Matrix Factorization in Action

Herman F. Tesso Tassang [1]

Abstract

Matching consumers with the most suitable products is crucial for enhancing user satisfaction and loyalty on various online platforms, such as e-commerce websites, streaming services, and social media. Recommender systems solve this problem by analyzing user interest patterns in products to provide users with personalized content and services. In this paper, we will gradually build our own prototype of a recommender system based on the Matrix Factorization Technique, using the Movieslens dataset with explicit feedback.

1. Introduction

During the last few decades, the demand for recommender(Lü et al., 2012) systems has increased significantly with the rise of platforms like YouTube, Amazon, Netflix, and many others(Koenigstein et al., 2012). Recommender systems are algorithms that filter out relevant information from a large amount of dynamically generated data, based on user preferences, interests, or observed behaviors.

The purpose of recommender systems is to suggest relevant items to users. There are two major paradigms for achieving this task. The first is the "content-based" approach(Schafer et al., 2007a), which creates profiles for each user or product and builds a model based on the available "**features**" that explain the observed between users and items. The program then uses these profiles to associate users with products.

The second approach is called "collaborative filtering,"(Schafer et al., 2007b) which is solely based on past interactions recorded between users and items. The recommendations are stored in the so called "*user-item interactions matrix*". The idea behind this approach is that past user-item interactions are sufficient to detect similar users and/or items and make predictions based on this estimated proximity. There are two primary classes of collaborative filtering: **memory-based** methods and **model-based methods**.

Memory-based methods work directly with the values of recorded interactions without assuming a model. They essentially rely on nearest neighbor search(e.g find closest users from user of interest and suggest the most popular items among these neighbours). On the other hand, model-based methods such as matrix factorization assume an underlying "generative" model that explains the user-item interactions and try to discover this model in order to make new predictions.

Some systems prompt users to provide ratings for items through the system interface, in order to construct and improve their models. These ratings are referred to as *"explicit user feedback*." Because it requires effort from users, explicit feedback usually comprises a sparse matrix, as each user is likely to have rated only a small percentage of possible items.

In this work, we will focus on building a recommender system using the matrix factorization approach, specifically with highly explicit feedback (ratings). The first section will discuss related work on recommender systems. The next two sections will describe the data used and the theoretical basis of matrix factorization(Koren et al., 2009), as well as the modeling process. The following two sections will be dedicated to the training, validation procedure, and recommendation process. Finally, we will discuss how to modify the model to cope with the "cold start" problem.

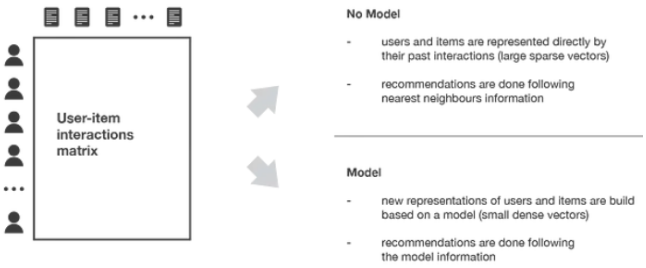


Figure 1. Overview of the Collaborative filtering paradigm

2. Data

2.1. Data Description

We will be using the MoviesLens datasets in our work. These datasets are widely recognized collections of movie ratings data provided by the GroupLens (University of Minnesota) . They are commonly used for research in recommendation systems and contain user ratings, movie meta-

data, and demographic information. This data is valuable for analyzing user preferences and developing recommendation algorithms. We will primarily focus on the 25M MovieLens dataset, which has the following characteristics:

- The dataset contains 25 million 5-star ratings.
- There are M=162,541 users and N=59,047 movies.
- Every user has rated at least 20 movies.
- each user has rated at least 20 movies.
- The dataset includes information about the movies, such as ID, title, and genres (19 genres are listed).
- There is no demographic information available.

2.2. Overall Trends in the Data

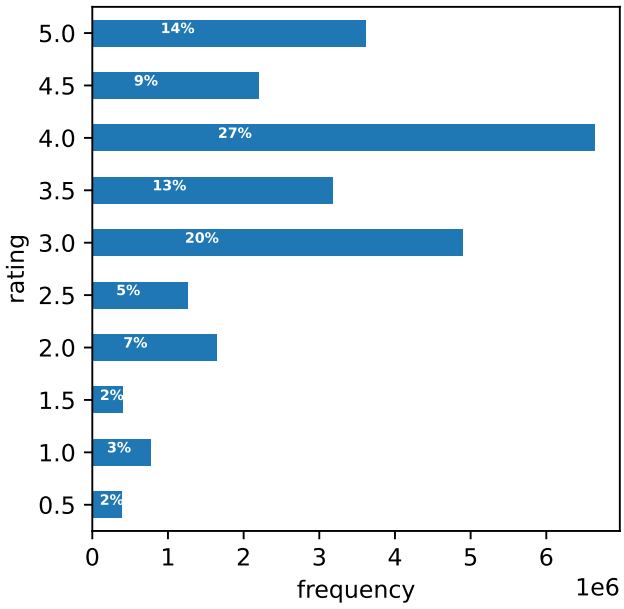


Figure 2. Rating distributions of the data

Figure 2 displays the rating distribution in the 25M MovieLens dataset. A noticeable pattern emerges where the majority of movies receive higher ratings. More specifically, ratings commonly fall within the range of 3 to 4 stars, suggesting that users generally rate movies positively. However, there is a long tail of movies with fewer ratings, encompassing both highly-rated and poorly-rated films. These observations underscore the importance of considering both movie popularity and individual user preferences when constructing our recommender system.

