# Capstone Project Report

**Github link: https://github.com/nyu-big-data/capstone-bdcs_66.git**

## Customer segmentation

To identify users with similar movie-watching styles, we implemented a MinHash-based Locality Sensitive Hashing (LSH) algorithm.

### 1. MinHash LSH Implementation

We defined "movie-watching style" as the set of movies rated by a user, regardless of the rating scores. Our implementation followed these main steps:

- **Data Preparation**: We extracted all `(userId, movieId)` pairs and repartitioned the dataset by `userId` to optimize the subsequent `groupBy` operation. For each user, we aggregated the set of rated movies using `groupBy().agg(collect_list("movieId"))`. Users who rated fewer than a threshold number of movies (e.g., 40 or 100) were filtered out to ensure more meaningful similarity comparisons.
- **MinHash Signature Generation**: We generated 128 independent hash functions of the form `h(x) = (a * x + b) % p` and computed a MinHash signature for each user. Each component of the signature is the minimum hash value observed over the user's rated movie set for that hash function.
- **Banding and Bucketing (LSH)**: Each MinHash signature was divided into 8 bands, each containing 16 consecutive hash values. Each band was hashed using SHA1 to create a bucket key. Users whose band hashes matched were grouped into the same bucket.
- **Candidate Pair Generation**: For each bucket (i.e. group of users with the same band hash in a band), we generated all possible user-user combinations.
- **Jaccard Similarity Computation**: For each candidate pair, we computed the exact Jaccard similarity $= \frac{A \cap B}{A \cup B}$ where A and B are the movie sets of the two users.
- **Top 100 Similar Pairs**: The top 100 user pairs were selected by sorting all candidate pairs based on their Jaccard similarity scores. The following is the top 100 pairs with a 40 threshold:

| user1 | user2 | sim* | user1 | user2 | sim* | user1 | user2 | sim* | user1 | user2 | sim* |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 198462 | 263013 | 1 | 9890 | 50336 | 0.984 | 242889 | 271504 | 0.978 | 7753 | 71778 | 0.955 |
| 40479 | 198462 | 1 | 167541 | 173768 | 0.982 | 77647 | 86967 | 0.978 | 168184 | 258335 | 0.955 |
| 38929 | 322547 | 1 | 112474 | 241351 | 0.981 | 282 | 27226 | 0.978 | 41157 | 50336 | 0.954 |
| 63309 | 168141 | 1 | 25084 | 294432 | 0.981 | 41012 | 300048 | 0.977 | 159358 | 198214 | 0.954 |
| 107031 | 263013 | 1 | 260734 | 262614 | 0.98 | 98837 | 294406 | 0.976 | 198214 | 275979 | 0.954 |
| 137812 | 181530 | 1 | 86967 | 294432 | 0.98 | 76755 | 103216 | 0.97 | 57536 | 251649 | 0.953 |
| 49647 | 59869 | 1 | 168141 | 286014 | 0.98 | 9890 | 41157 | 0.969 | 83880 | 261244 | 0.953 |
| 49647 | 63309 | 1 | 68284 | 168141 | 0.98 | 50336 | 134345 | 0.969 | 134572 | 159358 | 0.953 |
| 59869 | 63309 | 1 | 49647 | 255658 | 0.98 | 241273 | 296245 | 0.968 | 26612 | 142948 | 0.951 |
| 49647 | 168141 | 1 | 168141 | 255658 | 0.98 | 103228 | 180562 | 0.967 | 113046 | 294406 | 0.951 |
| 127960 | 195168 | 1 | 117141 | 124614 | 0.98 | 50012 | 174815 | 0.965 | 20860 | 189763 | 0.951 |
| 40479 | 263013 | 1 | 49647 | 286014 | 0.98 | 4989 | 288655 | 0.964 | 147443 | 167839 | 0.949 |
| 40479 | 107031 | 1 | 59869 | 286014 | 0.98 | 20860 | 39534 | 0.963 | 62868 | 134061 | 0.947 |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 59869 | 168141 | 1 | 59869 | 63366 | 0.98 | 216913 | 310906 | 0.961 | 68753 | 174216 | 0.947 |
| 60252 | 329440 | 1 | 59869 | 255658 | 0.98 | 53278 | 193817 | 0.961 | 65621 | 263838 | 0.946 |
| 107031 | 198462 | 1 | 49647 | 63366 | 0.98 | 63366 | 68284 | 0.96 | 198820 | 263838 | 0.946 |
| 77647 | 294432 | 0.998 | 49647 | 68284 | 0.98 | 68284 | 286014 | 0.96 | 135882 | 288844 | 0.946 |
| 19947 | 139527 | 0.997 | 63309 | 286014 | 0.98 | 255658 | 286014 | 0.96 | 86458 | 242505 | 0.945 |
| 71607 | 307748 | 0.99 | 59869 | 68284 | 0.98 | 63366 | 286014 | 0.96 | 234484 | 263838 | 0.945 |
| 67893 | 189763 | 0.988 | 63309 | 255658 | 0.98 | 68284 | 255658 | 0.96 | 73177 | 154923 | 0.945 |
| 30254 | 292994 | 0.987 | 63309 | 68284 | 0.98 | 156310 | 216913 | 0.96 | 63309 | 199922 | 0.943 |
| 84354 | 99176 | 0.986 | 278169 | 310906 | 0.98 | 63366 | 255658 | 0.96 | 59869 | 199922 | 0.943 |
| 9890 | 134345 | 0.985 | 63309 | 63366 | 0.98 | 60944 | 311877 | 0.959 | 168141 | 199922 | 0.943 |
| 41157 | 134345 | 0.985 | 63366 | 168141 | 0.98 | 242906 | 290377 | 0.957 | 49647 | 199922 | 0.943 |
| 25084 | 86967 | 0.984 | 25084 | 77647 | 0.979 | 203482 | 280374 | 0.956 | 77647 | 174815 | 0.942 |

