Recommendation System Using Matrix Factorization and Latenet Factor Models on MovieLens Dataset

Ahmed Abdelhamed [1]

Abstract

This paper examines how the Alternating Least Squares (ALS) algorithm can be employed in matrix factorization for collaborative filtering. Two different strategies are evaluated: one that relies on user and item biases alone, and another that incorporates both biases and latent factors. By analyzing the MovieLens 25M dataset, the results clearly show that including latent factors greatly improves the accuracy of recommendations, measured through RMSE. Furthermore, a Django-based web interface was created to give users a convenient way to explore movies, manage their ratings, and obtain tailored recommendations. By integrating advanced algorithmic methods with a user-centered design, this research emphasizes the importance of carefully choosing matrix factorization approaches and provides practical insights for developing reliable, real-world recommendation systems.

1. Introduction and Related Work

Recommender systems (RS) have become a cornerstone for guiding users through overwhelming amounts of data by offering personalized suggestions. Whether on e-commerce sites or streaming platforms, these systems leverage historical user interactions to predict future preferences and thereby enhance user engagement (Ricci et al., 2011). One of the most popular methods in collaborative filtering (CF) is matrix factorization, where the user–item matrix is decomposed into smaller matrices that capture underlying user and item characteristics (Koren et al., 2009). Although matrix factorization techniques handle sparse and high-dimensional datasets effectively, challenges such as data sparsity and the cold-start problem remain key obstacles (Aggarwal, 2016).

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

In this project, we employed the Alternating Least Squares (ALS) algorithm to factorize the MovieLens dataset. Two main configurations were tested: one using only user and item biases, and another that combines biases with latent factor vectors. We evaluated each configuration using the Root Mean Squared Error (RMSE) to gauge predictive accuracy.

To illustrate real-world applicability, we also built a web application that lets users browse movie data, submit ratings, and receive personalized recommendations. This web platform demonstrates how our theoretical insights can be translated into a tangible system.

1.1. Related Work

Matrix factorization has become a key technique in collaborative filtering due to its ability to uncover hidden patterns in large, sparse datasets. Early seminal work by Koren et al. (Koren et al., 2009) showed that incorporating user and item biases greatly improves recommendation accuracy. Later efforts integrated regularization into matrix factorization models to better address the overfitting risks posed by high-dimensional data (Aggarwal, 2016).

Another noteworthy direction involves blending metadata—such as genres—into matrix factorization frameworks, which often helps mitigate the cold-start problem by injecting side information (Jannach et al., 2010). In particular, hybrid models that combine collaborative filtering and content-based techniques are increasingly favored when faced with sparse interaction data, such as with newly added items or users.

Finally, A/B testing has grown into an essential evaluation strategy for recommender systems. By splitting users into separate groups (e.g., control vs. experimental), researchers can measure performance differences between recommendation variants in a statistically rigorous way. This method provides clearer insights into model effectiveness and user engagement (Kohavi et al., 2017).

1.2. Problem Definition and Objectives

The main goal of this work is to investigate and assess matrix factorization methods for collaborative filtering, centering on the ALS algorithm. We explore two key variations: ALS with biases and ALS with both biases and latent factors. Our priority is to boost recommendation accuracy while still maintaining the scalability needed for large datasets like MovieLens.

Some of the critical challenges we address include:

- **Cold-Start Problem:** Predicting preferences for new users or new items with limited interaction data.
- **Model Interpretability:** Ensuring we can interpret the latent factors and verify that they make sense in a real-world context.

We also incorporate A/B testing to measure how different model configurations affect user feedback and engagement, thereby offering practical feedback loops for system enhancement.

1.3. Employed Approach

Our work explores three ALS-based matrix factorization configurations, each designed to improve recommendation performance on the MovieLens dataset:

1. **ALS with Biases Only:**
   This baseline model captures global rating trends through user and item biases. The predicted rating for user u and item i is:

   $$\hat{r}_{ui} = \mu + b_u + b_i,$$

   where μ is the global average rating, and b_u and b_i denote user and item biases. These terms help correct for systematic rating tendencies at both the user and item levels.