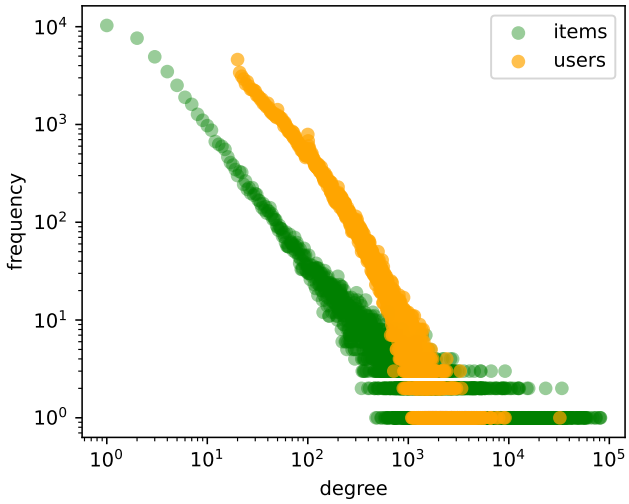


Figure 3. Rating distributions given movies/users

Figure 3 shows the distribution of ratings for movies and users. This distribution clearly follows a power-law pattern. It is observed that a small number of movies receive a disproportionately larger number of ratings compared to the majority of movies. Similarly, a small number of users have rated a large number of movies. This plot further emphasizes the significance of considering popularity bias in the following analysis.

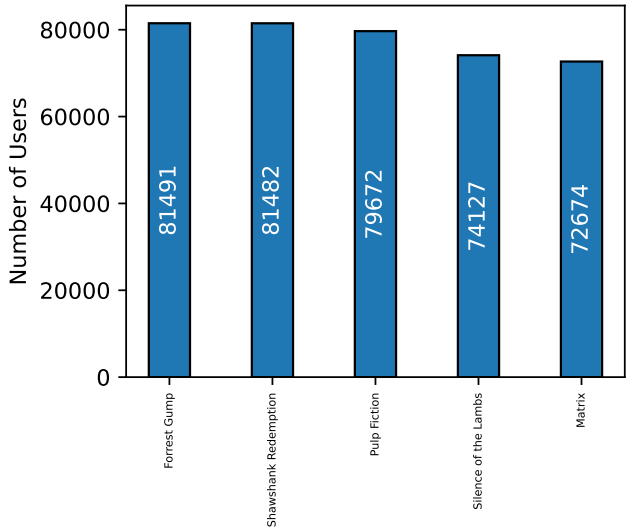


Figure 4. top popular movies

The last figure, **Figure 4** , represents the five most popular movies in our dataset. It shows the frequency of user ratings for each movie.

2.3. Data Structure

We used the dataset to create the necessary data structures for building our recommender model.

First, we mapped each user and item ID to an index, denoted by $1 \leq m \leq M$ and $1 \leq n \leq N$ respectively. We also stored the rating given by each user (indexed by m) for each movie (indexed by n) as "r_{mn}". Finally, we created the following tree data structures.

The main list consists of multiple lists, each containing tuples. Each list within the main list is indexed by m, where m is a value between 1 and M. These lists contain the indexes of movies that have been rated by the user indexed by m, along with their corresponding ratings in the form of (item, rating) pairs.

samples list : list of tuples of the form (m, n, r_{mn}) , $user - item - rating$.

data index by user: a list of lists. The main list consists of multiple lists, each containing tuples. Each list within the main list is indexed by m ($1 \leq m \leq M$). These lists contain the indexes of movies that have been rated by the user indexed by m, along with their corresponding ratings in the form of $(item, rating)$ pairs.

data index by movie: a list of lists.

The main list consists of a series of sublists, each containing tuples. Each sublist is indexed by n ($1 \leq n \leq N$) and contains the indices of the users who have rated the movie indexed by n, along with their corresponding ratings as pairs $(user, rating)$.

3. Model Building

3.1. Matrix Factorization

Matrix factorization is a model-based collaborative filtering approach that relies only on *user-item interaction* information. It assume a latent model supposed to explains these interactions. The algorithm consist in decomposing the huge and sparse user-item interaction matrix into a product of two smaller and dense matrices: a *"user-factor matrix"* (containing user representations) that multiplies a *factor-item matrix* (containing item representations).

The main assumption is that there is a low-dimensional latent space of features in which we can represent both users and items. In this space, the interaction between a user and an item can be obtained by computing the dot product of their corresponding dense vectors. However, we do not explicitly provide these features to our model, as can be done in content-based approaches (1). Instead, we allow the system to discover these useful features on its own and create its own representations of both users and items.

