Collaborative-Filter Based Recommender System for Movies

Howell Lu
hl4631@nyu.edu
Center for Data Science, New York University
New York City, NY, USA

Xiangyue Wang
xw1499@nyu.edu
Center for Data Science, New York University
New York City, NY, USA

Xiu Xie
xx2179@nyu.edu
Center for Data Science, New York University
New York City, NY, USA

ABSTRACT

In this final project, we build and evaluate a collaborative-filter based movie recommendation system using Spark's alternating least squares (ALS) method. We evaluated the model's performance by comparing it with that of a baseline popularity model as well as a single-machine implementation, and investigated the mistakes that our best-performing made. For code implementation and further references see GitHub repository or follow this link: https://github.com/nyu-big-data/final-project-group_80

ACM Reference Format:
Howell Lu, Xiangyue Wang, and Xiu Xie. 2022. Collaborative-Filter Based Recommender System for Movies. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION AND DATA

In recent years, the rise of e-commerce entities such as Amazon, video sharing platforms such as YouTube and streaming services such as Netflix has promoted both demand and development of collaborative-filter based recommender systems—algorithms that generate personalized recommendations based on users' past online activities. In the case of movie recommendation, the system utilizes user viewing and rating histories. In this project, we worked with the MovieLens dataset[3]. Collected by MovieLens.com and published by F. Maxwell Harper and Joseph A. Konstan, the dataset consists of movie ratings on a five-star scale and text-based movie tags from viewers. Only the ratings were used to develop the recommender system. The dataset has two versions. The small version contains 100,836 ratings across 9742 movies, generated by 610 users between March 29, 1996 and September 24, 2018. The large file contains 27,753,444 ratings across 58,098 movies, generated by 283,228 users between January 09, 1995 and September 26, 2018. Both datasets present missing entries. Each viewer in the small version has given out at least 20 movie ratings, and each viewer in the large version has at least 1 movie rating. Users were selected at random for inclusion, but since no demographic information is given, we do not know whether there exists imbalance in age, gender, race, or other demographics in the dataset.

Conference'17, July 2017, Washington, DC, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 DATA PRE-PROCESSING

First and foremost, we split each version of the datasets into train, test, and validation sets. Our methods are the following:

- Save around sixty percent of ratings for each viewer as the training set. This way, the training set contains at least some ratings from every viewer;
- Evenly split all the viewers into two groups;
- For the first group, assign each viewer's remaining forty percent of ratings to the test set;
- For the second group, assign each viewer's remaining forty percent of ratings to the validation set. This way, the test and validation sets are composed of distinct groups of users.

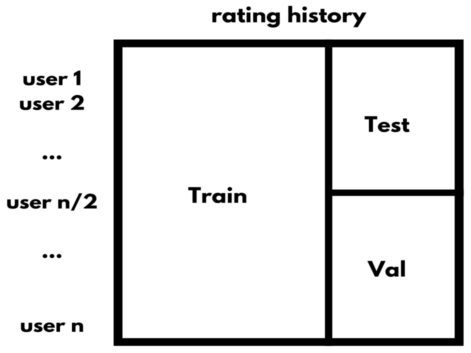

Figure 1: Graph of train, test, and validation partition.

To ensure that the results are stable, we further modified the large version of the dataset by filtering out movies with less than 30 ratings, based on the acknowledgement that a sample size of at least 30 is required to draw statistically significant conclusions.

3 EVALUATION METRIC

We chose the Mean Average Precision (MAP) as our model evaluation metric. MAP is a measure of how many of the recommended documents are in the set of true relevant documents, where the order of the recommendations is taken into account (i.e. penalty for highly relevant documents is higher)[1]. Having MAP as the evaluation metric is extremely advantageous when top recommendations are desired at the top of the list because it rewards you for such behavior and penalizes you more when incorrect guesses are higher up in the ranking. Meanwhile, precision is a similar metric to the

MAP with the only difference being that in calculating precision, the order of the recommendations is not taken into account. During Extension 1 of the project, MAP was not an available built-in metric, hence precision was used to evaluate that section alone.