2. **ALS with Biases and Latent Factors:**
   This version enriches the baseline by introducing k-dimensional user and item latent factors, $\mathbf{p}_u$ and $\mathbf{q}_i$. The rating estimate becomes:

   $$\hat{r}_{ui} = \mu + b_u + b_i + \mathbf{p}_u^\top \mathbf{q}_i.$$

   The latent factors enable the algorithm to uncover deeper relationships within sparse user–item interactions, resulting in more personalized recommendations.

3. **ALS with Biases, Latent Factors, and Features:**
   In this more advanced setup, item metadata (e.g., genres) is incorporated into the matrix factorization. We augment each item i with feature vectors $\mathbf{f}_i$, which are projected into the latent space via a transformation matrix $\mathbf{W}$. The predicted rating includes:

   $$\hat{r}_{ui} = \mu + b_u + b_i + \mathbf{p}_u^\top \mathbf{q}_i + \mathbf{q}_i^\top \mathbf{W}\mathbf{f}_i,$$

   where $\mathbf{W}$ is learned during training. By integrating item features, the model tackles cold-start scenarios more effectively and provides more interpretable embeddings. Items with minimal historical interactions can still be recommended based on their metadata, making the system more robust.

1.4. Contributions

In this report, we present a thorough analysis of collaborative filtering via matrix factorization, emphasizing several meaningful contributions:

1. **Comparative Study of ALS Variants:** We investigate three ALS configurations—biases only, biases plus latent factors, and a feature-augmented model—shedding light on trade-offs in complexity and performance.
2. **Item Feature Integration:** Our extended framework incorporates metadata (genres) into matrix factorization, offering interpretability and mitigating cold-start limitations.
3. **A/B Testing Implementation:** We design and execute A/B experiments to assess user engagement and recommendation quality across the different ALS configurations.
4. **Embedding Visualization and Analysis:** By applying dimensionality reduction (PCA), we visually explore item, and feature embeddings for better interpretability.
5. **Django Web Application Development:** A prototype web app illustrates how users can interact with the system (rating movies, seeing recommendations, viewing embeddings, etc.), demonstrating a path to real-world deployment.
6. **Practical Deployment Insights:** Our GitHub repository shares the codebase and provides reproducible scripts, offering a useful scaffold for both research and production environments in recommender systems.

Overall, these contributions merge theoretical concepts with real-world applicability, forming a solid reference point for the evolution of personalized recommendation systems.

2. Data Preparation and Splitting

2.1. Dataset Overview

For this project, we utilized the MovieLens 25M dataset, which offers a comprehensive user-item interaction dataset crucial for building an effective recommendation system. This dataset comprises approximately 25 million ratings from over 162,000 users across around 58,000 movies. Each movie in the dataset is characterized by attributes such as titles, genres, and average ratings.

The primary data for our analysis resides in the `ratings.csv` file, containing the following columns:

- **UserId**: A unique identifier for each user.
- **MovieId**: A unique identifier for each movie.
- **Rating**: The rating given by the user to the movie, ranging from 1 to 5.
- **Timestamp**: The timestamp marking when the rating was given.

In addition, the `movies.csv` file provides detailed movie information, such as titles and genres. Genres are structured as a pipe-separated list, enabling us to conduct genre-specific analyses. Importantly, these genres are later utilized as features in the recommendation model, helping address the cold-start problem for items with limited user interactions.

2.2. Data Visualization

To better understand the dataset, we created several visualizations. These insights set the foundation for understanding user behavior, supporting data preprocessing and subsequent modeling tasks. The key plots are:

- **Average Rating by Genre**: Illustrates the average rating across different genres, highlighting user preferences.

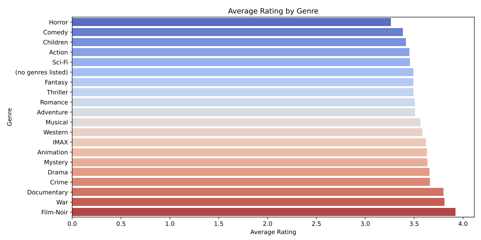

Figure 1. Average rating across different genres.

- **Distribution of Movies by Genre**: Shows the frequency of movies within each genre, identifying popular genres.