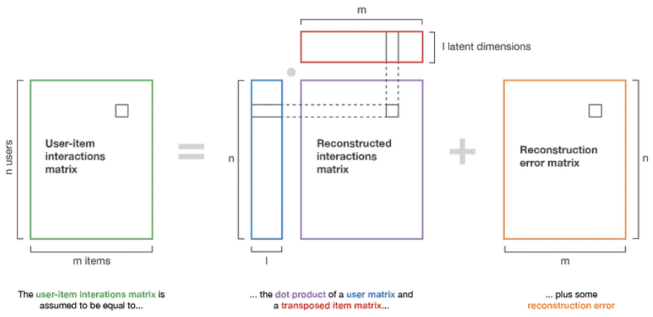


Figure 5. Graphic represention : Matrix factorization

3.2. Mathematical Formulation

Let's consider an interaction matrix $R \in \mathbb{R}^{M \times N}$ of ratings where only some items have been rated by each user (most interactions are set to $None$ to express the lack of rating). We want to factorize that matrix such that:

$$R \approx UV^T \tag{1}$$

However,since most of the observed variation in rating values is due to effects associated with either users or items known as biases independent of any actions; it would be unwise to explain the full rating value by only interaction of the form UV^T. Instead, equation (1) is reformulated as:

$$R \approx UV^T + b^{(u)} + b^{(i)} \tag{2}$$

where:

- $U \in \mathbb{R}^{M \times K}$ is the "*user matrix*" whose rows represent the M users.
- $V \in \mathbb{R}^{N \times K}$ is the "*item matrix*" whose row represent the N items.
- $user_m \equiv u_m$ user latent vector $\forall m \in \{1, ..., M\}$.
- $item_n \equiv v_n$ item latent vector $\forall n \in \{1, ..., N\}$.
- K is the dimension of the latent space
- $b^{(u)}$ *and* $b^{(i)}$ are user and vector biases respectively.

this lead to the estimate:

$$\hat{r}_{mn} = u_m^T v_n + b_m^{(u)} + b_n^{(i)} \tag{3}$$

Therefore, we model it as:

$$\mathcal{P}\left(r_{mn} | u_m, v_n, b_m^{(u)}, b_n^{(i)}\right) = \mathcal{N}\left(r_{mn}; u_m^T v_n + b_m^{(u)} + b_n^{(i)}, \lambda^{-1}\right) \tag{4}$$

with:

- λ the precision factor
- $\mathcal{P}(u_m) = \mathcal{N}\big(u_m; 0, \tau\mathbf{I}\big)$, $\mathcal{P}(v_m) = \mathcal{N}\big(v_n; 0, \tau\mathbf{I}\big)$ priors for u_m and v_n respectively.
- $\mathcal{P}(b^{(u)}) = \mathcal{N}\big(b^{(u)}; 0, \gamma\mathbf{I}\big)$, $\mathcal{P}(b^{(i)}) = \mathcal{N}\big(b^{(i)}; 0, \gamma\mathbf{I}\big)$ priors for $b^{(u)}$ and $b^{(i)}$ respectively.

The objective function:

We want to find $(U, V, b^{(u)}, b^{(i)})$ that maximize the regularized log likelihood:

$$\begin{aligned}\mathcal{L} =& \log \mathcal{P}\big(R|U, V, b^{(u)}, b^{(i)}\big) + \log \mathcal{P}\big(U\big) \\ &+ \log \mathcal{P}\big(V\big) + \log \mathcal{P}\big(b^{(u)}\big) + \log \mathcal{P}\big(b^{(i)}\big) \\ =& -\frac{\lambda}{2}\sum_{m}\sum_{n\in\Omega(m)}\Big(r_{mn} - (u_m^T v_n + b_m^{(u)} + b_n^{(i)})\Big)^2 \\ &- \frac{\tau}{2}\sum_{m} u_m^T u_m - \frac{\tau}{2}\sum_{n} v_m^T v_m \\ &- \frac{\gamma}{2}\sum_{m} b_m^{(u)2} - \frac{\gamma}{2}\sum_{n} b_n^{(i)2}\end{aligned} \tag{5}$$

Thus,

$$(U, V, b^{(u)}, b^{(i)}) = \underset{U,V,b^{(u)},b^{(i)}}{argmin}\ \big\{-\mathcal{L}(U, V, b^{(u)}, b^{(i)})\big\} \tag{6}$$

3.3. Learning Algorithm

The approach used to minimize equation (6) is called alternating least squares (ALS). The underlying idea is that even though equation (6) is not convex, we can solve the optimization problem optimally by fixing one unknown at a time.

