Capstone Project:
MovieLens Customer Segmentation & Collaborative-Filter Recommendation Systems

Authors: Bella Chang*, Kristina Fujimoto*, Emily Wang*

P. Wallisch (DS-GA 1004)

May 7, 2025

**New York University, Center for Data Science*

Code: https://github.com/nyu-big-data/capstone-bdcs_11

Customer Segmentation

Overview

In the first phase of our project, we aimed to identify the top 100 “movie twins”—pairs of users with highly similar movie-watching behaviors—using data extracted from the MovieLens dataset. We then evaluated the similarity of their rating patterns by comparing the average rating correlation of these identified pairs with that of 100 randomly selected user pairs.

Data Preprocessing and Movie Twin Identification

Our analysis began by focusing on the ratings.csv file, which includes both userId and movieId. We imported this dataset using Apache Spark and filtered it to retain only these two columns, as our notion of a user’s “movie-watching style” was defined by the set of movies they had rated.

To improve the quality of the similarity analysis, we excluded users who had rated fewer than five movies, reasoning that such sparse activity would provide limited insight into viewing preferences. With the filtered dataset, we applied Spark’s CountVectorizer to create a more compact representation of user activity, effectively encoding each user’s set of rated movies into a vector format. The resulting vectors were converted into a Spark DataFrame, which was then passed to Spark’s MinHashLSH algorithm.

We used the approxSimilarityJoin function to perform a self-join on the DataFrame of user vectors, identifying user pairs below a certain Jaccard distance threshold. We added a column computing Jaccard similarity as 1 - distance and sorted the resulting pairs in descending order of similarity to select the top 100 most similar user pairs.

Hyperparameter Choices

For the MinHashLSH similarity join, we used the default threshold of 0.5. This decision was guided by preliminary tests on a smaller subset of the data, which showed that this threshold yielded the 100 user pairs needed. We opted to retain this relatively lenient threshold even for the full dataset, preferring a broader inclusion of potential twins over an overly restrictive filter. Additionally, we increased the number of hash tables (numHashTables) from the default value of 1 to 5, aiming to reduce hash collisions and improve the quality of similarity matches.

Results and Observations

The results of this analysis are presented in the Appendix under “Movie Twin Similarity Analysis.” All of the top 100 user pairs achieved a Jaccard similarity score of 1.0, indicating that each pair had rated exactly the same set of movies. This outcome may be partly attributable to the chosen similarity threshold, but also reflects the presence of widely rated, popular films

within the dataset. Given that the dataset contains approximately 86,000 movies, it is plausible that many users shared identical subsets of highly popular titles.

Although we considered experimenting with different thresholds, our ability to do so was limited by computational constraints. Each Spark job typically required an hour or more to execute, making extensive hyperparameter tuning on threshold and number of hash tables infeasible. We therefore acknowledge this as a limitation and a potential avenue for future investigation.

Validation via Rating Correlation

To validate the relevance of our movie twin pairs, we compared the rating behaviors of these 100 user pairs with 100 randomly selected user pairs from the dataset.

We defined a function that, for each user pair, extracts the set of commonly rated movies and computes the correlation between their ratings—provided the pair had at least two movies in common. This threshold ensured that computed correlations were statistically meaningful. The function then aggregated these pairwise correlations and returned their average.

To establish a baseline, we implemented a separate function to randomly sample 100 user pairs and repeated the same correlation computation process. Our final results showed that the movie twin pairs had an average correlation of **0.1041**, while the randomly sampled pairs had an average correlation of **0.0409**. These results suggest that our method for identifying movie twins captures meaningful similarities in users' rating behaviors.

Movie Recommendation

Partitioning

To clean the data, we first removed any rows containing null values in the "userId," "movieId," or "rating" columns. We also filtered out movies with fewer than 5 ratings to avoid unreliable popularity estimates due to lack of rating data.

To partition our data, we used PySpark's randomSplit function to divide the cleaned dataset into training (80%), validation (10%), and test (10%) sets. To mitigate the cold start problem (in preparation for the ALS model in question 5), we further split the validation and test sets into 40% and 60% subsets each, based on movieId. The 40% subsets of validation and test sets were then joined back into the training set so that the model could learn from movies that might otherwise be unseen during training. The remaining 60% subsets are used for the final validation and test sets (with the previous 40% subsets withheld). Consequently, the training, validation, and testing datasets consisted of 29712679, 2003922, 2040499 rows, respectively, for the full dataset. Lastly, all three datasets were saved as separate CSV files, so that we can read these data frames directly for use in questions 4 and 5.

