Enhancing Movie Recommendations with Biases and Latent Factor Models: A Scalable Approach

Omer Ebead [1]

Abstract

This document presents a report on a recommender system that utilizes the Alternating Least Squares (ALS) algorithm for movie recommendations. The system was developed and evaluated using the MovieLens dataset, which contains 32 million ratings. The model achieved a Root Mean Square Error (RMSE) of 0.81.
Github Repo: Recommender System AIMS

1. Introduction

Recommender systems are vital for personalizing user experiences by analyzing behavior and delivering tailored suggestions, especially in the entertainment industry. They must balance popular and niche content to overcome the power law distribution in user interactions, ensuring diverse and relevant recommendations.

`http://icml.cc/`

2. Background

Recommender systems personalize user experiences and boost engagement, with companies like Amazon (5) and Netflix (8) leading in advanced, efficient algorithms.

2.1. Amazon's Recommender System (5)

Amazon first used user-based collaborative filtering but faced scalability issues. To address this, they developed item-to-item collaborative filtering, focusing on product relationships to improve accuracy and efficiency. This method, enhanced over time with user preferences, enables faster and more relevant recommendations.

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

2.2. The Netflix Prize (8)

In 2006, Netflix launched the Netflix Prize, offering 1,000,000 for a 10% improvement in its recommendation algorithm. The competition, using data from 480,189 users and 17,770 movies, spurred major innovations in recommender systems. In 2009, BellKor's Pragmatic Chaos (4) won with a 10.06% improvement using an ensemble of models, showcasing the power of collaborative filtering and inspiring more advanced personalization methods.

2.3. The Xbox Recommender System (6)

Microsoft's Xbox Live Marketplace uses a recommender system to personalize movie and game suggestions for millions of users, leveraging implicit feedback from Xbox consoles. The system combines offline and online modules, using bilinear models to represent users and items as vectors and predict preferences via their inner product. Bayesian inference refines these predictions to prevent overfitting. By maximizing a utility function, the system delivers relevant recommendations, performing better for games than movies due to movie data sparsity.

3. Methodology

3.1. Research Design and Method Selection

To select the best recommender system method, we assess how each approach meets the goal of accurately capturing user preferences while ensuring computational efficiency and scalability.

3.1.1. Research Objective

The main goal of this research is to develop a recommendation system that accurately reflects user preferences using historical user-item interaction data. The system must deliver personalized, scalable recommendations for large datasets, balancing accuracy and computational efficiency for real-world applications like movies, music, and products.

3.1.2. Justification for Method Choice

Different recommendation methods offer unique advantages, but certain approaches better align with the research goals.

Below is an evaluation of various methods and the rationale behind the chosen approach.

3.1.3. Feature-Based (Content-Based) Filtering

Pros: Leverages item or user attributes (e.g., genres, actors, keywords) to recommend similar items, making it effective for users with clear preferences.
Cons: Ignores user-to-user interactions, limiting recommendation diversity, and struggles with new users or items due to its dependence on predefined features.

3.1.4. Nearest Neighbor-Based Filtering

Pros: Identifies users or items with similar interaction patterns, enabling personalized recommendations. It is also straightforward to implement.
Cons: Struggles with scalability due to high computational costs on large datasets and lacks adaptability to changing user preferences.

3.1.5. Collaborative Filtering

Pros: Effectively captures latent user-item interactions, especially through matrix factorization, balancing accuracy and scalability. It also enhances recommendation diversity by leveraging extensive user interaction data.
Cons: Faces the *cold-start problem*, making it challenging to recommend items to new users or for new items with limited data. Regular model retraining is also required as the dataset grows.

3.1.6. Chosen Method: Collaborative Filtering

Collaborative filtering generates recommendations based on past user behavior, such as ratings and transactions, without needing explicit user profiles. Originally introduced by the creators of **Tapestry**, this domain-independent method effectively captures user-item interactions, often delivering more accurate results. However, it faces challenges like the **cold start problem**, struggling to recommend items to new users or for newly added products with limited data.

Collaborative filtering is mainly divided into two techniques:

- **Neighborhood Methods**: Identify relationships between users or items. For example, users who liked *"Saving Private Ryan"* might also enjoy other war films or Spielberg's movies due to similar rating patterns.

- **Latent Factor Models**: Map users and items into a shared latent feature space, typically with 20–100 factors, capturing both explicit (e.g., comedy vs. drama) and abstract user preferences, such as *serious* vs. *escapist* content.

3.2. Matrix factorization techniques for Latent factor models (1)

These methods represent both items and users as vectors of factors derived from item rating patterns. Each item i is associated with a factor vector $q_i \in \mathbb{R}$, and each user u has a factor vector $p_u \in \mathbb{R}$. The dot product of these vectors $q_i^T.p_u$ estimates the user's rating for the item, denoted $\hat{r} = q_i^T.p_u$.

The primary challenge is accurately mapping users and items to their respective factor vectors, particularly given the sparsity of the user-item rating matrix, which often has many missing values. Traditional techniques like Singular Value Decomposition (SVD) struggle with incomplete matrices. Early systems attempted to fill in these gaps through imputation (7) , but this approach can be costly and lead to inaccurate data.

Recent advancements focus on directly modeling observed ratings without imputing missing values, using a regularized approach to minimize the squared error on known ratings. The optimization function includes a regularization term to prevent overfitting, controlled by a parameter λ .

