# DSGA1004 - Big Data

Capstone Project: MovieLens Recommendation System

Amaan Mansuri (am14419), Ninad Chaudhari (nac8810),
Shravan Khunti (ssk10036)

May 7, 2025

## GitHub Repository

`https://github.com/nyu-big-data/capstone-bdcs-69`

## Customer Segmentation

### Q1: Finding Similar User Pairs (Movie Twins)

**Approach** To begin, we converted all the raw CSV files into Parquet format for faster I/O and efficient columnar access. Using our `preprocessed_grouping.sh` and `preprocessed_grouping.py` scripts (located in the q1 folder of our repository), we transformed the ratings data into a two-column format with `userId` and a corresponding list of `movieIds` representing the movies watched by each user. Importantly, we ignored the actual rating values at this stage and focused solely on the set of movies rated by each user.

We then used the `datasketch` library to generate MinHash signatures for each user, capturing their "movie-watching style" based on the set of movies they had rated. These signatures were indexed using **Locality Sensitive Hashing (LSH)** to efficiently retrieve similar users.

For every user, we queried the LSH index to find candidate pairs and computed the **Jaccard similarity** between the movie sets of each user pair. We filtered pairs to include only those where both users had rated more than a threshold number of movies, experimenting with thresholds of >5, >10, >20, >30, >40, and >50 movies to eliminate the possibility of unreliable estimates due to sparse data.

Finally, we sorted the user pairs by their estimated Jaccard similarity scores and extracted the **top 100 most similar user pairs (movie twins)** for each threshold. The results for threshold >50 are presented in Table 2 in the Appendix.

### Q2: Validation with Correlations

**Approach** To validate the quality of our top 100 most similar user pairs identified using MinHash (from Q1), we assessed whether their numerical ratings were more aligned compared to random user pairs. We used the Pearson correlation coefficient to quantify similarity in rating behavior.

First, we loaded the original ratings dataset and the CSV file containing the top 100 user pairs for various thresholds on the number of movies rated (e.g., >5, >10, >20, etc.). For each pair, we extracted the movies that both users had rated and retrieved their corresponding ratings.

We computed the Pearson correlation for all qualifying user pairs and averaged the values to obtain a single metric representing the consistency in rating behavior among these "movie twins." Informative warnings were printed when users had constant ratings, and exact matches (correlation = 1.0) were logged for inspection.

**Results** The results confirm a positive relationship between MinHash-based similarity and actual rating similarity. As the threshold on the minimum number of movies rated increases, the average correlation improves significantly, as shown in Table 1.

| **Threshold for Number of Movies Rated** | **Average Correlation** |
|---|---|
| >5 | -0.0014 |
| >10 | 0.2244 |
| >20 | 0.2402 |
| >30 | 0.2713 |
| >40 | 0.3710 |
| **>50** | **0.4101** |

Table 1: Average Pearson Correlation Coefficient for Similar User Pairs

**Random Correlation = 0.0234**

**Analysis** These results demonstrate a clear trend: as we increase the threshold for the minimum number of movies rated, the average correlation between similar user pairs also increases. We conducted this analysis at different thresholds to identify patterns in user similarity as the amount of available data grows. The correlation rises from -0.0014 at threshold >5 to 0.4101 at threshold >50, showing that MinHash similarity becomes increasingly aligned with rating behavior as users rate more movies, on the otherhand when users where selected at **random**, we got an **average correlation of 0.0234**.

Interestingly, for the lowest threshold (>5), the correlation is negative and lower than random pairs. This suggests that when users have rated very few movies, the MinHash similarity based solely on the set of movies rated may not accurately reflect their rating behavior. This finding highlights the importance of having sufficient data points to make meaningful similarity assessments and informed recommendations.

## Movie Recommendation

### Q3: Data Partitioning

**Approach** To prepare for building and evaluating our recommendation models, we partitioned the dataset into training, validation, and test sets. We began by loading the grouped user-to-movie mapping from the `grouped_user_movies.parquet` file, which contains a list of movie IDs rated by each user. We filtered out users with fewer than 5 rated movies to ensure each user had sufficient data for meaningful splits.

For each remaining user, we randomly shuffled their movie list and applied an 80/10/10 split: 80% of the movies were assigned to the training set, 10% to validation, and the remaining 10% to the test set. We maintained user-level granularity during the split to ensure all interactions within a single partition came from the same user, which avoids data leakage across sets.