Popularity Baseline

To implement our popularity baseline, we computed movie popularity score with a damping factor by applying the equation, (average rating * number of ratings) / (number of ratings + damping factor). The damping factor β reduces the influence of movies with very few ratings, ensuring that the popularity scores are more reliable and less susceptible to outliers. To compute ground truth for evaluation, we calculated each user's mean rating and filtered out all movies that are less than the user's mean rating (i.e. movies "not liked" by the user). These filtered ratings were then compared to the model's predicted recommendations. Instead of recommending the same globally popular movies to every user, we implemented a personalized popularity baseline by ranking all movies using damped popularity scores and filtering to include only the movies present in each user's test set. This ensures that recommendations for each user are limited to the set of movies they realistically could have rated in the test set, rather than the global top 100. By doing this, we cater the popularity-based recommendations to each user's context and ensure a fair evaluation against their personalized ground truth, avoiding bias toward globally popular movies the user never encountered. We hyperparameter tuned the damping factor β = [10, 100, 1000, 10000] on our validation set and found that all values yielded very similar results for all metrics (refer to Table 1 in appendix), with β = 1000 performing best. We used NDCG as our primary evaluation metric because it measures whether relevant items are recommended while also rewarding relevant movies when they are ranked higher, leading to a more realistic user experience for movie recommendations. For our test set, with β = 1000, our results were as follows:

- **NDCG@100:** 0.7879
- **Preicision@100:** 0.0573
- **Mean Average Precision@100:** 0.6602
- **Recall@100:** 0.9984

The NDCG value of 0.7879 is relatively high, demonstrating that the ranking of our recommendations is about 78.8% as good as an ideal ranking, with relevant movie recommendations being fairly high in the list. The precision of 5.73% is relatively low, meaning that the model also recommends irrelevant movies (that are not in the ground truth), which is expected for a popularity-based model that does not personalize results to individual users (unlike models such as ALS). However, the mean average precision is 66.02%, demonstrating that when the model does recommend relevant movies, it tends to rank them consistently early. Lastly, the recall value of 99.84% is extremely high, indicating that the popularity baseline model is able to retrieve nearly all relevant movies for each user within the top 100 recommendations.

ALS Model

We implemented Spark's alternating least squares (ALS) method to compare its recommendations compared to the popularity based model. First, we load the training, testing, and validation datasets from HDFS that were created in question 3. The function train_test_als was created which trains the ALS model on the training data, for a given rank regParam and maxIter. The model then predicts ratings on the testing data and the top 100 predictions per user were extracted using Window (which sorts by the predicted rating). For each user, the relevant movies were selected from the total movies each user rated by filtering for the movies rated above a user's mean rating. This was the personalized relevance threshold we established as this would allow the model to predict more accurately movies that users would actually like. Then, RankingMetrics was used to evaluate the model by matching the predicted and actual movies by user and retrieving the NDCG, precision, MAP and recall at 100.

To hyperparameter tune our ALS model, we ran the model with the validation data at ranks of 10, 100, and 150 and regParams of 0.01, 0.05, and 0.1 (Table 2). The performance was recorded for each combination and the model with the highest NDCG was selected as the best performing model. Finally, the model is then trained and tested with our best parameters on the test data. NDCG was used in selecting our best model as it is the most appropriate metric when the ranking order matters. With the same reasons mentioned in the popularity based model, NDCG is the most indicative. In addition, we used this metric in evaluating the popularity based model so we wanted to keep evaluation consistent between both models.

The best model was with 150 ranks and 0.01 reg_param, when run on the test dataset the ranking metrics were as follows:

- **NDCG@100:** 0.9720
- **Precision@100:** 0.0584
- **MAP@100:** 0.9456
- **Recall@100:** 0.9995

The NDCG value is quite high at 97.2%, suggesting that our model is very accurately recommending movies to users, especially providing movies that are more relevant higher up in the recommendations. The other metrics being high (MAP and recall) align with the results of NDCG in providing accurate recommendations. On the other hand, the precision is still quite low at 5.84% suggesting that similar to the popularity based model, some irrelevant movies are being recommended.

