Exploring Various ALS Approaches in Matrix Factorization for Collaborative Filtering: A Study on MovieLens

Yvan Carré [1]

Abstract

This paper explores the application of the Alternating Least Squares (ALS) algorithm to matrix factorization for collaborative filtering, highlighting two distinct approaches: one integrating only biases and the other combining both biases and latent vectors. Through a study using the MovieLens dataset, we demonstrate the significant impact of latent vectors on the accuracy and relevance of movie recommendations. This comparison between the two approaches offers essential insights for researchers and practitioners, emphasizing the importance of strategic choices in designing effective recommendation systems based on matrix factorization for collaborative filtering.

1. Introduction

In today's digital age, personalized recommendations have become ubiquitous, shaping our online experiences. Whether it's choosing a movie on Netflix, a product on Amazon, or a video on YouTube, recommendation systems play a central role in suggesting content tailored to our preferences. As a research field, recommendation systems have gained significant importance since the 1990s, attracting the attention of both researchers and companies (Ricci et al., 2011). The interest in these systems was further amplified by landmark events such as the Netflix Prize in 2006, which offered a million-dollar reward for improving Netflix's recommendation algorithm.

Recommendation systems aim to predict users' preferences based on their past interactions. For instance, Amazon utilizes product ratings to anticipate whether a user will like a particular book or other item. These systems often rely on matrix factorization techniques, a model that identifies latent structures within user-item rating data. Matrix factorization is particularly valued for its ability to handle large datasets and provide personalized recommendations.

However, an essential yet sometimes overlooked aspect of matrix factorization is the impact of biases. Biases can stem from specific user preferences or inherent trends in the products. Incorporating these biases into recommendation models can potentially enhance the accuracy and relevance of the suggestions provided to users.

This article delves into matrix factorization with and without biases, examining how accounting for biases can influence the performance of recommendation systems. We will compare both approaches to determine which one offers the best recommendations across different contexts. This research aims to deepen our understanding of recommendation systems and contribute to their continuous improvement, ultimately providing users with more personalized and relevant experiences.

2. Background

2.1. Recommender system

Recommendation systems have become ubiquitous in our daily lives, playing a crucial role in how users discover and interact with products, services, and content online. Whether on streaming platforms like Netflix, e-commerce sites like Amazon, or social networks like Facebook, these systems help personalize the user experience by suggesting items that may be of interest to them.

There are several approaches to designing recommendation systems, each with its own advantages and disadvantages.

Content-based filtering recommends items similar to those the user has liked in the past. It relies on analyzing item features and explicit user preferences. For example, if a user enjoyed movies of a certain genre or director, the system will recommend other films sharing similar characteristics.

Collaborative filtering, on the other hand, is based on

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

leveraging the behaviors and preferences of many users. It identifies users with similar tastes and recommends items based on what those similar users have liked. This approach is particularly effective for discovering new content that the user may not have found on their own, as it does not rely solely on item characteristics but also uses relationships between users.

Hybrid systems combine multiple recommendation techniques to improve the accuracy and relevance of suggestions. For example, they may combine content-based filtering and collaborative filtering to leverage the strengths of each approach.

Recommendation systems bring many benefits for both users and businesses. They personalize the user experience by offering content tailored to individual tastes and needs. They help users discover new products or content they may not have otherwise found. By providing relevant recommendations, they increase user engagement and loyalty to the platform. For businesses, well-targeted recommendations can increase sales and revenue by suggesting complementary or alternative products.

Among the different recommendation methods, collaborative filtering is particularly popular and effective. It is divided into two main subcategories: user-based collaborative filtering and item-based collaborative filtering. Each of these methods offers unique perspectives for improving the quality of recommendations.

In the rest of this article, we will explore collaborative filtering in more detail, examining its fundamental principles and different variants.

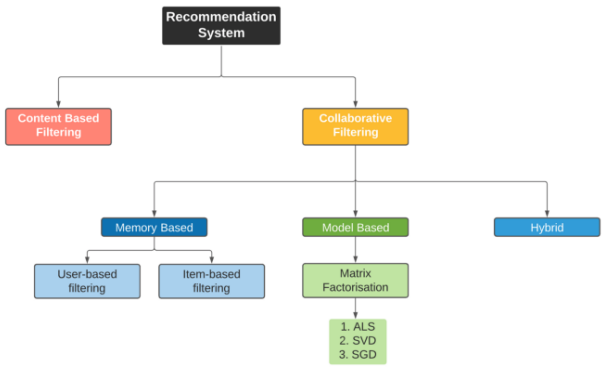

Figure 1. Recommender system

2.2. Collaborative Filtering

Collaborative filtering is an essential method used in recommendation systems to predict user preferences based on past interactions. There are various approaches to implement collaborative filtering, each with its own advantages and disadvantages.

