Final Project Report

DSGA 1004 Big Data

Linxia Li
Data Science
New York University
New York NY United States
ll4764@nyu.com

Xinyu Guo
Data Science
New York University
New York NY United States
xg693@nyu.com

Yuqin Wang
Data Science
New York University
New York NY United States
yw3875@nyu.com

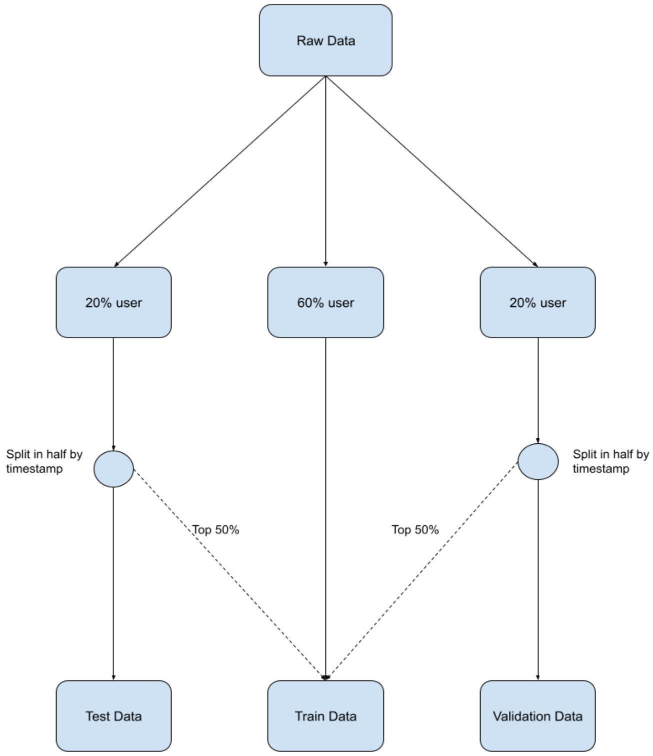
ABSTRACT

In this project, we focused on implementing models to build recommendation systems based on movie rating datasets, including the popularity baseline model and latent factor model. We also did extensions including the comparison to single-machine implementation and qualitative error analysis. We were writing our code using spark on peel. Our project is evenly divided as follows: All group members preprocessed data and discussed how to set evaluation criteria together; Then, Xinyu focused on building the popularity baseline model; Yuqin focused on building the ALS model; Linxia focused on completing two extensions. Finally, all group members get together and construct the final report. In the following sections of the report, we would discuss the processes in more detail. Our code is available on the GitHub repository: https://github.com/nyu-big-data/final-project-group_52

Data Preprocessing

We have two data sets to work on, a small one and a large one. Both have the same structure, so we process them in the same way. We first remove rows of the users who rated no more than 10 movies and rows of the movies having no more than 10 ratings to only include a large subset of users and ensure the robustness of the results. Then, we sorted each dataset based on the timestamp of the rating for each user for the preparation of the data partition in the next step.

For data partition, we split the user identities into training, validation, and test. Specifically, at the outset, we select the 60% of user identities into the training set. Then, we split the rest of the user identities in half and place them separately in the validation set and test set. Since our timestamp order has been ascending, we move the top 50% records of each user from the validation and test group back to the training set because it is more reasonable to train the model with earlier data and compare the predictive result with later action. In this way, users in the validation and test set are also in the training set. Otherwise, we would not be able to fully test user behavior.



