DS-GA 1004 Capstone Project

CAP 79: Rundong Guo, Jinran Jin, Nicole Lee, Qianqian Zhao

Github link: https://github.com/nyu-big-data/capstone-bdcs-79

Author contributions: Task 1&5: Rundong Guo; Task 2&4: Jinran Jin; Task 1&3: Nicole Lee, Qianqian Zhao. We worked on the final report together.

Consumer segmentation:

1. Top 100 Most Similar Pairs

To support customer segmentation through behavioral similarity, we developed a scalable PySpark pipeline to identify the top 100 most similar users‘ pairs termed “movie twins” based solely on the sets of movies they rated, ignoring rating values. Using the datasketch library, we applied a MinHash-based locality-sensitive hashing (LSH) algorithm to approximate Jaccard similarity efficiently. To handle the full MovieLens dataset with over 330,000 users, we adopted a batching strategy, dividing user MinHash signatures into groups of 10,000 for memory-efficient processing. Within each batch, an LSH index was built to identify candidate pairs, and exact Jaccard similarities were computed only for queried matches. We retained only pairs with at least 20 rated movies per user and eliminated redundant comparisons. After aggregating all batches, we extracted the top 100 pairs with the highest similarity, ranging from 1.0 to 0.9102.

This approach allowed us to scale to large data volumes while preserving accuracy, validating the use of MinHash-LSH for efficient and effective customer segmentation. I list the top 100 most similar pairs (including a suitable estimate of their similarity for each pair), sorted by similarity in question1_large_result.csv.

2. Correlation Comparison

Our validation process applied Pearson correlation analysis to assess whether algorithmically identified movie twins demonstrated truly similar rating patterns. Using PySpark's distributed computing framework, we processed the full MovieLens dataset in memory-efficient batches, calculating correlations between users who had rated at least five movies in common. The diagnostic results showed compelling evidence of our algorithm's effectiveness: top pairs exhibited an average correlation of 0.1966 compared to just 0.0644 for random pairs, representing a difference of 0.1322. The top pairs achieved a maximum correlation of 0.9994, exceeding the random pairs' maximum of 0.9468, while both groups showed similar negative correlation floors (-0.3281 and -0.9540 respectively). Statistical validation through Welch's t-test produced a t-statistic of 2.8011 with a p-value of 0.0056, firmly establishing the statistical significance of this difference. With 99 valid correlations from our top pairs and 100 from random pairs, we had sufficient data to conclude that our minHash-based similarity identification successfully captured meaningful patterns in user preferences that transcended algorithmic artifacts, confirming the validity of our customer segmentation approach.

Movie Recommendation

3. Train/Test/Validation Split

To ensure reliable model training and evaluation, we partitioned the full MovieLens ratings dataset using a random split strategy via partition_ratings.py, dividing the dataset into three distinct subsets: training (60%), validation (20%), and test (20%). This separation enables us to perform fair model selection and final evaluation while avoiding data leakage. The resulting partitions are saved back to HDFS in CSV format with headers, making them compatible with downstream Spark pipelines for both baseline and ALS model evaluation. After execution, we verified that the files were successfully saved to the specified HDFS paths and that the row counts approximately reflected the 60/20/20 proportions. These partitioned datasets were used for all subsequent model training, tuning, and evaluation.

4. Popularity Baseline Evaluation

Our baseline recommendation model implemented a straightforward popularity-based approach, recommending the same top 100 movies to all users based solely on how many ratings each movie received in the training dataset. Using PySpark's distributed computing capabilities, we processed the MovieLens dataset efficiently, first identifying the most-rated movies (with IDs 318, 356, 296, 2571, and 593 leading the list) and then evaluating this one-size-fits-all approach against user preferences in the validation set. Performance metrics revealed modest effectiveness: precisionAt10 of 0.0644, ndcgAt10 of 0.0833, and mapAt10 of 0.0378, with slightly lower precision but higher ndcg when

expanded to 100 recommendations (0.0366 and 0.1404 respectively). The sample evaluation data illustrated the inherent limitations of this approach—while some users had watched popular films like movie #318, many had unique preferences not captured by popularity alone, explaining the relatively low precision scores. This baseline establishes a performance floor against which our more sophisticated ALS-based collaborative filtering model will be compared, while confirming that even a simple model can capture some user preferences through the universal appeal of widely-watched films.

5. Latent Factor Model Documentation and Evaluation

After finishing the baseline model, we moved on to building a personalized recommendation system using Spark's ALS (alternating least squares) algorithm. The goal was to uncover hidden patterns between users and items by learning latent factors. We used the pyspark.ml.recommendation.ALS module and focused on tuning two key hyperparameters: the rank (which controls how many latent factors we learn) and the regularization parameter (which helps prevent overfitting). To make everything consistent, we used the same training, validation, and test splits we created earlier.We ran a grid search over several combinations of rank values {5, 10, 20} and regParam values {0.01, 0.1, 1.0}. Each model was trained with coldStartStrategy='drop' so we wouldn't get undefined metrics, and we evaluated performance on the validation set using Spark's built-in ranking metrics. The best model turned out to be the one with rank 20 and regParam 0.1, which got a precision@100 of 1.93e-5 and ndcg@100 of 1.70e-4.

When compared to the baseline's precision@100 of 0.0366 and ndcg@100 of 0.1404, the ALS model didn't do very well. We took that best ALS config and retrained it on the full training data, then tested it. The test set results were slightly better, with precision@100 of 4.87e-5 and ndcg@100 of 4.04e-4. We saved these final results in als_final_eval_clean.csv, and all the validation results from tuning are in als_hyperparam_clean.csv for transparency and reproducibility.