Specifically, for a fixed value of m, we compute the partial derivatives $\frac{\partial\mathcal{L}}{\partial u_m}$, and $\frac{\partial\mathcal{L}}{\partial b_m^{(u)}}$ and set them to zero. This allows us to solve for the optimal value (vector) of the quadratic problem. As a result, we obtain the following updates:

User vector update:

$$u_m = \lambda\Big(\sum_{n\in\Omega(m)} v_n v_n^T + \tau\mathbb{I}\Big)^{-1}\Big(\lambda\sum_{n\in\Omega(m)} v_n(r_{mn} - b_m^{(u)} - b_n^{(i)})\Big) \tag{7}$$

User bias update:

$$b_m^{(u)} = \frac{\lambda\sum_{n\in\Omega(m)}\Big(r_{mn} - (u_m^T v_n + b_n^{(i)})\Big)}{\lambda\Omega(m) + \gamma} \tag{8}$$

as previously, with fixed value of n, by computing the partial derivatives $\frac{\partial\mathcal{L}}{\partial v_n}$, and $\frac{\partial\mathcal{L}}{\partial b_n^{(i)}}$ and setting them to zero, we obtain the following updates for the items:

Item vector update:

$$v_n = \lambda\Big(\sum_{m\in\Omega(n)} u_m u_m^T + \tau\mathbb{I}\Big)^{-1}\Big(\lambda\sum_{m\in\Omega(n)} u_m(r_{mn} - b_m^{(u)} - b_n^{(i)})\Big) \tag{9}$$

Item bias update:

$$b_n^{(i)} = \frac{\lambda\sum_{m\in\Omega(n)}\Big(r_{mn} - (u_m^T v_n + b_m^{(u)})\Big)}{\lambda\Omega(n) + \gamma} \tag{10}$$

Note: $\Omega(m)$ *represents the set of movie indexes that has been rated by the user indexed by* m*.Similarly,* $\Omega(n)$ *is the set of user indexes that have rated the movie indexed by* n*.*

Algorithm 1 Alternating Least Square (ALS)

```
Input: data-structured, K , λ, τ, γ
Initialize U, V, b^(u), b^(i)
repeat
  for m = 0 to M − 1 do
    for (n,r) in data-index-by-user[m] do
      update user vector U[m :] with equation (7)
      update user bias b_m^(u) with equation (8)
    end for
  end for
  for n = 0 to N − 1 do
    for (m,r) in data-index-by-item[n] do
      update item vector V[n :] with equation (9)
      update item bias b_n^(i) with equation (10)
    end for
  end for
until convergence
```

4. Training and validation

4.1. experiments

In this subsection, we will experiment the optimization algorithm using a smaller MoviesLens dataset : the 100K MoviesLens dataset consist of 100,000 ratings and 9000 movies rated by 600 users.

This step is perform in order to achieve the following goals:

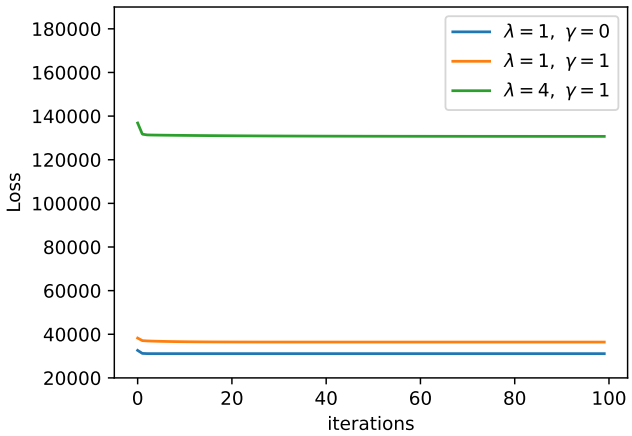
1) Quickly compare the reliability of the model that only involves biases to explain the ratings $\hat{r}_{mn} = b_m^{(u)} + b_n^{(i)}$ with our original model which also includes the latent trait vectors (equation (3)).
2) Have and overview of how hyperparameters K, γ, λ and τ can affect the overall performance of the model.

The performance of our models is assessed based on the **RMSE**:

$$\sqrt{\frac{1}{L}\sum_{m=1}^{M}\sum_{n\in\Omega(m)}\left(r_{mn}-(u_m^T v_n+b_m^{(u)}+b_n^{(i)})\right)^2}$$

with, L the total number of samples.

Experiment procedure: run the **ALS** algorithm (equation 1 with varying values of hyperparameters, record both the $Loss$ and $RMSE$ values at each iteration and use them to create plots representing the training evolution.



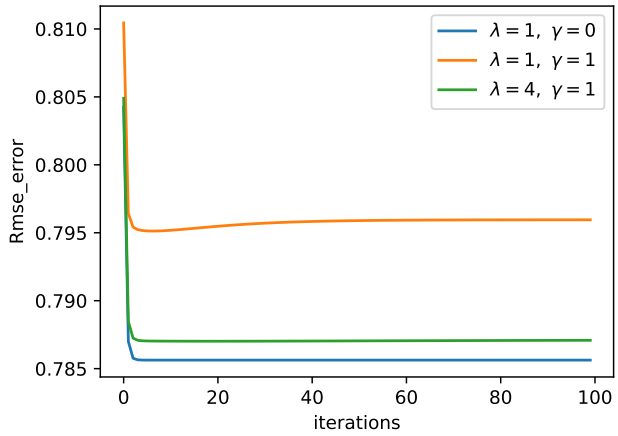
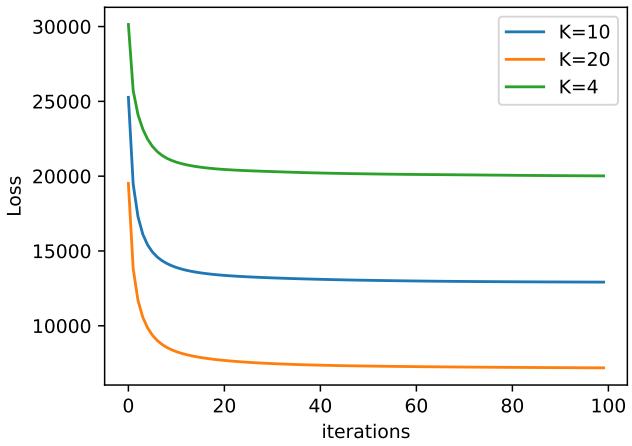


Figure 6. Training history for user+item biases update only on the 100K dataset, $\hat{r}_{mn} = b_m^{(u)} + b_n^{(i)}$.