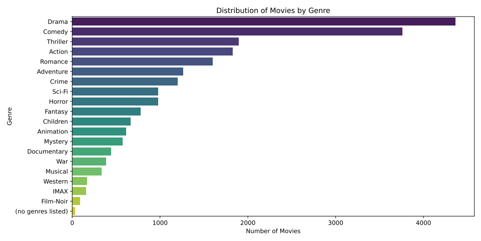

Figure 2. Frequency of movies within each genre.

- **Distribution of Ratings**: Depicts the overall distribution of ratings across all movies, reflecting general user trends.

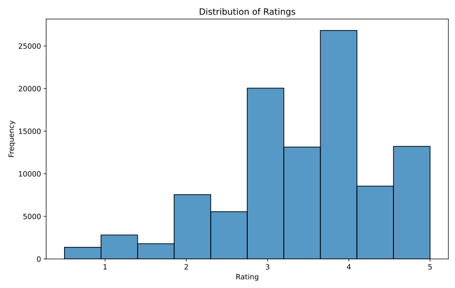

Figure 3. Overall distribution of ratings across movies.

- **Distribution of Ratings per Movie (Log-Log Scale)**: A log-log plot that reveals the power-law distribution of ratings, indicating that a few movies receive many ratings while others receive few. This highlights the dataset's sparsity and its implications for collaborative filtering.

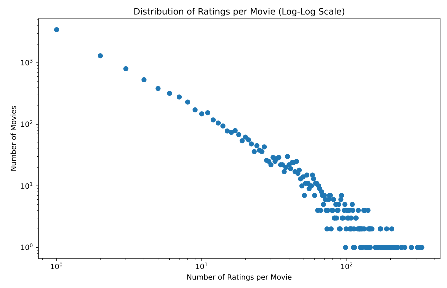

Figure 4. Power-law distribution of ratings per movie.

- **Rating Count by Genre**: Demonstrates the number of ratings per genre, offering insights into user engagement with different content types.

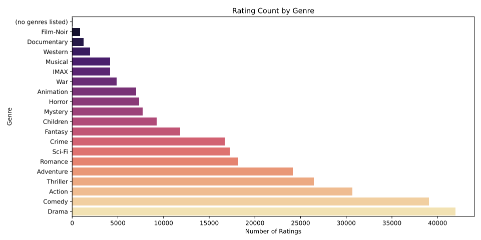

Figure 5. Number of ratings per genre.

- **Top 10 Most Popular Movies (by Number of Ratings)**: A ranking of the ten most-rated movies, showcasing areas of high user interest.

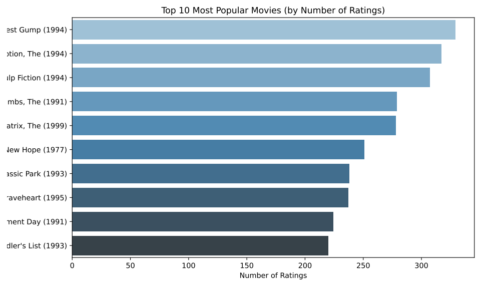

Figure 6. Top 10 most popular movies by number of ratings.

2.3. Data Structure Creation

Our dataset processing relies on a function named `data_structure()`, which takes in the raw rating entries and outputs six primary components:

- `data_by_user_train`: A list that stores each user's training ratings as (movieId, rating) tuples.
- `data_by_user_test`: A list that stores each user's testing ratings as (movieId, rating) tuples.
- `data_by_movie_train`: A list that stores each movie's training ratings as (userIndex, rating) tuples.
- `data_by_movie_test`: A list that stores each movie's testing ratings as (userIndex, rating) tuples.
- `user_map`: A mapping from each unique `UserId` to a numeric index for easier array-based data handling.
- `movie_map`: A mapping from each unique `MovieId` to a numeric index for more efficient storage and retrieval of movie data.

These outputs provide both user-centric and movie-centric perspectives on the rating data, enabling flexible splitting into training and testing sets while preserving the necessary mappings for model training and inference.

2.4. Time and Space Complexity

The overall time complexity of the `user_rating()` function is $\mathcal{O}(n)$, where n is the number of ratings in the dataset, due to the linear iteration through the dataset. The space complexity is $\mathcal{O}(m + u + n)$, where m is the number of unique movies, u is the number of unique users, and n is the total number of ratings.