4 BASELINE POPULARITY MODEL

The first model we tried to implement is a popularity-based, non-personalized recommendation model. The significance of implementing the popularity model is to have a benchmark that we can later compare to. For the popularity model, we selected the top 100 movies with 30 or more reviews ranked by ratings. For the large training set, the top five movies ranked by ratings are shown in the table below.

Movie ID	Average Rating	# Reviews
171011	4.487	853
159817	4.458	1384
318	4.424	97999
170705	4.400	984
191999	4.375	48

We then used Mean Average Precision (MAP) and precision as metrics to evaluate the popularity model on the test sets. For the small dataset, the MAP for the baseline model is 0.0287 and precision at query 100 is 0.07308. For the large dataset, we have MAP = 0.00596 and precision = 0.02047.

5 ALTERNATING LEAST SQUARES

Next, we used Spark's Alternating Least Squares (ALS) method to learn latent factor representations for users and items. The latent factor representation of users and ratings data produces low-rank, dense matrices that make the training process easy to parallelize and scale up. Given a user and their rating history, the model predicts which movies the user will be interested in by finding movies similar to those that they showed interest in in the past, as well as by finding movies enjoyed by users who gave similar ratings in their viewing history.

The results are as follows:

We then used Mean Average Precision (MAP) and precision as metrics to evaluate the latent factor model on the test sets. For the small dataset, the MAP for the ALS model is 0.00287 and precision at query 100 is 0.0730. For the large dataset, we have MAP = 0.00438 and precision = 0.0229.

Furthermore, we performed hyper-parameter tuning of the ALS model on the ranking factor, number of iterations, and the regularization factor using the validation set we generated earlier. We found that the best performing set of hyper-parameters requires Rank = 15, Iteration = 10, and Reg= 0.05. The table below demonstrates how MAP changes as iteration and regularization parameter varies for rank = 15.

reg	3 itr	5 itr	10 itr	15 itr
0.05	0.0001	0.0017	0.0049	0.0058
0.1	0.0000	0.0010	0.0040	0.0047
1	0.0000	0.0000	0.0000	0.0000

UserId	highest recommended movie ID	Predicted rating
2	72178	4.45
4	2571	4.88
5	171495	5.09
7	72178	4.65
10	95604	5.15
11	27423	4.74
14	140265	4.59
15	142452	5.11
19	72178	5.28
20	8336	3.017
21	100509	4.69
22	162596	5.31
23	120815	5.05
25	26073	5.70
29	93006	5.29
30	6072	4.96
31	132555	4.97
32	86574	5.23
39	72178	4.52
40	26914	5.62

However, the results of alternating least squares were not particularly impressive as the mean average precision was typically close to zero.

6 EXTENSION 1: COMPARISON TO SINGLE-MACHINE IMPLEMENTATION

As an extension, we measured how parallelism contributes to the runtime of the ALS model by comparing the Spark implementation to a single-machine implementation. The cluster results were already generated in the previous section. For the single machine implementation, we used a package named LensKit developed by Michael D. Ekstrand [2]. Precision is used as the evaluation metric in this section, given that lenskit does not offer MAP as a built-in metric. Based on predictions of the top 100 items for each user, the top 5 users by precision are:

User ID	# Recommendations	Precision
252364	100	0.74
63783	100	0.67
42494	100	0.60
191063	100	0.57
143817	100	0.49

Runtime wise, the single machine implementation took 8.87 seconds for the small dataset and 527.0124 seconds for large dataset; while the Spark implementation took 210.3 seconds for the small dataset, and 589.0711 for the large dataset. Contrary to our expectation, the cluster took longer than the single-machine implementation, which indicates that parallelism did not speed up the runtime. We suspect it is due to the tight integration with PyData tools such as Pandas that single machine possesses, that resulted in this discrepancy.

7 EXTENSION 2: QUALITATIVE ERROR ANALYSIS

Lastly, using our best-performing latent factor model, we investigated the mistakes that it made by analyzing the trends and genres of the users who produced the lowest-scoring predictions. We analyzed the values which resulted in a precision score of 0, compared to other values in the dataset and found quite a few patterns between the users. We noticed that users who had a precision of 0 had a few specific tendencies.