Matrix factorization techniques have gained popularity due to their scalability and accuracy, allowing for flexible modeling of diverse real-world scenarios. They utilize different types of input data, primarily high-quality explicit feedback (like star ratings), which often leads to a sparse matrix. Additionally, when explicit feedback is lacking, implicit feedback—derived from user behavior (e.g., browsing history, purchase records)—can be used to infer preferences, typically resulting in a denser matrix representation.

Two primary methods for minimizing the optimization equation are stochastic gradient descent (SGD) and alternating least squares (ALS).

3.2.1. Stochastic Gradient Descent (SGD)

Popularized by Simon Funk (2), SGD iteratively processes all ratings in the training set. For each rating, it predicts r_{ui} and computes the prediction error:

$$e_{ui} = r_{ui} - q_i^T p_u$$

The parameters are then updated in the opposite direction of the gradient:

$$(q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot p_u - \lambda q_i)$$

$$(p_u \leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda p_u)$$

This method is easy to implement and relatively fast but may not always be optimal.

3.2.2. Alternating Least Squares (ALS)

Since both q_i and p_u are unknown, the problem is non-convex. ALS addresses this by fixing one set of parameters while optimizing the other. It alternates between:

1. Fixing p_u to compute q_i using least-squares optimization.
2. Fixing q_i to compute p_u using least-squares optimization.

This iterative process continues until convergence.

Advantages of ALS:

- Parallelization: ALS can compute each q_i independently and each p_u independently, allowing for significant parallel processing.
- Handling Implicit Data: ALS is more efficient for datasets with implicit feedback, as it avoids the impracticality of processing each training case individually.

3.2.3. Adding Biases in Matrix Factorization

Matrix factorization enhances collaborative filtering by accommodating various data aspects and application-specific needs. A key consideration is the inclusion of biases to better capture rating variations that arise from user and item differences, independent of interactions.

Understanding Bias

In typical collaborative filtering data, certain users consistently rate items higher or lower than others, and some items are perceived as better or worse. This systematic variation can be accounted for by incorporating biases into the model, rather than relying solely on the interaction between user and item factors. The inclusion of biases modifies the original interaction model:

$\hat{r}_{ui} = b_i + b_u + q_i^T p_u$

This equation breaks down the observed rating into three components:

- Item Bias (b_i)
- User Bias (b_u)
- User-Item Interaction ($q_i^T p_u$)

By separating these components, each can capture relevant aspects of the rating signal.

The system learns to estimate these parameters by minimizing the following squared error function:

$$\min_{p,q,b} \sum_{(u,v)\in\kappa} (r_{ui} - b_u - b_i - q_i^T p_u)^2 + \lambda(\|p_u\|^2 + \|q_i\|^2 + b_u^2 + b_i^2)$$

Here, κ represents the known user-item pairs. The regularization term λ helps prevent overfitting by penalizing large parameter values.

Accurately modeling these biases is crucial, as they capture much of the signal in user-item interactions, leading to improved recommendation accuracy. More sophisticated bias models have also been developed to enhance performance further.

4. Results

4.1. Overview

In this chapter, we present the implementation of a recommender system based on ALS. The goal of this implementation is to predict user ratings for items (movies), leveraging both user and item biases to enhance accuracy. By employing collaborative filtering methods, specifically focusing on a probabilistic approach to matrix factorization, we aim to improve the quality of recommendations provided to users in a dynamic environment.

4.2. MovieLens Dataset (3)

The dataset used in this project is the MovieLens dataset, a widely recognized resource for developing and testing movie recommendation systems. This dataset was collected by GroupLens, a research lab at the University of Minnesota, which operates MovieLens as a research-oriented site. The data was gathered with the aim of building robust movie recommendation services, and it provides valuable insights for exploring collaborative filtering approaches.

The MovieLens dataset contains both 5-star ratings and free-text tags for movies, allowing us to analyze user preferences based on explicit feedback (ratings) and implicit interest indicators (tags). In total, it comprises 32 million ratings and over 2 million tag applications across more than 82,000 movies, contributed by more than 200,000 users. Notably, each user included in the dataset has rated at least 20 movies, ensuring a sufficient level of engagement for reliable analysis.

No demographic information, such as age or gender, is included. Users are represented solely by anonymized IDs, preserving privacy while allowing for pattern discovery and recommendation modeling.

4.2.1. User Behavior Distribution

The distribution of users ratings exhibits a power law characteristic, where a small number of users contribute a large proportion of ratings. This behavior is visually represented in Figure 1, which shows the frequency of ratings given by users.

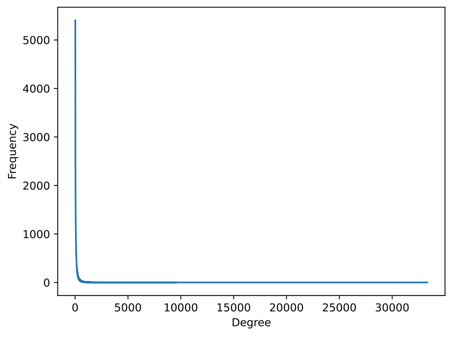

Figure 1. Degree Distribution for users

As we can see, a few users are highly active, contributing significantly to the dataset, while the majority provide relatively few ratings. When applying a logarithmic transformation to this distribution, it resembles Zipf's law, indicating that the frequency of user ratings decreases rapidly as we move down the list of users (Figure 2).