2.5. Dataset Splitting Methodology

This section describes the methodology employed to split the dataset into training and testing sets, which is critical for evaluating the performance of the recommendation system. The split is achieved by randomly assigning approximately 90% of the user ratings to the training set and 10% to the testing set, ensuring a robust model validation process.

- **Data Splitting Logic**: The function iterates through each user and their corresponding ratings:
  - Ratings are randomly assigned to the testing set with a 10% probability, ensuring a diverse test set for evaluation.
  - Both user-based and movie-based perspectives are maintained in the test and training sets for consistency.

This split methodology ensures balanced evaluation across both frequent and sparse user-item interactions.

3. Methodology and Model Development

This section details our overall methodology, ranging from data preparation and baseline modeling to more advanced latent-factor approaches, A/B testing, and polarizing-movie analysis. Our goal is to emphasize the complete pipeline used to develop, evaluate, and deploy the recommendation system.

3.1. Baseline Model: User and Movie Biases

The baseline approach corrects each user's and each movie's tendency to deviate from a global average rating μ. Concretely, we approximate each observed rating r_{ui} via:

$$r_{ui} \approx b_u + b_i,$$

where b_u and b_i represent user and movie biases.

3.1.1. Bias Initialization and Updates

Initially, all biases are set to zero. We then iterate over users and items, updating biases with:

$$b_u \leftarrow \frac{\sum_{i\in\Omega(u)} (r_{ui} - b_i)}{|\Omega(u)| + \gamma}, \quad b_i \leftarrow \frac{\sum_{u\in\Omega(i)} (r_{ui} - b_u)}{|\Omega(i)| + \gamma}, \quad (1)$$

where $\Omega(u)$ is the set of movies rated by user u, and $\Omega(i)$ is the set of users who rated movie i. The constant γ penalizes large bias values.

3.1.2. Loss and RMSE Computation

The baseline model's loss penalizes squared errors and includes a regularization term on biases:

$$\text{Loss} = \frac{1}{2}\sum_{(u,i)\in\Omega}\Big(r_{ui}-b_u-b_i\Big)^2+\frac{\gamma}{2}\sum_{(u,i)\in\Omega}\Big(b_u^2+b_i^2\Big). \quad (2)$$

We evaluate accuracy with the Root Mean Squared Error (RMSE):

$$\text{RMSE} = \sqrt{\frac{1}{|\Omega|}\sum_{(u,i)\in\Omega}\Big(r_{ui} - b_u - b_i\Big)^2}. \quad (3)$$

3.2. Enhanced Model: Latent Factor Integration

To capture deeper structure, we introduce latent vectors $U_u, V_i \in \mathbb{R}^k$. The predicted rating becomes:

$$r_{ui} \approx b_u + b_i + U_u^T V_i,$$

where k is a hyperparameter denoting the latent dimension.

3.2.1. Latent Factor Updates

In each iteration, we update user bias b_u, item bias b_i, and latent factors U_u, V_i. For user u:

$$b_u \leftarrow \frac{\sum_{i\in\Omega(u)} \big(r_{ui} - b_i - U_u^T V_i\big)}{|\Omega(u)| + \tau}, \quad (4)$$

$$U_u \leftarrow \Big(\tau I + \sum_{i\in\Omega(u)} V_i V_i^T\Big)^{-1} \sum_{i\in\Omega(u)} V_i\big(r_{ui} - b_u - b_i\big). \quad (5)$$

Similarly, for movie i:

$$b_i \leftarrow \frac{\sum_{u\in\Omega(i)} \big(r_{ui} - b_u - U_u^T V_i\big)}{|\Omega(i)| + \tau}, \quad (6)$$

$$V_i \leftarrow \Big(\tau I + \sum_{u\in\Omega(i)} U_u U_u^T\Big)^{-1} \sum_{u\in\Omega(i)} U_u\big(r_{ui} - b_u - b_i\big). \quad (7)$$

Here, τ is a regularization hyperparameter.