The resulting datasets were converted into Pandas DataFrames and saved as separate Parquet files (`train.parquet`, `val.parquet`, `test.parquet`) for efficient access in later stages. This pre-splitting strategy allows for reproducibility and easier tuning of future models.

**Results:** The final counts of interactions in each split were as follows:

- **Train**: 26,906,772 interactions
- **Validation**: 3,361,115 interactions
- **Test**: 3,512,044 interactions

This partitioning approach ensures that our recommendation models are trained on a substantial portion of each user's preferences while reserving sufficient data for validation and testing. By maintaining separate files for each partition, we streamlined the workflow for subsequent modeling tasks and enabled consistent evaluation across different model configurations.

### Q4: Popularity Baseline Model

**Approach** As a baseline recommender, we implemented a popularity-based model where items are ranked based on their adjusted popularity scores. The scoring formula we used is:

$$P[i] \leftarrow \frac{\sum_u R[u,i]}{|\mathrm{R}[:,i]| + \beta} \tag{1}$$

Here, $R[u,i]$ is the rating given by user $u$ to item $i$, $|\mathrm{R}[:,i]|$ denotes the number of users who rated item $i$, and $\beta$ is a regularization term that prevents items with very few ratings from dominating the rankings. This adjustment ensures a fair balance between item popularity and robustness to low interaction counts.

The model was built and evaluated using Dask for efficient distributed computation. We precomputed statistics on each movie's total rating sum and count, then iteratively computed popularity scores for a range of $\beta$ values. For each value of $\beta$, we generated the top-k recommendations and evaluated them using **NDCG@k** (Normalized Discounted Cumulative Gain), a ranking metric that accounts for the position of relevant items in the recommendation list.

We grouped validation and test data by user and treated their held-out movie interactions as ground truth. For each user, relevance was defined as binary (1 if the item appeared in the user's true set, 0 otherwise). We evaluated DCG and IDCG to compute the NDCG for each user, then averaged over all users.

Finally, we selected the $\beta$ that maximized **validation NDCG@k** and evaluated the model on the test set using the same value. The process was visualized via a line plot of NDCG scores against beta values, and results were saved for reproducibility. This model served as a strong, interpretable baseline to compare against more complex models like ALS.

**Results:** After tuning the regularization parameter, we found that $\beta = 500$ produced the best performance, with an **NDCG@100** score of **0.18290** on the test set.

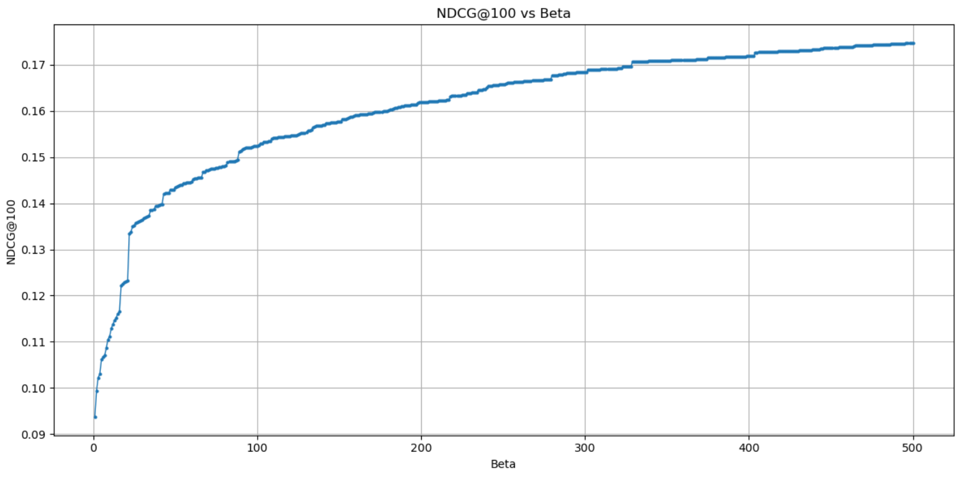

Figure 1: NDCG@100 scores for different values of regularization parameter $\beta$

Figure 1 shows how the NDCG@100 score varies with different values of $\beta$. The optimal value balances between promoting highly-rated items and penalizing those with few ratings. This baseline model provides a solid foundation for comparison with more sophisticated approaches in the next section.

