Collaborative-Filter Based Modeling for Movie Recommendation

DS-GA 1004 Final Project Report

Link to GitHub repository: https://github.com/nyu-big-data/final-project-group_1

Yeong Koh
Center for Data Science
New York University
yk2678@nyu.edu

Yoon Tae Park
Center for Data Science
New York University
yp2201@nyu.edu

ABSTRACT

In this final project, we applied the tools we have learned in *DS-GA 1004 Big Data* to build and evaluate a collaborative-filter based recommender system based on a small and a large size **MovieLens** datasets (Harper and Konstan, 2015). We utilized the Spark engine to initially construct a baseline model and later compared its performance with the performance of our ALS recommender system whose hyperparameters were tuned to optimize performance. Additionally, we compared the model with **LensKit** (Ekstrand, 2020)**,** a single-machine implementation, and compared the efficiencies of the two models by measuring the time complexity of model fitting time as a function of data set size and evaluated the models' accuracies.

1 Introduction

A common task of recommender systems is to improve customer experience through personalized recommendations based on the history of explicit and/or implicit feedback. Collaborative filtering is a technique commonly used for recommender systems, which filters out items that a user might like on the basis of reactions by similar users and fills in the missing entries of a user-item association matrix (Hu at el., 2008). `spark.ml` supports model-based collaborative filtering, in which users and items are described by a small set of latent factors that can be used to predict missing entries (spark.ml, 2022). It also uses the alternating least squares algorithm to learn these latent factors that can be used to predict the expected preference of a user for an item (ALS, 2022).

2 Dataset Implementation

Two versions of the dataset were provided for this project: a small sample (ml-latest-small, 9,000 movies and 600 users) and a larger sample (ml-latest, 58,000 movies and 280,000 users).

Both small and large datasets were split into training, validation and test sets by userId. Our training dataset contained full records of movie ratings for 60% of randomly selected users in each dataset. 40% of the users whose entire histories were not selected for training were split evenly into test and validation sets by userId, achieving an overall 60-20-20 split into training, validation and test sets by userId. Then, for the users in the test and validation sets, we did a time-based partitioning where 60% of older records of movie ratings were used for training and the remaining 40% of the data were used for test and validation.

Using this data partitioning strategy, every user had at least some observations in the training set where 60% of the users in each of our datasets had their entire histories observed at training time and the remaining 40% users who belong in the test and validation sets had 60% of their older histories observed at training time. The partitioned datasets were then exported to separate csv files so that the same data are used for all model implementations and our results are reproducible.

3 Evaluation Method

Normalized Discounted Cumulative Gain (NDCG) at k, which measures how many of the first k recommended items are in the set of true relevant items averaged across all users, was used to evaluate

the accuracies of both the popularity baseline and ALS models.

NDCG provides insight on the quality of recommendations by taking into account the position of an element in the list of predictions. The mathematical definition for this metric is the following (NDCG, 2022):

$$NDCG(k) = \frac{1}{M}\sum_{i=0}^{M-1}\frac{1}{IDCG(D_i,k)}\sum_{j=0}^{n-1}\frac{rel_{D_i}(R_i(j))}{\log(j+2)}$$

Where

$$n = \min\left(\max\left(Q_i, N_i\right), k\right)$$

$$IDCG(D,k) = \sum_{j=0}^{\min(|D|,k)-1}\frac{1}{\log(j+2)}$$

Here, M refers to the set of users in the dataset D, which consists of N items, or the ground truth relevant movies, and Q refers to the recommended items in order of decreasing relevance.

Since our evaluations are based on predictions of the top 100 items for each user, k was set to 100.

4 Results

4.1 Popularity Baseline Model

Our baseline model was created by taking the average of the ratings for each movie and recommending the same top 100 movies with the highest mean ratings to all users.

Using the NDCG metrics, the accuracy scores for the small and large datasets were as follows:

Baseline Result

	Validation Score (NDCG)	Test Score (NDCG)
Small	1.4637002341920377e-06	8.27496138413598e-06
Large	1.5136328001142606e-07	4.285425181771657e-07

This method, which produces the same ranking for all users, does not provide enough personalization for each user, but it simply serves as a baseline for the construction of our ALS models as explained in the next section.

