Movie Recommendation System

DSGA 1004: Big Data
Aneesh Shetye[1], Anubha Singh[2], Anchal Agrawal[3]
New York University
[1]axs10302@nyu.edu [2]as18806@nyu.edu [3]aa11597@nyu.edu

Introduction

Recommendation systems are essential for helping users navigate through a vast array of content, they personalize suggestions based on individual preferences, enhancing user engagement and satisfaction. This capstone project focuses on recommendation systems on the Movielens dataset to build two main components: a collaborative-filter-based recommender and a customer segmentation mechanism.

In its first part, the project identifies users with similar movie-watching habits, emphasizing patterns over ratings. We compute these by using rated movies as a proxy for individual movie ratings and further show that the correlation in rating scores are indeed more for users who have rated similar movie titles (in our case movie-ids). This can be thought of as a consequence of movie ratings being an external type of feedback- the fact that a user has rated a movie is a testament to the fact that it has elicited feelings strong enough to drive the user to do so.

The second part of our project aims to recommend movies to users. As a baseline, we test the popularity-based recommendation model which recommends the highest rated movies (across all users) to every user. Further, we use the Alternating Least Squares (ALS) Model from Spark's ml library to get latent representations of user and movies and .

Dataset

The provided MovieLens dataset includes two versions: the small dataset consists of approximately 9,000 movies and 600 users (links.csv, movies.csv, ratings.csv, tags.csv). The full dataset consists of 86,000 movies and 330,000 users (links-large.csv, movies-large.csv, ratings-large.csv, tags-large.csv, and genome-scores-large.csv).

Part 1: Customer Segmentation

Top 100 pairs

Here we have implemented the `get_top_100` function, designed to identify the top 100 pairs of similar users based on their movie preferences. We start by utilizing Locality-Sensitive Hashing (LSH) and user MinHashes to efficiently approximate nearest neighbors. Upon initialization, we create a dictionary to store the top user pairs and their similarity scores, initializing all scores to 0. Our function then iterates through each user, finding their similar users using LSH and computing Jaccard similarity scores between pairs. For each user pair, if the similarity score exceeds the lowest score among the top 100 pairs, it updates the dictionary accordingly, ensuring it maintains the most similar user pairs. Ultimately, the function returns the updated dictionary containing the top 100 similar user pairs based on their movie preferences.

Calculating Correlation

Here we have the `calc_corr` function, which computes the correlation between the ratings of shared movies among pairs of users. It operates by iterating through each user pair provided as input, extracting their respective movie lists and corresponding ratings. For each pair, it identifies the common movies and constructs dictionaries mapping movies to their ratings for both users. If there are no common movies or only one common movie, the correlation is set to 0. Otherwise, it calculates the correlation coefficient between the ratings of common movies. The resulting correlation coefficients are collected and returned, offering insights into the similarity of preferences between user pairs based on their shared movie ratings.

Results

When we perform the operations above, and calculate the following:

```
corr_random = calc_corr(random_100,
    user_movies, user_ratings)
```

Here, we find that the correlation between random100 (paired users), movies that the user has rated and the ratings that the user has given is as follows:

Correlation	Dataset
0.67	Small
0.825	Big

Table 1: Correlations

Part 2: Movie Recommendation

Pre-processing

1. **Filtering Users with Few Ratings:** Initially, we excluded users from our dataset if they had fewer than 3 ratings to ensure that we have sufficient data per user.
2. **Removing Duplicate Ratings:** After filtering out users, we then eliminated any duplicate ratings given by the same user for the same item to ensure that each user-item interaction is unique and avoids skewing the analysis with redundant data.
3. **Splitting into Train, Test, and Validation Sets:** The next step involved dividing our pre-processed dataset into three separate sets: training, testing, and validation. This split was based on the userId such that there is a balanced representation of users across all sets.
   - **Weighting and Distribution:** We allocated 60% of the data to the training set, 20% to the testing set, and 20% to the validation set.
   - **User Representation:** We ensured that every user was present in each of the three sets at least once. This strategy is designed to mitigate the "cold start" problem, where new users or items have insufficient data for effective modeling. By including each user in every set, we enable the model to learn from and generalize across a diverse range of user behaviors.

Model

Baseline Model

A popularity-based recommendation model operates by suggesting items to users based on their overall popularity or frequency of selection by all users within a dataset. To establish this model, we start by calculating the popularity of items using metrics like the number of views, purchases, or ratings they have garnered. This calculation assigns each item a score or ranking that reflects its popularity among users. When generating recommendations, the model simply presents users with the most popular items based on these calculated scores. This method doesn't take into consideration individual user preferences or behaviors; instead, it offers the same set of popular items to all users uniformly.