## Q5: Latent Factor Model using ALS

**Approach: Explicit Feedback Model** To develop a more robust recommendation system, we implemented the **Alternating Least Squares (ALS)** algorithm using explicit feedback with PySpark's `pyspark.ml.recommendation` module. ALS models user-item interactions by learning latent factors for both users and items through matrix factorization. We focused on tuning three key hyperparameters: **rank** (number of latent dimensions), **maxIter** (maximum number of training iterations), and **regParam** (regularization parameter). Specifically, we tested all combinations of configurations with rank $\in \{10, 20, 40, 60\}$, maxIter $\in \{10, 15, 20\}$, and regParam $\in \{0.01, 0.05, 0.1, 0.2\}$.

The model was trained on the preprocessed training set, and top-100 recommendations were generated for each user. Evaluation was conducted using **NDCG@100** (Normalized Discounted Cumulative Gain), a metric that accounts for both the relevance and position of recommended items. We compared the predicted rankings against ground truth item lists from the validation and test sets.

Our best-performing configuration was:

- **Rank**: 60
- **Max Iterations**: 15
- **Regularization**: 0.05
- **Validation NDCG@100**: 0.01382
- **Test NDCG@100**: 0.01497

These results demonstrate that ALS effectively captures collaborative patterns in user preferences and provides a stronger baseline than random or popularity-based approaches.

**Approach: Implicit Feedback Model** In addition to the explicit feedback model, we implemented an **implicit-feedback variant** of the ALS algorithm, leveraging both user ratings and tag interactions as a proxy for user engagement and preference. To enhance the confidence signal, we enriched the training data using both the `ratings.csv` and `tags.csv` files. The ratings.csv file contains 0.5 to 5.0 star ratings, while tags.csv includes user-generated tags that serve as qualitative indicators of interest or familiarity with a movie.

Inspired by the formulation presented in Hu et al., 2008 (IEEE), we modeled implicit preference confidence using the formula:

$$c_{ui} = 1 + \alpha \cdot r_{ui} + \beta \cdot \text{tagged}_{ui} \tag{2}$$

Here, $r_{ui}$ is the explicit rating, and $\text{tagged}_{ui}$ is a binary indicator denoting whether the user tagged the movie. The parameters $\alpha$ and $\beta$ were tuned to modulate the influence of ratings and tags on the final confidence score.

We used Spark's ALS with `implicitPrefs=True` and performed hyperparameter tuning over:

- **Rank**: 60
- **Max Iterations**: 15
- **Regularization**: 0.05
- **Alpha (rating confidence multiplier)**: 40
- **Tag Bonus**: 5

The model generated top-100 movie recommendations per user, which were evaluated using **NDCG@100**. The implicit model significantly outperformed the explicit variant, yielding:

- **Validation NDCG@100**: 0.18188
- **Test NDCG@100**: 0.18971

These results highlight the power of incorporating soft signals like tags to enrich implicit feedback, and demonstrate that properly tuned confidence-weighted ALS models can capture nuanced user preferences more effectively than their explicit counterparts.

**Analysis:** Comparing the three recommendation approaches reveals interesting patterns in model performance and recommendation diversity. The popularity baseline achieved an NDCG@100 of 0.18290, significantly outperforming the explicit ALS model (NDCG@100: 0.01497), while the implicit ALS model (NDCG@100: 0.18971) performed comparably to the baseline.

However, this higher score for the baseline came at the cost of recommendation diversity, as the baseline primarily suggested the most popular movies with minimal personalization. The explicit ALS model, despite its lower numerical performance, demonstrated a much more diverse recommendation range by learning user-specific preferences through latent factors.

One plausible explanation for this apparent contradiction is the Missing Not At Random (MNAR) effect in user behavior. Users tend to watch and rate popular movies at a higher frequency, creating a bias in the training data. This leads to situations where recommending popular movies (as the baseline does) produces higher NDCG scores, even though they may not represent truly personalized preferences.