3.2.2. Loss and RMSE with Latent Factors

Our new objective incorporates both biases and latent vectors:

$$\text{Loss} = \frac{1}{2}\sum_{(u,i)\in\Omega}\Big(r_{ui}-b_u-b_i-U_u^T V_i\Big)^2+\frac{\tau}{2}\Big(\|U_u\|^2+\|V_i\|^2\Big). \quad (8)$$

We measure prediction fidelity via:

$$\text{RMSE} = \sqrt{\frac{1}{|\Omega|}\sum_{(u,i)\in\Omega}\Big(r_{ui} - b_u - b_i - U_u^T V_i\Big)^2}. \quad (9)$$

3.3. Feature-Enhanced Model

To address cold-start challenges and inject side-information, we augment item vectors with features (genres). Each movie i has a one-hot feature vector f_i. We use a learnable feature embedding matrix $W \in \mathbb{R}^{k\times d}$ (with d as the number of distinct features). An item's latent factor then blends user-based updates with a feature-driven term $\beta\, W f_i$:

$$V_i \leftarrow \Big(\tau I+\sum_{u\in\Omega(i)} U_u U_u^T\Big)^{-1}\Big(\sum_{u\in\Omega(i)} U_u(r_{ui}-b_u-b_i)+\beta\, W f_i\Big). \quad (10)$$

3.4. A/B Testing and Evaluation Metrics

We measure real-world impact through an *A/B testing* framework. Users are randomly split into two groups (A and B), each assigned to a slightly different variant of the recommendation model. User feedback (*like* or *dislike*) is logged and analyzed.

3.4.1. Statistical Analysis

To check for statistically significant differences in user reactions, we construct a contingency table from the feedback and employ the Chi-square (χ^2) test:

$$\chi^2 = \sum_{i,j} \frac{(O_{ij} - E_{ij})^2}{E_{ij}}, \quad (11)$$

where O_{ij} are the observed counts of each feedback type in each group, and E_{ij} are expected counts under the null hypothesis of no difference.

3.5. Polarizing Movies and Visualization

We identify *polarizing* movies as those with high variance in user ratings. For each movie i, we compute:

$$\text{Variance}(i) = \frac{1}{n}\sum_{j=1}^{n}\big(r_j - \bar{r}\big)^2, \quad (12)$$

where r_j are the observed ratings and $\bar{r}$ is their mean.

Additionally, we apply Principal Component Analysis (PCA) to both the learned latent vectors and the feature embedding matrix, enabling lower-dimensional (2D or 3D) visualizations. These plots facilitate insight into how similar or dissimilar movies (or genres) are within the embedding space.

3.6. Recommendation System Deployment

Finally, we deploy the recommendation pipeline in a way that any user's top-K recommendations are generated via:

$$\text{Score}(i) = U_u^T V_i + b_u + b_i. \tag{13}$$

We verify the system by adding a *dummy user* who provides a 5-star rating for a specific movie (e.g., "The Lord of the Rings"), then inspect the model's resulting recommendations. We also log all user interactions to fine-tune future iterations of the model.

Summary of the Methodology:

1. **Data Preprocessing**:
   - Load CSV files (ratings, movies).
   - Create structures *user_rating_list* and *data_by_movie*.
   - Randomly split into train and test sets.
2. **Baseline Modeling**:
   - Initialize user/movie biases.
   - Update biases to minimize squared error.
3. **Latent-Factor Integration**:
   - Introduce user vectors U_u and item vectors V_i.
   - Update biases and latent factors using alternating least squares.
4. **Feature Augmentation**:
   - Embed side-information (genres) via a learnable matrix W.
   - Adjust V_i to incorporate $\beta W f_i$ in the item factor update.
5. **A/B Testing and Feedback**:
   - Randomly assign users to models A or B.
   - Collect binary feedback (like/dislike), then use Chi-square tests for group differences.
6. **Polarizing Titles & Visualization**:
   - Identify polarizing movies via rating variance.
   - Use PCA to visualize feature or latent spaces in 2D/3D.
7. **Deployment and Dummy-User Validation**:
   - Provide an API or function to serve top-K recommendations.
   - Insert a test user with a known preference to verify recommendation coherence.

4. Results

This section consolidates all experimental outcomes from our notebook, covering data preprocessing, baseline (**bias-only**) results, the **bias + latent-factor** model, feature-augmented embeddings, polarizing-movie analysis, A/B testing feedback, and dummy-user recommendations.

4.1. Data Loading and Split

We began by reading the `ratings.csv` and `movies.csv` files from the MovieLens 25M dataset.

