Big Data Final Project: Movie Recommendation System

BAOSEN LUO[*] and YING WANG[*], New York University, USA

We use Apache Spark framework to build and evaluate a collaborative-filter based recommendation system using the MovieLens dataset. Along the way, we find the best-performing baseline and latent factor models with hyper-parameter tuning, and conduct a qualitative error analysis with visualization.

Additional Key Words and Phrases: MovieLens, Popularity Model, Latent Factor Model, Ranking Metrics, UMAP

1 INTRODUCTION

Recommender is a model that predicts users' preference over products. It is widely used in music, movie, e-commerce, and social media platforms, where a large number of products are created and consumed every day. A good recommender that yields accurate and personalized recommendations is crucial to the service providers' user experience, business growth, and monetization. In this report, we document our methodology and findings of building and evaluating a recommender system using the MovieLens dataset. The details of our implementation is stored in https://github.com/nyu-big-data/final-project-group_180.

2 DATASET

2.1 MovieLens

The MovieLens dataset collects users' interaction with the MovieLens online recommender system that is run by University of Minnesota researchers [1]. It provides two versions of the data samples: a small sample (ml-latest-small, 9742 movies and 610 users) and a larger sample (ml-latest, 58098 movies and 283228 users). In this project, we use the small sample for development purpose and use the larger sample for the final model building and evaluation. We focus on the rating.csv file to build our recommender system, where each record corresponds to one rating of one movie by one user with timestamp. Other information, such as movie tags (tags.csv) and movie genres(movie.csv), is utilized and analyzed in the error analysis in Section 6.

2.2 Train/Val/Test Split

We first collect the distinct user id list, then split users into 'training', 'validation', and 'testing'. Users in 'training' have all of their rating records assigned to the training set. Users in 'validation' and 'testing' have the latest three movies they rated assigned to the validation and testing set respectively, and have the rest of the records assigned to the training set. In this way, we ensure that (i) all the users have some rating history in the training set, so that we can avoid the cold-start problem, (ii) evaluation records for users in the evaluation set are chronologically after their training counterparts, (iii) evaluation records at validation set and test set do not overlap, so that the final model's performance on testing set is not biased.

Since the small-dataset and full-dataset have different characteristics, we apply slightly different strategies here. Each user in the small dataset rates at least 20 movies, so we could just follow the partition strategy described above. For the full dataset, however, the minimum rating per user is only one. To tackle this issue, we first identify the frequent users

[*]Both authors contributed equally to this research. Baosen Luo implemented and wrote the baseline model and the ALS model. Ying Wang conducted and wrote the results and error analysis. Both of us brainstormed on the data prepossessing and ran hyper-parameter search for all models.

Authors' address: Baosen Luo, bl3243@nyu.edu; Ying Wang, yw3076@nyu.edu, New York University, New York, USA.

who rated at least 4 movies and the infrequent users who rated less than 4 movies. We follow the previous splitting strategy for the frequent users whereas putting all the infrequent users into the training set.

3 EVALUATION CRITERIA

3.1 Ranking Metrics Definition

A ranking system[1] usually deals with a set of M users: $U = \{u_0, u_1, ..., u_{M-1}\}$. Each user u_i having a set of N (we set N=3 here) ground truth relevant documents: $D_i = \{d_0, d_1, ..., d_{N-1}\}$, and a list of Q (we set Q=100 here) recommended documents, in order of decreasing relevance: $R_i = \{r_0, r_1, ..., r_{Q-1}\}$. We select three ranking metrics to evaluate our recommender's performance. The underlying idea is to compare the true relevant documents set with the model's recommended set. Let's first define a function which, provided a recommended document and a set of ground truth relevant documents, returns a relevance score for the recommended document.

