Movie Recommender System Using Matrix Factorization

Eliud Koto [1]

Abstract

With the massive growth in both the demand for and supply of digital content, consumers often struggle to find material that truly matches their interests. Recommender engines help address this challenge by analyzing explicit and implicit user behavior, enabling digital platforms to assess user preferences and suggest relevant content more effectively. This paper discusses the matrix factorization technique used to develop a movie recommender system. It explores the mathematical derivation, implementation and optimization of ALS algorithm. Github Repo

1. Introduction

Over the years, content creation has grown exponentially with the help of advanced technologies and tools. In the digital space, there is now an almost infinite amount of content, increasing every second. With the rapid adoption of Artificial Intelligence (AI) in creative fields, we can expect even more content, far more than we could ever consume. With so many options available, accessing relevant content becomes a challenge for consumers.

Another major issue is that among all available products, a few popular items dominate and overshadow less popular or newer ones(Koenigstein et al., 2012). In other words, digital content often follows a power-law distribution. This creates a unique challenge for both users and producers. Users face an overwhelming number of choices and may struggle to discover what they would actually enjoy, hurting their overall experience. Meanwhile, less popular or recent products may never become visible to potential consumers due to this distribution.

The solution to these challenges is the use of recommender engines. Recomender engines help match consumers with relevant content. Each user has unique tastes that may change over time, and such factors must be considered when building a such systems. This ensures a better user experience by providing options aligned with individual interests.

Recommender engines typically achieve this through two methods: content filtering and collaborative filtering (Koren et al., 2009). Content filtering builds profiles for products or users to characterize their attributes, allowing the system to associate users with appropriate content. Its main limitation is the need for explicit data, which sometimes can be difficult to obtain or may not be available. The second method, collaborative filtering, analyzes interactions between users and different items to uncover new user–product relationships. Due to its ability to efficiently handle sparse data while delivering accurate predictions, matrix factorization has become a dominant strategy in collaborative filtering. This approach gained popularity during the Netflix Prize competition, where it significantly outperformed traditional techniques such as nearest-neighbor models(Koren et al., 2009). This paper will focus on this method, specifically examining the use of matrix factorization in developing a movie recommender system.

2. Problem Statement

The goal of this project is to build an optimized movie recommender system using the matrix factorization technique. Traditional collaborative filtering methods estimate user preferences based on ratings of similar items or similar users. This requires calculating similarities between all pairs of users or all pairs of movies, which becomes increasingly computationally expensive as data grows. In addition, real-world user–item matrices are usually sparse, meaning such methods may struggle to produce accurate predictions.

Matrix factorization provides a more robust alternative. By leveraging sparse rating matrices, this approach characterizes both users and items through multiple latent factors inferred from observed rating patterns. As a result, it can deduce preferences even when users share no overlapping rated items, addressing a major limitation of traditional collaborative filtering.

[1]African Institute for Mathematical Sciences (AIMS) South Africa, 6 Melrose Road, Muizenberg 7975, Cape Town, South Africa. Correspondence to: Eliud Koto <eliud@aims.ac.za>.

3. Exploratory Data Analysis (EDA)

3.1. Dataset Summary

The paper uses two sets of MovieLens dataset, 100k Movielens dataset to build the first model with biases only and the 32M MovieLens dataset for the latent factor model. This dataset contain 5-star rating and various metadata about the movies.

DESCRIPTION	TOTAL
RATINGS	32,000,204
USERS	200,948
MOVIES	87,585
HIGHEST RATINGS PER USER	33,332
HIGHEST RATINGS PER MOVIE	102,929
LOWEST RATINGS PER USER	20
LOWEST RATINGS PER MOVIE	1
START DATE	09/01/1995
END DATE	12/10/2023

Table 1. MovieLens 32M Data Summary

DESCRIPTION	TOTAL
RATINGS	100,836
USERS	610
MOVIES	9742
HIGHEST RATINGS PER USER	33,332
HIGHEST RATINGS PER MOVIE	102,929