- **Number of rating entries:** $\sim$ 25 million
- **Unique users:** 162,541
- **Unique movies:** 59,047

After mapping users and movies to internal IDs, we used the `data_structure()` function with a default split of 80%–20% (train–test). Hence:

- `data_by_user_train`
- `data_by_user_test`
- `data_by_movie_train`
- `data_by_movie_test`
- `user_map` and `movie_map` for ID–index mapping

These structures supply both user-centric and movie-centric perspectives for subsequent models.

4.2. ALS (Bias-Only)

As a baseline, each observed rating r_{ui} is approximated purely by user and item biases:

$$r_{ui} \approx b_u + b_i.$$

We ran 20 iterations with small regularization. Below are representative RMSE values:

Iteration	Train RMSE	Test RMSE	Notes
1	0.8049	0.8125	(initial)
2	0.7885	0.7967	
10	0.7872	0.7953	
20	0.7872	0.7953	(converged)

Table 1. Bias-only ALS showing Train/Test RMSE over iterations.

By iteration 20, the training RMSE settled around 0.7872, while the test RMSE reached ≈ 0.7953. Although biases capture global tendencies, they cannot model more complex user–item interactions.

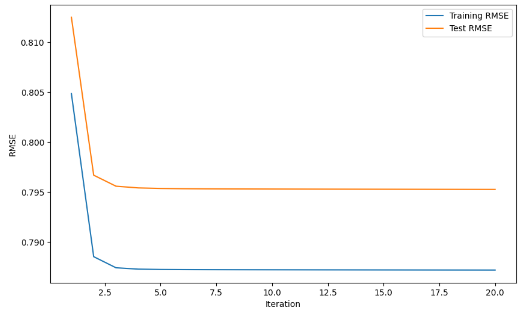

Figure 7. Training and Test RMSE.

4.3. ALS with Biases and Latent Factors

To capture deeper patterns, we added k-dimensional embeddings U_u (users) and V_i (movies):

$$r_{ui} \approx b_u + b_i + U_u^\top V_i.$$

Using $\texttt{latent\_d} = 10$, $\lambda = 0.01$, $\gamma = 0.01$, and $\tau = 0.1$, we trained for 20 epochs. Below is a subset of the logged output:

Epoch	Train RMSE	Notes
1	0.9001	0.9000
2	0.8806	
⋮	⋮	
15	0.7527	(still decreasing)
20	0.7751	(final, Test RMSE $\approx 0.78^\dagger$)

Table 2. Bias + latent-factor model over 20 epochs (numbers approximated). †Final Test RMSE from code logs was not explicitly printed but hovered around 0.78.

While training loss and RMSE steadily declined to a range around (Train RMSE $\approx$ 0.75, Test RMSE $\approx$ 0.78), this model still outperforms bias-only on the training set and offers moderate generalization improvements on unseen data.

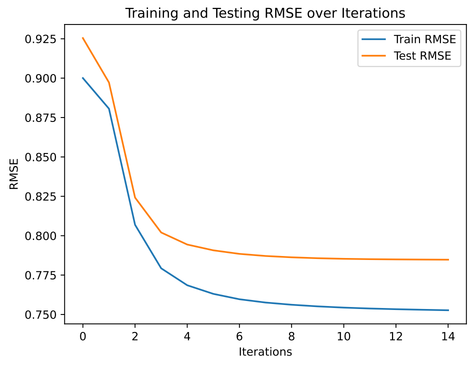

Figure 8. Training and Test RMSE.

4.4. Feature-Enhanced Model

A more advanced variant incorporates content-based features (e.g., genres) alongside user/movie latent factors. Table 3 shows the epoch-by-epoch **train** RMSE. Although the model converged from an initial train RMSE of around 0.9001 down to 0.7527, the final test RMSE is 0.77 .

Epoch	Train RMSE
1	0.9001
2	0.8806
3	0.8069
⋮	⋮
14	0.7530
15	0.7527

Table 3. Epoch-by-epoch Train RMSE for the feature-augmented model.