**2. Validation via Rating Correlation**

To validate the MinHash-based similarity:

- We computed the **Pearson correlation of rating scores** for two sets of user pairs:
    1. The top 100 most similar pairs found via MinHash LSH.
    2. 100 randomly selected user pairs from the dataset.
- To make comparisons more meaningful, we applied thresholds on the number of movies rated by each user. This reduced the chance of falsely high similarity scores between users who happened to rate just a few movies in common, such as 3 to 5 movies.

| Pair Type | Average Pearson Correlation |
|---|---|
| Top 100 (MinHash) 40 threshold | 0.37 |
| Top 100 (MinHash) 40 threshold | 0.17 |
| Top 100 (MinHash) 100 threshold | 0.54 |
| Top 100 (MinHash) 100 threshold | 0.07 |

- These results show that user pairs identified through MinHash-based similarity have **consistently higher rating correlation** than randomly paired users.
- Furthermore, when increasing the filtering threshold, the **difference in correlation becomes more pronounced**. This supports the idea that MinHash is more informative when applied to users with more rating histories.
- Although our similarity metric did not consider rating scores directly, the strong correlation among user pairs selected by the algorithm shows that **watching/rating history alone** is a powerful indicator of similar user behavior.

## Movie recommendation

**3.Train/Validation/Test Set Partition and Preprocessing**

- We randomly split the data into training, validation, and test sets using a 6:2:2 ratio.
-To ensure balanced representation across users, we implemented a stratified splitting procedure where, for each user, their rated movies were randomly assigned to the training,

validation, and test sets based on the specified ratio. This was achieved by assigning random row numbers to each user's ratings using a fixed random seed, and then applying deterministic thresholds to allocate rows to the corresponding splits. No filtering was applied to the ratings based on value during this phase, as the purpose was to retain the full user–item interaction set for unbiased model training and evaluation in later stages.
- We define relevance as ratings **greater than or equal to 3**, under the assumption that ratings below this threshold indicate user dissatisfaction or disinterest. This threshold was selected based on common practices in collaborative filtering where mid-to-high ratings are treated as indicators of relevance.

**4. Baseline Model Evaluation**

In this section, we implemented a simple yet effective **popularity-based baseline recommendation model**. The model ranks movies based on their average ratings in the training set. However, to address small-sample bias—where movies with only a few high ratings may appear more relevant than they are in reality—we applied **Bayesian regularization** using a hyperparameter β. This results in a **regularized average rating** for each movie, computed as:

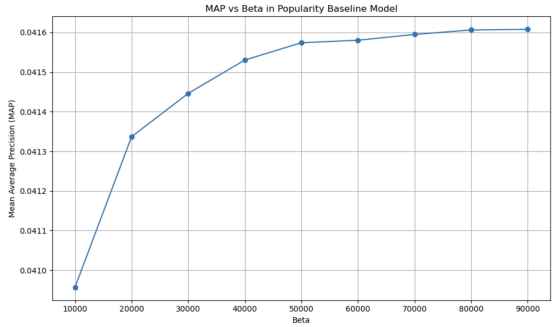
$$p[i] = \frac{\Sigma u\, R[u,i]}{|R[:,i]|+\beta}$$

This formula smooths the scores of low-frequency movies toward zero, ensuring that heavily rated movies are not overshadowed by a few outliers.