We uploaded csvs into SQLLite and performed queries on them to analyze tendencies of the group with the lowest precision and found a vast array of tendencies.

tag	(Worst Rated #)	Total Tags	% of Movies	Best Rated #	Total Tags
sci-fi	9346	9400	99.43%	9317	99.12%
atmospheric	6386	6430	99.32%	6405	99.61%
action	6088	6219	97.89%	5973	96.04%
comedy	5581	5923	94.23%	5431	91.69%
surreal	5197	5299	98.08%	5244	98.96%
based on a book	5080	5294	95.96%	5013	94.69%
twist ending	4831	4864	99.32%	4796	98.60%
funny	4785	4844	98.78%	4678	96.57%
visually appealing	4322	4333	99.75%	4321	99.72%
dystopia	4203	4268	98.48%	4192	98.22%
dark comedy	3918	4026	97.32%	3902	96.92%
stylized	3796	3804	99.79%	3797	99.82%
classic	3754	3778	99.36%	3756	99.42%
fantasy	3742	3829	97.73%	3699	96.60%
thought-provoking	3740	3753	99.65%	3736	99.55%
psychology	3661	3691	99.19%	3651	98.92%
romance	3613	3867	93.43%	3482	90.04%
quirky	3599	3625	99.28%	3591	99.06%
time travel	3358	3446	97.45%	3291	95.50%

Firstly, the median number of reviews for user is 30. However, within the group with the worst reviews, the median number of reviews is actually 17. Another interesting factor is how the tags are distributed.

Furthermore, we noticed that the median rating was at 4.0 and the average rating at 3.53. The reviews of the poorly rated viewers have similar ratings as their median is 4.0 with a median of 3.55. So there is not much of a difference in that field. However, we lastly noticed that the top 8 ranked users reviewed nearly the same amount of movies as the bottom 130k users and therefore we wanted to see what percentage of movies of each genre was watched. Although cursory, it appears that the users with the worst precision prefer to watch more action films, comedy films, twist endings and romantic movies rather than our best rated cohort.

The final tendency that was noticed about the viewing habits of those who were poorly predicted was the fact that the movies that they had watched were generally more common than the movies watched by those who were better rated. The median rating of a movie watched by the worst ranked was 13132, but the median rank of movies watched by users with over 10 percent precision was 5142.

The tendencies that were noted about poorly predicted users was that they watched less movies than usual but watched more popular movies whilst having a preference for movies that are comedies, actions and funny.

8 CONCLUSION

In this paper, we implemented a baseline popularity model and an Alternating Least Squares latent factor model using the dataset MovieLens to provide personalized movie recommendations to users. We evaluated both models' performance on the data using precision and MAP as metrics. We searched for the optimal hyper-parameters of the latent factor model. We measured the speedup generated by using parallel machines via Spark instead of a single machine using LensKit. And lastly, we performed qualitative error analysis.

9 ACKNOWLEDGEMENTS

The team would like to acknowledge Professor Brian McFee and the teaching assistants of DS-GA 1004 Spring '22 for the support they have provided throughout the course of this project. The Spark and lenskit documentations have also proven to be extremely helpful when implementing the different models.

10 CONTRIBUTION

The team distributed the workload evenly throughout the entire project, from the initial benchmark construction to the final report write-up. Communication was easy between us and each of us adhered to the timelines and deadlines that were agreed upon during group meetings. Howell implemented the ALS model, tuned the hyper-parameters, contributed to the baseline model, and implemented the second extension. Xiangyue incorporated feedback from the checkpoint and re-did the train-test split, contributed to the popularity baseline model and extension 1, and worked on the write-up. Xiu developed the initial train-test split script, implemented the first extension, and contributed to the report write-up. Each member reviewed the programming scripts uploaded to the shared git repository to ensure quality.

REFERENCES

[1] Evaluation metrics - rdd-based api.
[2] Michael D. Ekstrand. 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)*, 2020.
[3] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4), dec 2015.