Leveraging MinHash LSH and ALS for User Segmentation and Personalized Movie Recommendations

Luke Sullivan, lcs9595

GitHub Repository

I. Dataset Overview

This dataset consists of available movie ratings data. Specifically, of 5-star rating and free-text tagging activity from MovieLens. It contains ratings and tag information for every movie from over 330,000 users. The data spans from January 1995 to July 2023. Users included in this dataset were selected randomly and are represented by anonymized IDs with no demographic information.

Key components:

- **Ratings**: 33,832,162 5-star ratings with half-star increments.
- **Tags**: 2,328,315 tag applications.
- **Movies**: Information on 86,537 movies, including titles, genres, and external links.
- **Users**: Data from 330,975 anonymized users.

Utilizing NYU Dataproc, the dataset was accessed, collected, and retrieved from the Hadoop Distributed File System (HDFS).

II. Customer Segmentation

The primary goal of this model was to identify groups of users with similar movie consumption patterns. The user's "movie watching style" was based on the set of movies each user has rated and the value of the score they provided. This approach focuses on and captures a user's viewing history, while understanding their preferences in a movie. The objective was to uncover pairs of users, termed "movie twins," who share the highest degree of overlap in their similar rated movie sets.

Model Setup

The process begins by loading the ratings dataset and performing initial data filtering to focus on active users and sufficiently rated movies. Users with fewer than the minimum of 20 ratings are removed, and movies rated by fewer than 15 users are also filtered out.

Critically, user movie sets are defined based on movies they have "liked." A liked movie is determined by a rating that meets or exceeds a threshold of 3.0 ratings (ranges from 0.5-5.0). Only these liked movies form the sets used for similarity comparison.

MinHash LSH for Candidate Pair Identification: For each user, a MinHash signature is generated based on their set of liked movie IDs. The MinHash algorithm creates a compact representation of each user's set, with 256 permutations to determine the signature size. These MinHash signatures are then indexed into a Locality Sensitive Hashing (LSH) structure.

LSH allows for querying of candidate similar pairs without having to compare every user with every other user. The LSH index groups users whose MinHash signatures are likely to be similar above the Jaccard similarity threshold, best set at 0.3. After LSH identifies candidate pairs, the exact Jaccard similarity is calculated for each pair based on their actual sets of liked movies. Pairs whose exact Jaccard similarity meet the threshold of 90% are retained.

Validation

The primary validation method for this customer segmentation (or movie twin identification) approach is to assess whether the users identified as similar by MinHash LSH (based on liked movies) also exhibit similar overall rating behaviors. This is achieved by calculating and comparing Pearson correlations of their movie ratings.

Correlation of MinHash-Identified Twin Pairs: For the top N movie twin pairs identified through the MinHash LSH process (those with high Jaccard similarity on their liked movie sets), their Pearson correlation coefficient is calculated. This correlation is computed using all movie ratings provided by these two users for movies they have both rated, not just the liked movies used for the MinHash step.. The average Pearson correlation across these top MinHash-identified pairs is then calculated.

Baseline Correlation (Random Pairs): To provide a benchmark, a set of 100 random user pairs is generated from the dataset using the same correlation methods. The selected random pairs were not already identified as twin pairs by the MinHash LSH process.

The effectiveness of the MinHash LSH approach in identifying users with similar tastes is determined by comparing the average Pearson correlation of the movie twin pairs against the average Pearson correlation of the random pairs. If the movie twin pairs provide a higher average correlation, it suggests that the Jaccard similarity on sets of liked movies, identified by MinHash LSH, is a valid understanding of similarity and tastes in ratings behavior.

Results

The MinHash LSH approach, followed by exact Jaccard similarity calculation, successfully identified 100 movie twin pairs based on their sets of liked movies (rated >= 3.0). Notably, many of the top identified pairs displayed a Jaccard similarity of 1.00, indicating identical sets of liked movies.

Correlation of Movie Twin Pairs: The average Pearson correlation for these top 100 MinHash-identified movie twin pairs was 0.3123.

- It's important to note that this average was calculated based on 99 pairs for which a valid Pearson correlation could be established (i.e., they met the *min_common_ratings* criteria and had sufficient variance in ratings).

Correlation of Random Baseline Pairs: The average Pearson correlation for these randomly selected pairs was 0.2214.

- This average was based on 12 random pairs that met the criteria for valid correlation calculation.