To evaluate the effectiveness of this baseline, we used the **Mean Average Precision (MAP)** metric. we performed the following steps:

1. Selected the top 100 movies with the highest regularized ratings.
2. Treated these top-100 as the predicted recommendation list for all users.
3. Defined relevance using a rating threshold of ≥3, meaning only items with a rating of 3 or above were considered relevant by the user.
4. For each user in the validation set, we extracted the list of movies rated ≥3 as the ground truth and compared it against the top-100 predictions using Spark's RankingEvaluator

To tune the regularization hyperparameter β, we evaluated MAP scores across a set of candidate β values. The results are visualized in the figure below:




As seen in the graph, MAP consistently improves with higher β values and begins to flatten around β = 90,000. So our **best β identified**: **90,000**, which yields the highest MAP of **0.0416** on the validation set.

**Final Result:**

Train Set MAP: 0.0676; Test Set MAP: 0.0456; Validation Set MAP: 0.0416

**5. Documentation of latent factor model's hyper-parameters and validation (All MAP values shown in the following figures refer to the performance on the validation set during the tuning process.)**

We assume that the **ml-latest-small** dataset is representative of the **ml-latest** dataset. This implies that the optimal hyperparameters (or model type) identified on the smaller dataset should be similar to those for the larger dataset. Therefore, we first performed hyperparameter tuning on **ml-latest-small** to determine promising candidate configurations. We then applied and fine-tuned parameters close to these candidates on the larger dataset.

The first step in building the model was to choose the appropriate type of latent factor model: either an explicit feedback model or an implicit feedback model. As shown in Figures 1 and 2, the implicit feedback model outperforms the explicit feedback model by achieving a higher mean average precision (MAP) with a lower required rank.

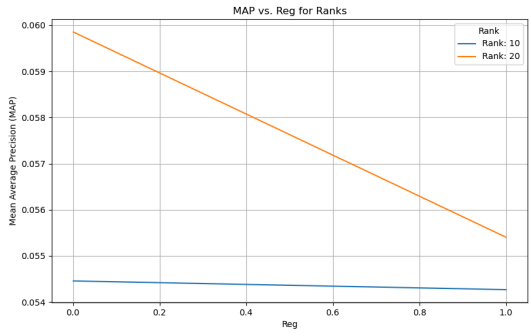



Figure 1

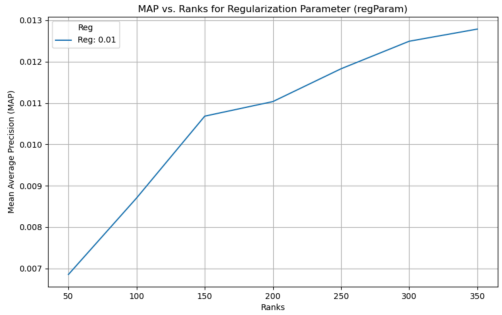



Figure 2

This indicates that the implicit model is both more effective and computationally efficient for our use case. To determine the optimal hyperparameter configuration, we followed a two-step tuning strategy: we fixed one hyperparameter (either rank or regularization) and tuned the other, starting from a wide range and narrowing down. First, we fixed the rank at 20 based on the results in Figure 2, and then tuned the regularization parameter. As shown in Figures 3 and 4, the optimal regularization value is 0.03.

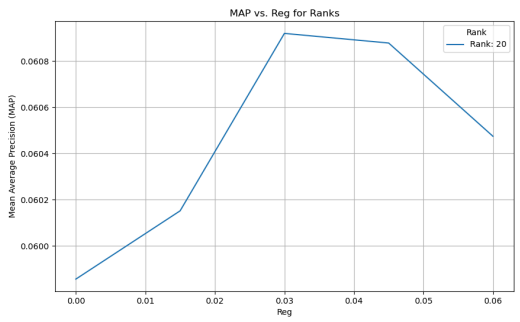



Figure 3

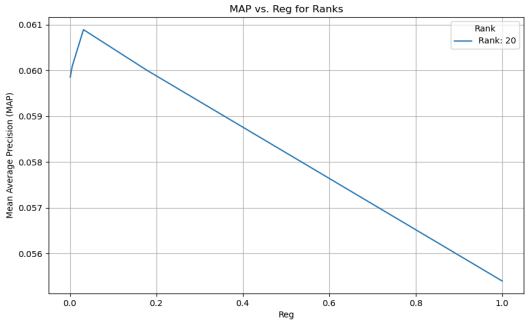



Figure 4

With regularization fixed at 0.03, we verified that the best rank remains 20, as confirmed in Figures 5 and 6.

