Recommender System for MovieLens Dataset

CECILIA WU, LUCAS TAO, BILLY SUN, Center for Data Science, New York University

In this project, our team leveraged Spark engine to build a recommendation system for movies based on users' explicit feedback. The system was built using the baseline popularity model and the latent factor model. By tuning hyperparameters of regularization, alpha, maxIter, and rank, we identified the best performing Alternating Least Squares(ALS) model in our case and compared performance with that of baseline. On top of that, we leveraged other strategies such as fast search, and quantitative error analysis to further improve the performance and visualize models' output.

ACM Reference Format:
Cecilia Wu, Lucas Tao, Billy Sun. 2022. Recommender System for MovieLens Dataset. 1, 1 (May 2022), 6 pages.

1 INTRODUCTION

With an explosion of user behavior related data, both explicit and implicit, recommendation utilizing big data techniques to provide personalized item suggestions has been more and more widely adopted. Compared to traditional baseline model, where suggestions are universal, matrix factorization based method aims to provide customized recommendations fulfilling different people's tastes. By factorizing the original interaction matrix into a user matrix (rows for users and columns for latent factors), and a item matrix (rows for latent factors and columns for items), we essentially decompose the interaction matrix into the product of two rectangular matrices of lower dimensionality. Similar to the idea of principle components in the PCA method, the number of latent factors decides how much abstract information is contained in the factorized version - having more latent factors help us secure more information, but more likely to observe fittingm while fewer latent factors prevents overfitting, but would mean less power for personalized recommendation. The repository of our code can be found here https://github.com/nyu-big-data/final-project-group_40.

2 DATA PREPARATION

There were several common data splitting strategies in recommendation models mentioned in Meng el al. including Leave One Last, Random Split, and Temporal User Split. Leave one last split takes the last rating by timestamp of each user into the validation and random split randomly selects the training and testing for each user. Temporal user split is similar to leave one last but takes a percentage of the last interactions of each user to testing/validation. Our data splitting strategy was based on the temporal user split because it provides more stable validation by take a percentage of the ratings instead of one and respect the temporal order unlike the random split.

The dataset was split into training, validation, and testing set. Aviod-ing the user cold start problem, all the users in validation and test were in the training set. To respect the temporal order, the training data was generated by taking the earliest 70% of the ratings of each user with respect to the timestamp. If a user had less than 4 ratings, all of his or hers rating would be put in to training dataset. The remain 30% of the data was split in half in terms of the users.

Author's address: Cecilia Wu, Lucas Tao, Billy Sun, Center for Data Science, New York University.

3 COLLABORATIVE FILTER

3.1 Baseline

The baseline model used was the popularity based model from the Lecture. We modeled the user and movie interaction as R[u,i].u denoted the userID and i denoted the movieID. The prediction for user u was made by taking the first 100 items sorted by descending items/user interaction score R[u,i]. Below is the formulas used in the popularity model.

$R[u,i] = \mu + b[i] + b[u]$

$\mu = \frac{\sum_{u,i} R[u,i]}{|R|+\beta_g}$

$b[i] = \frac{\sum_u R[u,i]-\mu}{|R[:,i]|+\beta_i}$

$b[u] = \frac{\sum_i R[u,i]-\mu-b[i]}{|R[u]|+\beta_u}$

μ is the average rating over the all user interactions. b[i] is the average users difference from average rating μ for item i. b[u] is the average items difference from the sum of average rating and average user difference (μ + b[i]) for user u.

β is a damping factor or prior that makes the model favor items with more ratings. In our model, β_g, β_i, β_u = 1

3.2 Alternating Least Squares

In our work, we used Alternating Least Squares (ALS) as our matrix factorization method. Slightly different from other similar methods such as Funk SVD, ALS uses L2 regularization, and it minimizes two loss functions in an alternating fashion. Specifically, it first holds user matrix unchanged and conducts gradient descent on item matrix, and performed the same trick again vice versa - holding item matrix unchanged and conducted gradient descent on user matrix. Leveraging hyperparameter tuning such as adding punishment by modifying the regularization term, using different number of latent factors by twisting the rank, as well as other factors help us achieve the best balance. Lastly, evaluations for different configurations are provided and compared.

