Capstone Project - Group 69

Sampreeth Avvari* **Barath Rama Shankar*** **Dhruv Sridhar***
spa9659@nyu.edu, br2543@nyu.edu, ds7395@nyu.edu
New York University

Abstract

Personalized recommendation systems are essential in today's digital world, enhancing user experiences by delivering tailored content. This capstone project uses the MovieLens dataset, a benchmark in recommender systems research, to develop and evaluate a collaborative filtering-based recommendation system and a customer segmentation system. The project includes two main tasks: identifying user pairs with similar movie-watching styles using a MinHash-based algorithm and building a movie recommender system. We developed a system to identify "movie twins"—pairs of users with similar movie-watching styles—using a MinHash-based algorithm implemented with datasketch and pandas.

For the movie recommendation system, we partitioned the data into training, validation, and test sets, starting with a popularity baseline model and progressing to a more sophisticated approach using Spark's alternating least squares (ALS) method. We fine-tuned hyperparameters such as latent factor dimensions and regularization parameters, and evaluated the models using ranking metrics to ensure accurate and meaningful recommendations. Our final report details the implementation, evaluation results, and contributions of each team member, along with necessary software documentation and instructions.

Customer Segmentation

Customer segmentation is a crucial task in personalized recommendation systems, allowing us to group users with similar behaviors and preferences. For this project, we focused on identifying "movie twins"—pairs of users with highly similar movie-watching styles. By leveraging the MovieLens dataset, we employed a MinHash-based Locality Sensitive Hashing (LSH) approach to efficiently compute user similarities based on the set of movies they rated, regardless of the ratings themselves.

Pandas-Greene

Methodology To begin, we loaded the large version of the MovieLens dataset, retaining only the `userId` and `movieId` columns. The preprocessing step involved grouping movie IDs by user IDs to create a dictionary of user-specific movie sets. Using the `datasketch` library, we created MinHash signatures for each user by hashing the movie IDs they rated. Specifically, we used 30 permutations to generate these signatures, which balance the trade-off between computational efficiency and accuracy.

Creating MinHash signatures for the user-movie sets was a computationally intensive task. Using 30 permutations for each user's movie set, the process took approximately 6 minutes and 19 seconds. This step involved iterating through each user's movie set, updating the MinHash object with encoded movie IDs, and storing the resulting MinHash signatures. Despite the time required, this preprocessing step is crucial for the subsequent LSH algorithm to efficiently identify similar users.

Next, we implemented the LSH algorithm to find similar users. The LSH scheme was configured with a threshold of 0.9 and 6 bands, which effectively grouped users with similar MinHash signatures. By inserting the MinHash signatures into the LSH index and querying each user against this index, we identified pairs of users with similar movie-watching styles. This process was computationally intensive but efficient, taking approximately 10 seconds for insertion and 27 seconds for querying on a dataset with 330,975 user pairs.

Hyperparameter Choices The hyperparameters for the LSH algorithm, including the number of permutations (30), the similarity threshold (0.9), and the number of bands (6), were chosen to balance accuracy and computational efficiency. Higher permutations provide finer granularity but increase computation time. The similarity threshold determines the minimum Jaccard similarity required to consider two users similar. A higher threshold reduces false positives. The number of bands affects the LSH index's precision and recall. We found that 6 bands provided a good balance, allowing us to efficiently identify 100 pairs of similar users.

Increasing the number of permutations (num_perm) generally increases the time required to create MinHash signatures. For example, with 500 permutations, MinHash creation time was significantly longer at 33 minutes

Table 1: Hyperparameter Choices and Performance Metrics

SlNo.	num_perm	MinHash Time	Threshold	num_bands	LSH Time	Sort Time	Pairs Found	Avg Correlation
1	150	12:09	0.9	4	00:12	00:03	547642	0.4011816751916741
2	**30**	**06:19**	**0.9**	**6**	**00:10**	**00:27**	**22215721**	**0.7348021838603093**
3	15	7:01	0.9	2	00:09	00:06	1830546	0.6459987961922578
4	30	07:24	0.9	4	00:15	00:06	3170697	0.6778928586400977
5	500	33:00	0.9	8	01:12	-	543531	0.32810841878011643
6	500	33:00	0.9	4	01:15	-	542102	0.2464566386545519
7	500	33:18	0.3	8	01:19	-	542102	0.32810841878011643

compared to just over 6 minutes for 30 permutations. While higher permutations provide finer granularity and potentially more accurate similarity detection, they also increase computation time, as seen with 500 permutations leading to lower average correlation for the top 100 pairs compared to lower permutations.