4.2 Alternating Least Squares (ALS) Model

Our recommendation model used Spark's parallel alternating least squares (ALS) method from Spark's machine learning library (pyspark.ml). We fit our training data to the ALS model to learn the latent factor representations for users and items, or movies, that can be used to predict the expected preference for each user and make top 100 item predictions.

Hyperparameters We scaled the regularization parameters such as the dimension of the latent factors *(rank)* and regularization parameters *(regParam)* to optimize the model's performance and checked for convergence by observing the model's evaluation scores against the number of maximum iterations *(maxIter)*. The ranges of the hyper parameters that we used to tune our model are the following.

Range of hyper parameters used in small dataset

rank	50	100	200
regParam	1e-05	5e-05	1e-04	5e-04
maxIter	5	10	15

Range of hyper parameters used in large dataset

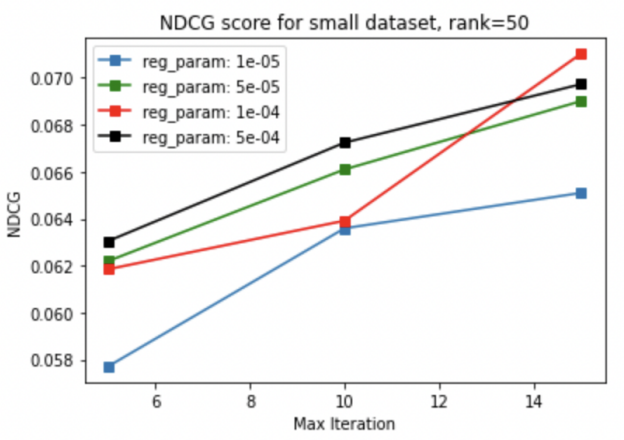
rank	100	200	300
regParam	0.001	0.005	0.01	0.05
maxIter	5	10	15

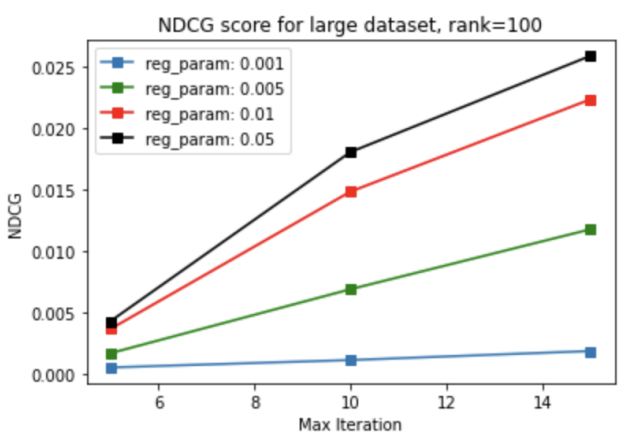
After performing a grid search over the above parameters for the two datasets, we found the parameters that achieved the highest accuracy score based on the Normalized Discounted Cumulative Gain (NDCG) at k metrics to have a rank of 100 and regParam of 1e-04 at maxIter of 15 for the small dataset and a rank of 300 and regParam of 0.01 at maxIter of 15 for the large dataset.

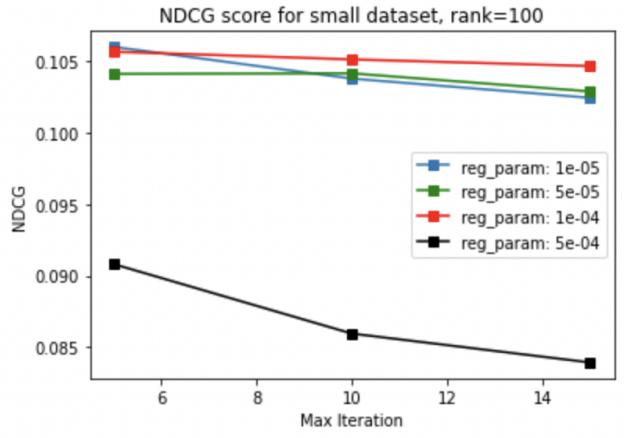
Best hyper parameter setting for small and large datasets