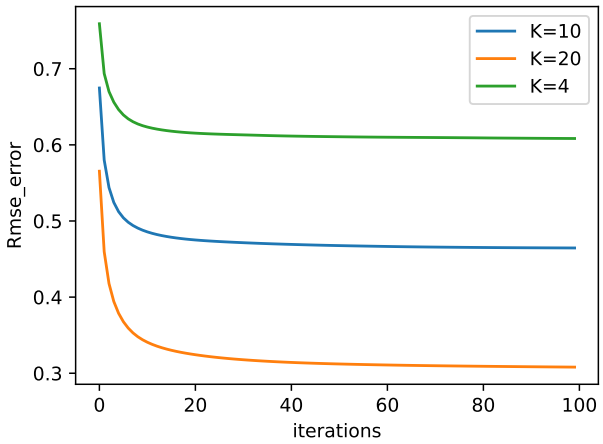


Figure 7. Training history for user+item biases+ latent vectors (equation (3)) on the 100K dataset.

.

Discussion:

Based on this small experiment, it appears that the models that incorporate latent factors show improvement compared to the models that rely solely on user and item biases. Different values of hyperparameters lead to significant differences in the performance of the models, as measured by RMSE and overall loss values. Specifically, it seems that for the model that includes user interaction (**Figure 7**), increasing the dimension of the latent space K results in a more flexible model. These observations emphasize the importance of carefully selecting the hyperparameters for training on the full 25M ratings dataset in order to achieve an effective model.

4.2. Training on the 25M MoviesLens Dataset.

Before training the model on the entire 25M dataset, it is necessary to determine which hyperparameters work well,

as discussed previously. These hyperparameters should allow for achieving good performance without overtraining the data.

To accomplish this, the data will be split into a training set and a test set. The training set will be used for the fitting procedure, optimizing the parameters, while the test set will be used to check for overfitting by visualizing the recorded train/test loss and RMSE over training iterations.

4.3. Splitting Strategy

To split the data, all the samples have been shuffled before being divided into 80% for the training set and 20% for the test set. The data structures, as described previously (see 2.3) have been reconstructed from the training set.Consequently, both the user and item data structures might contain empty lists, but that's also fine.

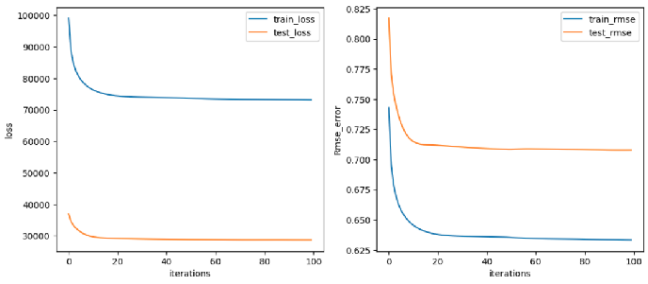


Figure 8. Training/test loss and RMSE history on the full 25 million ratings

4.4. Choose of hyperparameters

Several models have been trained using various values of K, λ, γ *and* tau. To evaluate each model, we plot the loss and RMSE history on the test set in order to determine if there is any overfitting occurring.

Figure 8 shows the obtained results of some trials.

After conducting multiple trials, we observed that overfitting occurs when K is set to 20 or higher. Additionally, we discovered that models with a latent dimension of $K = 10$ generally perform well, as they converge to lower test RMSE values, as long as reasonable values of K, λ, γ *and* τ are used. Based on these findings, we have selected the following hyperparameters for the entire 25M dataset.