Table 2. MovieLens 100K Data Summary

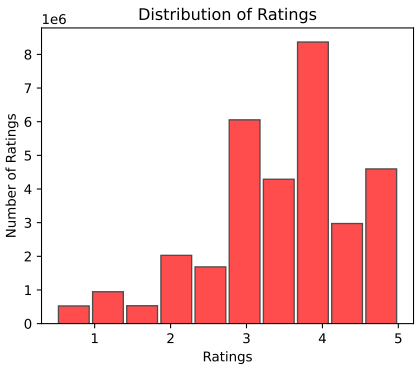

Figure 1. 32M MovieLens Ratings Distribution

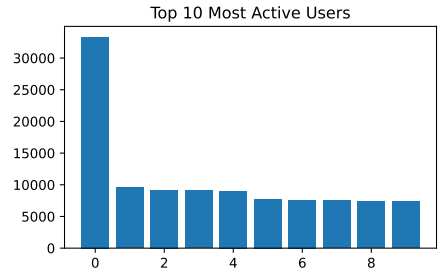

Figure 2. 32M MovieLens Top 10 Users Distribution

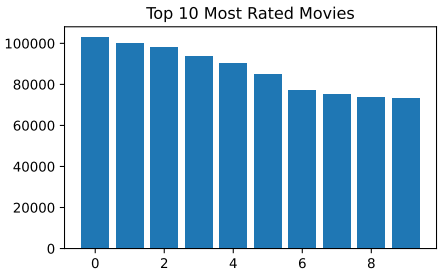

Figure 3. 32M MovieLens Top 10 Movies Distribution

3.2. Power Law Distribution

As discussed earlier, items and users typically follow a power-law distribution, where a small number of movies receive a very large number of ratings and a small number of users rate many movies. In this dataset, the movie with the highest number of ratings has 102,929 ratings, and one user has rated 33,332 movies. Overall, Figure 4 illustrates that a small subset of users and movies accounts for a disproportionate share of total interactions. This pattern aligns with Zipf's Law, which states that a few users or items contribute the majority of interactions in a system. .

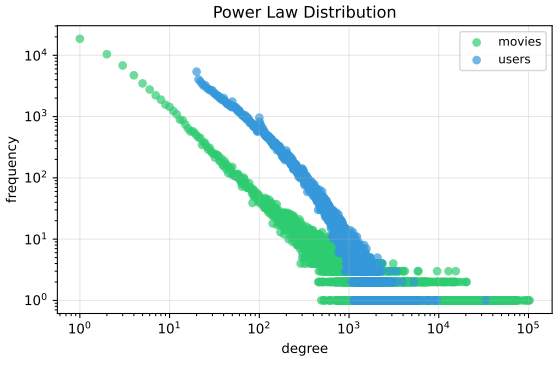

Figure 4. 32M MovieLens Power Law Distribution

3.3. Data Indexing

To optimize the matrix factorization model, the first step involved restructuring the MovieLens data. Users and movies were mapped to numerical indices to allow the model to efficiently access and iterate over them. The updated data structure included a dictionary mapping each user to an index, a list storing user IDs so the original ID could be retrieved from the index, and a sparce matrix, basically list of lists with tuples that recorded which movie index each user rated and the corresponding rating. A similar structure was created for movies.

3.4. Train-Test Split

The data was split into training and test sets using an 80–20 ratio. This split allows the model to learn from the training data and identify optimal parameters, while the test set is used to evaluate how well the model can predict a user's rating for unseen items. This evaluation helps determine whether the recommender system can generalize effectively and provide relevant suggestions for content a user has not previously interacted with.

A key consideration in this process was ensuring that the sparse matrix retained the same shape, with only the ratings split between the training and test sets. Additionally, because the dataset follows a power-law distribution, the split needed to be done in a way that avoided bias toward heavy users. To preserve this distribution and maintain realistic user behavior, the leave-last-out approach was used for splitting the data. Figure 5. shows the result of the split on the 100K MovieLens dataset.