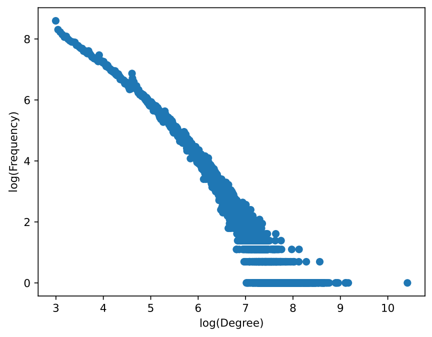

Figure 2. Zipf's law for users

4.2.2. Movie Popularity Distribution

Similarly, the distribution of movie ratings also follows a power law. Figure 3 illustrates the frequency of ratings received by movies, highlighting that a small number of movies are rated highly, while most receive few ratings.

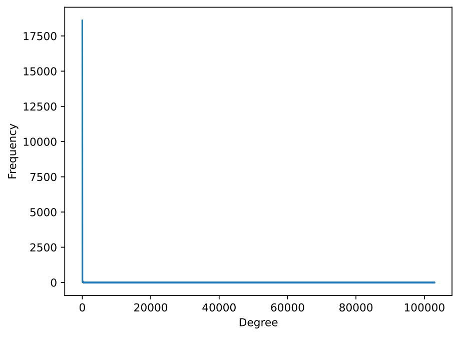

Figure 3. Degree Distribution for movies

This distribution further reinforces the notion that certain movies gain significant attention and ratings, while the majority remain relatively obscure. Again, a logarithmic transformation of this distribution reveals a Zipf-like pattern, where the rank of the movie corresponds inversely to its frequency of ratings (Figure 4).

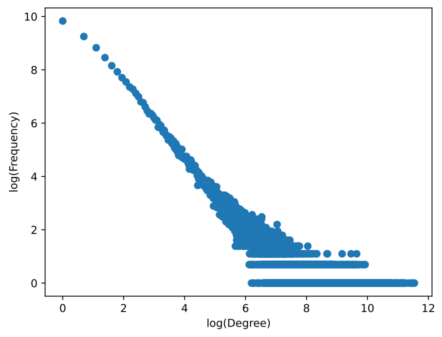

Figure 4. Zipf's law for movies

4.2.3. Implications for the Recommender System

Understanding these distributions is crucial for the design of our recommender system. The skewed nature of user and item interactions suggests that traditional recommendation techniques may struggle with sparsity, especially for less popular items or infrequent users. Therefore, our approach will incorporate mechanisms to address these challenges, ensuring that the system provides relevant recommendations even for items with limited data.

4.2.4. Ratings Distributions

Exploring the distribution of ratings is essential for comprehending the data landscape. Figure 5 shows the overall distribution of movies ratings.

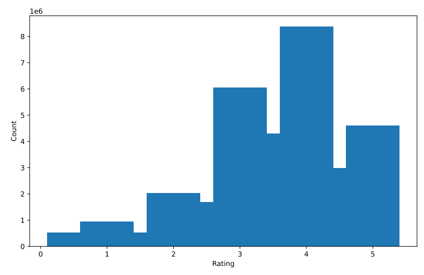

Figure 5. Overall ratings distributions

Additionally, I analyzed ratings based on genres, which offers deeper insights into user preferences. I used 2 genres, Comedy (Figure 6) and Drama (Figure 7).

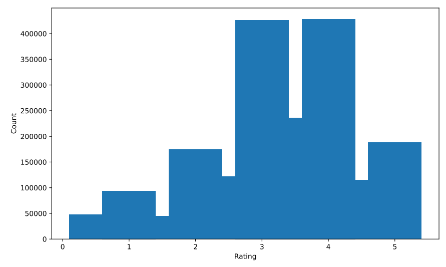

Figure 6. Comedy Movies ratings distributions

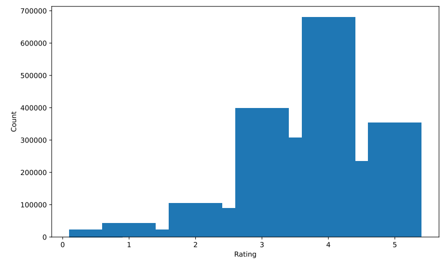

Figure 7. Drama Movies ratings distributions

4.3. Handling ALS efficiently

4.3.1. Data Structures

I utilize two data structures to manage and process the recommendation data:

1. DataStructureForRecommendation
This data structure is designed for general-purpose recommendation tasks without considering the temporal aspect of user interactions.

2. DataStructureForRecommendationSorted
This data structure extends the basic structure by incorporating timestamps, enabling chronological sorting of user interactions. This is particularly useful for creating train-validation-test splits based on temporal order . It has the same attributes as the **DataStructureForRecommendation** but differs in populating of the data and the train,test splits.

4.3.2. Train-Test Split

DataStructureForRecommendation:

- Training Set: Approximately 90% of the ratings.
- Test Set: Approximately 10% of the ratings.
- Method: Randomly assign each rating to the training or test set based on a 90% probability threshold.

DataStructureForRecommendationSorted:

- Training Set: 70% of the earliest ratings.
- Validation Set: 20% of the subsequent ratings.
- Test Set: 10% of the latest ratings.
- Method: Sort each user's or movie's ratings chronologically by timestamp before splitting.