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

3.2 Mean Average Precision (MAP)

MAP measures how many of the recommended items are in the ground-truth set, where order of the recommendations is taken into consideration.

$$MAP = \frac{1}{M} \sum_{i=0}^{M-1} \frac{1}{|D_i|} \sum_{j=0}^{Q-1} \frac{rel_{D_i}(R_i(j))}{j+1} \tag{2}$$

3.3 Precision

Precision measures how many of the first k recommended documents are in the ground-truth set. Order is irrelevant in this metric. We set k = 100 here.

$$p(k) = \frac{1}{M} \sum_{i=0}^{M-1} \frac{1}{k} \sum_{j=0}^{min(|D|,k)-1} rel_{D_i}(R_i(j)) \tag{3}$$

3.4 Normalized Discounted Cumulative Gain (NDCG)

Similar to MAP, NDCG measures how many of the first k recommended documents are in the ground-truth set while considering the order. We set k = 100 here.

$$\begin{gathered} NDCG(k) = \frac{1}{M} \sum_{i=0}^{M-1} \frac{1}{IDCG(D_i, k)} \sum_{j=0}^{n-1} \frac{rel_{D_i}(R_i(j))}{ln(j+1)} \\ \text{where,} \\ n = min(max(|R_i|, |D_i|), k) \\ IDCG(D, k) = \sum_{j=0}^{min(|D|,k)-1} \frac{1}{ln(j+1)} \end{gathered} \tag{4}$$

[1]https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.mllib.evaluation.RankingMetrics.html

Table 1. Baseline Hyperparameter Tuning Results

Damping Factor	Val MAP (small)	Val MAP (full)
1	0.0109	0.0055
5	0.0136	0.0066
10	0.0129	0.0079
20	0.0136	0.0109
50	**0.0141**	0.0131
100	0.0140	**0.0147**

Table 2. Baseline Testing Set Performance

	MAP	Precision	NDCG
Small Version	0.01028	0.00516	0.05091
Full Version	0.01496	0.00376	0.04598

4 BASELINE

The baseline is a popularity model that makes the same recommendations for all users. The underlying idea is to first calculate the average rating per movie, adjusted by a damping factor, and then recommend the top k (k = 100 in our case) movies to all the users. The popularity score for each movie is:

$$p[i] \leftarrow (\sum_u R[u,i])/(|R[:,i]| + \beta) \tag{5}$$

To find the best hyperparameter, namely the damping factor β, we run grid search on the validation set. The best damping factors for the small and full version are 50 and 100 respectively, and the results of different damping factors are displayed in Table 1. We then evaluate the best model on the testing set, displayed in Table 2.

5 LATENT FACTOR MODEL

Collaborative Filtering is a state-of-art technique used for recommendation system. The underling idea is that people tend to like the items that are similar to what they have consumed or those consumed by other similar users. Based on this idea, the latent factor model considers the original rating (or utility) matrix as an approximate product of two lower-rank matrices representing users and items. These two matrices are viewed as the latent factors for users and items, hopefully capturing the implicit similarity. The machine learning library of Spark applies the alternating least squares (ALS) algorithm to learn these latent factors. [2]. ALS algorithm has many hyper-parameters and different choices affect its performance significantly. In this report, we focus on the factorization rank and the regularization parameter. Note that higher rank leads to a more sophisticated model and larger regularization parameter discourages overfitting.

Based on the experiments with the small dataset, we find that the ALS model probably requires more iterations to converge. Thus, we increase the maximum number of iterations to 30 for the experiments on the full dataset and obtain an improvement in the validation MAP. The hyper-parameter tuning results are recorded in Table 3 and Table 4. The best model with highest MAP for the small dataset has rank = 160 and reg = 0.001, and that for the full dataset has rank = 300 and reg = 0.01. The evaluation results on the testing data are shown in Table 5.

[2]https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.recommendation.ALS.html

Table 3. ALS Hyperparameter Tuning Results (small version, maxiter=10)

Val MAP	reg = 0.001	reg = 0.005	reg = 0.01	reg = 0.1	reg = 0.5	reg = 1
rank = 10	0.0003	0.0034	0.0034	0.0003	0	0
rank = 20	0.0008	0.0014	0.0015	0.001	0	0
rank = 30	0.0021	0.0015	0.0037	0.0008	0	0
rank = 40	0.0037	0.0026	0.0045	0.0008	0	0
rank = 80	0.0054	0.0045	0.0056	0.0016	0	0
rank = 160	**0.0145**	0.0106	0.0087	0.0017	0	0

Table 4. ALS Hyperparameter Tuning Results (full version, maxiter=30)