The implicit model's performance approaching that of the baseline can be attributed to its design philosophy. By incorporating rating magnitudes and tag interactions as confidence weights, we essentially amplified signals from highly-rated and tagged items – many of which are popular movies. This approach creates a hybrid effect: while maintaining some personalization through ALS, the confidence weighting pushes recommendations toward items with stronger user engagement patterns, which often correlate with popularity.

## Contributions:

**Amaan:** Hyperparameter tuning, top 100 users (movie twins), MinHash LSH implementation, Pre-processing (user grouping), Correlation implementation

**Ninad:** Hyperparameter tuning, ALS (Implicit and Explicit) model implementation, Pre-processing for implicit model confidence generation

**Shravan:** Hyperparameter tuning, Train/Test/Val split implementation, Baseline (popularity based model implementation), documentation

## Appendix

### Table of Top 100 Most Similar User Pairs

Table 2: Top 100 Most Similar User Pairs with Jaccard Similarity

| userId1 | userId2 | estimated_jaccard | num_of_user1_ratings | num_of_user2_ratings |
|---:|---:|---:|---:|---:|
| 19947.0 | 139527 | 1.0 | 1049 | 1050 |
| 41157.0 | 134345 | 1.0 | 64 | 65 |
| 137812.0 | 181530 | 1.0 | 419 | 419 |
| 77647.0 | 294432 | 1.0 | 3280 | 3273 |
| 38929.0 | 322547 | 1.0 | 70 | 70 |
| 167541.0 | 173768 | 0.9921875 | 56 | 55 |
| 242889.0 | 271504 | 0.9921875 | 135 | 136 |
| 25084.0 | 294432 | 0.984375 | 3224 | 3273 |
| 25084.0 | 77647 | 0.984375 | 3224 | 3280 |
| 41012.0 | 300048 | 0.984375 | 389 | 394 |
| 84354.0 | 99176 | 0.984375 | 70 | 69 |
| 71607.0 | 307748 | 0.984375 | 200 | 198 |
| 25084.0 | 86967 | 0.984375 | 3224 | 3263 |
| 9890.0 | 134345 | 0.984375 | 64 | 65 |
| 9890.0 | 41157 | 0.984375 | 64 | 64 |
| 67893.0 | 189763 | 0.984375 | 79 | 80 |
| 60944.0 | 311877 | 0.9765625 | 285 | 291 |
| 20860.0 | 189763 | 0.9765625 | 80 | 80 |
| 119099.0 | 194288 | 0.9765625 | 155 | 166 |
| 9890.0 | 50336 | 0.9765625 | 64 | 63 |
| 242906.0 | 290377 | 0.9765625 | 67 | 70 |
| 103228.0 | 180562 | 0.96875 | 60 | 58 |
| 198214.0 | 289867 | 0.96875 | 64 | 62 |
| 198214.0 | 275979 | 0.96875 | 64 | 63 |
| 86967.0 | 294432 | 0.96875 | 3263 | 3273 |
| 106707.0 | 198214 | 0.96875 | 67 | 64 |
| 62868.0 | 134061 | 0.96875 | 57 | 54 |
| 30254.0 | 292994 | 0.96875 | 302 | 300 |
| 77647.0 | 86967 | 0.96875 | 3280 | 3263 |
| 259490.0 | 315961 | 0.96875 | 85 | 82 |
| 212802.0 | 263838 | 0.96875 | 51 | 55 |
| 20860.0 | 67893 | 0.9609375 | 80 | 79 |
| 83880.0 | 261244 | 0.9609375 | 3066 | 3212 |
| 20860.0 | 278152 | 0.9609375 | 80 | 75 |
| 41157.0 | 50336 | 0.9609375 | 64 | 63 |
| 4989.0 | 288655 | 0.9609375 | 56 | 54 |
| 47902.0 | 94671 | 0.9609375 | 90 | 88 |
| 143514.0 | 261096 | 0.9609375 | 162 | 158 |
| 50336.0 | 134345 | 0.9609375 | 63 | 65 |
| 155094.0 | 189763 | 0.9609375 | 76 | 80 |
| 112474.0 | 241351 | 0.9609375 | 538 | 528 |
| 134572.0 | 198214 | 0.953125 | 62 | 64 |
| 81527.0 | 123851 | 0.953125 | 94 | 100 |
| 134345.0 | 290377 | 0.953125 | 65 | 70 |
| 20860.0 | 112294 | 0.953125 | 80 | 75 |
| 20860.0 | 39534 | 0.953125 | 80 | 79 |
| 273826.0 | 278152 | 0.953125 | 74 | 75 |
| 81745.0 | 106707 | 0.953125 | 70 | 67 |
| 46734.0 | 135003 | 0.953125 | 2282 | 2199 |
| 159358.0 | 198214 | 0.953125 | 63 | 64 |
| 73177.0 | 154923 | 0.953125 | 53 | 54 |
| 41157.0 | 290377 | 0.953125 | 64 | 70 |
| 67893.0 | 278152 | 0.953125 | 79 | 75 |
| 81745.0 | 198214 | 0.953125 | 70 | 64 |
| 65621.0 | 263838 | 0.953125 | 54 | 55 |
| 39534.0 | 189763 | 0.953125 | 79 | 80 |