4.4. ALS using only Biases

The model begins by considering user and item biases. The bias for item m given user u is calculated as follows:

$$b_m^{(u)} = \frac{\lambda \sum_{n\in\Omega(m)} \left(r_{mn} - b_n^{(i)}\right)}{\lambda|\Omega(m)| + \gamma}$$

The loss can be calculated using :

$$Loss = \frac{\lambda}{2} \sum_{mn} \left(r_{mn} - \left(b_m^{(u)} + b_n^{(i)}\right)\right)^2$$

Where b_n^i is the bias for user n given item i and to calculate it we can follow the same logic as b_m^u.
Root mean square also can be calculated using :

$$RMSE = \sqrt{\frac{1}{N} \sum_{mn} \left(r_{mn} - \left(b_m^{(u)} + b_n^{(i)}\right)\right)^2}$$

4.4.1. Vectorized Extension of ALS with Bias Only

The ALS algorithm using biases only can be significantly optimized through vectorization. By representing user-movie interactions as vectors, the algorithm can perform batch computations, reducing computational overhead and accelerating convergence.

Bias Updates (1) Instead of updating biases individually, user and item biases can be updated in a vectorized manner:

$$b^{(u)} \leftarrow \frac{\lambda \cdot \sum(\mathbf{S_u} - b_k^{(i)})}{\lambda \cdot |\mathbf{S_u}| + \gamma} \tag{1}$$

$$b^{(i)} \leftarrow \frac{\lambda \cdot \sum(\mathbf{M_i} - b_p^{(u)})}{\lambda \cdot |\mathbf{M_i}| + \gamma} \tag{2}$$

where k is movies indices watched by user u, p is the users indices who watched movie i. S,M is the users-rating matrix and Movies rating matrix respectively. $b^{(u)}$ and $b^{(i)}$ are user and item bias vectors, λ is the regularization parameter, and γ controls bias adjustment.

Algorithm 1 Vectorized ALS with Bias Only

```
1: Initialize user bias $b^{(u)} \leftarrow \mathbf{0}$ of size $E$ (number of users)
2: Initialize item bias $b^{(i)} \leftarrow \mathbf{0}$ of size $F$ (number of items
   ”movies”)
3: Initialize iteration counter $t \leftarrow 0$
4: Maximum number of iterations T
5: repeat
6:     for $e = 1$ to $E$ do
7:         Retrieve user $e$’s ratings: ratings $\leftarrow \mathbf{S}[e]$
8:         Retrieve movie indices: k $\leftarrow$ movies_user$[e]$
9:         Retrieve item biases: $y \leftarrow b^{(i)}$[k]
10:        Compute bias sum: bias_sum $\leftarrow \lambda\cdot$(ratings$-y$)
11:        Aggregate bias: bias_sum $\leftarrow \sum$ bias_sum
12:        Update user bias: $b_e^{(u)} \leftarrow \frac{\text{bias\_sum}}{\lambda\cdot|\text{ratings}|+\gamma}$
13:    end for
14:    for $f = 1$ to $F$ do
15:        Similar vectorized update for item bias $b_f^{(i)}$
16:    end for
17:    Increment iteration counter: $t \leftarrow t + 1$
18: until $t \geq T$ or convergence
19: Return user bias $b^{(u)}$, item bias $b^{(i)}$
```

Vectorized Loss Calculation (2) The loss function can also be vectorized:

$$L = \frac{\lambda}{2N} \sum_{n \in users} \sum \left(S_u - (b^{(u)} + b_k^{(i)})\right)^2 \tag{3}$$

where N is the total number of ratings and is used for normalization.

Algorithm 2 Loss Calculation for ALS with Bias

```
1: Initialize overall loss: $L \leftarrow 0$
2: Initialize counter: $n \leftarrow 0$
3: for each user $e$ do
4:     Retrieve movie indices: $k \leftarrow$ movies_user$[e]$
5:     Retrieve ratings: ratings $\leftarrow \mathbf{S}[e]$
6:     Compute predicted ratings: $y \leftarrow b_e^{(u)} + b_k^{(i)}$
7:     Compute error: $\epsilon \leftarrow$ ratings $- y$
8:     Compute squared error: $e \leftarrow \epsilon^2$
9:     Accumulate loss: $L \leftarrow L + \sum e$
10:    Update counter: $n \leftarrow n + |\text{ratings}|$
11: end for
12: Return $\frac{\lambda}{2} \cdot \frac{L}{n}$        ▷ Final Mean Squared Error (MSE)
```

Vectorized RMSE Calculation (3) Similarly, RMSE can be computed efficiently:

$$\text{RMSE} = \sqrt{\frac{1}{N} \sum_{n \in users} \sum \left(S_u - (b^{(u)} + b_k^{(i)})\right)^2} \tag{4}$$

Vectorization reduces the need for explicit loops, leverages optimized linear algebra operations, and significantly improves performance on large datasets.

