DSGA 1004: Final Project- Recommender System

Group 14: Shriya Murthy Akella (N15762877), Shambhavi Sachin Rege(N15006238)
Github Link: https://github.com/nyu-big-data/final-project-group_14

Introduction

In this project we are building a recommender system using the MovieLens Dataset. We are leveraging big data technologies in order to train our models faster and more efficiently. We have tried to contrast various approaches in order to understand the difference between each. We start off by implementing a popularity-baseline and then try the ALS model.

Splitting Data

In order to split our data into train, validation and test set we have based our idea on the ideology of using historical data to predict the trends of the future data. In order to achieve this we have used the Timestamp feature to sort our data and we have used older data in our training set. We have made sure there is no overlap between test and validation data either. We have achieved this by randomly splitting our users' data into 0.4, 0.3, 0.3(train-val-test). Then we have taken out the values that are less than equal to the 60th percentile of our timestamp and put it back into the training dataset while ensuring that there is no overlap between validation and test data. The users in val and test are also present in the test data.

We have used parquet files in order to save the ml-latest dataset to easily store huge amounts of data.

Choice of Evaluation Criteria

Precision is a fraction of relevant instances amongst the retrieved instances. It is based on the understanding and the measure of relevance. Precision@k is defined as the number of predictions that the model gets right, where k is the number of recommendations per user. Mean average precision@k is the mean of average of all precisions in the instances where a recommendation is a true positive over all the users . Mean precision@k also considers the rank of predictions and is a better suited metric than precision at k. Mean Average Precision does not specify "k". We have chosen these metrics because they not only tell us about the relevant recommendations made to the user but also consider the rank of these recommendations for better clarity.

Evaluation of Popularity-Baseline Model on Small and Full Dataset

First we start off by constructing the popularity baseline model. The idea is to recommend the highest rated movies to all the users. We took the top 100 movies from our training data and recommended these movies to all the users in the val and test set.

The validation results are as follows:

Metric	Small	Large
MAP@100	0.000108855940 27115100	3.928538912960 79E-07
MAP	4.454777750210 49E-05	2.676213626564 06E-07
Precision@100	0.001276595744 680850	4.584768655051 92E-06

The test results of this are as follows:

Metric	Small	Large
MAP@100	5.672535368794 39E-05	5.144010082661 68E-07
MAP	3.486010976829 35E-05	3.364791835323 97E-07
Precision@100	0.001276595744 6808500	3.195202280882 86E-06

Latent Factor Model's Hyperparameters

Models are built and fitted using ALS from pyspark ml library. Top 100 recommendations among all the items are made for each of the users using the function recommendForUserSubset() function to compare with the ground truth label. The groundtruth labels are movies that are rated by each user in the validation set.

We used two hyperparameters to tune the model: rank, which is the number of the user/item latent factors and regularization parameter (reg).

The below table shows the results on the validation set:

Rank	**reg=0.001**	**reg=0.01**	**reg=0.1**
10	0.014734042553191487	0.019521276595744687	0.016542553191489357
20	0.027765957446808515	0.032234042553191492	0.024095744680851062
100	0.039574468085106639	0.045585106382978720	0.028776595744680850
200	0.039095744680851070	0.045744680851063826	0.029255319148936164

Evaluation of Latent Factor Model on Small and Full Dataset

From the table above, we can see that the best performance achieved was for rank 200 and regularization parameter 0.01. So we have used these hyperparameters in order to evaluate our small and large datasets. We can see that there is considerable improvement with respect to the popularity baseline model. This indicates that the latent factor model does a better job than the baseline model.

Metric	**Small**	**Large**
Precision@100	0.038011363636363635	0.016457212956308392
MAP	0.014824395568406569	0.006451061758654441 5
NDCG@100	0.085462402733753 81	0.042735521828359 68

Extension: Comparison to single machine Implementation

We implemented the single machine implementation of light fm on our dataset. “LightFM is a Python implementation of recommendation algorithms for both implicit and explicit feedback. It also makes it possible to incorporate both item and user metadata into the traditional matrix factorization algorithms. It represents each user and item as the sum of the latent representations of their features, thus allowing recommendations to generalize to new items (via item features) and to new users (via user features).” - Light fm documentation.

The dataframe for the user-movie interactions is converted to a sparse matrix representation. The LightFM model is trained on the training split with the weighted approximate rank pairwise loss. It maximizes the rank of positive examples by repeatedly sampling negative examples until rank violating one is found. It is useful in applications where only positive interactions are present and optimizing the top of the recommendation list (precision@k) is desired.

The following are the results for the extension:

Metric	**Small**	**Large**
Precision@100	0.06898305	0.02
Time in seconds	18	240

Compared with the results from the Spark’s parallel ALS Model:

Metric	Small	Large
Precision@100	0.03801136363 6363635	0.01645721295 6308392
Time in seconds	660	5000

We can see that the time taken for the model fitting for the light FM model is lesser than the time taken for fitting the ALS model, for both the small and the large datasets. The precision@100 is also better for both the small dataset and large dataset in case of the light FM over the Spark's parallel ALS model.

Contributions

Shriya Murthy Akella: Partitioning of data, Baseline Model, ALS model and their documentation

Shambhavi Sachin Rege: Partitioning of data, ALS model, LightFM model and their documentation

References

[1] Learning to Rank Sketchfab Models with LightFM(https://www.ethanrosenthal.com/2016/11/07/implicit-mf-part-2/)

[2] LightFM Documentation Examples (https://making.lyst.com/lightfm/docs/examples/movielens_implicit.html)

[3] Spark Documentation: Collaborative Filtering (https://spark.apache.org/docs/3.0.1/ml-collaborative-filtering.html)

[4] Spark Documentation: Evaluation Metrics-RDD-based API (https://spark.apache.org/docs/3.0.1/mllib-evaluation-metrics.html#ranking-systems)