4 HYPERPARAMETER TUNING

4.1 Find the Best Hyperparameters

We aim to explore rank, regularization, alpha, and max-iteration parameters of the ALS modelusing evaluation Metrics Precision at K, Mean Average Precision (MAP), and Normalized Discounted Cumulative Gain (NDCG) as metrics.

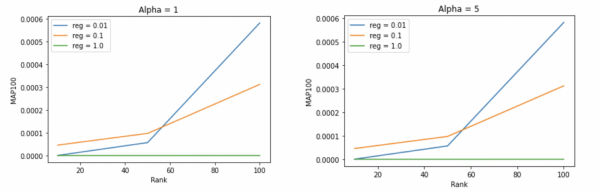
With an educated guess, we first performed a parameter search on a combination of hypyerparameters shown in Table 1.

Table 1. **First Grid Search Run**

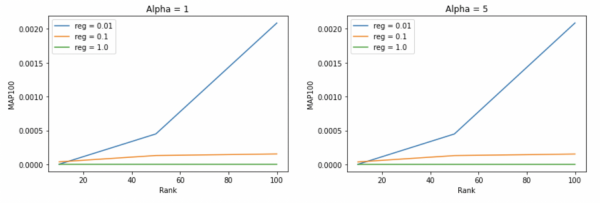
rank	10,50,100
regParam	0.01,0.1,1
maxIter	5, 10,15
alpha	1,5

For illustration purpose, we included the performance comparison for cases when maxIter = 5, 10, 15, which was our initial attempt. We further tuned hyperparameters based on the performance of our initial attempt.

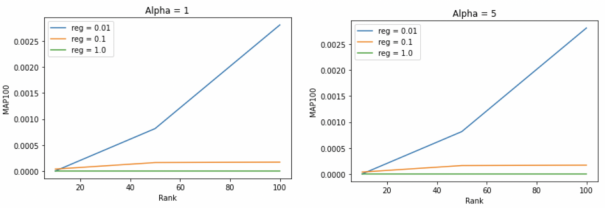
When maxIter = 5:



When maxIter = 10:



When maxIter = 15:



We observed model performance increases with greater rank, max iteration and smaller regularization; we verified that alpha indeed has no performance change. Since alpha governed the baseline confidence in preference observations of implicit feedback, we had no implicit feedback in MovieLens dataset, thus model performance was not depended on alpha. Nonetheless, we found out that smaller alpha improves runtime efficiency. Setting alpha=1 instead of alpha=5 expedites runtime by 50 percent when holding other parameters unchanged.

Due to resource constraints and in consideration of others using the cluster, we did not have the leisure of testing rank by a fixed increment (e.g. 50). Instead, we proposed to explore whether rank was dependent on user or movie size of the dataset. Borrowed from common practices of feature selection, we tested out small and full datasets with rank proportional to $\sqrt{user\ size}$ and $\sqrt{item\ size}$ according to Table 2.

Table 2. **sqrt(n) w.r.t size of user item on small full dataset**

approximate rank	small	full
user	$\sqrt{610} = 24 \approx 25$	$\sqrt{283228} = 532 \approx 500$
movie item	$\sqrt{9742} = 98 \approx 100$	$\sqrt{58098} = 241 \approx 250$

Table 3. **Rank VS. Size with MaxIter 15, regParam 0.01**

Data	Rank	PrecisionAt100	MAP100	NDCG100
small	25	0.02531	0.00699	0.05104
small	100	0.02613	0.01188	0.07158
full	500	0.01474	0.00612	0.04505
full	250	0.01374	0.00578	0.04105

Our preliminary results in Table 3 showed that greater rank always gave better model performance, at least for our dataset. We had to reject our hypothesis that parameter rank might be dependent on user or movie size.