$$K = 10, \quad \lambda = 1, \quad \gamma = 0.1, \quad \tau = 1$$

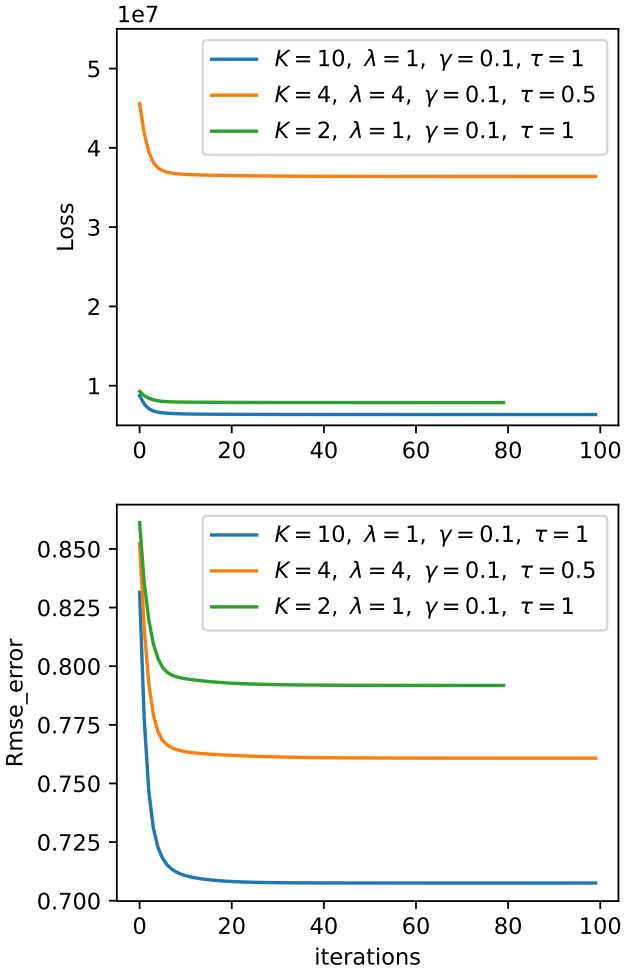


Figure 9. Errors on the test set for different values of the hyperparameters

5. Recommendations

The optimization step for all 25 million ratings has been completed. As a result, we now have access to the learning representations of both the users and items i.e optimized latent vectors as well as the optimized users and items biases. However, before we proceed, let's take a look into the latent space.

5.1. Item embeddings

In this subsection, we consider the model characterised by,

$$K = 2,\ \ \lambda = 1,\ \ \gamma = 0.1,\ \ \tau = 1,\ (*)\ \ \ (figure(9))$$

whose the dimension of the latent space is 2. The goal is to be able to visualize the item trait vectors v_n in this latent space. We anticipate that similar movies will be embedded together and that opposite movies will be embedded in opposite parts of the space.

The plot **fig9** show the embeddings of *"Children's"* and *"Horror"* genre movies.

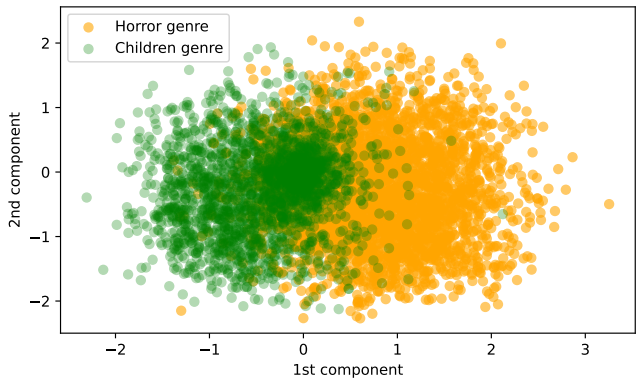


Figure 10. 2D item (movies) embeddings

5.2. Polirizing Movies

In recommender system, polarizing movies are those that generate extreme ratings from users. These movies receive a significant number of very high ratings from some users and very low ratings from others, with few ratings falling in between. Some users will rate them very highly and want similar recommendations, while others will rate them very poorly and want the opposite.Consequently, they quickly narrow down a user's taste once you know whither they like or dislike them.

It turn out that identifying polarizing movies amounts to search for movies with unusually long vectors. Mathematically this involves extracting movies whose latent vectors have the highest Euclidean norm values $\|v_n\|$.

Remember that the dimension of the latent space we used for training on the full 25 million ratings is $K = 10$ (see (*)). First, the movie data has been filtered by removing those that have received less than 50 ratings. Then, the five most polarizing movies have been extracted based on this rule.

title	genres
Twilight (2008)	Drama|Fantasy|Romance|Thriller
Twilight Saga: New Moon, The (2009)	Drama|Fantasy|Horror|Romance|Thriller
Twilight Saga: Eclipse, The (2010)	Fantasy|Romance|Thriller|IMAX
Twilight Saga: Breaking Dawn - Part 1, The (2011)	Adventure|Drama|Fantasy|Romance
Twilight Saga: Breaking Dawn - Part 2, The (2012)	Adventure|Drama|Fantasy|Romance|IMAX

Figure 11. Top most polarizing movies in the dataset

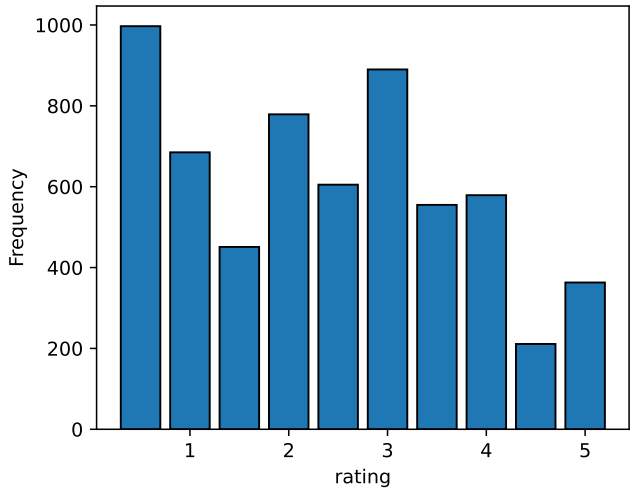


Figure 12. Rating distributions of the most polarizing movie:*"Twilight Saga: New Moon"*