The number of bands (num_bands) in the LSH algorithm also impacts both the processing time and accuracy. More bands can improve the precision of the LSH index, reducing false positives by effectively narrowing down the candidate pairs. However, increasing the number of bands also requires more computational resources. For instance, using 8 bands with 500 permutations resulted in an LSH time of over 1 minute, while 6 bands with 30 permutations maintained a reasonable balance between processing time and correlation accuracy, achieving a high average correlation of 0.7348. Conversely, fewer bands, such as 2 bands, resulted in a lower but still substantial average correlation of 0.646, with relatively faster processing times.

Overall, balancing the number of permutations and bands is essential for optimizing both the performance and accuracy of the customer segmentation task. While higher permutations and bands can lead to more precise results, they also significantly increase computation time, which must be managed based on the specific requirements and available resources.

Correlation Analysis To validate the identified pairs of similar users, we calculated the average correlation of their movie ratings. For each pair of users, we extracted their ratings for common movies and computed the Pearson correlation coefficient. The average correlation for the top 100 similar pairs was 0.7348, indicating strong similarity in their rating patterns. For comparison, we randomly selected 100 pairs of users and computed their average correlation, which was significantly lower at 0.0323. This stark difference highlights the effectiveness of our LSH-based approach in identifying genuinely similar users.

Table 2: Performance Metrics for Best Model

Dataset	Total Pairs	Top 100 Correlation	Random 100 Correlation
Large	22215721	0.7348	0.0323
Small	151	0.2156	0.1511

The performance of the best model was evaluated on both the large and small datasets using the same hyperparameters: 30 permutations, a threshold of 0.9, and 6 bands. On the large dataset, the model found over 22 million similar pairs with a high average correlation of 0.7348 for the top 100 pairs, significantly higher than the average correlation of 0.0323 for 100 random pairs. This indicates that the model effectively identifies genuinely similar users in a large dataset.

In contrast, the same model applied to the small dataset found only 151 similar pairs, with a much lower average correlation of 0.2156 for the top 100 pairs and 0.1511 for 100 random pairs. The lower correlations on the small dataset suggest that the model's effectiveness is more pronounced with larger datasets, where there are more users and interactions to accurately capture user similarity. The disparity in performance between the large and small datasets highlights the importance of dataset size in the accuracy of customer segmentation models using MinHash and LSH techniques. The large dataset provides a richer set of interactions, enabling more precise identification of similar user pairs.

Results The results of our LSH-based customer segmentation demonstrate its efficiency and accuracy. The time taken for the LSH calculations was notably quick, with insertion and querying completing in 10 seconds and 27 seconds, respectively, for 330,975 user pairs. The average correlation for the top 100 pairs was substantially higher than that of random pairs, confirming the validity of our segmentation method. These findings underscore the potential of MinHash-based LSH for scalable and effective customer segmentation in large datasets.

Pyspark-Hadoop

To perform customer segmentation using PySpark, we utilized HashingTF and MinHashLSH available from pyspark.ml.feature

First, we created movie sets for every user using the collect_set function from PySpark. Since we were using collect_set, a groupBy on userId was performed. To optimize the partitions, we repartitioned the DataFrame based on userId. This allowed us to obtain every unique user with a list of all the movies they rated. We then used HashingTF to convert the movie sets into feature vectors. These feature vectors were passed to MinHashLSH for fitting. Subsequently,

approxSimilarityJoin was used to compute the Jaccard distance between all users. Finally, we sorted this DataFrame to get the top 100 user pairs based on similarity.