From results of the initial attempt, we implemented a greedy approach to raise max iteration to 20, 25 and 30, and lower regularization to 0.001. The best model performance was found at: maxIter = 30, regParam = 0.01, and rank = 500. Based on the best configuration we obtained as above, we tested our model's performance on our test set. Root Mean Square Error (RMSE) was checked for potential overfit. Results from our best ALS parameters greatly outperformed that of our baseline model.

Table 4. **Performance of Best Parameter**

Data	PrecisionAt100	MAP100	NDCG100	RMSE
val	0.01431	0.00615	0.04577	0.88931
test	0.01474	0.00612	0.04505	0.88919

Table 5. **Comparison of 0.01 and 0.001 reg in Rank 250 with 15 maxIter**

regParam	PrecisionAt100	MAP100	NDCG100
0.01	0.01107	0.00406	0.03096
0.001	0.00721	0.00341	0.02246

4.2 Performance of baseline model

	Small Test Set	Large Test Set	Small Validation Set	Large Validation Set
Precision at 15	0.0375956284153005	0.0260351671884753	0.0424043715846994	0.0260221010843159
Precision at 25	0.0319999999999999	0.0191833139659677	0.0354098360655737	0.0191603958049416
Precision at 100	0.0121639344262295	0.0069795051329687	0.0133770491803278	0.0069108105705990
MAP at 15	0.0135201349955448	0.0075160385323441	0.0180145658834183	0.0075601142847988
MAP at 25	0.0096144119924909	0.0048957467170430	0.0122820741247914	0.0049200717696348
MAP at 100	0.0026832507184029	0.0013482747269400	0.0033829110034699	0.0013484062376329

4.3 Compare performance between baseline and ALS

As we could easily tell by comparing the results to Table 5, ALS with the best hyperparameters completely outperformed the performance of baseline model, with MAP100 = 0.006, while MAP100 = 0.0013 for baseline model. It proved that with the appropriate hyperparameters selected, ALS model was able to provide more customized recommendations for users. However, although we had made such improvement, we didn't believe this is the converged result for our ALS model. Due to time contraints and limited computing resources we have, we didn't have time to further tune up the hyperparameters of maxIter and rank.

5 EXTENSION

5.1 Fast Search

For the fast search, the Annoy (Approximate Nearest Neighbors Oh Yeah) was used to enabling fast inference compare to brute force inner product. The Annoy library was particularly used to search for similar user/items for matrix factorization recommendation algorithms. The number of trees was a hyper-parameter for Annoy. The higher number of trees meant higher accuracy compared to brute force and longer inference time. The hyper-parameter (number of trees) for Annoy was tuned so that the precision was approximately same as brute force search.
Precision = Recommended Movie that user has interaction/ Recommended Movie
For inference, 100 user from validation set is randomly selected. The average inference time per user and the average precision was calculated.

Table 6. **Brute Force Search**

Model	Average Inference Time	Average Precision
Rank 10	0.000911	0.0038
Rank 100	0.004093	0.0013
Rank 250	0.005727	0.0031
Rank 500	0.011278	0.0125

Table 7. **Annoy Search**

Model	Average Inference Time	Average Precision
Rank 10 tree = 5	0.000094	0.0034
Rank 100 tree = 8	0.000103	0.0025
Rank 250 tree = 10	0.000182	0.0033
Rank 500 tree = 100	0.003186	0.01125

5.2 Latent Representation Visualization

We extended our project scope to visualize the high-dimensional learned item representation generated from our best ALS model. Two methods, T-distributed Stochastic Neighbor Embedding (t-SNE) and Uniform Manifold Approximation and Projection for Dimension Reduction (UMAP), were implemented with the help of sklearn and umap-learn packages.