The first approach, memory-based collaborative filtering, relies on the direct storage and manipulation of user-item interaction data. It can be subdivided into two main categories: user-based filtering and item-based filtering. In user-based filtering, a user's preferences are estimated based on those of similar users. For example, if two users have similar interaction histories, they are likely to have similar tastes for new items. Similarly, in item-based filtering, similarities between items are calculated from user ratings, enabling the recommendation of items similar to those already liked by a user.

Next, model-based collaborative filtering involves building a statistical model from user-item interaction data. Algorithms in this approach, such as matrix factorization, are trained on interaction data to predict user ratings for unrated items. This approach is often more robust than memory-based filtering, but it generally requires more computation and resources.

Finally, hybrid collaborative filtering combines different recommendation approaches to improve the accuracy and robustness of recommendations. For example, a hybrid system may combine collaborative filtering with content-based filtering using item feature information to complement user-item interaction data. This approach allows leveraging the strengths of each method while mitigating their respective weaknesses.

Advantages of Collaborative Filtering

- **No Need for Domain Knowledge**: Collaborative filtering dispenses with the necessity for information regarding the items themselves, relying solely on users' interaction data.
- **Personalization**: Recommendations in collaborative filtering are tailored to individual users' tastes, derived from the preferences and behavior of similar users, thereby enhancing user satisfaction and engagement through personalized suggestions.
- **Cold Start Problem Mitigation**: Collaborative filtering effectively addresses the "cold start" issue, wherein new items or users lack substantial interaction history. By leveraging analogous users or items, these algorithms furnish relevant recommendations even in such scenarios.

Challenges of Collaborative Filtering

- **Cold Start Problem**: The absence of adequate data for new users or items hampers accurate recommendations, constituting a significant challenge.

- **Sparsity**: Large-scale systems often exhibit sparse user-item interaction matrices, wherein users rate only a small fraction of available items, posing a challenge to recommendation quality.

- **Scalability**: With the proliferation of users and items, the computational complexity involved in identifying similar users or items escalates, presenting scalability challenges.

Despite these challenges, collaborative filtering remains a widely used and effective technique in recommendation systems due to its simplicity and ability to provide personalized recommendations without needing extensive item metadata.

In the following section, we will delve into matrix factorization techniques, which address some of the limitations of traditional collaborative filtering methods and offer improved performance for recommendation systems.

2.3. Matrix Factorization

Matrix factorization, particularly through the Alternating Least Squares (ALS) method, has emerged as a powerful technique in the field of recommendation systems. This approach is particularly effective for addressing collaborative filtering problems, where the goal is to predict user preferences or ratings for unrated items.

ALS is an optimization technique that seeks to factorize a user-item data matrix into two matrices of latent factors: one for users and one for items. By iteratively adjusting these matrices to minimize the error between predictions and actual ratings, ALS allows for discovering meaningful latent representations of users and items.

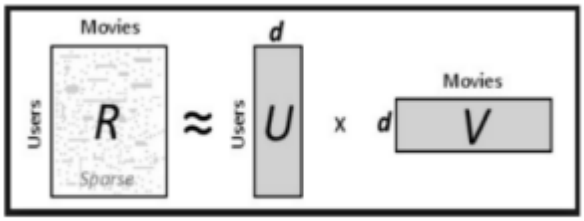

Figure 2. Matrix Factorisation Illustration

In the following sections, we will explore how this method can be extended to account for user and item biases, as well as the challenges and opportunities associated with these extensions.

2.3.1. ALS With Bias Only

When employing ALS (Alternating Least Squares) with bias only, we're incorporating bias terms for users and items to encapsulate their inherent tendencies or preferences, irrespective of specific interactions. These biases, denoted as $b^{(u)}$ for users and $b^{(i)}$ for items, serve as individual offsets, akin to adjusting the baseline rating for each user and item. Conceptually, this approach enables us to account for users' general rating inclinations and items' inherent appeal, thereby refining our predictions by aligning them with the overall trends observed in the data. Geometrically, it's akin to shifting the entire rating scale for each user and item, allowing us to capture their respective biases and better model their influence on ratings without delving into intricate user-item interactions.

Let's consider a rating matrix R where r_{mn} represents the rating given by user m to item n. We seek to approximate this matrix R using bias terms $b^{(u)}$ for users and $b^{(i)}$ for items, knowing that r_{mn} follows $\mathcal{N}(b^{(u)} + b^{(i)}, \lambda^{-1})$.

