Project Report
Big Data (DS-GA 1004): Capstone Project (Group:83)

Repository Link: https://github.com/nyu-big-data/capstone-bdcs-83
Group Members: Gabriel Nixon (gr2513), Harshit Soni (hs5666), Aditya Desai (ad7602)

A1. Customer Segmentation

Objective and Methodology (scripts/final_minhash.py):

The first deliverable focused on customer segmentation by identifying pairs of users who exhibited the most similar movie-watching patterns. For simplicity and scalability, similarity was defined by the set of movies rated by each user, without considering the specific rating values. The goal of this phase was to identify pairs of users with highly overlapping movie-watching behavior and to validate whether such behavioral similarity also corresponded to aligned rating preferences.

Data Preparation (scripts/user_movie_sets.py)

To facilitate this analysis, we constructed a user-movie interaction set for each user:

1. The ratings data was ingested and converted into a structured format where each user was associated with the set of movies they had rated.
2. To ensure comparisons and reduce noise, we filtered out users with fewer than five rated movies.
3. The resulting user-movie sets were stored in Parquet format, which supports efficient storage and high-performance distributed processing in Spark.

To efficiently estimate pairwise user similarity over a large dataset, we applied MinHash combined with Locality Sensitive Hashing (LSH). This approach allowed us to approximate the Jaccard similarity between the movie sets rated by each user while significantly reducing computational complexity.

1. MinHash Signature Generation: Using the datasketch library, we computed MinHash signatures for each user's set of rated movies. Each movie ID was encoded into a hash function, producing a signature that preserves similarity properties.
2. Locality Sensitive Hashing (LSH): We constructed an LSH index with a similarity threshold of 0.5 and 64 permutations, inserting each user's MinHash signature into the index. This enabled us to efficiently query for candidate pairs of similar users without performing a full pairwise comparison.
3. Candidate Pair Generation and Similarity Filtering: For each user, we queried the LSH index to retrieve other users with potentially similar viewing patterns. We then computed the actual Jaccard similarity between their MinHash signatures. Duplicate and self-pairs were excluded. The resulting user pairs were ranked by similarity, and the top 100 most similar pairs were selected for further analysis. The link to our entire result file is here.

user1	user2	Jaccard
16036	201412	1.0
107105	163339	1.0
90605	160998	1.0
...	...	...
...	...	...
2667	170026	1.0
212	173282	1.0
115135	170222	1.0

Interpretation:

The top 100 most similar user pairs were identified using MinHash-based Jaccard similarity, all with a perfect score of 1.0, indicating identical sets of rated movies. To balance data diversity and reliability, we initially tested higher thresholds such as 3,000 ratings per user, but found it overly restrictive. We settled on a minimum of 5 ratings, which preserved a broader range of user behaviors. While the perfect Jaccard scores suggest strong behavioral similarity, Pearson correlation validation confirmed that such similarity does not always imply aligned rating preferences.

A2: Validation Using Pearson Correlation

Objective and Methodology: (scripts/correlation_validation.py)

While Jaccard similarity identifies users who rated many of the same movies, it does not capture whether those users rated the movies similarly. To validate whether behavioral similarity (co-watching) translated into preference similarity, we computed the Pearson correlation coefficient between the ratings of user pairs.

1. Data Aggregation: We extracted the ratings for all users appearing in the top 100 similar pairs and structured them into a user-movie-rating mapping.
2. Twin Pair Correlation Calculation: For each of the top 100 similar pairs, we identified the set of commonly rated movies. Pairs with fewer than two shared ratings or constant rating values were excluded to ensure statistical validity. The Pearson correlation coefficient was computed for the remaining pairs.
3. Random Pair Benchmarking: To establish a baseline, we randomly sampled 100 user pairs from the dataset and computed their Pearson correlations using the same procedure.

Average correlation for top 100 similar pairs: 0.1509; Average correlation for 100 random pairs: 0.0997

Validation of Similarity Results:

To validate the MinHash-based similarity, we computed Pearson correlations between ratings for the top 100 similar pairs and 100 random pairs. The similar pairs showed an average correlation of 0.1509, compared to 0.0997 for random pairs. While both correlations were modest, the higher value for similar pairs confirms that behavioral similarity modestly reflects preference alignment. This highlights the need to combine co-watching patterns with preference-based models like ALS for better recommendation accuracy.

A3: Train/Validation/Test Split

Objective and Methodology: (scripts/split_ratings.py)

Before building recommendation models, we partitioned the ratings data into training, validation, and test sets. This approach enables robust model evaluation and simplifies experimentation by avoiding repeated random splits. The splitting strategy followed a leave-k-out temporal split:

1. Users with fewer than five ratings were excluded to ensure sufficient data for meaningful recommendations.
2. For each remaining user:
    - All ratings were sorted chronologically.
    - The last two ratings were reserved—one for the validation set and one for the test set.
    - The remaining ratings were allocated to the training set.
3. The split was performed in 80:10:10 ratio.

This method ensures that every user in the validation and test sets has corresponding ratings in the training set, mitigating cold-start issues commonly encountered in collaborative filtering.