This approach worked perfectly with the small dataset. However, when we ran it on the large dataset, we encountered various errors in Dataproc, such as lost node errors and killed worker errors.

Optimizations Attempted: We attempted several optimizations to address the errors encountered during processing. First, we reduced the dataset size by removing users who had rated fewer than 20 movies. Despite this reduction, the errors persisted.

Additionally, we experimented with various hyperparameters in the PySpark functions, such as the number of features in HashingTF, the number of hash tables in MinHashLSH, and the threshold in approxSimilarityJoin. These adjustments aimed to decrease the computational workload, but they did not significantly improve performance.

Even when processing only 10% of the large dataset (approximately 8,000 unique users), we still encountered issues with nodes getting lost, which ultimately halted the execution. These challenges prompted us to explore alternative methods for customer segmentation. Due to these issues, we decided to use datasketch and pandas for customer segmentation, which provided a more stable environment for handling the large dataset.

Movie Recommendation

Data Partition

To ensure unbiased evaluation of our recommendation system, we partitioned the MovieLens dataset into training, validation, and test sets using PySpark. We began by loading the dataset and retaining the essential columns: `userId`, `movieId`, and `timestamp`.

Using the `percent_rank` function, we partitioned the data by `userId` and ordered it by `timestamp`. Each user's rating history was split into 80% for training and 20% for testing. The training data was further divided into 80% for training and 20% for validation.

The resulting partitions were combined into single DataFrames and saved as Parquet files for efficient storage and retrieval. This approach preserves the temporal order of ratings, ensuring robust and reliable recommendations.

Popularity Baseline Model

To establish a benchmark for our recommendation system, we implemented a popularity baseline model as asked. This model recommends the most popular movies to all users, serving as a simple yet effective baseline for comparison with more complex algorithms.

We began by calculating the popularity of each movie in the training set. The popularity score was defined as the sum of ratings for a movie divided by the number of ratings it received, adjusted by a regularization term to account for movies with fewer ratings. This approach helps mitigate the bias towards movies with a small number of high ratings.

Next, we ranked the movies based on their popularity scores and selected the top 100 movies. These top movies were then recommended to all users in the validation and test sets.

To evaluate the performance of the popularity baseline model, we used several ranking metrics. For each user in the validation and test sets, we compared the recommended movies against the movies they had actually rated. The key metrics used for evaluation included Mean Average Precision (MAP), Precision at 100, and Normalized Discounted Cumulative Gain (NDCG) at 100.

The popularity baseline model provides a straightforward benchmark, highlighting the effectiveness of recommending widely liked movies. While simple, this model sets a foundation for comparing more sophisticated recommendation algorithms.

ALS

ALS Fine-Tuning To optimize the performance of our recommendation system, we conducted fine-tuning of the Alternating Least Squares (ALS) model using the MovieLens dataset. The main objective was to identify the best hyperparameters for the model, specifically the rank and regularization parameters, by evaluating them on a smaller subset of the dataset.

We began by loading the validation and test datasets, ensuring they were in the appropriate format for training the ALS model. We experimented with various hyperparameters, including ranks of 10, 50, 100, and 200, and regularization parameters of 0.01, 0.1, and 1. Using a cross-validation approach with five folds, we assessed the model's performance to identify the best combination of parameters.

Our evaluation metric of choice was Mean Average Precision (MAP), which measures the precision of the recommendations. After extensive experimentation, we found that the best parameters for the small dataset were a rank of 200 and a regularization parameter of 0.01. These parameters yielded the highest MAP, indicating the model's effectiveness in generating accurate recommendations.

With the optimal parameters identified, we trained the ALS model on the larger dataset to ensure scalability and robustness. This fine-tuning process allowed us to enhance the recommendation system's performance by leveraging the most effective hyperparameter settings, ensuring high-quality and relevant recommendations for users.