| userId1 | userId2 | estimated_jaccard | num_of_user1_ratings | num_of_user2_ratings |
|---:|---:|---:|---:|---:|
| 98555.0 | 304729 | 0.9453125 | 55 | 58 |
| 112294.0 | 189763 | 0.9453125 | 75 | 80 |
| 135882.0 | 288844 | 0.9453125 | 70 | 74 |
| 83880.0 | 86967 | 0.9453125 | 3066 | 3263 |
| 98839.0 | 136037 | 0.9453125 | 54 | 59 |
| 98839.0 | 134061 | 0.9453125 | 54 | 54 |
| 134061.0 | 136037 | 0.9453125 | 54 | 59 |
| 189763.0 | 206872 | 0.9453125 | 80 | 84 |
| 66348.0 | 134345 | 0.9453125 | 60 | 65 |
| 41157.0 | 66348 | 0.9453125 | 64 | 60 |
| 41157.0 | 74091 | 0.9453125 | 64 | 57 |
| 9890.0 | 242906 | 0.9453125 | 64 | 67 |
| 11487.0 | 62855 | 0.9453125 | 60 | 58 |
| 122890.0 | 134345 | 0.9453125 | 64 | 65 |
| 8746.0 | 286001 | 0.9453125 | 133 | 135 |
| 41157.0 | 122890 | 0.9453125 | 64 | 64 |
| 134631.0 | 223888 | 0.9453125 | 2198 | 2340 |
| 155094.0 | 206872 | 0.9453125 | 76 | 84 |
| 67893.0 | 155094 | 0.9453125 | 79 | 76 |
| 68753.0 | 174216 | 0.9453125 | 520 | 500 |
| 92747.0 | 189763 | 0.9453125 | 76 | 80 |
| 86458.0 | 242505 | 0.9453125 | 52 | 55 |
| 44343.0 | 198214 | 0.9453125 | 61 | 64 |
| 50336.0 | 66348 | 0.9453125 | 63 | 60 |
| 74091.0 | 134345 | 0.9453125 | 57 | 65 |
| 76755.0 | 103216 | 0.9453125 | 386 | 398 |
| 147443.0 | 167839 | 0.9453125 | 59 | 56 |
| 26617.0 | 147443 | 0.9453125 | 57 | 59 |
| 31266.0 | 81745 | 0.9453125 | 65 | 70 |
| 44343.0 | 106707 | 0.9453125 | 61 | 67 |
| 81745.0 | 304728 | 0.9453125 | 70 | 68 |
| 7753.0 | 71778 | 0.9453125 | 109 | 108 |
| 40884.0 | 122168 | 0.9453125 | 52 | 52 |
| 174815.0 | 294432 | 0.9375 | 3389 | 3273 |
| 265793.0 | 294516 | 0.9375 | 99 | 93 |
| 66348.0 | 304729 | 0.9375 | 60 | 58 |
| 181179.0 | 275876 | 0.9375 | 101 | 104 |
| 234484.0 | 263838 | 0.9375 | 52 | 55 |
| 134345.0 | 287308 | 0.9375 | 65 | 58 |
| 39534.0 | 112294 | 0.9375 | 79 | 75 |
| 92747.0 | 155094 | 0.9375 | 76 | 76 |
| 39534.0 | 67893 | 0.9375 | 79 | 79 |
| 56575.0 | 123851 | 0.9375 | 90 | 100 |
| 9890.0 | 290377 | 0.9375 | 64 | 70 |