Val MAP	reg = 0.00001	reg = 0.0001	reg = 0.001	reg = 0.01
rank = 50	0	0	0.0003	0.0014
rank = 100	0	0.0001	0.0005	0.0044
rank = 200	0.0001	0.0005	0.0024	0.0067
rank = 300	0.0003	0.0019	0.0036	**0.0071**
rank = 400	0.0009	0.0037	0.0045	0.0070

Table 5. ALS Testing Set Performance

	MAP	Precision	NDCG
Small Version	0.01189	0.00434	0.04658
Full Version	0.00741	0.00286	0.02978

6 ERROR ANALYSIS

Although the ALS model on the full dataset gives worse evaluation scores compared to the baseline model, we cannot conclude that our ALS model doesn't learn anything. Our evaluation metrics don't apply to the cases where a user indeed likes the recommendations generated by the model, but the selected ground-truth set happens to be disjoint with the prediction set. Therefore, a qualitative error analysis is necessary to evaluate the model in depth.

To examine the quality of the learned hidden representations of users and items by the ALS model, we visualize these high dimensional hidden representations in two-dimensional space by UMAP [2] and t-SNE [3]. Due to the large size of the dataset, we only use 1% users and 10% movies for visualization, and find that they provide a good representation of the entire data. A brief description of these two dimensionality reduction techniques is as below.

- t-Distributed Stochastic Neighbouring Entities (t-SNE): It maps the original data from a high-dimensional space to a lower dimensional embedding by minimizing the KL divergence between the pairwise similarity of the points in the original space and that in the new embedding. It is available in the scikit-learn python library[3]. We use PCA to reduce the dimension to 50 before t-SNE transformation Following scikit-learn's suggestion.
- Uniform Manifold Approximation and Projection (UMAP): It's similar to t-SNE. The major difference is that t-SNE focus on local structures while UMAP also tries to preserve the global structure. It is available in the UMAP python library [4].

[3]https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html
[4]https://umap-learn.readthedocs.io/en/latest/

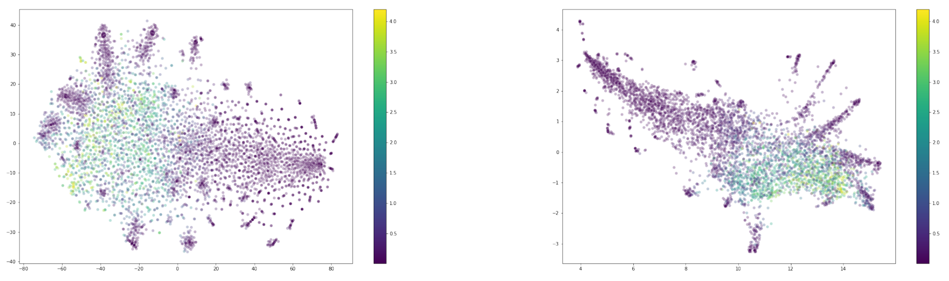
Fig. 1. t-SNE (left) and UMAP (right) visualization of the item factor with color representing popularity

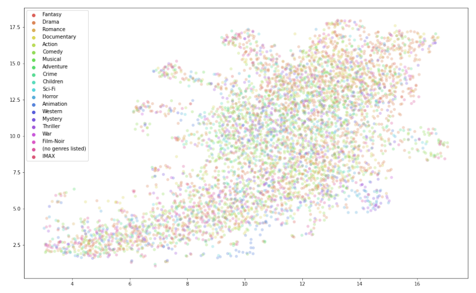

Fig. 2. UMAP visualization of the item factor with color representing genre

T-SNE and UMAP generate similar results as illustrated by Fig 1. We decide to use UMAP for most of the later experiments because it is more computational effricient than t-SNE.

6.1 Learned Item Representation

Fig 1 suggests that the learned item representation captures the popularity of the movies. Each point represents a movie and its associated color represents the popularity score that is calculated in the baseline model. In both t-SNE and UMAP figures, popular movies with brighter colors tend to cluster together, while less popular movie with darker color scatter in the rest of the figure. This suggests popular movies have similar hidden representations in the ALS model whereas unpopular movies have more diverse representations. This is not surprising because there are many reasons for a movie to become "less popular": It might have bad overall quality, is too new to have stable rating, or have a special theme that only a small group of fans will like.