ALS Model Training After fine-tuning the ALS model on the small dataset, we proceeded to train the model on the larger MovieLens dataset using the optimal parameters identified: a rank of 200 and a regularization parameter of 0.01. This step ensures that our recommendation system is robust and scalable, capable of handling extensive data while providing high-quality recommendations. The training process involved the following steps:

1. **Data Preparation:** We loaded the large training dataset from Parquet files.

2. **Model Training:** Using the PySpark implementation of ALS, we trained the model with the identified best parameters. The model was configured to handle implicit feedback

and mitigate the cold start problem by dropping users and items not seen in the training set.

3. **Evaluation:** Although the primary focus was on training, we also ensured that the model's predictions were evaluated using key metrics such as Mean Average Precision (MAP) and Normalized Discounted Cumulative Gain (NDCG). This evaluation step, while not detailed here, involves comparing the predicted rankings of movies for users with their actual preferences, ensuring the model's effectiveness.

Training the ALS model on the large dataset ensures that the recommendation system can scale efficiently and maintain high performance, providing users with personalized and relevant movie suggestions.

Evaluation

Fine-tuning result To identify the optimal hyperparameters for the ALS model, we conducted extensive fine-tuning using the small MovieLens dataset. We experimented with different values for the rank and regularization parameters, evaluating the performance of each configuration using key metrics such as RMSE and Mean Average Precision (MAP). The table below summarizes the results of our experiments, highlighting the best performing model, which had a rank of 50 and a regularization parameter of 0.1.

Table 3: ALS Fine-Tuning Results

Rank	RegParam	RMSE	mAP
10	0.01	1.0439	0.9375
10	0.1	0.8868	0.9403
10	1.0	1.2925	0.9391
50	0.01	1.1817	0.9379
50	0.1	0.8811	0.9411
50	1.0	1.2921	0.9391
100	0.01	1.2837	0.9372
100	0.1	0.8850	0.9416
100	1.0	1.2921	0.9391
200	0.01	1.3960	0.9377
200	0.1	0.8846	0.9416
200	1.0	1.2921	0.9391

Table 4: Result on Large

Model	Rank	RegParam	RMSE	mAP
Popularity Baseline	NA	NA	0.8666	0.0035678
ALS	200	0.1	0.8846	0.345789

Conclusion

This capstone project successfully developed and evaluated a personalized recommendation system using the MovieLens dataset. The project involved two main tasks: customer segmentation using a MinHash-based Locality Sensitive Hashing (LSH) approach and building a collaborative filtering-based recommendation system with Spark's Alternating Least Squares (ALS) method.

For customer segmentation, we efficiently identified "movie twins" by leveraging MinHash signatures and LSH, achieving a high correlation among top similar user pairs. In the recommendation system, we began with a popularity baseline model and advanced to an ALS model. Fine-tuning the ALS model on a small dataset allowed us to identify optimal hyperparameters, which were then applied to train the model on a larger dataset. The fine-tuned ALS model demonstrated superior performance in providing accurate and personalized movie recommendations.

Overall, this project highlights the effectiveness of combining efficient data processing techniques with robust machine learning algorithms to create scalable and accurate recommendation systems. The results demonstrate significant improvements over baseline models, validating our approach and methodology.

References

- Kula, Maciej. "Metadata embeddings for user and item cold-start recommendations." *arXiv preprint arXiv:1507.08439* (2015).
- Chai, T. and Draxler, R. R.: Root mean square error (RMSE) or mean absolute error (MAE)? – Arguments against avoiding RMSE in the literature, *Geosci. Model Dev.*, 7, 1247–1250, https://doi.org/10.5194/gmd-7-1247-2014, 2014.
- Distinguishability, Consistent. "A Theoretical Analysis of Normalized Discounted Cumulative Gain (NDCG) Ranking Measures." (2013).
- (2009) Mean Average Precision. In: LIU L., ÖZSU M.T. (eds) Encyclopedia of Database Systems. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-39940-9_3032.
- Takács, Gábor, and Domonkos Tikk. "Alternating least squares for personalized ranking." *Proceedings of the sixth ACM conference on Recommender systems*. 2012.