The objective is to build a model that finds the maximum likelihood estimate for user and item biases. That is, we aim to minimize the function $\mathcal{L}$ defined by:

$$\mathcal{L} = \frac{\lambda}{2} \sum_{m} \sum_{n \in \Omega(m)} (r_{mn} - b_m^{(u)} - b_n^{(i)})^2 + \frac{\gamma}{2}(b_m^{(u)})^2 + \frac{\gamma}{2}(b_n^{(i)})^2 \quad (1)$$

The regularization term γ is added to prevent overfitting.

Here's the process described:

- Initialize user and item biases to zero.
- For each user m, update user biases for $b_m^{(u)}$ while fixing the item biases.

$$b_m^{(u)} = \frac{\lambda \sum_{n \in \Omega(m)} (r_{mn} - b_n^{(i)})}{\lambda |\Omega(m)| + \gamma} \quad (2)$$

- For each item n, update item biases for $b_n^{(i)}$ while fixing the item biases.

$$b_n^{(i)} = \frac{\lambda \sum_{m \in \Omega(n)} (r_{mn} - b_m^{(u)})}{\lambda |\Omega(n)| + \gamma} \quad (3)$$

- Repeat these steps until convergence.

This method optimizes user and item biases to produce robust predictions and is very useful for capturing global

trends. However, it's limited in its ability to model complex interactions between users and items. For more precise recommendations, models that include both biases and latent vectors are often used.

Algorithm 1 ALS with Bias Only

Input:

−Users rating U_{ir} and Items ratings V_{ur}
−M number of users and N number of items
−Hyperparameters λ, γ
−Maximum iterations T

Initialize user bias $b^{(u)}$ with zeros and size M
Initialize item bias $b^{(i)}$ with zeros and size N
Initialize iteration counter $t = 0$
repeat
  for m in $1 \dots M$ **do**
    for (i, r) in $U_{ir}[m]$ **do**
      Compute bias user
    end for
    Update user bias $b_m^{(u)}$ with bias user
  end for
  for n in $1 \dots N$ **do**
    for (u, r) in $V_{ir}[n]$ **do**
      Compute bias item
    end for
    Update item bias $b_n^{(i)}$ with bias item
  end for
  Increment iteration counter: $t \leftarrow t + 1$
until $t \geq T$ or convergence
Output: Learned user bias $b^{(u)}$, item bias $b^{(i)}$

2.3.2. With Bias and Latent Vectors

In the previous section, we explored matrix factorization with biases using the ALS (Alternating Least Squares) algorithm, a powerful technique for modeling user-item interactions in recommendation systems. Now, we will enhance our model by incorporating latent vectors, allowing us to capture more complex relationships between users and items.

Latent vectors are hidden representations of users and items in a reduced-dimensional space, introduced to model interactions that are more intricate than those captured solely by biases. By adding latent vectors, the model is enriched, enabling it to grasp subtle similarities between users and items, even in the absence of explicit shared attributes.

Geometrically, this involves projecting users and items into a latent space of dimension k. In this space, each user m is represented by a k-dimensional feature vector u_m, and each item n is represented by a k-dimensional feature vector v_n. The key idea is that the dot product $u_m^T v_n$ in this latent space approximates the interaction between user m and item n.

To illustrate, consider an example with $k = 2$. Each user and item can be visualized as a point in a two-dimensional space. The closer these points are to each other, the higher the predicted interaction or rating. The direction and magnitude of the vectors u_m and v_n encode specific preferences and characteristics of users and items. For instance, if two users have similar preferences, their vectors u_m will be close to each other in this latent space. Similarly, items with similar attributes will have vectors v_n that are close.

Additionally, the angles between these vectors indicate the type of relationship. For example, if two vectors form a small angle, it suggests that the corresponding user and item are likely to have a high interaction value, i.e., the user will probably rate the item highly. Conversely, a larger angle indicates a lower expected interaction.

This geometric interpretation helps us understand how the model captures latent factors driving user-item interactions. The biases $b_m^{(u)}$ and $b_n^{(i)}$ account for user and item-specific tendencies, such as a user generally rating items higher or an item generally receiving higher ratings. The latent vectors u_m and v_n then capture additional nuances, such as the user's preference for a specific genre or the item's appeal to a particular demographic.