Algorithm 3 RMSE Calculation for ALS with Bias

```
1: Initialize RMSE accumulator: $RMSE \leftarrow 0$
2: Initialize counter: $n \leftarrow 0$
3: for each user $e$ do
4:     Retrieve movie indices: k $\leftarrow$ movies_user$[e]$
5:     Retrieve actual ratings: ratings $\leftarrow \mathbf{S}[e]$
6:     Compute predicted ratings: $y \leftarrow b_e^{(u)} + b_k^{(i)}$
7:     Compute error: $\epsilon \leftarrow$ ratings $- y$
8:     Compute squared error: $e \leftarrow \epsilon^2$
9:     Accumulate squared error: $RMSE \leftarrow RMSE + \sum e$
10:    Update counter: $n \leftarrow n + |\text{ratings}|$
11: end for
12: Return $\sqrt{\frac{R}{n}}$        ▷ Final Root Mean Squared Error
```

4.4.2. Loss Function Results

The loss functions for both data structures are illustrated in Figures 8 and 9. For the **Basic DS**, Figure 8 displays the training and testing loss, while Figure 9 shows the training and validation loss for the **Sorted DS**. In both cases, the loss values decrease monotonically, indicating effective model convergence.

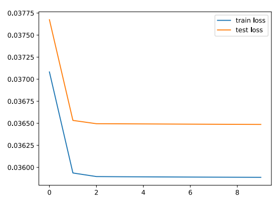

Figure 8. Training vs. Testing Loss with only biases for Basic Data Structure

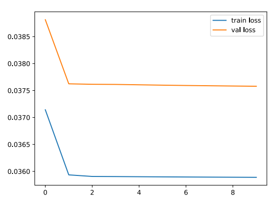

Figure 9. Training vs. Validation Loss with only biases for Sorted Data Structure

4.4.3. RMSE RESULTS

The resulting RMSE demonstrates the model's performance when only biases are utilized as shown in Figures 10 and 11.

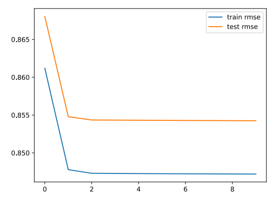

Figure 10. Training vs. Test RMSE using Biases Only for Basic DS

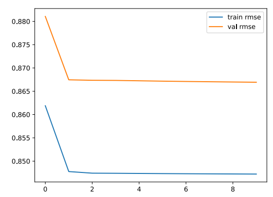

Figure 11. Training vs. Validation RMSE using Biases Only for Sorted DS

4.5. ALS with Biases and Latent Factors

To improve the model's performance, latent factors are incorporated into the Alternating Least Squares (ALS) framework alongside user and item biases. This enhancement allows the model to capture complex user-item interactions and provide more accurate recommendations.

The bias for item m given user u is updated as:

$$b_m^{(u)} = \frac{\lambda \sum_{n\in\Omega(m)} \left(r_{mn} - \left(u_m^\top v_n + b_n^{(i)}\right)\right)}{\lambda|\Omega(m)| + \gamma} \tag{5}$$

The latent factors for user u are computed as follows:

$$u_m = \left(\lambda \sum_{n\in\Omega(m)} v_n v_n^T + \tau I\right)^{-1} \left(\lambda \sum_{n\in\Omega(m)} v_n \left(r_{mn} - b_m^{(u)} - b_n^{(i)}\right)\right)$$

$$v_n = \left(\lambda \sum_{i\in\Omega(n)} u_i u_i^T + \tau I\right)^{-1} \left(\lambda \sum_{i\in\Omega(n)} u_i \left(r_{nm} - b_m^{(u)} - b_n^{(i)}\right)\right)$$

4.5.1. VECTORIZED ALS WITH BIASES AND LATENT FACTORS

The ALS algorithm is further optimized by vectorizing the computation of user and item latent factors. This reduces computational overhead and speeds up convergence.

Vectorized Bias Updates User and item biases are updated using vectorized operations:

$$b^{(u)} \leftarrow \frac{\lambda \cdot \sum \left(S_u - (U_u^\top V_k + b_k^{(i)})\right)}{\lambda \cdot |S_u| + \gamma} \tag{6}$$

$$b^{(i)} \leftarrow \frac{\lambda \cdot \sum \left(M_i{}^\top - (U_p^\top V_i + b_p^{(u)})\right)}{\lambda \cdot |M_i| + \gamma} \tag{7}$$

where U and V represent user and item latent matrices, respectively.

Latent Factor Updates (4) User and item latent factors are updated as follows:

$$U_u = \left(\lambda(V_k \cdot V_k^T) + \tau I\right)^{-1} \left(\lambda V_k \left(S_u - b^{(u)} - b_k^{(i)}\right)\right)$$

$$V_i = \left(\lambda(U_p \cdot U_p^T) + \tau I\right)^{-1} \left(\lambda U_p \left(M_i - b^{(i)} - b_p^{(u)}\right)\right)$$

Algorithm 4 Vectorized ALS with Biases and Latent Factors

```
1: Initialize user bias $b^{(u)} \leftarrow \mathbf{0}$ and item bias $b^{(i)} \leftarrow \mathbf{0}$
2: Initialize user latent matrix $U$ and item latent matrix $V$
3: Initialize iteration counter $t \leftarrow 0$
4: Maximum number of iterations $T$
5: repeat
6:     for $e = 1$ to $E$ do
7:         Retrieve user $e$'s ratings: ratings $\leftarrow \mathbf{S}[e]$
8:         Retrieve movie indices: k $\leftarrow$ movies_user$[e]$
9:         Compute error: $\epsilon \leftarrow$ ratings $- (b_e^{(u)} + b_k^{(i)})$
10:        Update user latent factors: $x \leftarrow \lambda V_k \cdot \epsilon$
11:        Update matrix: $y \leftarrow \lambda V_k V_k^\top + \tau I$
12:        $U_e \leftarrow \cdot y^{-1} \cdot x$
13:    end for
14:    for $f = 1$ to $F$ do
15:        Similar vectorized update for item latent factors $V_f$
16:    end for
17:    Increment iteration counter: $t \leftarrow t + 1$
18: until $t \geq T$ or convergence
19: Return user latent matrix $U$, item latent matrix $V$, user bias $b^{(u)}$, item bias $b^{(i)}$
```