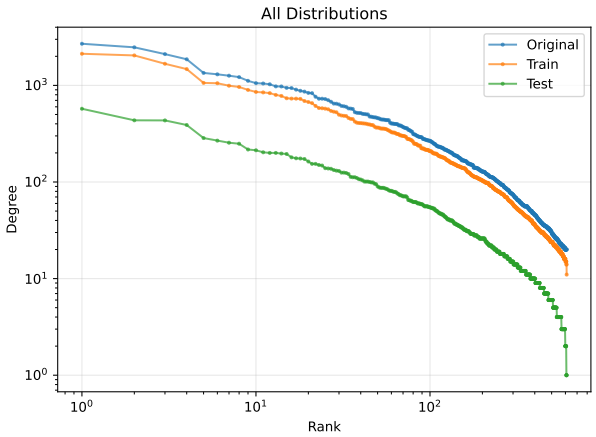

Figure 5. 100K MovieLens test-train split Power Law Distribution

4. Matrix Factorization using Alternating Least Squares (ALS)

4.1. Building Loss Function

Matrix factorization uses observed rating patterns to infer user and item vectors(Koren et al., 2009). These vectors capture latent features that help predict missing ratings in the matrix. For example, a user vector $\mathbf{u}_m$ may encode the interests of user m across several latent dimensions, such as preference for animation or drama movies, while an item vector $\mathbf{v}_n$ represents the characteristics of item n for the same dimensions. The dot product $\mathbf{u}_m^\top \mathbf{v}_n$ reflects the interaction between user m and item n, indicating the user's preference for that item. The estimate of the rating r is given by

$$\hat{r}_{mn} = \mathbf{u}_m^\top \mathbf{v}_n \tag{1}$$

The model should minimize the error, defined as the difference between the predicted rating $\hat{r}_{mn}$ and the actual rating r_{mn}. To ensure the model generalizes well, regularization is typically applied to prevent overfitting. Importantly, much of the variation in observed ratings often comes from user bias b_m or item bias b_n that are unrelated to user–item interactions. Therefore, these biases should be incorporated into the loss function to capture their contribution to the final rating. The resulting loss function is given by:

$$\begin{aligned} L = \sum_{(mn)\in\kappa} \Big(r_{mn} - \mu - b_m - b_n - U_m^\top V_n\Big)^2 \\ + \lambda\Big(\|U_m\|^2 + \|V_n\|^2 + b_m^2 + b_n^2\Big) \end{aligned} \tag{2}$$

The paper will adopt a Bayesian framework to model the observed ratings. The assumption is that each rating is generated from an underlying latent factor model with Gaussian noise. The likelihood function is given by:

$$p(r_{mn} \mid \mathbf{u}_m, \mathbf{v}_n, b_m^{(u)}, b_n^{(i)}) = \mathcal{N}\big(r_{mn}\,; \hat{r}_{mn}; \lambda^{-1}\big) \tag{3}$$

Converting this to a log-likelihood function, we have:

$$\begin{aligned} \log p(U, V, b^{(u)}, b^{(i)} \mid R) = -\frac{\lambda}{2} \sum_{(m,n)\in\kappa} \big(r_{mn} - \hat{r}_{mn}\big)^2 \\ - \frac{\lambda_u}{2}\sum_m \|\mathbf{u}_m\|^2 - \frac{\lambda_v}{2}\sum_n \|\mathbf{v}_n\|^2 \\ - \frac{\lambda_b}{2}\left(\sum_m (b_m^{(u)})^2 + \sum_n (b_n^{(i)})^2\right) + \text{c} \end{aligned} \tag{4}$$

where:

$$\hat{r}_{mn} = \mathbf{u}_m^\top \mathbf{v}_n + b_m^{(u)} + b_n^{(i)} \tag{5}$$

4.2. Minimizing Loss Function

The model will use alternating least squares (ALS) algorithm to minimize the loss function. This algorithm works by fixing the user vectors $\mathbf{u}_m$ and updating the item vectors $\mathbf{v}_n$ through a series of least-squares problems, and then fixing $\mathbf{v}_n$ while updating $\mathbf{u}_m$. These steps are repeated iteratively until convergence. ALS is computationally efficient and effective, as it supports parallelization.