The approximation of the rating matrix R, with r_{mn} following $\mathcal{N}(u_m^T v_n + b_m^{(u)} + b_n^{(i)}, \lambda^{-1})$, allows us to define our objective. The goal is to minimize the regularized loss function $\mathcal{L}$:

$$\mathcal{L} = \frac{\lambda}{2} \sum_m \sum_{n \in \Omega(m)} (r_{mn} - (u_m^T v_n + b_m^{(u)} + b_n^{(i)}))^2 + \frac{\tau}{2} \sum_m u_m^T u_m + \frac{\tau}{2} \sum_n v_n^T v_n + \frac{\gamma}{2} b_m^{(u)2} + \frac{\gamma}{2} b_n^{(i)2} \quad (4)$$

The regularization terms γ and τ are added to prevent overfitting, ensuring that the model generalizes well to unseen data.

Here's the process described:

- Initialize user and item biases to zero.
- Initialize user latent vector with $\mathcal{N}(0, \frac{1}{\sqrt{k}})$ and size (M, k) and item latent vector with $\mathcal{N}(0, \frac{1}{\sqrt{k}})$ and size (k, N) .
- For each user m, update user biases for $b_m^{(u)}$ while

fixing the item biases.

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

- For each user m, update user latent vector for u_m while fixing the item biases.

$$u_m = \left(\lambda \sum_{n \in \Omega(m)} v_n^T v_n + \tau I \right)^{-1} \left(\lambda \sum_{n \in \Omega(m)} v_n (r_{mn} - (b_m^{(u)} - b_n^{(i)}) \right) \quad (6)$$

- For each item n, update movie latent vector for b_n while fixing the user biases.

$$b_n^{(i)} = \frac{\lambda \sum_{m \in \Omega(n)} (r_{mn} - u_m^T v_n + b_m^{(u)})}{\lambda |\Omega(n)| + \gamma} \quad (7)$$

- For each item n, update movie latent vector for v_n while fixing the user biases.

$$v_n = \left(\lambda \sum_{m \in \Omega(n)} u_m^T u_m + \tau I \right)^{-1} \left(\lambda \sum_{m \in \Omega(n)} u_m (r_{mn} - (b_m^{(u)} - b_n^{(i)}) \right) \quad (8)$$

- Repeat these steps until convergence.

The impact of dimension k is significant. By increasing k, we enhance the model's ability to capture complex relationships between users and items. A higher k allows for a finer representation of interactions, distinguishing more finely between user and item preferences and characteristics. However, an excessively high dimension k can lead to overfitting of the model and poor generalization to new data.

Algorithm 2 ALS with Bias combined with latent vectors

Input:

−Users rating U_{ir} and Items ratings V_{ur}
−M number of users and N number of items
−Hyperparameters λ, γ, k
−Maximum iterations T

Initialize user bias $b^{(u)}$ with zeros and size M
Initialize item bias $b^{(i)}$ with zeros and size N
Initialize user vector u with $\mathcal{N}(0, \frac{1}{\sqrt{k}})$ and size (M, k)
Initialize item vector v with $\mathcal{N}(0, \frac{1}{\sqrt{k}})$ and size (k, N)
Initialize iteration counter $t = 0$
repeat
  for m in $1 \dots M$ **do**
    for (i, r) in $U_{ir}[m]$ **do**
      Compute bias user
    end for
    Update user bias $b_m^{(u)}$ with bias user
  end for
  for m in $1 \dots M$ **do**
    for (i, r) in $U_{ir}[m]$ **do**
      Compute latent user vector
    end for
    Update user vector u_m with latent user vector
  end for
  for n in $1 \dots N$ **do**
    for (u, r) in $V_{ir}[n]$ **do**
      Compute bias item
    end for
    Update item bias $b_n^{(i)}$ with bias item
  end for
  for n in $1 \dots N$ **do**
    for (u, r) in $V_{ir}[n]$ **do**
      Compute latent item vector
    end for
    Update item vector v_n with latent item vector
  end for
  Increment iteration counter: $t \leftarrow t + 1$
until $t \geq T$ or convergence
Output: Learned user vector u_m, user bias $b^{(u)}$, item vector v_n, item bias $b^{(i)}$