Below is our workflow for splitting data:

Figure 1: **Working flow of train-val-test split.**

Beta Selection

We build a baseline model and an ALS model for our recommender system. For the baseline model, we introduce a constant parameter called beta, which represents the extra observations with the utility matrix.

$$P[i] \leftarrow \left(\sum_u R[u, i]\right) / \left(|R[:, i]| + \beta\right)$$

Figure 2: **Function of the popularity baseline.**

Choice of evaluation criteria

1.1 Baseline Model

For baseline mode, we selected Mean Average Precision (MAP) and Root Mean Square Error (RMSE) as our evaluation criteria. MAP is defined as the measure of the number of the recommended documents that are in the set of true relevant documents, where the order of the recommendations is taken into account [1]. To implement the function for MAP, we followed its definition and compare the rating of each movie per user to the top 100 movies suggested by the baseline model: if that movie's rating by the user is higher than the overall average rating of the top 100 movies, we assign it with a relevance score = 1; otherwise, we assign it with a relevance score = 0 and calculate the true ratio.

$$MAP = \frac{1}{M}\sum_{i=0}^{M-1}\frac{1}{N_i}\sum_{j=0}^{Q_i-1}\frac{rel_{D_i}(R_i(j))}{j+1}$$

$$rel_D(r) = \begin{cases} 1 & \text{if } r \in D, \\ 0 & \text{otherwise.} \end{cases}$$

Figure 3: **Functions of the ALS and RMSE.**

RMSE is the standard deviation of the prediction errors, which refers to how concentrated the data is around the line of best fit [2]. Similar to our implementation of MAP, RMSE is also implemented based on the rating record of each user. We first find the movies rated by the user that is also in the list of top 100 movies. Next, we compute the squared residuals between each user's movie rating and the average rating of the same movie. Then we average the sum of the squaring results we got, and take the square root of it to get RMSE.

1.2 ALS Model

We select Mean Average Precision (MAP), Precision at k, and Normalized Discounted Cumulative Gain (ndcg) as our evaluation criteria for the Alternating Least Squares (ALS) model. ALS model factors the user-to-item matrix A into the user-to-feature matrix U and the item-to-feature matrix M. The unknown row dimension is given as a parameter to the algorithm and is called latent factors. The number of latent factors is defined as rank, which implies the number of features the model uses [3].

MAP is the same as the one used in the baseline model. However, instead of writing the specific algorithm for implementation, we directly call the function of MAP from the package pyspark.mllib.evaluation.RankingMetrics directly.

Precision at k is a measure of how many of the first k recommended documents are in the set of true relevant documents averaged across all users. NDCG at k is a measure of how many of the first k recommended documents are in the set of true relevant documents averaged across all users. Similar to MAP, both metrics are called directly from the package pyspark.mllib.evaluation.RankingMetrics.

Hyper-parameter Tuning

For the baseline model, we try beta from the interval [0.1, 0.15], which is separated ten times evenly, and apply each of them to the validation dataset. The MAP scores stay constant regardless of the value of beta, but the RMSE score is lowest at beta = 0.1222 (RMSE = 0.9077), so we chose it as our beta for the testing data set. Finally, the testing data reaches 0.0030 for MAP score and 0.9118 for RMSE score.

For the ALS model, we try various combinations of rank and regParam and tune the parameters that yield the best result. According to the results, the best rank is 15, and the best regParam is 0.05. Below is our documentation of the latent factor model's hyper-parameter:

Hyperparameter					
rank	regParam	Dataset	MAP	Precision at 100	NDCG
5	0.05	Val	0.0429	0.034	0.1666
		Test	0.0391	0.0332	0.1611
5	0.1	Val	0.0435	0.0339	0.1672
		Test	0.0406	0.0335	0.1628
5	0.5	Val	0.0425	0.0329	0.1618
		Test	0.0402	0.0331	0.1606
10	0.05	Val	0.0456	0.0345	0.1761
		Test	0.0484	0.0346	0.1798
10	0.1	Val	0.0453	0.0349	0.1768
		Test	0.0488	0.0348	0.181
10	0.5	Val	0.0437	0.0355	0.1741
		Test	0.0463	0.0353	0.1775
15	0.05	Val	0.0464	0.0347	0.1768
		Test	0.0464	0.0345	0.177
15	0.1	Val	0.0456	0.035	0.1764
		Test	0.0447	0.0347	0.1747
15	0.5	Val	0.0437	0.0356	0.1741
		Test	0.0449	0.359	0.1761