Vectorized Loss Calculation

$$L = \frac{\lambda}{2N} \sum_{n\in users} \sum \left(S_u - (U_u^\top V_k + b^{(u)} + b_k^{(i)})\right)^2 \tag{8}$$

Vectorized RMSE Calculation

$$\text{RMSE} = \sqrt{\frac{1}{N} \sum_{n\in users} \sum \left(S_u - (U_u^\top V_k + b^{(u)} + b_k^{(i)})\right)^2} \tag{9}$$

4.5.2. Loss and RMSE Results

Figures 12 and 13 illustrate the training, test, and validation loss after integrating latent factors. Similarly, Figures 14 and 15 show the corresponding RMSE improvements. As for the test set on the Sorted DS, the model produces 0.81 RMSE.

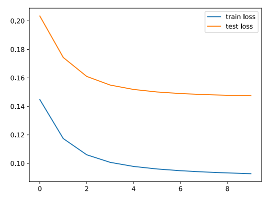

Figure 12. Training vs. Test Loss using Biases and Latent Factors for the Basic DS

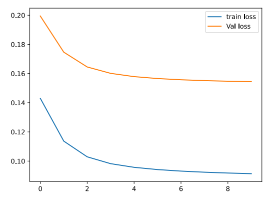

Figure 13. Training vs. Validation Loss using Biases and Latent Factors for the Sorted DS

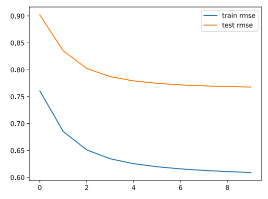

Figure 14. Training vs. Test RMSE using Biases and Latent Factors for the Basic DS

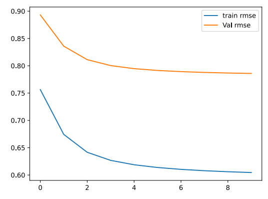

Figure 15. Training vs. Validation RMSE using Biases and Latent Factors for the Sorted DS

4.6. Hyper-parameter selection

The hyper-parameter selection experiment was done on the 100K MovieLens dataset.

4.6.1. Training Iterations

Chosing the number of training iterations depends on several factors :

- Convergence: ALS updates the user and item latent factors iteratively. Each iteration attempts to minimize the loss function, which measures the difference between the actual ratings and the predicted ratings. Generally, more iterations allow the algorithm to converge better to the optimal solution. I also noticed that the RMSE decreases faster in the first 5 iterations and then the fall in its value tends to be slower. So i was in a choice of more than 5 iterations and i chose 10 which allows me to see and insure of the convergence part.
- Avoiding Over-fitting: While more iterations can improve training performance, excessive iterations may lead to over-fitting, where the model learns noise in the training data rather than the underlying patterns.

4.6.2. Lambda λ

This is the regularization parameter that helps prevent over-fitting. A larger value of λ adds more penalty to the loss function for larger values of the latent factors, effectively discouraging the model from fitting noise in the training data. I used grid search to find the best lambda in (0.01, 0.1, 0.2, 0.5) and the best was 0.5 when combined with gamma and tau which will be discussed next.

4.6.3. Tau τ

This is often used to control the step size in the updates of latent factors, affecting the convergence speed and stability of the training process. I used grid search to search for the best tau in (0.5 , 1 , 2) and the best was 0.5 when integrated with gamma and lambda.

4.6.4. Gamma γ

The choice of γ affects how much influence the biases have during the training process. I used grid search to search for the best gamma in (0.01, 0.1, 0.2) and the best was 0.1 when integrated with lambda and tau.

4.6.5. Number of Latent Factors (K)

I want to find a balance between model complexity and performance—specifically, how well the model generalizes to unseen data. I tried with different values of K. I used grid search to find the best K, I used (2, 4, 8, 16, 32,64). I trained the model on each value of K and calculate the RMSE for them. Lambda, gamma and tau for this experiment were selected using the previous selected values. I had to make a trade off because higher values of K is expensive in term of computation, so i had to see the point in which a relatively small value of K will result in a reasonable RMSE value, so i chose K to be 32. Figure 16 is a plot for the distribution of K vs their corresponding RMSE values.

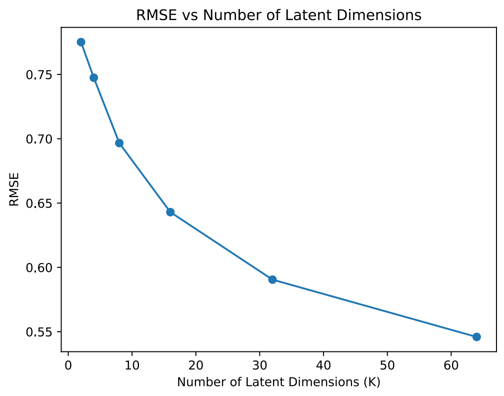

Figure 16. RMSE vs the Number of latent dimensions

5. Prediction

I created a dummy user and I made him watch Harry Potter and the Sorcerer's Stone and give it a rating of 5 stars. I recommended him the movies based on the equation:

$$s_i^{(n)} = d_m \cdot v_n + 0.05 * b_n^{(i)}$$

where $s_i^{(n)}$ represents the predicted rating (score) for movie i and d_m represents the latent factor for the dummy user. I multiply with 0.05 to reduce the contribution of the item bias because some popular movies (high bias) will always sit on the top.
I sorted the predictions in descending order and show the titles of the highest predictions.