2.3.3. ADDING FEATURES

When an item has no interactions with any users, it poses a significant challenge for recommending that item, commonly known as the cold start problem. To mitigate this issue, we will consider that each item has associated features, and the model will incorporate these features in the recommendation process. More concretely, we aim to obtain an item vector v_n given its features f_i. This can be

expressed as

$$v_n \sim \mathcal{N}\left(\frac{1}{\sqrt{F_n}} \sum_{i \in \mathcal{F}(n)} f_i, \tau I\right)$$

and

$$f_i \sim \mathcal{N}(0, \beta I).$$

Regarding the algorithm, the inclusion of a new component f_i impacts the expression for updating the item vector v_n:

$$\begin{aligned} v_n &= \left(\lambda \sum_{m \in \Omega(n)} u_m u_m^T + \tau I\right)^{-1} \\ &\times \left(\lambda \sum_{m \in \Omega(n)} (r_{mn} - b_m^{(u)} - b_n^{(i)}) u_m + \frac{\tau}{\sqrt{F_n}} \sum_{i \in \mathcal{F}(n)} f_i\right), \\ f_i &= \left(\sum_{n \in \mathcal{M}_i} \frac{\tau}{\sqrt{F_n}} + \tau_f\right)^{-1} \left(\tau \sum_{n \in \mathcal{M}_i} v_n\right). \end{aligned} \tag{9}$$

By incorporating the features of the items into our model, we aim to improve recommendations for items with few or no interactions by emphasizing their intrinsic attributes.

2.4. Evaluation

Evaluating recommendation systems is a crucial step to measure their performance and effectiveness in providing relevant recommendations. Among the various metrics used, the Root Mean Square Error (RMSE) is one of the most popular and widely adopted.

RMSE (10) is a statistical measure of the difference between the values predicted by the recommendation model and the actual observed values. Specifically, it evaluates the discrepancy between the ratings predicted by the recommendation system and the actual ratings given by users. RMSE can be used to tune hyperparameters and select the best model by minimizing the error between predicted ratings and actual ratings. The formula for RMSE is as follows:

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (r_{ui} - \hat{r}_{ui})^2} \tag{10}$$

For ALS with bias only:

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} \left(r_{ui} - (b_m^{(u)} + b_n^{(i)})\right)^2} \tag{11}$$

And for ALS with bias combined with latent vectors:

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} \left(r_{ui} - (u_m^T v_n + b_m^{(u)} + b_n^{(i)})\right)^2} \tag{12}$$

3. Methods

3.1. Data

3.1.1. Presentation

To evaluate the performance of the matrix factorization models without and with bias, a simulation study is performed. For this study, we focus on **movie recommendations**, utilizing the MovieLens dataset (ml-25m). This dataset is commonly used for testing recommender systems and contains rating data collected from the MovieLens website for research purposes between January 09, 1995, and November 21, 2019.

3.1.2. Dataset description : MovieLens

The MovieLens dataset (ml-25m) describes 5-star rating and free-text tagging activity from MovieLens, a movie recommendation service. It contains 25,000,095 ratings for 62,423 movies from 162,541 users. Users were selected at random for inclusion, with all selected users having rated at least 20 movies. No demographic information is included; each user is represented by an ID, and no other personal information is provided.

The data are contained in the files genome-scores.csv, genome-tags.csv, links.csv, movies.csv, ratings.csv, and tags.csv. More details about the contents and use of all these files follow.

Constructing the origin matrix of the MovieLens dataset (ml-25m) with 162,541 users and 62,423 movies results in a matrix with 10,146,296,843 elements. Given that we have 25,000,095 ratings, the sparsity (13) of this matrix is 99.75%.

$$\text{Sparsity} = 1 - \frac{\text{Non-zero elements}}{\text{Total number of elements}} \tag{13}$$

3.1.3. Specificity

Each movie belongs to one or more of 20 different categories (Figure 3). The movie with the least number of ratings is *Most Wanted (1997)* with 16 ratings, averaging 2.28. The movie with the most number of ratings is *Forrest Gump (1994)* with 81,491 ratings, averaging 4.04. The smallest rating for a movie is 0.5, and the highest rating is 5.0, while the mean rating is 3.53. The smallest number of ratings by a user is 20, and the highest number is 32,202.

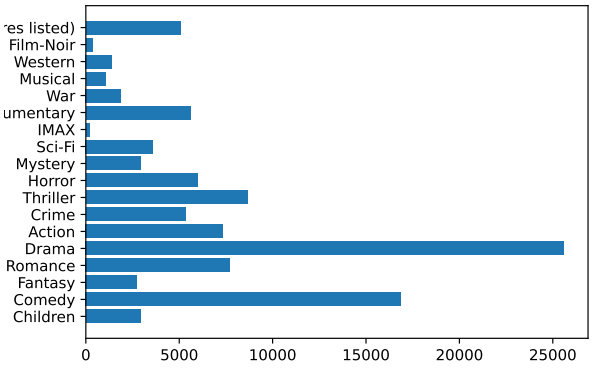

Figure 3. Movies category Distribution

3.1.4. Power-Law Distribution

The distribution of ratings in the MovieLens dataset follows a power-law distribution, which is characterized by a small number of movies and users receiving a large number of ratings, while the majority receive relatively few. This type of distribution is common in many real-world networks and datasets, where a few items or users dominate in terms of frequency or popularity.