5.2.1 Parameters.

(1) t-SNE
- number of components = 2
- perplexity = 30
- early exaggeration = 12
- learning rate = 'auto' = max(sample size/ early exaggeration/ 4, 50)
- number of iteration = 1000
- number iteration without progress before abortion = 100
- initialization embedding = 'pca'
- distance metric = 'euclidean'

(2) UMAP
- number of components = 2
- number of neighbors = 200
- minimum distance = 0.8
- distance metric = 'euclidean'

5.2.2 Cluster by Genre Tag. Our initial attempt was to color each item by a genre of the most significant movie-genre relevance score. However, the tags.csv dataset had 1128 uniquely tagged genres and map to unique 834 genres in our best model item representation matrix. The genre groups were indistinguishable in a 2-D scatter plot, so we ought to cluster by smaller groups. One area of future work would be grouping genres by word match or relevance score; for instance group tags "alien", "aliens", and "alien invasion" into one genre.

5.2.3 Cluster by Popularity. We defined popularity by median rating of each movie item to color-code dimension reduction scatter plots Figure 1 and Figure 2. A facet grid of scatter plots for t-SNE was also included in the appendix Figure 3

From both plots, we could observe that less popular movies (colored in red and orange hue) skew to the left of the spectrum, while more popular movies scatter to the right. The UMAP embedding Figure 2 has five neighborhoods outside of the mean cluster. The cluster on the left upper corner is mostly occupied by "mediocre movies" of around 3.0 median ratings, while the upper center cluster is composed of movies around 4.0 median rating.

6 CONCLUSION AND FUTURE WORK

Based on the results, we found that the baseline model would be an entry model to consider when providing recommendations. However, if we were trying to provide more personalized recommendations, as well as considering scalability, matrix factorization based model would definitely be a better choice. In our exploration, we spent a lot of time tuning hyperparameters to find the best configration. And even before we finalized this paper, we had been getting better results as we tuned up maxIter and rank of ALS model. This proved that ALS models would have a much higher performance ceilings compared to the baseline model if given enough time and resources, although how we partitioned the dataset also played a role of defining such ceiling.

Regarding future work, we could try different partition methods. We considered timestamp in our case, which is good. However, we were not sure if we selected the right proportion of data to put into