Results and Interpretation:

Total users processed: 307,413

Split	Number of Ratings
Training Set	33,165,105
Validation Set	307,413
Test Set	307,413

The resulting data splits were saved in Parquet format to enable efficient loading and optimized processing in Spark-based workflows. By preserving the temporal order of ratings and maintaining consistent user presence across splits, this approach supports realistic and stable evaluation of recommendation models.

A4: Popularity Baseline Evaluation

Objective and Methodology: (scripts/final_popularity.py)

Before implementing a personalized recommendation model, we established a popularity-based baseline to provide a simple yet informative benchmark for recommendation performance. This model ranks movies solely based on their overall popularity and average ratings in the training set, without considering individual user preferences. The popularity baseline pipeline followed these steps:

1. Identify Popular Movies: From the training data, we computed the number of ratings (*count*) and the average rating for each movie. Movies were ranked by rating count. The top 100 movies were selected as the baseline recommendations.
2. Generate Recommendations: Every user in the test set was recommended the same top 100 movies.
3. Prepare Ground Truth (True Items): For each user, the most recent ratings were extracted as the true items for evaluation. This ensured that the evaluation considered movies rated after the training data, reflecting a realistic recommendation scenario.
4. Evaluation Metrics: Evaluated the popularity model using Precision@100, NDCG@100, and MAP.

5. Validation RMSE: As an additional check, we computed the Root Mean Squared Error (RMSE) between the predicted and actual ratings for validation purposes, though this metric is secondary for ranking-based baselines.

Results and Interpretation:

Metric	Value
MAP	0.0895
NDCG@100	0.2491
Precision@100	0.0100
Validation RMSE	1.0291

Number of users evaluated: 53,575

The popularity model achieved moderate NDCG and MAP scores but low Precision@100, which is expected for non-personalized baselines that do not account for individual preferences. While the model provides reasonable recommendations for widely liked movies, it lacks the ability to tailor suggestions to specific user tastes, highlighting the need for more sophisticated models like collaborative filtering.

A5: Collaborative Filtering with ALS

Objective and Methodology: (scripts/als_final.py)

Following the popularity-based baseline, we implemented a collaborative filtering model using Alternating Least Squares (ALS) to generate personalized movie recommendations. ALS is a widely used matrix factorization technique that learns latent factor representations for users and items, capturing patterns in user preferences.

We used the ALS algorithm from the pyspark.ml.recommendation module, which factors the user-item rating matrix into lower-dimensional latent factor representations. These factors make prediction of unrated items for each user.

The following hyperparameters were tuned to optimize performance:

1. **Rank (latent dimensions):** [10, 20, 50]
2. **Regularization Parameter:** [0.01, 0.05, 0.1]
3. **Cold Start Strategy:** Drop (to avoid including users/items unseen during training).

Hyperparameter Tuning:

The ALS model was trained on the training set and evaluated on the validation set across all hyperparameter combinations. For each model, we generated top-100 recommendations for each user and evaluated the results using Mean Average Precision (MAP) as the primary metric.

Evaluation Protocol:

The best-performing model on the validation set was then evaluated on the test set using MAP (Mean Average Precision), NDCG@100 (Normalized Discounted Cumulative Gain at rank 100), and Precision@100 (Proportion of relevant items among the top 100 recommendations)

Hyperparameter Tuning Results:

Rank	RegParam	Validation MAP
10	0.01	0.0155
10	0.05	0.0159
10	0.1	0.0166
20	0.01	0.0101
20	0.05	0.0177
20	0.1	0.0184
50	0.01	0.0114
50	0.05	0.0159
50	0.1	0.0193 (Best)

The best ALS configuration achieved a validation MAP of 0.0193 with rank = 50 and regParam = 0.1.

Test Set Results and Interpretation:

Number of users evaluated: 307,284

Metric	Value
MAP	0.0167
NDCG@100	0.0595
Precision@100	0.0027

The ALS model outperformed the popularity baseline in MAP and NDCG, reflecting improved ranking quality through personalized recommendations. However, the Precision@100 remained relatively low, which is common in large, sparse recommendation datasets where only a small fraction of possible user-item pairs are relevant. The recall and hit rate indicate that for approximately 26.7% of users, at least one relevant item appeared in the top 100 recommendations, suggesting reasonable coverage even though precision was limited. The relatively modest MAP improvement compared to the popularity model highlights the challenges of cold-start problems, sparse data, and the limitations of implicit feedback modeling. Nevertheless, ALS provided a meaningful enhancement over the naive baseline.

Team Contributions

All members of Group 83 collaborated closely throughout the capstone project, ensuring consistent progress and rigorous validation across all deliverables. Specific responsibilities were distributed as follows:

- Gabriel Nixon led the design and implementation of the MinHash + LSH-based similarity pipeline, including the computation of Jaccard similarities, Pearson correlation validation. He also contributed to the ALS model development and hyperparameter tuning.
- Harshit Soni focused on the data preprocessing pipeline, including filtering, train/validation/test splitting, and Parquet data handling to ensure efficient storage and scalability. He also contributed to the popularity baseline model and evaluation metrics implementation.
- Aditya Desai worked on the popularity baseline evaluation, metric computation, and provided support for debugging the ALS training pipeline. He also assisted in refining the evaluation scripts and generating validation metrics.

While individual tasks were divided to ensure efficiency, all members participated in code reviews, testing, and the interpretation of results.