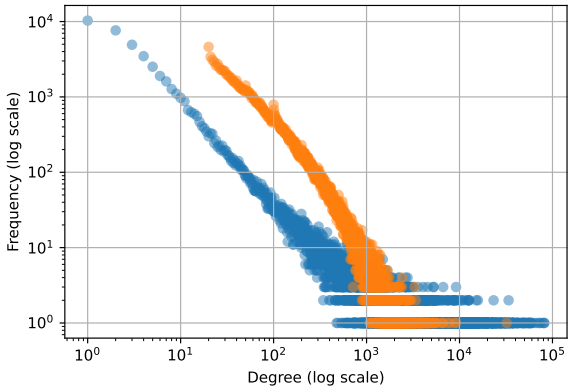

Figure 4. Power-Law Distribution of Ratings

Figure 4 illustrates the power-law distribution of ratings, where both the degree (number of ratings per movie and number of ratings per user) and the frequency of those degrees are plotted on a log-log scale. This visualization confirms that a small number of movies receive an exceptionally high number of ratings(orange curve) (a small number of users contributing a large number of ratings (blue curve)), which significantly influences the dataset's overall distribution. Understanding this distribution is crucial for developing and evaluating recommender systems, as it affects how algorithms predict and recommend items to users.

Statistic	Value
Number of users	162,541
Number of movies	62,423
Number of ratings	25,000,095
Least ratings by a user	20
Most ratings by a user	32,202
Lowest rating for a movie	0.5
Highest rating for a movie	5.0
Least ratings for a movie	16
Most ratings for a movie	81,491
Movie with the most ratings	*Forrest Gump (1994)*
Movie with the least ratings	*Most Wanted (1997)*

Table 1. Descriptive statistics of the MovieLens dataset.

3.2. Training and Evaluation

To utilize this dataset, several transformations were conducted to create data structures that are easy to manipulate. The goal is to create two data structures: a list of lists of tuples. These structures respectively represent users with the different movies they have rated along with their ratings, and movies with the different users who have rated them and the ratings given.

Regarding the training of different models, we split the dataset into training data (90%) and test data (10%). The objective is to minimize the cost function, and each model was evaluated using RMSE (Root Mean Square Error).

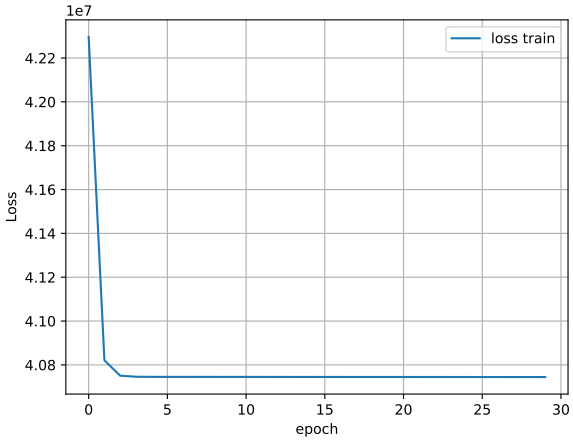

Figure 5. Loss Evolution ALS bias only

We observed that for our model with bias only, the RMSE curve does not decrease significantly and remains