Table 1. Results of a dummy user who gave *Harry Potter and the Sorcerer's Stone (2001)* a 5-star rating.

Movie ID	Title
69844	Harry Potter and the Half-Blood Prince (2009)
54001	Harry Potter and the Order of the Phoenix (2007)
5816	Harry Potter and the Chamber of Secrets (2002)
88125	Harry Potter and the Deathly Hallows: Part 2 (2011)
81834	Harry Potter and the Deathly Hallows: Part 1 (2010)

5.1. Movies Train Vectors Space

To visualize how similar movies are positioned in the model's latent space, I plotted the first two latent dimensions of selected movie vectors. Unlike dimensionality reduction techniques like PCA, this approach directly uses the latent factors learned by the model. Additionally, each movie's latent vector is scaled slightly by its bias to account for popularity effects.

This visualization reveals how movies with similar characteristics—such as the *Twilight Saga* series (Fantasy/Romance) and the *Saw* series (Horror)—naturally cluster together in the latent space.

Figure 17 shows the distribution of these movie embeddings in two dimensions.

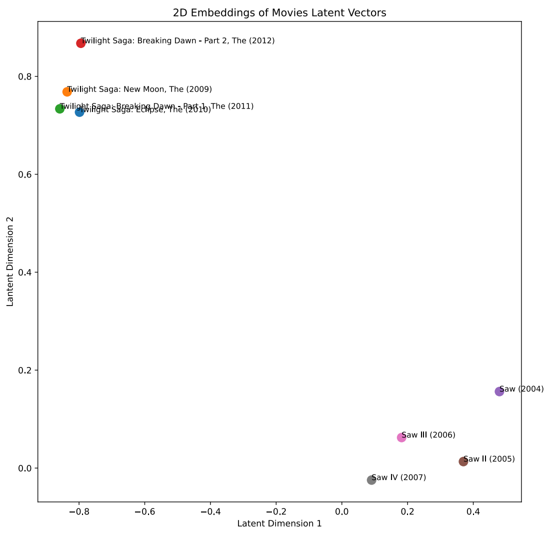

Figure 17. 2D Embeddings of Twilight Saga series vs Saw series in the latent space

5.2. Polarizing Movies

5.2.1. Directly from the Dataset

A polarizing movie is one that evokes strong opinions, either positive or negative, rather than moderate or indifferent reactions. To identify the most polarizing movies, I calculated the **standard deviation** and the **count** of ratings for each movie. Movies with a rating count of 20 or fewer were excluded to ensure reliability. A higher standard deviation indicates greater polarization, reflecting diverse and conflicting user opinions. The count of ratings serves as a measure of the movie's popularity. By sorting the movies in descending order of standard deviation, I identified the most polarizing movies, and by sorting in ascending order, I identified the least polarizing ones. Tables 2 and 3 present the most and least polarizing movies, respectively.

Table 2. Most Polarizing Movies Using The Dataset directly

TITLE	STD DEV	COUNT
TWISTED PAIR (2018)	1.90	31
LOOSE CHANGE: 2ND ED. (2006)	1.86	31
WHAT THE BLEEP! (2006)	1.85	27
FATEFUL FINDINGS (2013)	1.84	79
ONE DIRECTION: THIS IS US (2013)	1.80	41
KIDS OF SURVIVAL (1996)	1.77	24
ANATOMY OF A LOVE SEEN (2014)	1.76	21
SOUND OF MUSIC LIVE (2013)	1.71	30
SANTA WITH MUSCLES (1996)	1.71	163
BLACK IS KING (2020)	1.70	32

Table 3. Least Polarizing Movies Using The Dataset directly

TITLE	STD DEV	COUNT
THE NET (2016)	0.35	24
SNIPER, THE (1952)	0.36	29
TWO LIVES (ZWEI LEBEN) (2012)	0.39	24
THE LOST LEONARDO (2021)	0.41	23
LOUIS THEROUX: LOUIS AND THE NAZIS (2003)	0.42	24
THE MYSTERY OF HENRI PICK (2019)	0.42	23
SWEAT (2020)	0.43	22
THE HOUSE OF FEAR (1945)	0.43	21
NEVER WEAKEN (1921)	0.44	28
RETURN TO SENDER (2004)	0.45	21

5.2.2. USING THE LATENT VECTORS

To further analyze movie polarization, I computed predictions for each movie based on the users who watched them. Subsequently, I calculated the **standard deviation** of these predictions for each movie. A higher standard deviation indicates a wider range of predicted ratings, signifying diverse and potentially conflicting user opinions—hence, greater polarization. Additionally, I calculated the **Euclidean norm** of each movie's latent vector. The hypothesis posits that movies with higher latent norms should exhibit higher standard deviations (at least above the mean standard deviation). To test this hypothesis, I selected the **top 10 movies with the highest latent norms** and calculated the average of their standard deviations. Similarly, I selected the **bottom 10 movies with the lowest latent norms** and calculated the average of their standard deviations. These average standard deviations were then plotted as horizontal lines on the distribution of standard deviations, as shown in Figure 18.