Table 1: **Tuning results of the ALS model.**

Evaluation

2.1 Popularity Baseline Model on small and full datasets

We try 10 betas ranging from *np.linspace(0.1, .15, 10)* We finally choose the beta of 0.15 for the small dataset and choose beta of 0.1222 for the full dataset because they would bring the lowest RMSE and highest MAP during the parameter tuning step. We still can observe that the baseline model for the recommender system is not good. The reason behind this is that the baseline model is not personalized. We recommend the same movie list to everyone, which would result in a higher RMSE.

	MAP	RMSE
small_val	0.0027	0.8989
small_test	0.0026	0.8915
full_val	0.0031	0.9077
full_test	0.0030	0.9118

Table 2: **Scores of the popularity baseline model.**

2.2 ALS Model on small and full datasets

Here we have our documentation of results for the ALS model. As we can see, the MAP in the ALS model is higher than the baseline model, which indicates that ALS performs better.

	MAP	Precision at k	NDCG
small_val	0.0437	0.0356	0.1741
small_test	0.0449	0.0359	0.1761
full_val	0.0463	0.0347	0.1815
full_test	0.0458	0.036	0.1803

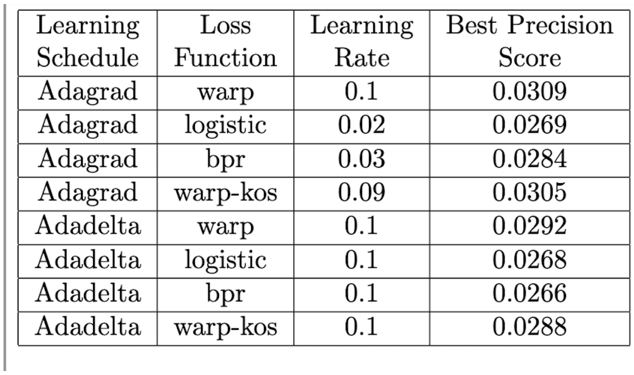
Table 3: **Scores of the ALS model.**

Extension

3.1 Comparison to Single-Machine Implementation: *LightFM*

To compare how the recommender system performs with the different numbers of machines used, we implement LightFM. LightFM is a Python algorithm that allows us to incorporate both item and user data into the traditional matrix factorization algorithms, making it possible to generalize to new items (via item features) and new users (via user features) [4]. Implementing the LightFM, we try different loss functions, including logistic, BPR (Bayesian Personalized Ranking 1 pairwise loss), WARP (Weighted Approximate-Rank Pairwise loss), and k-OS WARP (k-th order statistic loss). The learning rate of the model was adjusted from 0.01 to 0.1, increased by 0.01 for each trial. We also try learning schedules of adagrad and adadelta. The result is shown in Table 1 below. It implies that with WARP as the loss function, learning rate = 0.1, and adagrad as learning schedule, we reach the highest precision score at k = 100, which is 0.0309. Therefore, we select it as our LightFM model.

Working on the training data, we choose the number of threads from 1 to 9 and fit the LightFM model to each of them. The result is shown in Table 2 below. It shows that lightfm trains the fastest with 7 threads computing parallelly, completing with only 0.820 seconds. Hence, we finally apply the 7-thread LightFM model to both the training data and the testing data. The former achieves a precision score = 0.0305 and the latter achieves a precision score = 0.0314.