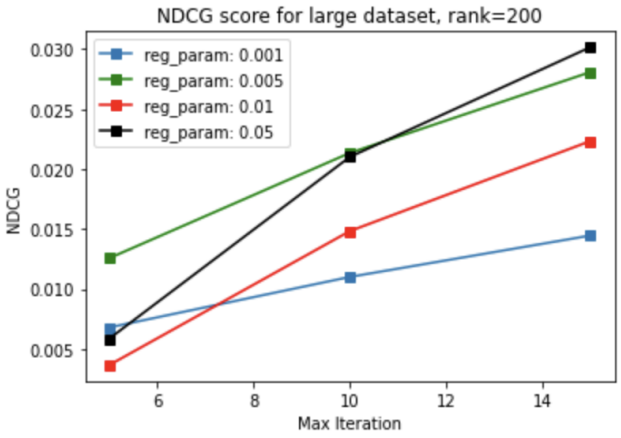
	rank	regParam	maxIter
Small	100	1e-04	15
Large	300	0.01	15

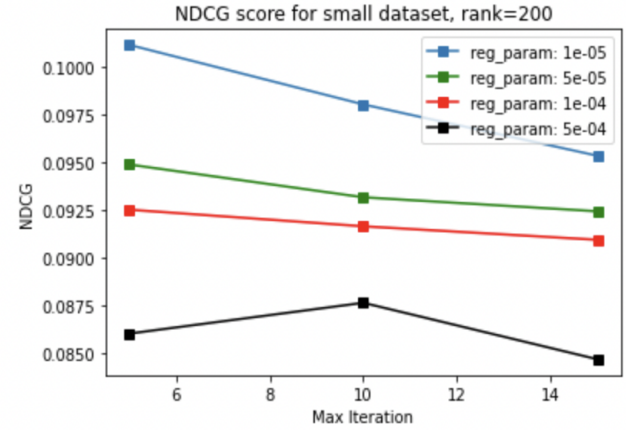
Based on the optimal values of hyper parameters to our ALS model, accuracy scores (NDCG) for our validation set were 0.10462 for the small dataset and 0.04155 for the large dataset. We also found that this result converges, as increasing maxIter increases the accuracy score but slowly, and may continue to converge as maxIter gets larger. A summary of our results can be found in the plots below.

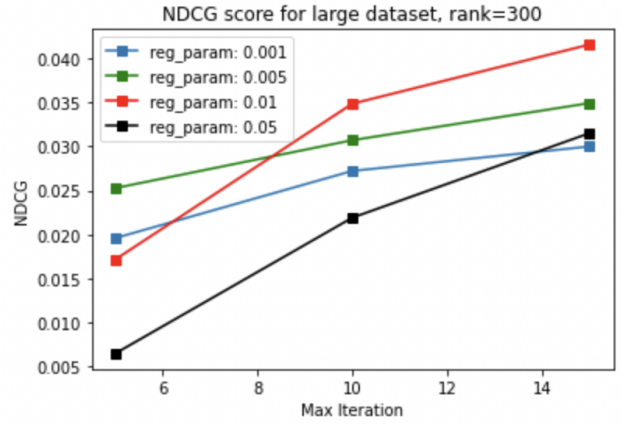

NDCG score for small dataset, rank=50
reg_param: 1e-05
reg_param: 5e-05
reg_param: 1e-04
reg_param: 5e-04
NDCG
Max Iteration


NDCG score for large dataset, rank=100
reg_param: 0.001
reg_param: 0.005
reg_param: 0.01
reg_param: 0.05
NDCG
Max Iteration


NDCG score for small dataset, rank=100
reg_param: 1e-05
reg_param: 5e-05
reg_param: 1e-04
reg_param: 5e-04
NDCG
Max Iteration


NDCG score for large dataset, rank=200
reg_param: 0.001
reg_param: 0.005
reg_param: 0.01
reg_param: 0.05
NDCG
Max Iteration


NDCG score for small dataset, rank=200
reg_param: 1e-05
reg_param: 5e-05
reg_param: 1e-04
reg_param: 5e-04
NDCG
Max Iteration


NDCG score for large dataset, rank=300
reg_param: 0.001
reg_param: 0.005
reg_param: 0.01
reg_param: 0.05
NDCG
Max Iteration

Evaluation on Test Sets We evaluated our test dataset using the best hyperparameter setting that resulted in the highest scores on our validation datasets. NDCG scores for test datasets were 0.10054 and 0.04122 for the small dataset and large dataset, respectively.

