyspark.mllib** which is oriented for RDDs. **pyspark.mllib.evaluation** contains more evaluation metrics including **P@k**, **ndcgAt**, **recallAt** and **MAP** which were not available in **pyspark.ml**. Since our original dataset was in data frame format, we adopted **pyspark.ml** to fit recommender model, and used **pyspark.mllib** for evaluation.

In Baseline Popularity model we do not take user likings into the account while making predictions, hence it is not a personalized movie recommendation model. So, For Baseline Popularity Model, We used simple spark data frame operations to predict top 100 highest average rated movies which have at least 50 ratings count. This technique was used because it helped in identifying the most popular movies from the whole training data and all the movies with high rating, but low number of ratings were removed. Next all user's movies were identified and compared with the top 100 popular movies. Then with the help of RankingMetrics we computed mean average precision, average precision, NDCG and average recall of the baseline popularity model.

For latent factor model implementation, we used Alternating Least Square (ALS) algorithm which was built in PySpark library. It factors the utility matrix into user-to-item matrix  $U$  and item-to-user matrix  $V$ . The ALS algorithm uncovers the latent factors that explain

the observed users to item ratings and tries to find optimal factor weights to minimize the least squares between predicted and actual ratings. It's also designed to run in parallel fashions which significantly contributes to the efficiency of training. We fit the model and predict the movies for all the users in our test set and calculate the Ranking Metrics to evaluate our Model.

#### 4. Evaluation Metrics

We have tried to adopt all common evaluation metrics for recommender systems, in order to present an overall performance measurement. Here is a brief list and interpretation of metrics used:

- **Precision at k:** Number of the top k recommended items which are in the set of true relevant item sets, averaged across all users (the order of recommendation is not relevant).
- **MAP:** Mean Average Precision, number of all recommended items which are in the set of true relevant item sets, averaged across all users. Penalty is assigned according with order.
- **NDCG:** Normalized Discounted Cumulative Gain, number of the top k recommended items which are in the set of true relevant item sets, averaged across all users, and considered of orders
- **RMSE:** Rooted mean square error between predicted and ground truth rating, of all possible user-item interactions.

#### 5. Baseline Popularity v/s Latent Factor Model

Table.1 shows the comparison between Baseline Popularity Model and the Latent Factor Model Results on small and large dataset.

Metrics	Baseline	ALS
MAP	0.0381668928989208	0.00041876403532256434
Precision at 100	0.11630769230769232	0.003076923076923077
ndcgAt 100	0.1312073223414145	0.002555138145732898
recallAt 100	0.01441449989212212	0.00019181873933005154

Metrics	Baseline	ALS
MAP	0.007268647780525474	1.1385886586450231e-05
Precision at 100	0.015443804907603737	9.887322551081344e-06
ndcgAt 100	0.01427201628489011	1.865572122533339e-05
recallAt 100	0.008372671003483398	1.581697497323456e-05

**Table.1 Baseline and ALS model comparison on a) Movielens small dataset b) Movielens large dataset**

From the table we can observe that without the hyperparameter tuning the ALS model is not as accurate as the Baseline model although the former is personalized recommending model. So we perform hyperparameter tuning to improve the model.

#### 6. HyperParameter Tuning

We used grid search to tune 2 hyperparameters: regularization parameter and ranks of latent factor matrices. To tune our model, we created an als model, fit our model on training dataset and perform transformation on validation dataset and evaluate the model on test dataset. For evaluation metrics, we have tried 2 methods, First is the RMSE function which is within the same pyspark.ml; Second is using ranking metrics(**P@k**, **ndcgAt**, **recallAt** and **MAP**) to determine the best parameters. The latter method gave better results. The parameters used to perform grid search were **rank: [100,125,150,175,200]** and **regParam: [0.01, 0.1, 1, 10]**. With the help of hyperparameter tuning we were able to significantly improve the ranking metrics. It could be seen in the results table below. The results of hyperparameter tuning are in Table. 2 and Table.3. All the results are conducted on test dataset.