The configured MinHash LSH approach successfully identified 100 user pairs whose movie rating preferences were, on average, significantly more correlated than those of random pairs. The distribution of Jaccard scores, coupled with the high rate of valid correlations for MinHash pairs, indicates that strong Jaccard similarity in filtered viewing histories is a good indicator of shared taste.

III. Movie Recommendation

This model uses an Alternating Least Squares (ALS) algorithm, specifically configured for implicit feedback, to generate personalized movie recommendations. It combines user-movie interactions from both movie ratings and user-applied tags to improve the recommendation quality.

Data Partitioning

The data consists of both user movie ratings and movie tags (metadata about each movie) that undergo a temporal splitting process. Both ratings and tags datasets are first sorted by timestamp to ensure a chronological order.

Training and Test Split: The model aims for 75% training, 15% validation, and 15% test split. This is achieved by calculating approximate quantiles on the row Id for the 80th and 90th percentiles.

Optimizations: An approximate quantile method was used to determine the split points based on the row Id, which is suitable for large datasets where exact quantiles can be computationally expensive. A relative error of 0.01 is specified for balance.

Caching/Persistence: The initial sorted tags and ratings dataframes are explicitly not persisted before splitting, as they are used once by the split function and then discarded.

Model Setup

The project sets up and trains two distinct models for comparison: a Popularity Baseline and an ALS-based collaborative filtering model.

Data Preprocessing:

- Ratings and tags data are loaded from HDFS paths, with schemas applied.
- Rows with null values in key columns are dropped.

- Data is sorted by timestamp and assigned a row Id (*rowId*) for consistent temporal splitting.

Popularity Baseline Model: Recommends the same set of globally most popular movies to all users. Movie popularity is determined by counting the number of occurrences (interactions) for each movieId in the training ratings data.

The top 100 most popular movies from the training set are identified. This fixed list of *k* movies constitutes the recommendations for every user in the validation and test sets for whom predictions are needed.

Combined Implicit ALS Model: Using the PySpark implementation of Alternating Least Squares (ALS)**,** both ratings and tag applications from the training splits are treated as implicit user interactions.

Rating interactions are assigned a weight of 1.0 (*RATING_INTERACTION_VALUE = 1.0*).
Tag application interactions are assigned a weight of 0.5(*TAG_INTERACTION_VALUE = 0.5*).

- These weighted interactions are unioned, and then their values are summed up per user-movie pair to create a single combined interaction score. This forms the aggregated data which is cached before training.

Hyperparameters: The model is configured with key parameters that were determined through a tuning process. The most critical hyperparameters are:

- **rank**: Set to 15, this defines the number of latent features used to represent users and movies.
- **regParam**: Set to 0.1, this is the regularization parameter (λ) applied to prevent overfitting by penalizing large latent factor values.

These specific values were identified as the best parameters through a prior grid search process, where the model was evaluated across a range of potential hyperparameter combinations on a validation set to find the configuration yielding the optimal performance metrics. Notably, users who were not present in the training data were excluded to prevent cold start problems (coldStartStrategy="drop").

Validation

Both the Popularity Baseline and the ALS models are evaluated using ranking-based metrics against the held-out validation and test sets. The ratings data provides the ground truth for these evaluations.

- Ground truth for evaluation consists of movies rated highly (>= 4.0) by users in both the validation and test sets. Recommendations from both models are compared against this ground truth.
- Precision, recall, NCDG and mean average precision (all at k=100) were calculated allowing for a direct comparison of performance.

Precision	0.1281
Recall	0.8346
NDCG	0.5907
Mean Average Precision	0.3265

Precision	0.1037
Recall	0.7488
NDCG	0.5138
Mean Average Precision	0.2887

Figure [1]: Performance metrics on both models validation sets (1,003 users with positive ratings)

Utilizing the grid search method for tuning the model, the metrics revealed by the validation set were directly used to interpret the most efficient values for hyperparameters. The test set was prioritized for the experiment results.

Results

The evaluation of both the models yielded distinct performance characteristics when comparing their effectiveness on the test dataset.

Popularity Baseline Model:
Ground truth (users with items rated >= 4.0) was successfully generated for 322,090 users. Recommendations were generated for 328,666 users, and the evaluation was performed on the 322,090 users common to both recommendations and ground truth.