Spark ALS: NDCG on best hyper parameter setting

	Validation Score (NDCG)	Test Score (NDCG)
Small	0.104622195435061	0.100537868622676
Large	0.0415544526529172	0.0412236825750406

Compared to the popularity baseline models, our ALS model offers a higher degree of personalization as indicated by the improvement in the accuracy scores for both datasets.

5 Extension (Single-Machine Implementation)

In this section, we conducted a comparison of Spark's parallel ALS model to LensKit(Ekstrand, 2020), which is a single-machine implementation.

Dataset To ensure that the datasets used in Spark's parallel ALS model and LensKit are the same, we imported the partitioned datasets that were used for Spark ALS model implementation to build our new models.

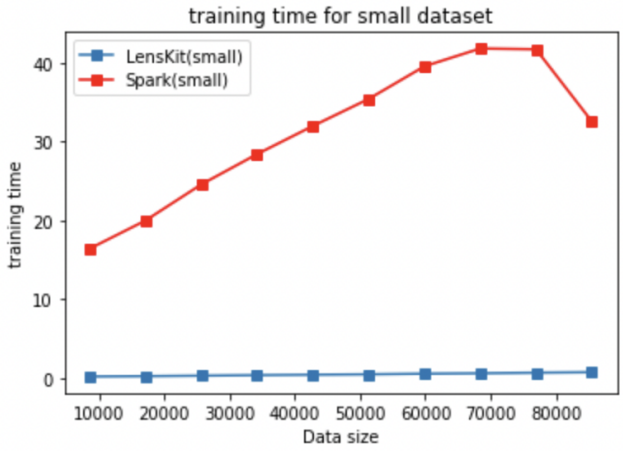
Model To make the comparison more reliable, we used the ALS model provided by LensKit. LensKit's ALS model requires the same data schema for constructing and evaluating the model.

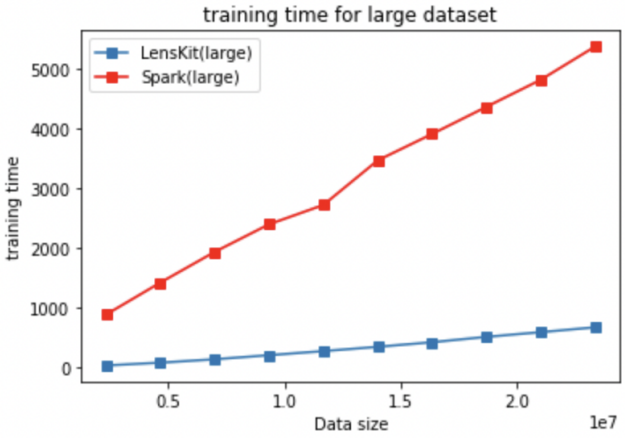
Hyper parameters For hyper parameters, we used the best parameter setting conducted in Spark's ALS model. This makes the comparison more reasonable as the setup for LensKit is exactly the same as Spark. Therefore, we used a rank of 100, regParam of 1e-04, with maxIter at 15 for the small dataset and a rank of 300, regParam of 0.01, with maxIter at 15 for the large dataset.

Evaluation metric To evaluate efficiency, we measured the time it takes to fit an ALS model at various sizes of the training dataset. For accuracy, we used NDCG, to make the analysis consistent with Spark's ALS model.

5.1 Model Efficiency

We measured model fitting time as a function of training data set size. As shown in the plots below, we can see that the time it takes to fit an ALS model scales linearly with the dataset size, for both Spark and LensKit. For small dataset, training time in Spark showed some efficiency in full dataset, but this may be due to the relatively small size of the entire dataset. Comparing Spark to LensKit, LensKit outperformed in training time for both small and large datasets. This may imply that for the datasets used for this project, single-machine implementation outperforms Spark in model efficiency.