Evaluation of Popularity Based Model and ALS Model

Both the popularity based model and ALS model performed surprisingly well, but the ALS model performed slightly better. While there was only slight improvement in the precision and recall, the NDCG and the MAP of ALS are much higher than the popularity based model. These improvements between the popularity based model and the ALS model are as expected since the latter captures personal preferences. This shows that ALS ranks relevant movies higher meaning users are more likely to see personalized relevant recommendations at the top of the list. ALS also allows us to fine-tune the model more compared to the popularity based model where we can only change the damping factor. While it performs better, there are limitations to consider such as addressing the cold start problem by allowing some test and validation data to be seen in the model training process and potential bias towards movies that have more ratings (ignoring more niche movies).

Contributions

Together, we all worked really hard on completing this capstone. All three of us met to work on Question 1 (customer segmentation), which we first implemented in Python. Bella then converted it to work on Spark. Kristina worked on the correlation code. All three of us worked on trying to run the similarity code and correlation code for customer segmentation on the big dataset, but Bella was able to debug the correlation code to run. Emily worked on the partitioning and popularity baseline code. All of us worked on the ALS model and hyperparameter search. All three of us worked on debugging together and writing the report.

Appendix

Movie Twin Similarity Analysis

userA	userB	similarity
141384	290112	1.0
316199	316301	1.0
117965	76751	1.0
6353	66577	1.0
249671	254507	1.0
158896	317196	1.0
110447	280091	1.0
246448	27807	1.0
317196	55950	1.0
46874	6022	1.0
127410	36694	1.0
119037	184266	1.0
155065	828	1.0
179738	330375	1.0
101817	89797	1.0
158896	285618	1.0
212700	77609	1.0
192594	90442	1.0
69366	84310	1.0
16527	2214	1.0
118377	215389	1.0
277178	316199	1.0
247121	94323	1.0
123839	67915	1.0
113896	84479	1.0
100934	86415	1.0
76751	77489	1.0
152837	2214	1.0
174438	76594	1.0
159411	225000	1.0
123850	7694	1.0
141384	174700	1.0
75676	81540	1.0
212806	27485	1.0
147456	208742	1.0

264872	89790	1.0
121191	124526	1.0
212806	315367	1.0
264872	306830	1.0
64597	9824	1.0
228812	81540	1.0
182114	255872	1.0
264872	303242	1.0
238160	259389	1.0
118377	330375	1.0
164810	228812	1.0
182114	318044	1.0
289753	306830	1.0
204665	69366	1.0
164050	303026	1.0
204018	81540	1.0
195278	277929	1.0
124526	75676	1.0
195278	204018	1.0
238160	9824	1.0
212893	84479	1.0
220926	298682	1.0
155065	320166	1.0
313887	89797	1.0
2214	273370	1.0
122955	329289	1.0
204665	89797	1.0
148904	221462	1.0
27807	77489	1.0
176923	307269	1.0
318177	35551	1.0
254189	285618	1.0
326394	35290	1.0
23717	316301	1.0
126365	178146	1.0
110447	110963	1.0
179121	90442	1.0
281537	86130	1.0
215389	254189	1.0
164050	91387	1.0

236843	81540	1.0
160573	84479	1.0
123839	161529	1.0
195278	313299	1.0
123839	81540	1.0
273406	314990	1.0
145769	79181	1.0
15736	62793	1.0
133032	246463	1.0
27807	65054	1.0
159411	46874	1.0
123119	298307	1.0
115701	29361	1.0
174700	84546	1.0
110963	7694	1.0
178146	236843	1.0
117965	79181	1.0
328749	9824	1.0
117965	15736	1.0
204665	294081	1.0
160184	178146	1.0
208742	224779	1.0
164050	8632	1.0
330375	35551	1.0
212893	9183	1.0
+--------+---------+----------+

Table 1: Validation Set Results with Hyperparameter Tuning for Popularity Baseline Model

	NDCG@100	Precision@100	Mean Average Precision@100	Recall@100
β: 10	0.7875	0.0573	0.6602	0.9984
β: 100	0.7879	0.0573	0.6602	0.9984
β: 1000	0.7879	0.0573	0.6602	0.9984
β: 10000	0.7875	0.0573	0.6607	0.9984

Table 2: NDCG in Hyperparameter Tuning of ALS

	10 Ranks	100 Ranks	150 Ranks
0.01 reg_param	0.916	0.967	0.972
0.05 reg_param	0.915	0.953	0.957
0.1 reg_param	0.906	0.914	0.914