The UMAP visualization with genre coloring in Fig 2, however, doesn't show any interpretable pattern. Note that we also don't input genre information explicitly into our recommendation system. By investigating several UMAP plots with different hyper parameters, we observe that none of the visualization results reflect movie genres and it's very likely that the ALS model doesn't implicitly learn the genre information.

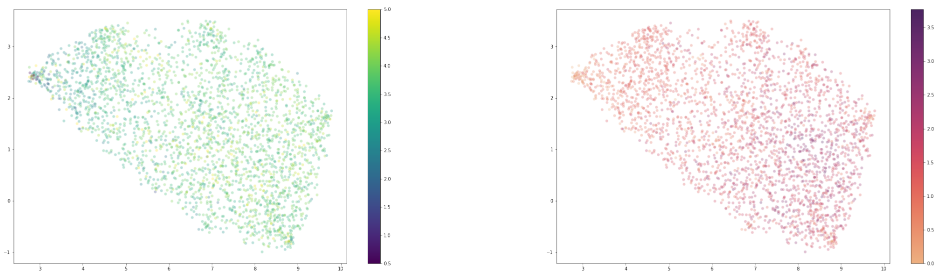

Fig. 3. UMAP visualization of the user factor with color representing average rating (left) and number of ratings (right)

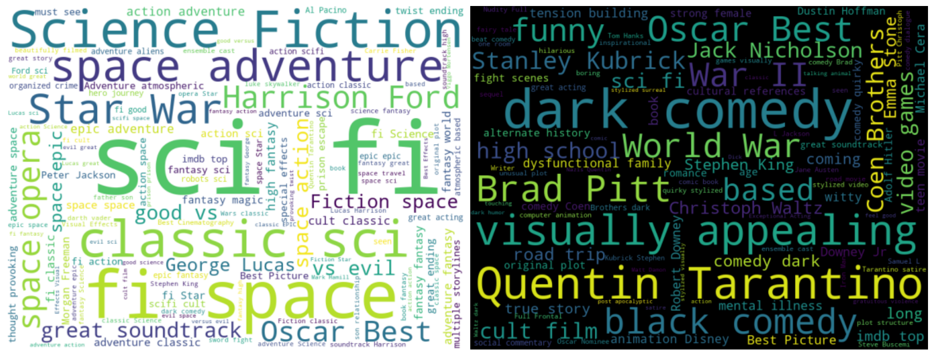

Fig. 4. Word clouds of confident (left) and unconfident (right) users

6.2 Learned User Representation

We don't observe a clear pattern in the UMAP results of the learned user representation shown in Fig 3. The green points representing users with different average ratings seem to scatter randomly in the left figure. The right figure, where color represents the log10 of the number of ratings that each user gives, shows a vague boundary between the infrequent users on the left top corner (in light red) and frequent users on the right bottom corner (in dark red). But no obvious clusters or trends are observed in all UMAP plots with different hyper-parameters or a larger proportion of users. This indicates that our model struggles to learn a useful user representation and sheds light on why our model fails to beat the baseline model.

6.3 Analysis of the Predictions

We call the users who has at least one movie in ground-truth set that appears in the first 50 recommendations the "confident" users and the users who doesn't have overlapping interests with first 200 recommendations "unconfident" users. Then, we draw word cloud (or word frequency plot) for the tags associated with correctly predicted movies for confident users and word cloud for the tags associated with the movies in ground-truth set for unconfident users, as shown in Figure 4. The confident users are more related to sci-fi while unconfident users are more related to various topics including religion, war and celebrities.

REFERENCES

[1] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.* 5, 4, Article 19 (dec 2015), 19 pages. https://doi.org/10.1145/2827872

[2] John Healy Leland McInnes and James Melville. 2020. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. (2020). https://arxiv.org/pdf/1802.03426.pdf

[3] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing Data using t-SNE. *Journal of Machine Learning Research* (Nov 2008). https://lvdmaaten.github.io/publications/papers/JMLR_2008.pdf