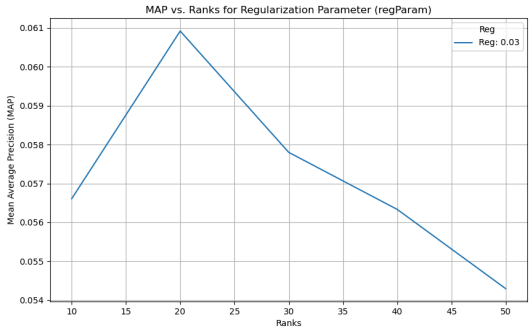



Figure 5

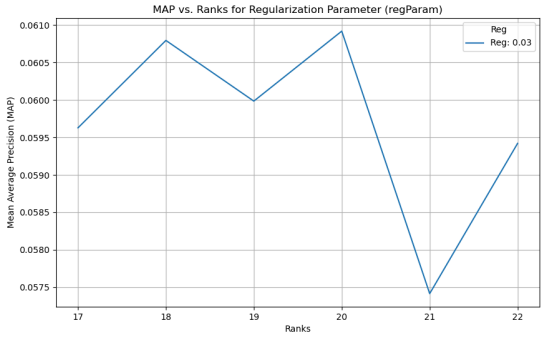



Figure 6

Based on the results from the smaller dataset, we defined a focused hyperparameter search space for the full dataset: **rank** ∈ [17, 18, 19, 20, 21, 22, 23] and **regularization** ∈ [0.01, 0.02, 0.03, 0.04, 0.05], centered around the previously identified optimal values. I initially fixed the rank at 20 and experimented with different regularization values. As shown in the figure 7, the MAP score continued to increase as the regularization parameter increased. I then explored a broader range using `np.arange(0.05, 0.55, 0.10)` and observed that MAP peaked at 0.25 before starting to decline. Ultimately, I found that the best regularization value was 0.2, as indicated in figure 9. With this value, the optimal rank remained 20, as indicated in figure 10.

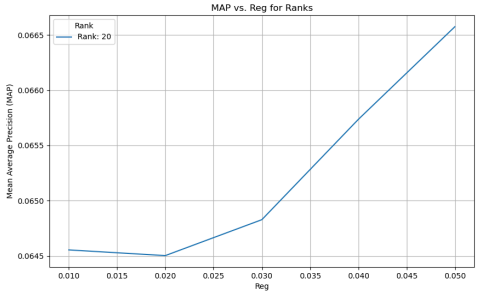



Figure 7

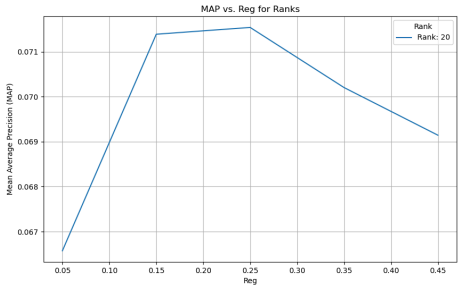



Figure 8

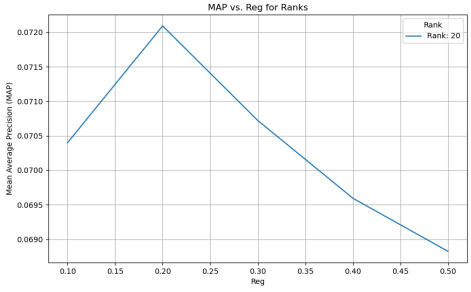



Figure 9

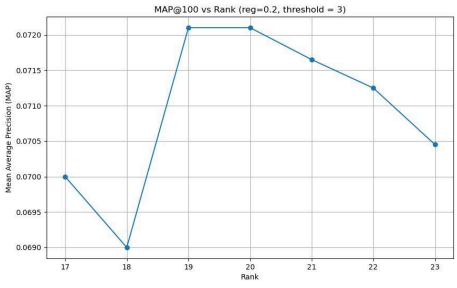



Figure 10

## 6. Evaluation of latent factor model

The latent factor model with rank 20 and a regularization parameter of 0.2 achieves a MAP of 0.167 on the training set, but only 0.0720 on the validation set and 0.0722 on the test set. The large gap between training and validation performance indicates overfitting and high variance, while the close alignment between validation and test suggests the model is at least consistent in its generalization performance. A MAP of around 7% means that, on average, 7% of the recommended movies are relevant and well-ranked for each user.