(i) Before hyperparameter training: -

Iteration	RMSE	MAP	PA	NDCG	Recall
10	0.9048930872645341	0.00978364643024941	0.03972307692307692	0.07236193227820072	0.10302455581625031
15	0.9208599048883793	0.008719369859523051	0.03529230769230771	0.06736585084566271	0.10179554157618477

(ii)After starting hyperparameter tuning

Iteration	RMSE	MAP	PA <sub>t</sub>	NDCG	Recall
10	1.4920 050997 02937	0.01953 3966626 636874	0.05206 1538461 538485	0.11108 9408892 17304	0.17476 7831943 14033
15	1.4089 751025 23959	0.02320 7688995 88743	0.05193 8461538 46156	<b>0.11713</b> <b>2725578</b> <b>76755</b>	<b>0.17238</b> <b>9212475</b> <b>22646</b>

Table.2 The best rank was 175 and the best regParam was 0.01 on Small Dataset

(i) Before hyperparameter training

Iteration	RMSE	MAP	PA <sub>t</sub>	NDCG	Recall
10	0.7817 637332 880522	0.00114 2633716 6653446	0.00064 3018219 2932087	0.00506 3676005 7387814	0.01751 6983693 89073
15	0.7817 637332 27067	0.00114 2394606 6489875	0.00064 3056247 4568668	0.00506 3116065 795393	0.01751 7013874 97299

(ii)After starting hyperparameter tuning

Iteration	RMSE	MAP	PA <sub>t</sub>	NDCG	Recall
10	0.6911 283941 016878	0.08623 6542702 10764	0.01630 3662492 441907	0.16209 6654394 9737	0.23568 2670967 2368
15	0.6925 147698 10245	0.08787 4126507 4142365	0.01758 2014736 5894147	<b>0.18254</b> <b>6987118</b> <b>876664</b>	<b>0.24258</b> <b>4871151</b> <b>86161</b>

Table. 3 The best rank was 200 and the best regParam was 0.01 on Large Dataset

From the above results we could see, after hyperparameter tuning, the accuracy of our model is significantly improved giving us **ndcg** of **0.12** and **0.18**, similarly for recall **0.17** and **0.24** on small and large datasets respectively.

## 7. Single Machine Implementation

In this section, we introduce lightfm package for recommendation system implementation in Python and compare our experiment results. We also utilized pandas and SciPy in intermediate steps for data structure transformation.

### 7.1 Implementation with LightFM package

lightfm is a package designed for various types of recommender systems. It covers functions for both implicit and explicit feedback models, and can be applied to collaborative filtering models, hybrid models as well as cold-start recommendations.

In this project we only explored the collaborative filtering part of the lightfm package. A summary of implementation steps is listed as follows:

- Input dataframe of interactions, convert to initial utility matrix with `pandas.pivot_table`, then convert to coordinate format with `scipy.sparse.csr_matrix` which is the acceptable format for fitting lightfm models.
- Training and test split with built-in function `lightfm.cross_validation`. We set split ratio as 0.8/0.2 and split into train and test set, corresponding with data splitting method for Spark ALS model.
- Fit dataset into a lightfm instance, with corresponding hyperparameters selected from Section 6. We conducted experiments on Weighted Approximate Rank Pairwise Loss Personalized Ranking for better performance comparison. The time for model fitting was noted and then compared with ALS model.
- Get test score of precision at  $k = 100$  and take mean of it. And compared this with ALS MAP Ranking metric.

### 7.2 Performance comparison with Spark's ALS model

We ran lightfm model on 50% (50418,4) and 100% (100836, 4) of small dataset, with the same hyperparameter combinations as in Spark ALS. Some instant observations are:

- LightFM in comparison with ALS model, gives better performance in terms of both the accuracy and model fitting time, on same dataset and hyper parameters.
- The optimal hyperparameter combinations for Spark ALS and LightFM do not agree. This may result from different splitting methodologies of original dataset
- LightFM and ALS when compared on different dataset sizes produces same trend i.e., LightFM is faster than ALS, but when We tried to run a huge dataset like the Movielens large dataset on both the models. LightFM gave us the error of memory full and were not able to predict movie recommendations whereas ALS model was easily able to process the large dataset. Hence, LightFM is better on small datasets, but we cannot implement this model on huge datasets (which is usually the case in most recommender

systems problems). Also, lightfm only contains limited built-in evaluation metrics. So, this makes ALS preferable when dealing with large datasets.