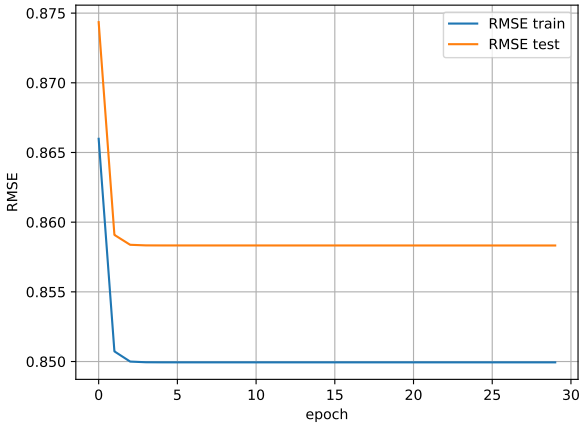

Figure 6. RMSE Evolution ALS bias only

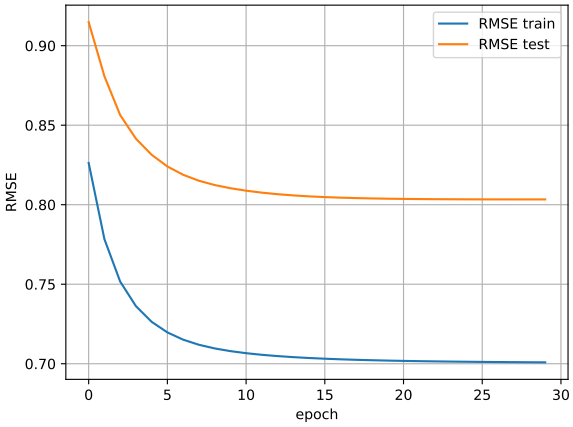

Figure 8. RMSE Evolution ALS bias + Latent vector

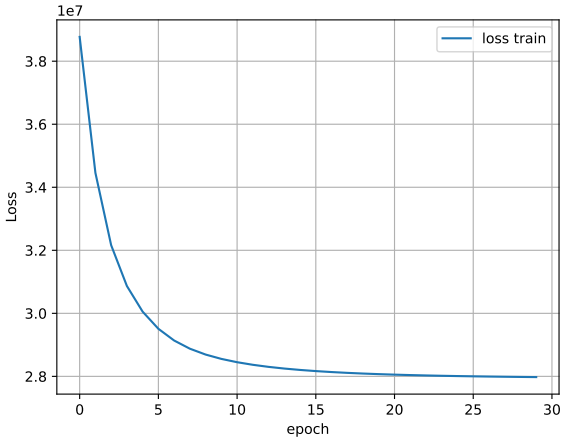

Figure 7. RMSE Evolution ALS bias + Latent vector

relatively constant after a few iterations. This indicates that the model fails to capture complex relationships within the data. In contrast, with our other model, we see a more substantial improvement in the RMSE curve, which demonstrates the enhanced ability of these models to capture the nuances of user preferences.

4. Resultats

We present here the different results of our model's predictions. We create a new user who will rate a certain number of movies, and we will conduct a study to better understand these results.

We choose the movie "Lord of the Rings" and give it the highest rating, which is 5. The prediction process is as follows: using our model, we obtain a latent vector for the user and a bias vector for this same user. We then combine these with our item latent vectors and item biases to get predictions for each movie. We then retrieve the movies with the highest ratings, which constitute our predictions. The relationship used to obtain these scores is as follows:

$$score\_for\_item[n] = u_{new} \cdot v_n + fact \times b_n^{(i)} + b_{new}^{(u)} \tag{14}$$

The term $fact$ in the equation represents a customization factor according to the user's preferences. It is there to discriminate against global trends. Empirically, the value is 0.05, but if we want to prioritize global trends by default, it is equal to 1.

The expected result here is, of course, movies of the same category as this one, namely (Action—Adventure—Drama—Fantasy). The table 2 shows the different predictions of our model.

To visualize our embeddings, we used a dimensionality reduction technique called PCA, which allows us to move from a k-dimensional representation (k=10) to a 2-dimensional representation. This enables us to visualize our embeddings accurately, further reinforcing the analysis that the model can capture the different specificities of the movies.

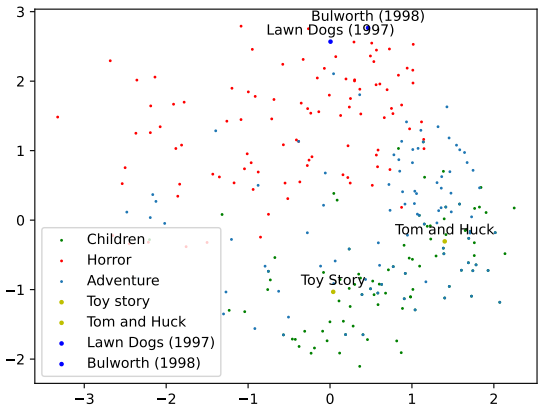

Figure 9. Visualisation of our embedding with feature embedding

Software and Deployment

To bring our results to life, we have decided to design a web application that provides a more immersive visual experience. The application is written in Python and deployed on the Streamlit server.

Use Case

- **Search for a Movie**: The user can enter a movie title and view a summary of the movie along with its category.
- **Provide Ratings**: The user can rate movies they have enjoyed.
- **Visualize Ratings**: The user can view a list of the movies they have rated.
- **Delete Ratings**: The user can delete the ratings they have given to movies.
- **Get Recommendations**: The user can receive movie recommendations based on the movies they have rated.