Algorithm 1 Alternating Least Squares (ALS)

Initialize user latent factors U and item latent factors V.
repeat
  for each user m **do**
    Fix U, update user bias, b,user trait V.
  end for
  for each item n **do**
    Fix V, update item bias, b,item trait V.
  end for
until maximum number of iterations is reached

5. Model

5.1. Model with Bias Only

While attempting to infer latent vectors from the interaction between users and items, it is crucial to consider the role that bias plays in the observed ratings. These ratings are influenced not only by user–item interactions but also by user bias and item bias. This first model uses the 100K dataset to estimate user bias b_m and item bias b_n using the ALS algorithm. The user bias update rule is given by:

$$b_m = \frac{\lambda \sum_{n \in \Omega(m)} \left(r_{mn} - (\mathbf{u}_m^\top \mathbf{v}_n + b_n^{(i)}) \right)}{\lambda |\Omega(m)| + \gamma} \tag{6}$$

The item bias update rule is given by:

$$b_n = \frac{\lambda \sum_{m \in \Omega(n)} \left(r_{mn} - (\mathbf{u}_m^\top \mathbf{v}_n + b_m^{(u)}) \right)}{\lambda |\Omega(n)| + \gamma} \tag{7}$$

where λ is the regularization factor and γ is the scaling factor on the count of occurrence.

To evaluate this model, this RMSE (root mean squared error) function was used.

$$\text{RMSE} = \sqrt{\frac{1}{|\Omega|} \sum_{(m,n) \in \Omega} (r_{mn} - (b_m + b_n))^2} \tag{8}$$

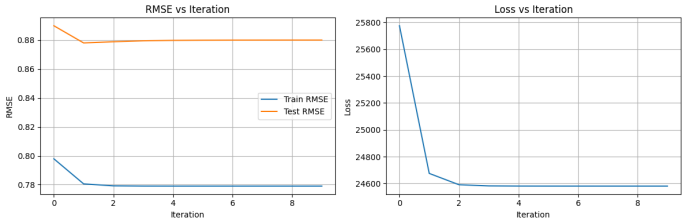

Figure 6. Bias Only RMSE

5.2. Model with Latent Factors and Biases

Similarly, the second model to optimize latent factors and biases used the ALS algorithm. Latent Factors Updates are computed with the following functions:

User Factors ($\mathbf{u}_m$)

$$\mathbf{u}_m \leftarrow \left(\lambda \sum_{n \in \Omega_m} \mathbf{v}_n \mathbf{v}_n^T + \tau I \right)^{-1} \left(\lambda \sum_{n \in \Omega_m} \mathbf{v}_n (r_{mn} - b_m - b_n) \right) \tag{9}$$

Item Factors ($\mathbf{v}_n$)

$$\mathbf{v}_n \leftarrow \left(\lambda \sum_{m \in \Omega_n} \mathbf{u}_m \mathbf{u}_m^T + \tau I \right)^{-1} \left(\lambda \sum_{m \in \Omega_n} \mathbf{u}_m (r_{mn} - b_m - b_n) \right) \tag{10}$$

To evaluate this model, this RMSE (root mean squared error) function was used.

$$\text{RMSE} = \sqrt{\frac{1}{|\kappa|} \sum_{(m,n) \in \kappa} \left(r_{mn} - \left(b_m^{(u)} + b_n^{(i)} + U_m^\top V_n \right) \right)^2} \tag{11}$$

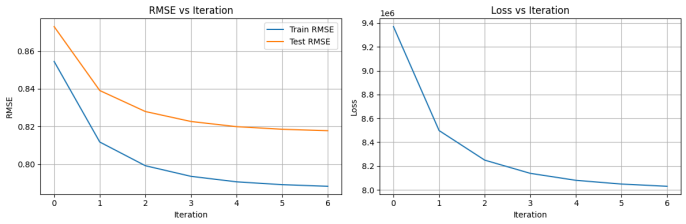

Figure 7. 32M MovieLens RMSE

5.3. Results

In this section, we will discuss the result of my models and explore it's performance.

References

Koenigstein, N., Nice, N., Paquet, U., and Schleyen, N. The xbox recommender system. In *Proceedings of the sixth ACM conference on Recommender systems*, pp. 281–284, 2012.

Koren, Y., Bell, R., and Volinsky, C. Matrix factorization techniques for recommender systems. *Computer*, 42(8): 30–37, 2009.

A. You *can* have an appendix here.

You can have as much text here as you want. The main body must be at most 8 pages long. For the final version, one more page can be added. If you want, you can use an appendix like this one, even using the one-column format.