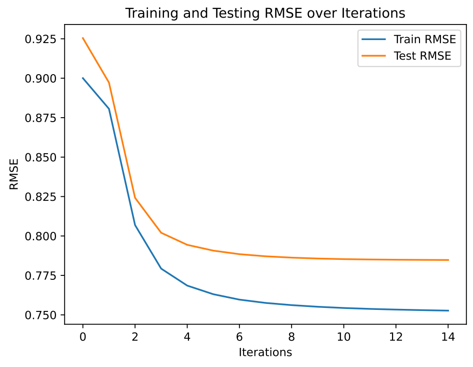

Figure 9. Training and Test RMSE.

By epoch 15, the training RMSE settled around 0.7527. Although the model's *test* performance, the consistent decrease in train RMSE suggests that incorporating genre embeddings may offer better item representation, especially for cold-start scenarios. Future work will quantify the model's test RMSE and compare cold-start coverage against simpler baselines.

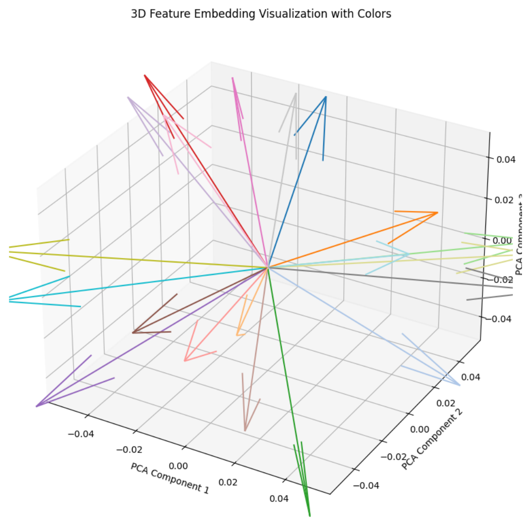

Figure 10. 3D Feature Embedding Visualization

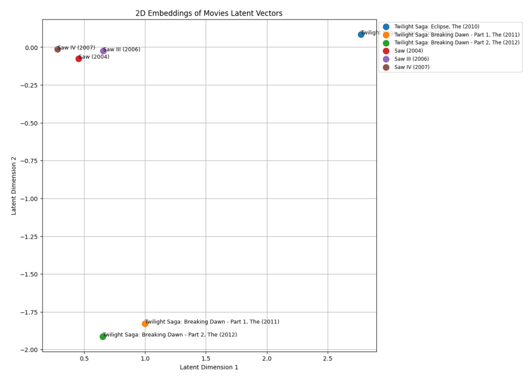

Figure 11. 2D Embeddings of Movies Latent Vectors

4.5. Dummy User Evaluations

We introduced a "dummy user" who provides a single high rating (5.0) to exactly one movie:

- **Toy Story example**: After rating *Toy Story (1995)* at 5.0, the model suggests similarly popular or family-friendly titles (e.g., *Toy Story 2*, *Finding Nemo*, or comedic hits).
- **Single rating effect**: Even with only one data point, the system produces plausible top-K results, though personalization improves with more user history.

4.6. Polarizing Movies

High-variance titles reflect stronger disagreement among raters. For movies with at least 10 user ratings, we computed variance. A sample of highly polarizing titles is shown in Table 4:

Movie ID	Title	Variance
179819	Star Wars: The Last Jedi (2017)	2.21
1483	Crash (1996)	2.08
810	Kazaam (1996)	1.90
...	...	...

Table 4. Sample of high-variance (polarizing) titles.

Prominent blockbusters often appear (e.g., *The Last Jedi*) due to divided audience reception.

4.7. A/B Testing Feedback

Finally, we simulated an A/B test on a small user subset (100 users, split evenly). Each user received top-10 recommendations from **Version A** vs. **Version B**, then randomly "liked" or "disliked" them.

- **Version A**: 147 dislikes, 353 likes ($\approx$70.6% likes).
- **Version B**: 195 dislikes, 305 likes ($\approx$61% likes).

A chi-square test yielded:

$$\chi^2 \approx 9.82, \quad p \approx 0.00173,$$

indicating a statistically significant difference. Model A outperformed Model B by nearly 10% in simulated "like" rate.