Learning Schedule	Loss Function	Learning Rate	Best Precision Score
Adagrad	warp	0.1	0.0309
Adagrad	logistic	0.02	0.0269
Adagrad	bpr	0.03	0.0284
Adagrad	warp-kos	0.09	0.0305
Adadelta	warp	0.1	0.0292
Adadelta	logistic	0.1	0.0268
Adadelta	bpr	0.1	0.0266
Adadelta	warp-kos	0.1	0.0288

Table 4: **Comparison of result accuracy.**

Number of Threads	Running Time (seconds)
1	1.8552
2	1.2853
3	1.1369
4	1.0538
5	1.0321
6	1.1724
7	0.8201
8	1.0724
9	1.1281

Table 2: **Comparison in efficiency.**

3.2 **Qualitative Error Analysis: *UMAP***

UMAP is aiming to generate a high-dimensional graph representation of the dataset and optimize a low-dimensional graph to be similarly constructed. We used UMAP to reduce the dimension and visualize the latent factor model. The figure below shows the UMAP visualization of the ALS model, clustering by "genre". When we choose n_neighbors = 15 and min_dist = 0.1, we can see that purple points tend to locate in the lower left, while green points tend to locate in the upper right corner. It shows that the latent factors are clustering by their genre.

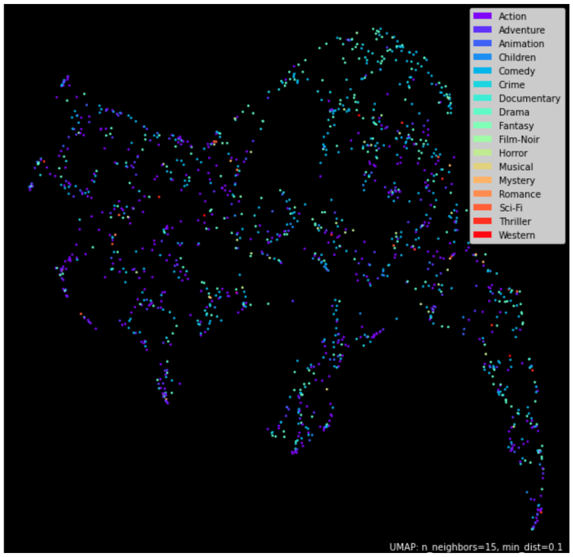


Figure 4: **Result of UMAP.**

Conclusion and Future works

According to our results, It is hard to build up a perfect recommender system and predict movie preferences without further personalization. The latent factor model generates a better prediction compared to the popularity baseline mode for both the small and full datasets. And both models perform better on larger datasets than on smaller datasets. The reason could be that when we have more information, i.e. historical data, we would have better knowledge and understanding of the system and thus will be more likely to capture the future.

A great limitation we face is the largest memory and the computing ability of the machine we use, especially in the case of the large data set. Since for each set of data, we need to firstly split the data into the training data, validation data, and the testing data based on user ID before building the model, we need enough memory space to store the data sets and need strong computing ability to process the train-test split by user. Similarly, the evaluation of the models we build also needs the machine to deal with a great amount of computation. Currently, both of the tasks take us hours to complete. If we have access to machines with a larger memory capacity and better computing ability, we can tune our models better by justifying more hyperparameters. Also, currently we only include the rating record of the users in our model. With a higher-quality machine, we are able to include more information in our recommender system, such as the links, tags and genres of the movies, which would definitely improve the performance of each model.

REFERENCES

[1] "Evaluation Metrics - RDD-Based API." Evaluation Metrics - RDD-Based API - Spark 3.0.1 Documentation, https://spark.apache.org/docs/3.0.1/mllib-evaluation-metrics.html#ranking-systems.

[2] RMSE: Root mean square error. Statistics How To. (2021, May 31). Retrieved May 17, 2022, from https://www.statisticshowto.com/probability-and-statistics/regression-analysis/rmse-root-mean-square-error/

[3] *MLlib - Evaluation Metrics.* Evaluation Metrics - MLlib - Spark 1.5.0 Documentation. (n.d.). Retrieved May 17, 2022, from https://spark.apache.org/docs/1.5.0/mllib-evaluation-metrics.html

[4] Kapadia, S. (2020, February 26). Recommendation system in python: Lightfm. Medium. Retrieved May 17, 2022, from https://towardsdatascience.com/recommendation-system-in-python-lightfm-61c85010ce17