5.2 Model Accuracy

We measured model accuracy by computing NDCG scores for both Spark and LensKit's ALS model. As shown in the tables below, we found that Spark showed better performance compared to LensKit. There could be some possibilities that the ALS model and NDCG logic on both Spark and LensKit are different. However, this result is quite surprising, as all settings were held the same, and we expected the result to be similar as well.

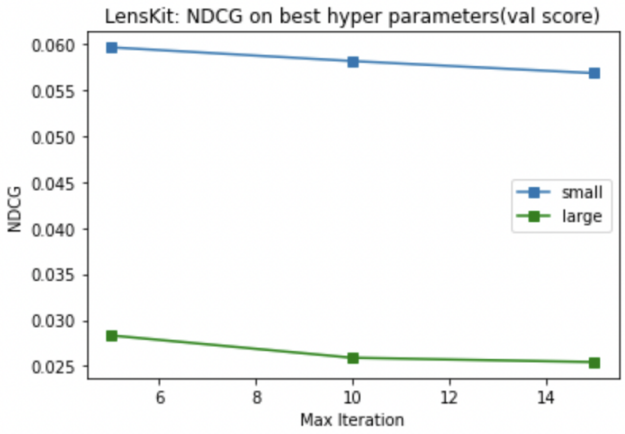
Model accuracy on small dataset

	Validation Score (NDCG)	Test Score (NDCG)
Spark(ALS)	0.104622195435061	0.100537868622676
LensKit(ALS)	0.056810394150239	0.075083593589482

Model accuracy on large dataset

	Validation Score (NDCG)	Test Score (NDCG)
Spark(ALS)	0.0415544526529172	0.0412236825750406
LensKit(ALS)	0.0254438621689313	0.0254189041756357

Also, we've checked for convergence of our Lenskit ALS models. In the below plot, we can see that our model converges as max iteration increases for both datasets.



6 Conclusion and Future work

Our analysis found that compared to the popularity baseline models, our ALS recommender system with optimized hyper parameters outperformed in accuracy scores for both small and large datasets. We also found that LensKit, a single-machine implementation, is more efficient in training time compared to Spark's parallel implementation, but its accuracy score is lower than Spark's ALS recommender system. This suggests that LensKit can be a good tool for light experiment on constructing recommender systems.

In the future, it could be helpful to improve our model's efficiency by adopting fast search methods, using a spatial data structure. Another improvement would be improving accuracy score by adding a cold-start scenario. This can be done by using supplementary metadata, such as tags, genres, etc. Finally, it could be helpful to conduct qualitative error analysis, investigating the mistakes on our model. This can be done in numerous ways, such as investigating the user with the lowest-scoring predictions, visualizing the learned item representation by dimensionality reduction techniques, and clustering users and looking for common trends in each cluster's consumption behavior.

Collaboration Statement

All team members contributed roughly equally to the project. All members worked on partitioning the rating data into training, validation, and test samples, and building a popularity baseline model for both small/large datasets. Yeong Koh contributed to constructing Spark's alternating least squares (ALS) method in both small/large datasets, hyper parameter tuning the model, and building a ranking-metric for all models. Yoon Tae Park contributed to constructing ALS model on single-machine implementation by using LensKit, and measuring both efficiency and resulting accuracy on Spark's parallel ALS model to a single-machine implementation.

REFERENCES

[1] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent \Systems (TiiS) 5, 4: 19:1–19:19.

[2] Michael D. Ekstrand. 2020. LensKit for Python: Next-Generation Software for Recommender Systems Experiments. In Proceedings of The 29th ACM International Conference on Information and Knowledge Management (CIKM '20). ACM, New York, NY, USA, 8 pages.

[3] Y. Hu, Y. Koren and C. Volinsky. 2008. Collaborative Filtering for Implicit Feedback Datasets. Eighth IEEE International Conference on Data Mining, 2008, pp. 263-272, doi: 10.1109/ICDM.2008.22.

[4] spark.ml, 2022. Spark Machine Learning Library (MLlib) Guide.

[5] ALS, 2022. Spark ALS model documentation.

[6] NDCG, 2022. Spark Evaluation Metrics - RDD-based API