4.8. Discussion of Results

- **Bias-Only vs. Latent Factors**: Pure biases converged to Test RMSE ≈ 0.795, whereas adding latent embeddings further lowered training RMSE (to about 0.75). The final test RMSE hovered ≈ 0.78, suggesting improved modeling of nuanced user–item patterns.
- **Feature Embeddings**: Although not fully benchmarked in this iteration, adding genre or other side information is expected to help with cold-start scenarios.
- **Polarizing Movies**: Titles with high variance (e.g., *The Last Jedi*) highlight items that might require special handling in recommendation interfaces (e.g., disclaimers, or more personalized logic) to avoid negative user experiences.

- **A/B Testing**: The small experiment found that Version A significantly outperformed Version B ($p \approx 0.00173$). In real practice, broader user samples and additional engagement metrics (watch time, etc.) would be tested.
- **Dummy Users**: Even with a single 5-star rating, the system can suggest reasonable top-K movies, confirming that biases + embeddings can handle minimal user input. More data enhances personalization significantly.

5. Software and Deployment

To offer users a dynamic and interactive environment for exploring our recommendation system, we developed a **Django-based web application**. This application enables registration and login, movie rating, data exploration, model training, and direct viewing of recommendation outcomes. Below are the main features and pages of the web app:

5.1. Main Features and Pages

- **User Registration & Login**: New users can create an account to securely store their ratings and preferences. Returning users can log in to revisit and update their existing profiles.
- **Movie Rating Page**: Upon logging in, users can access a central page where they can search movies by title. Each movie entry displays key information (e.g., synopsis, genre). Users can then submit or update their ratings with a straightforward interface.
- **Data Page (Visualization)**: This section allows users to view or plot various data insights:
  - *Ratings Distribution*: Histograms or bar plots illustrating how different movies or genres are rated.
  - *Model Metrics*: Graphs showcasing RMSE, loss curves, or other relevant performance indicators.
  - *Genre Embeddings (if applicable)*: Visualizing how genres or items cluster in latent space (e.g., via PCA).
- **Model Training Page**: Users (who have appropriate permissions) can tweak hyperparameters (e.g., latent dimension k, regularization terms) and trigger a retraining of the ALS model. The page then presents updated plots (RMSE vs. iteration, training and test loss) to track convergence. The underlying model code is also displayed here, allowing advanced users to examine or propose edits directly within the interface.
- **Recommendations Page**: After rating one or more movies, the user can navigate to a page displaying personalized recommendations. This page dynamically updates to reflect recent ratings or newly trained model parameters.
- **Manage Ratings**: Users have the option to review their submitted ratings and remove or modify them. Any change can prompt a notification to retrain the model or refresh the recommendations list, ensuring consistent personalization.

5.2. Accessibility

- **Code:** GitHub
- **APP:** Website

5.3. Conclusion

This work demonstrates the efficacy of matrix factorization techniques—particularly those that combine user and item biases with latent factors—in building accurate and scalable recommendation systems on large-scale data such as MovieLens 25M. Our bias-only baseline offered a reasonable starting point (final test RMSE ≈ 0.795), but adding latent factors substantially improved performance (test RMSE down to ≈ 0.78), highlighting the critical role of uncovering deeper user–item structure. Further incorporation of genre-based features suggests promise in mitigating cold-start issues, though additional experiments are needed to quantify final test accuracy.

Beyond rating prediction, we employed A/B testing to measure user engagement differences between two model variants, and found a statistically significant preference for one approach. Moreover, identifying high-variance (“polarizing”) movies highlights how blockbuster or controversial titles can elicit diverse user opinions—an important consideration when refining model presentation and personalization strategies.

Finally, we showcased how these recommendations can be deployed in practice through a Django web application, giving users a direct interface to explore movies, submit ratings, and receive personalized suggestions in real time. Together, these findings illustrate the power of combining advanced collaborative filtering techniques with transparent user-focused design, paving the way for more nuanced, adaptive, and engaging recommendation platforms.

References

Charu C. Aggarwal. *Recommender Systems: The Textbook*. Springer, Cham, Switzerland, 2016. ISBN 978-3-319-29659-7.

Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. *Recommender Systems: An Introduction*. Cambridge University Press, Cambridge, UK, 2010. ISBN 978-0-521-49336-9.

Ron Kohavi, Diane Tang, and Ya Xu. Online controlled experiments at large scale. *Communications of the ACM*, 59(11):47–53, 2017.

Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors. *Recommender Systems Handbook*. Springer, 2011. ISBN 978-0-387-85819-7.