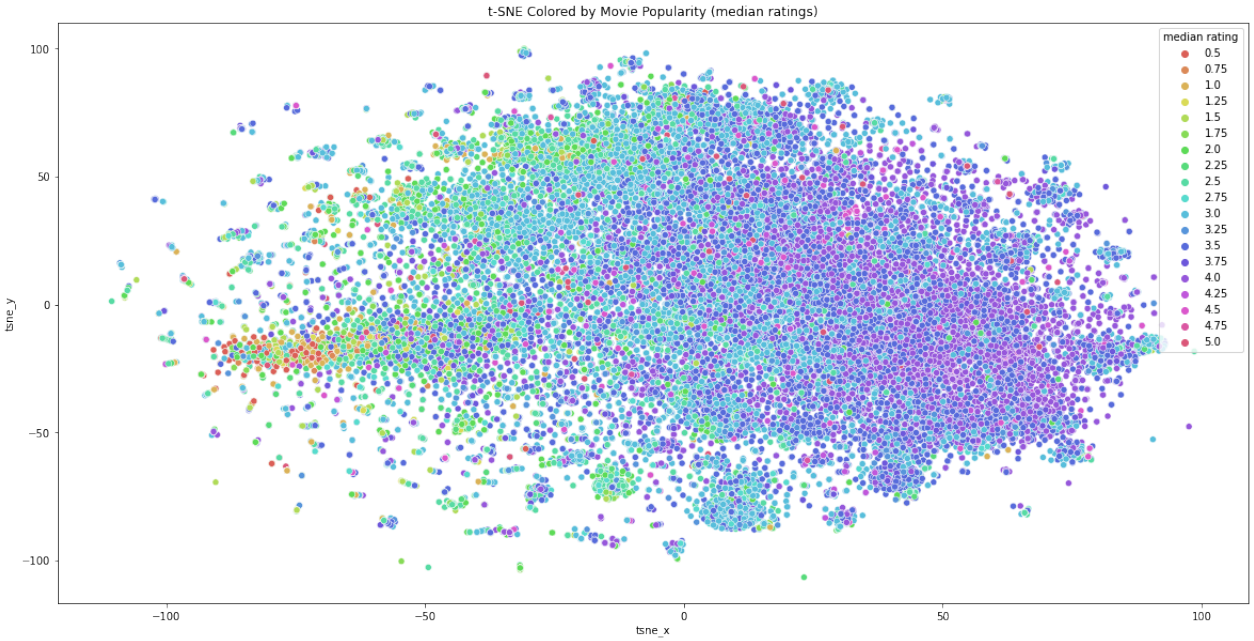


Fig. 1. **t-SNE Embedding**

Fig. 2. **UMAP Embedding**

validation and test set, which we believed would also impact the performance. And future work could be focusing on identifying a better partition method, as well as leaving more time and resources for hyperparameter tuning to approach the ceiling of the model.

7 CONTRIBUTION

Billy Sun - Data Preparation, Baseline Model, Fast Search, ALS Model Saving

Lucas Tao - Codes of evaluations on small and full datasets with respect to both popularity-based and latent factor models. Result

comparison between ALS and baseline models. Hyperparameter tuning of ALS to identify the best hyperparameter configurations. Cecilia Wu - Baseline Evaluation, ALS Hyperparameter search, Latent Representation Visualization

[1]

REFERENCES

[1] Zaiqiao Meng, Richard McCreadie, Craig Macdonald, and Iadh Ounis. 2020. Exploring data splitting strategies for the evaluation of recommendation models. In *Fourteenth ACM conference on recommender systems*. 681–686.

8 LATENT REPRESENTATION VISUALIZATION

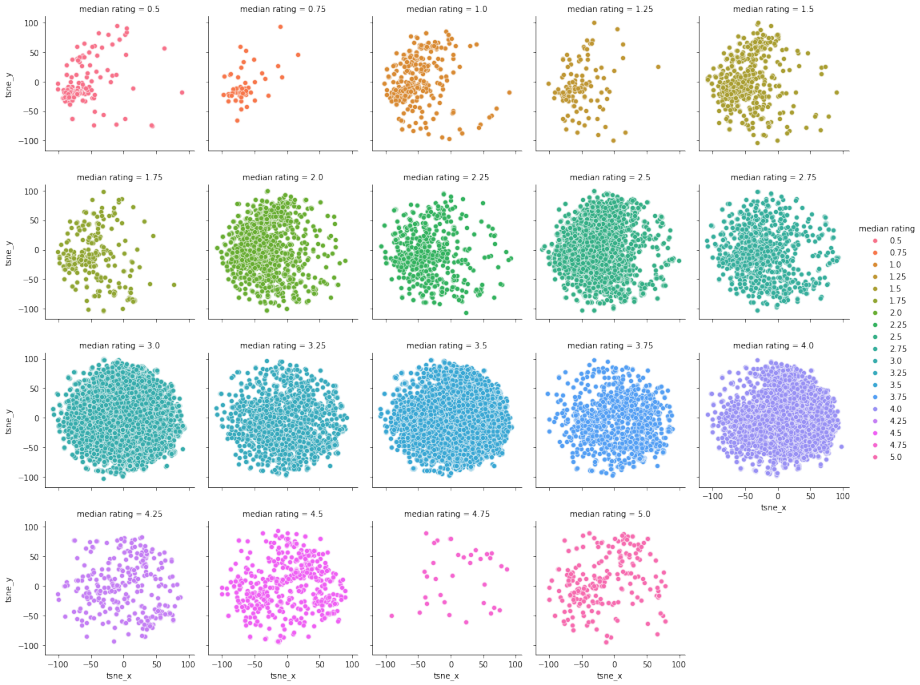


Fig. 3. **Fecet Grid for t-SNE**