Accessibility

Link to the web app: Click here to access

Link to the GitHub repository: GitHub repository

Conclusion

In conclusion, our study delves into the nuanced distinctions between two approaches: one incorporating solely biases and the other amalgamating both biases and latent vectors within the framework of the Alternating Least Squares (ALS) algorithm for matrix factorization in collaborative filtering. Our analysis, conducted using the MovieLens dataset, unequivocally demonstrates the profound impact of latent vectors on the precision and pertinence of movie recommendations.

These findings underscore the paramount significance of strategic decision-making in crafting efficacious recommendation systems rooted in matrix factorization for collaborative filtering. By offering these insights, our study seeks to steer researchers and practitioners towards more resilient and pertinent methodologies, adept at meeting the dynamic needs of users in the realm of content recommendations.

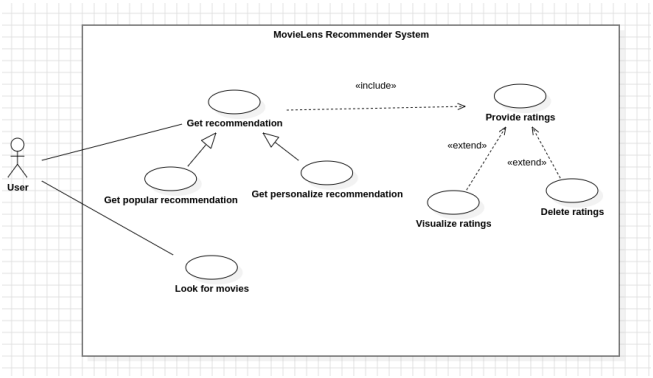

Figure 10. Use case

Figure 11. Scan me

References

[1] Bell, R. M. and Koren, Y. (2007). Lessons from the Netflix Prize Challenge. *ACM Sigkdd Explorations Newsletter*, 9(2), 75–79.

[2] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by Latent Semantic Analysis. *Journal of the American Society for Information Science*, 41(6), 391.

[3] Eckart, C. and Young, G. (1936). The Approximation of One Matrix by Another of Lower Rank. *Psychometrika*, 1(3), 211–218.

[4] Jannach, D., Zanker, M., Felfernig, A., and Friedrich, G. (2010). Recommender Systems: An Introduction. Cambridge University Press.

[5] Jing, L., Wang, P., and Yang, L. (2015). Sparse Probabilistic Matrix Factorization by Laplace Distribution for Collaborative Filtering. In *IJCAI*, pp. 1771–1777.

[6] Koren, Y. (2010a). Collaborative Filtering with Temporal Dynamics. *Communications of the ACM*, 53(4), 89–97.

[7] Koren, Y. (2010b). Factor in the Neighbors: Scalable and Accurate Collaborative Filtering. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 4(1), 1.

[8] Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix Factorization Techniques for Recommender Systems.

Table 2. Prediction

MOVIES	GENRES
LORD OF THE RINGS: THE RETURN OF THE KING, THE (2003)	ACTION—ADVENTURE—DRAMA—FANTASY
LORD OF THE RINGS: THE FELLOWSHIP OF THE RING, THE (2001)	ADVENTURE—FANTASY
LORD OF THE RINGS: THE TWO TOWERS, THE (2002)	ADVENTURE—FANTASY
STAR WARS: EPISODE III - REVENGE OF THE SITH (2005)	ACTION—ADVENTURE—SCI-FI
STAR WARS: EPISODE IV - A NEW HOPE (1977)	ACTION—ADVENTURE—SCI-FI
GRATEFUL DEAD MOVIE, THE (1977)	DOCUMENTARY—MUSICAL
STAR WARS: EPISODE V - THE EMPIRE STRIKES BACK (1980)	ACTION—ADVENTURE—SCI-FI
STAR WARS: EPISODE II - ATTACK OF THE CLONES (2002)	ACTION—ADVENTURE—SCI-FI—IMAX
HOBBIT: AN UNEXPECTED JOURNEY, THE (2012)	ADVENTURE—FANTASY—IMAX
HOBBIT: THE DESOLATION OF SMAUG, THE (2013)	ADVENTURE—FANTASY—IMAX
JAPON (A.K.A. JAPAN) (JAPÓN) (2002)	DRAMA
LAST OF THE MOHICANS, THE (1920)	ADVENTURE—DRAMA
HORACE AND PETE (2016)	COMEDY—DRAMA
BEYOND (2003)	ANIMATION—SCI-FI
STAR WARS: EPISODE VI - RETURN OF THE JEDI (1983)	ACTION—ADVENTURE—SCI-FI
BRANDED TO KILL (KOROSHI NO RAKUIN) (1967)	ACTION—CRIME—DRAMA