To evaluate the effectiveness of this popularity-based model, we compare the items recommended by the model with the items that users have actually rated or interacted with. This comparison allows us to assess how well the recommended popular items align with user preferences. The evaluation process focuses on calculating accuracy by measuring the percentage of correctly recommended items out of all user interactions. For instance, in the context of movie recommendations, we might evaluate the model's accuracy by comparing the top 100 recommended movies to the movies that users have rated. The resulting accuracy score provides a quantitative measure of the model's performance in terms of recommending popular items that resonate well with user preferences across the dataset.

Metric	*mAP*
Train	0.0694
Validation	0.0275
Test	0.0285

Table 2: mAP Results for Baseline Model

Latent Factor Model

Collaborative filtering is a popular technique in recommender systems that addresses missing entries in a user-item association matrix. Spark.ml offers model-based collaborative filtering, representing users and products with latent factors to predict missing entries. This is achieved through the alternating least squares (ALS) algorithm, which learns these latent factors efficiently.

$\hat{r}_{ui} \approx f(User = u, Item = i)$

Here we tuned rank (number of latent factors) and regularization factors (penalty coefficient) in order to find the best parameter combination. Specifically, we tried rank = [5, 20, 50] and reg = [0.01, 0.1, 1]. The results we obtained were as follows:

Rank	RegParam	Validation RMSE	Validation mAP
5	0.01	1.050	0.0447
5	0.1	0.911	0.0464
5	1	1.355	0.0377
20	0.01	1.079	0.0427
20	0.1	0.906	0.0459
20	1	1.356	0.0377
50	0.01	1.106	0.0414
50	0.1	0.909	0.0458
50	1	1.355	0.0377

Table 3: RMSE and mAP for Different Ranks and RegParams

After reviewing the data, we found that the best RMSE result occurred with a regularization parameter of 0.1 and a rank of 20, giving us a value of 0.906. Similarly, the highest Validation mAP score of 0.0464 was obtained with a rank of 5 and a regularization parameter of 0.1. Across different ranks, we consistently observed better results when using a regularization parameter of 0.1.

These findings suggest that lower ranks and moderate regularization parameters tend to produce better RMSE values. Similarly, for mAP, lower ranks and average regularization parameters appear to yield the best results.

Comparing the models

The popularity-based baseline model relies solely on item popularity metrics, such as views, purchases, or ratings, to generate recommendations. It offers uniform recommendations to all users, disregarding individual preferences. In contrast, the latent factor collaborative filtering model leverages sophisticated techniques to capture user-item interactions. By representing users and items with latent factors and optimizing them using the ALS algorithm, this model

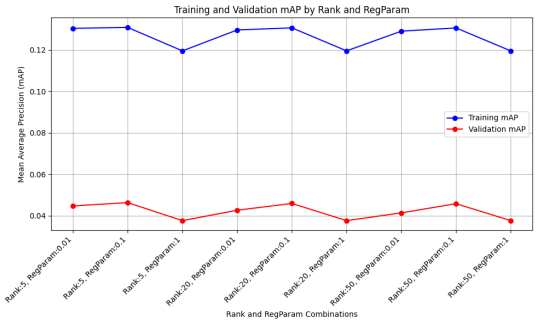

Figure 1: Training Vs Validation mAP for ALS Model

learns from observed interactions to predict missing entries in user-item matrices.

In terms of effectiveness, the baseline model's recommendations are straightforward but lack personalization, potentially leading to less relevant suggestions for individual users. On the other hand, the collaborative filtering model aims to provide more tailored recommendations by capturing underlying patterns in user preferences and item characteristics.

After evaluating the results, it is seen that the ALS model outperforms the simple baseline model. With a mAP of 0.046, the ALS model shows better performance compared to the baseline model's mAP of 0.027.

Conclusion

In Part 1 of this project, we aimed to segment customers based on their movie-watching styles using MinHash. To validate our results, we compared the average correlation of numerical ratings within the movie twins against that of 100 randomly selected pairs from the dataset. Our results revealed a correlation of 0.825 on the larger dataset.

In Part 2 of the project, we investigated two types of recommendation models: a popularity based model and a collaborative filtering based recommendation model using ALS.

File Description

- **Part 1:**
  - Outputs for Small: `customer-segmentation.ipynb`
  - Outputs for All: `customer-segmentation(all).ipynb`
- **Part 2:**
  - Data Filtering and Splitting: `split_prep.py`
  - Baseline Model: `popularity_baseline.py`, `utils.py`, `test_similarity.py`
  - ALS Model: `als_new.py`

References

Spark. 2022. Collaborative Filtering.