The performance metrics for the Popularity Baseline at K=100 were as follows:

Precision	0.0421
Recall	0.1270
NDCG	0.1013
Mean Average Precision	0.0303

Figure [2]: Performance metrics on Popularity Baseline test set

ALS model:
The ground truth data remained consistent, with 322,090 users having positively rated items. A notable difference is in the number of users with recommendations generated; the ALS model provided recommendations for 23,355 users. This reduction is attributable to the coldStartStrategy="drop" setting, which omits users or items from the test set that were not encountered during the model training phase. The evaluation was ultimately performed on 23,196 users present in both the recommendations and ground truth.

The performance metrics for the ALS model at K=100 were:

Precision	0.1586
Recall	0.2437
NDCG	0.2752
Mean Average Precision	0.0837

Figure [3]: Performance metrics on ALS model test set

Comparing the test set performance, the combined implicit ALS model demonstrated notably stronger results across all ranking metrics when compared to the Popularity Baseline model. The ALS model achieved a precision of **0.1586** versus the Popularity model's **0.0421**, and an NDCG of **0.2752** compared to **0.1013**. Similar improvements were observed for recall and mean average precision. This indicates that the approach of the ALS model, leveraging both ratings and tags, was significantly more effective at recommending relevant items to users it could provide predictions for, despite covering a smaller subset of test users due to the cold start strategy. The performance lift suggests that the complexity and personalization of the ALS model provides a reasonable benefit over simple popularity-based recommendations for users known to the model.

IV. Appendix

Model Python Code Files

1. Customer Segmentation Code File
2. Movie Recommendation Code FIle

Top 100 Jaccard Similarities Results

```
--- Top 100 'Movie Twin' Pairs (UserID1, UserID2, Jaccard Similarity) ---
1. User IDs: (18356, 313668), Jaccard Similarity: 1.0000
2. User IDs: (169845, 308490), Jaccard Similarity: 1.0000
3. User IDs: (97685, 227884), Jaccard Similarity: 1.0000..
… (proceeds with same 1.0 Jaccard Similarity for each pair)
..68. User IDs: (65538, 77318), Jaccard Similarity: 1.0000
69. User IDs: (77647, 294432), Jaccard Similarity: 0.9984
70. User IDs: (241273, 296245), Jaccard Similarity: 0.9910
71. User IDs: (25084, 86967), Jaccard Similarity: 0.9866
72. User IDs: (86967, 294432), Jaccard Similarity: 0.9835
73. User IDs: (77647, 86967), Jaccard Similarity: 0.9820
74. User IDs: (25084, 294432), Jaccard Similarity: 0.9819
75. User IDs: (167541, 173768), Jaccard Similarity: 0.9818
76. User IDs: (25084, 77647), Jaccard Similarity: 0.9804
77. User IDs: (50012, 174815), Jaccard Similarity: 0.9770
78. User IDs: (41012, 300048), Jaccard Similarity: 0.9761
79. User IDs: (159785, 319480), Jaccard Similarity: 0.9750
80. User IDs: (48673, 107031), Jaccard Similarity: 0.9750
81. User IDs: (53426, 330736), Jaccard Similarity: 0.9714
82. User IDs: (242889, 271504), Jaccard Similarity: 0.9709
83. User IDs: (115400, 144661), Jaccard Similarity: 0.9677
84. User IDs: (195935, 319748), Jaccard Similarity: 0.9667
85. User IDs: (71302, 147789), Jaccard Similarity: 0.9667
86. User IDs: (71302, 279965), Jaccard Similarity: 0.9667
87. User IDs: (187276, 319748), Jaccard Similarity: 0.9667
88. User IDs: (293880, 319748), Jaccard Similarity: 0.9667
89. User IDs: (13012, 56180), Jaccard Similarity: 0.9667
90. User IDs: (13012, 264494), Jaccard Similarity: 0.9667
91. User IDs: (71302, 138579), Jaccard Similarity: 0.9655
92. User IDs: (32448, 259106), Jaccard Similarity: 0.9643
93. User IDs: (245387, 249077), Jaccard Similarity: 0.9600
94. User IDs: (178961, 200620), Jaccard Similarity: 0.9600
95. User IDs: (132494, 266741), Jaccard Similarity: 0.9600
96. User IDs: (177837, 245387), Jaccard Similarity: 0.9600
97. User IDs: (13195, 124266), Jaccard Similarity: 0.9600
98. User IDs: (13195, 224936), Jaccard Similarity: 0.9600
99. User IDs: (125637, 204124), Jaccard Similarity: 0.9600
100. User IDs: (13195, 291625), Jaccard Similarity: 0.9600
```

- Full listed results available on GitHub.