The less polarizing movies are those that tend to have high popularity indicating that they are well-liked by a large number of users. The followings are the 5 less polarizing movies in the dataset:

title	genres
Pulp Fiction (1994)	Comedy|Crime|Drama|Thriller
Shawshank Redemption, The (1994)	Crime|Drama
Forrest Gump (1994)	Comedy|Drama|Romance|War
Silence of the Lambs, The (1991)	Crime|Horror|Thriller
Matrix, The (1999)	Action|Sci-Fi|Thriller

Figure 13. Top less polarizing movies

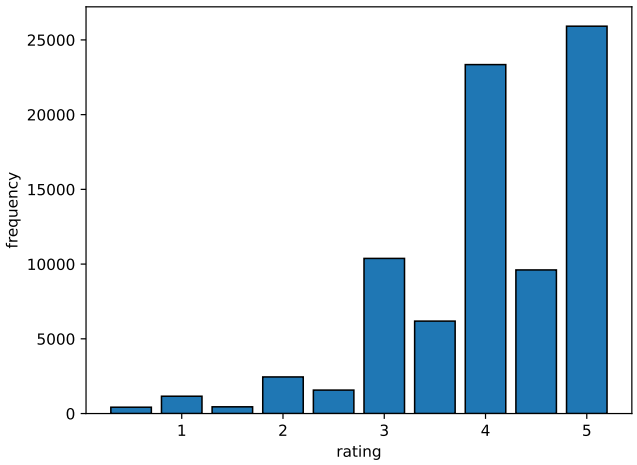


Figure 14. Rating distributionsof the less polarizing movie:*"Forrest Gump"*

5.3. Recommendation Process

We designed this model with the specific goal of providing personalized content to users based on their ratings of certain movies. The recommendations are generated based on a kind of *user recommendation score* that is not influenced by the user's bias.

Workflow for movies recommendations to a user:

1) Learn the latent vector of this user using *ALS* algorithm (see 1) and the optimized item matrix V of the underlying model.

2) Compute the user's score for the movie "n" according to the formula:

$$Score_{item_n} = u_{user}^T v_n + \alpha \times b_n^{(i)}, \ \ \forall n \in \{1, ..., N\}$$

where:

- u_{user} is the learned user latent vector of the user
- α is a factor use to downplay the contribution of item bias to cope with the movie popularity and/or high bias effect.

3) Finally, the scores are ranked and the movies associated with highest score sit at the top of the recommendations.

Figure (15) shows the recommendations for a *"dummy user"* that gave the movie "*Lord the rings*" five stars. The factor α have been choose after a several trials and the value $\alpha = 0.02$ have been shown to be an acceptable value.

title	genres
Star Wars: Episode IV - A New Hope (1977)	Action|Adventure|Sci-Fi
Star Wars: Episode V - The Empire Strikes Back...	Action|Adventure|Sci-Fi
Star Wars: Episode VI - Return of the Jedi (1983)	Action|Adventure|Sci-Fi
Star Wars: Episode I - The Phantom Menace (1999)	Action|Adventure|Sci-Fi
Lord of the Rings: The Fellowship of the Ring,...	Adventure|Fantasy
Star Wars: Episode II - Attack of the Clones (...	Action|Adventure|Sci-Fi|IMAX
Lord of the Rings: The Two Towers, The (2002)	Adventure|Fantasy
Lord of the Rings: The Return of the King, The...	Action|Adventure|Drama|Fantasy
Harry Potter and the Prisoner of Azkaban (2004)	Adventure|Fantasy|IMAX
Star Wars: Episode III - Revenge of the Sith (...	Action|Adventure|Sci-Fi
Harry Potter and the Goblet of Fire (2005)	Adventure|Fantasy|Thriller|IMAX
Harry Potter and the Order of the Phoenix (2007)	Adventure|Drama|Fantasy|IMAX
Star Wars: The Clone Wars (2008)	Action|Adventure|Animation|Sci-Fi
Harry Potter and the Half-Blood Prince (2009)	Adventure|Fantasy|Mystery|Romance|IMAX
Harry Potter and the Deathly Hallows: Part 1 (...	Action|Adventure|Fantasy|IMAX
Harry Potter and the Deathly Hallows: Part 2 (...	Action|Adventure|Drama|Fantasy|Mystery|IMAX
Hobbit: An Unexpected Journey, The (2012)	Adventure|Fantasy|IMAX
Hobbit: The Desolation of Smaug, The (2013)	Adventure|Fantasy|IMAX
The Hobbit: The Battle of the Five Armies (2014)	Adventure|Fantasy
Star Wars: Episode VII - The Force Awakens (2015)	Action|Adventure|Fantasy|Sci-Fi|IMAX

Figure 15. Personalized recommendations provide by the recommender prototype: Top twenty.

6. Add Features information

As it only consider past interactions to make recommendations, collaborative filtering approaches like matrix factorization suffer from the **"cold start problem"**: it is impossible to recommend anything to new users or to recommend a new item to any users.