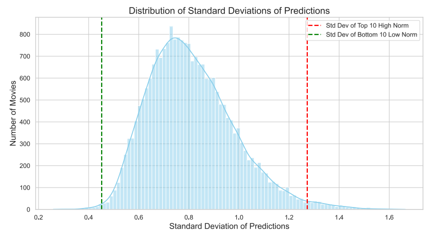

Figure 18. Distribution of standard deviation of predictions

Tables 4 and 5 present the most and least polarizing movies, respectively, based on the standard deviation of predicted ratings and the latent norm of movie vectors. A higher standard deviation and latent norm indicate greater variability in user preferences, reflecting stronger polarization. Conversely, lower values suggest more consistent user opinions and lower polarization.

Table 4. Most Polarizing Movies using the Predictions and Latent Norm

TITLE	STD DEV	LATENT NORM
FATEFUL FINDINGS (2013)	1.578272	6.141332
BIRDEMIC: SHOCK AND TERROR (2010)	1.289856	6.133512
MANOS: THE HANDS OF FATE (1966)	1.252253	5.617301
ROOM, THE (2003)	1.322111	5.336578
BARENAKED IN AMERICA (1999)	1.115469	5.242451
SILENT NIGHT, DEADLY NIGHT PART 2 (1987)	0.896086	5.159497
MAN WHO SAVES THE WORLD	1.147672	5.012164
EXPELLED (2008)	1.431452	4.959569
TWISTED PAIR (2018)	1.457564	4.827200
SAMURAI COP (1989)	1.236650	4.800586

Table 5. Least Polarizing Movies using the Predictions and Latent Norm

TITLE	STD DEV	LATENT NORM
HOUSE OF FURY (2005)	0.262895	0.407881
SPIDER FOREST (GEOMI SUP) (2004)	0.594735	0.640812
PSYCHOMETRY (2013)	0.458959	0.779359
ESCAPE FROM ALCATRAZ (1979)	0.450477	0.817690
DOUGH (2015)	0.390635	0.833405
HOW TO PLAN AN ORGY IN A SMALL TOWN (2015)	0.391778	0.845272
CATCH ME IF YOU CAN (2002)	0.518425	0.855818
FILMS TO KEEP YOU AWAKE: THE BABY'S ROOM (2006)	0.443902	0.858348
THE HOUSE OF FEAR (1945)	0.390803	0.859228
WEATHER GIRL (2009)	0.618350	0.876803

6. Software and Deployment

To showcase the effectiveness of our recommendation system, we developed a user-friendly web application that offers an interactive and immersive experience. The application is built with **Nextjs** and **Python Flask** ,enabling seamless interaction and real-time recommendations.

6.1. Accessibility

- **Web Application:** Click here to access the web app.
- **Frontend Repository:** Frontend Source code in Github
- **Backend Repository:** Backend Source code in Github
- **Model Repository:** Model Source code in Github

7. Future Work

To further improve the recommendation system's performance and personalization, several enhancements can be explored:

- **Content-Based Features:** Integrate item metadata (genres, directors, actors) into the latent factor model to address the cold-start problem and enrich recommendations.
- **Context-Aware Recommendations:** Incorporate contextual data (user demographics, time, location) to deliver more dynamic and personalized suggestions.
- **Temporal Dynamics:** Model evolving user preferences and item popularity over time using time-based decay or sequential models.
- **Scalability:** Optimize ALS with parallel or distributed computing to handle large-scale datasets efficiently.
- **Multi-Modal Data:** Incorporate diverse data sources (posters, trailers, reviews) for richer item representation and improved accuracy.

Exploring these directions will create a more robust, scalable, and user-centric recommendation system.

8. Conclusion

In conclusion, this study compares two ALS-based collaborative filtering approaches: one using only biases and the other combining biases with latent factors. Analysis on the MovieLens dataset shows that incorporating latent factors significantly improves the accuracy and relevance of movie recommendations. These findings highlight the importance of strategic model design in building effective and scalable recommendation systems, guiding researchers and practitioners toward more robust solutions that adapt to evolving user preferences.

References

[1] DataJobs. Recommender systems (netflix), 2013. Accessed: January 11, 2024.

[2] Simon Funk. Netflix update: Try this at home, 2006. Accessed: 2024-01-12.

[3] GroupLens Research. Movielens dataset, 2024. `https://grouplens.org/datasets/movielens/`.

[4] Yehuda Koren. Matrix factorization techniques for recommender systems. *IEEE Computer*, 42(8):30–37, 2009. Accessed: January 11, 2024.

[5] Greg Linden, Brent Smith, and Jeremy York. Two decades of recommender systems at amazon.com. In *Proceedings of the 13th ACM Conference on Recommender Systems (RecSys '19)*. Amazon Science, 2019. Accessed: January 11, 2024.

[6] Ulrich Paquet and Noam Koenigstein. A scalable collaborative filtering approach for large-scale recommender systems on xbox live. In *Proceedings of the 6th ACM Conference on Recommender Systems (RecSys)*, pages 281–284. ACM, 2013. Accessed: January 11, 2024.

[7] Maksims Volkovs, Reid McLeod, and Timo Poutanen. Two-stage recommender systems. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 388–389. ACM, 2018.

[8] Wikipedia. Netflix prize. https://en.wikipedia.org/wiki/Netflix_Prize, 2024. Accessed: 2024-01-11.