Comparisons between lightfm and Spark ALS, on mean of precision at k, MAP, time, and accuracy, over ranks and regularization parameter and different max iteration are shown in Table. 4, 5, 6, and 7.

RegParam	Rank 100	Rank 125	Rank 150	Rank 175	Rank 200
0.01	0.386197	0.385590	0.384410	0.386049	0.388393
0.1	0.397918	0.398607	0.398902	0.398328	0.399000
1	0.397820	0.397049	0.398869	0.399902	0.392656
10	0.400918	0.400393	0.400852	<b>0.401115</b>	0.400574

Table. 4 LightFM Epochs = 15 , pAt 100, {"rank": [100,125,150,175,200], "regParam": [0.01, 0.1, 1, 10]}  
Dataset- Whole Small Dataset  
Hyperparameter tuning took 1625.933 seconds  
Best rank: 175, best reg: 10.0  
Evaluation on test data: 0.3957377076148987  
Final model training and fitting took 6.172 seconds

RegParam	Rank 100	Rank 125	Rank 150	Rank 175	Rank 200
0.01	0.383607	0.383180	0.382738	0.380623	0.382344
0.1	0.399131	0.395410	0.397344	0.398836	0.396541
1	0.397754	0.395295	0.395115	0.398656	<b>0.399656</b>
10	0.396902	0.397803	0.398180	0.395918	0.396934

Table. 5 LightFM Epochs = 15 , pAt 100, {"rank": [100,125,150,175,200], "regParam": [0.01, 0.1, 1, 10]}  
Dataset- 50% Small Dataset  
Hyperparameter tuning took 1001.974 seconds  
Best rank: 200, best reg: 1.0  
Evaluation on test data: 0.39436066150665283  
Final model training and fitting took 2.441 seconds

RegParam	Rank 100	Rank 125	Rank 150	Rank 175	Rank 200
0.01	0.013289	0.013139	0.013796	0.013589	<b>0.014587</b>
0.1	0.005627	0.005709	0.005709	0.005849	0.005758
1	0.000009	0.000009	0.000009	0.000009	0.000009
10	0.000004	0.000004	0.000004	0.000004	0.000004

Table. 6 ALS MaxIter= 15 , pAt 100, {"rank": [100,125,150,175,200], "regParam": [0.01, 0.1, 1, 10]}  
Dataset- Whole Small Dataset  
Hyperparameter tuning took 3171.655 seconds  
Best rank: 200, best reg: 0.01  
Evaluation on test data: 0.014543443354773252  
Final model training and fitting took 160.865 seconds

RegParam	Rank 100	Rank 125	Rank 150	Rank 175	Rank 200
0.01	<b>0.013638</b>	0.012732	0.013223	0.012420	0.013303
0.1	0.006451	0.006454	0.006594	0.006539	0.006714
1	0.000006	0.000006	0.000006	0.000006	0.000006
10	0.000048	0.000049	0.000047	0.000049	0.000047

Table. 7 ALS MaxIter= 15 , pAt 100, {"rank": [100,125,150,175,200], "regParam": [0.01, 0.1, 1, 10]}  
Dataset- 50% Small Dataset  
Hyperparameter tuning took 2368.511 seconds  
Best rank: 100, best reg: 0.01  
Evaluation on test data: 0.010160484429774293  
Final model training and fitting took 19.67 seconds

Note, we have used **precision\_at\_k(100).mean()** for **LightFM** and **meanAveragePrecision()** for **ALS**, and we get maximum accuracy of **0.401115** and **0.014587** and minimum time for model fitting, **2.441 seconds** and **19.67 seconds, respectively**. But as mentioned above this is the case for small dataset, when dealing with large datasets LighFM model is not efficient as it need more memory whereas ALS treats all datasets in the same way.