This drawback can be addressed in different way:

- recommend new items (movies) to random users
- recommend new items to must active users
- recommend new items to set of various users

However, a more sophisticated approach has been utilized in this study to address the issue with new movies. This approach involves incorporating side information about the movies into the model. The concept behind this is that when new items are introduced, the model can still provide informed recommendations based on the known features of the items, even without interaction data. In the 25M MovieLens dataset, the features information includes movie genres (*Action, Horror, Children's...etc*). Therefore, we include genre information in the model, allowing the algorithm (ALS) to learn its own representation of the features (feature vectors).

6.1. Mathematical Formulation

The model (4)

$$\mathcal{P}\big(r_{mn}|u_m, v_n, b_m^{(u)}, b_n^{(i)}, f_i\big) = \mathcal{N}\big(r_{mn}; u_m^T v_n + b_m^{(u)} + b_n^{(i)}, \lambda^{-1}\big) \tag{11}$$

with:

- $\mathcal{P}(u_m) = \mathcal{N}\big(u_m; 0, \tau\mathbf{I}\big)$, prior for u_m .
- $\mathcal{P}(f_i) = \mathcal{N}\big(f_i; 0, \beta\mathbf{I}\big)$, $prior\ for\ f_i$
- $\mathcal{P}(v_m) = \mathcal{N}\big(v_n; \frac{1}{\sqrt{F_n}} \sum_{i \in features(n)} f_i, \tau\mathbf{I}\big)$, $prior\ for\ v_n$.
- $\mathcal{P}(b^{(u)}) = \mathcal{N}\big(b^{(u)}; 0, \gamma\mathbf{I}\big)$, $\mathcal{P}(b^{(i)}) = \mathcal{N}\big(b^{(i)}; 0, \gamma\mathbf{I}\big)$ priors for $b^{(u)}$ and $b^{(i)}$ respectively.

The objective function:

$$\begin{aligned}
\mathcal{L} &= \log \mathcal{P}\big(R|U, V, b^{(u)}, b^{(i)}\big) + \log \mathcal{P}\big(U\big) + \log \mathcal{P}\big(V\big) \\
&\quad + \log \mathcal{P}\big(F\big) + \log \mathcal{P}\big(b^{(u)}\big) + \log \mathcal{P}\big(b^{(i)}\big) \\
&= -\frac{\lambda}{2} \sum_m \sum_{n \in \Omega(m)} \Big(r_{mn} - (u_m^T v_n + b_m^{(u)} + b_n^{(i)})\Big)^2 \\
&\quad - \frac{\tau}{2} \sum_m u_m^T u_m - \frac{\tau}{2} \sum_m f_i^T f_i \\
&\quad - \frac{\tau}{2} \sum_n \big(v_m - \frac{1}{\sqrt{F_n}} \sum_{i=1}^{num_f} f_i\big)^T \big(v_m - \frac{1}{\sqrt{F_n}} \sum_{i=1}^{num_f} f_i\big) \\
&\quad - \frac{\gamma}{2} \sum_m b_m^{(u)2} - \frac{\gamma}{2} \sum_n b_n^{(i)2}
\end{aligned} \tag{12}$$

proceeding the same way as in **3.3**, we get the following updates at each step:

User vector update: No change happen, see equation (7).

User bias update: see equation (8).

Item vector update:

$$v_n = \lambda\Big(\sum_{m \in \Omega(n)} u_m u_m^T + \tau\mathbb{I}\Big)^{-1} \Big(\lambda \sum_{m \in \Omega(n)} u_m (r_{mn} - b_m^{(u)} - b_n^{(i)}) + \tau f\Big) \tag{13}$$

with $f = \frac{1}{\sqrt{F_n}} \sum_{i \in features(n)} f_i$.

Item bias update: refer to equation (10)

feature vector update:

$$f_i = \frac{\sum_{n \in \Omega(\mathcal{F}(i))} \Big(v_n \frac{1}{\sqrt{F_n}} - \frac{1}{F_n} \sum_{j \in \mathcal{F}_n, j \neq i} f_j\Big)}{\Big(\sum_{n \in \Omega(\mathcal{F}(i))} \frac{1}{F_n} - 1\Big)} \tag{14}$$

where:

- $\Omega(\mathcal{F}(i))$ is the set of index of movies containing the feature i.
- $\mathcal{F}_n$ is the set of features of the movie index by n.
- f_j is the feature vector of the feature j.

Note: "$features(n)$ " is the set of feature factors of the movie indexed by n , F_n is the number of feature factors of the movie indexed by n and num_f is the total number of features (movie genres) in the dataset.

Algorithm 2 ALS (with feature information)

Input: data-structured, K , λ, τ, γ
Initialize U, V, F, $b^{(u)}$, $b^{(i)}$
repeat
  for $m = 0$ **to** $M - 1$ **do**
    for (n,r) in data-index-by-user[m] **do**
      update user vector $U[m:]$ with equation (7)
      update user bias $b_m^{(u)}$ with equation (8)
    end for
  end for
  for $n = 0$ **to** $N - 1$ **do**
    for (m,r) in data-index-by-item[n] **do**
      update item vector $V[n:]$ with equation (13)
      update item bias $b_n^{(i)}$ with equation (10)
    end for
  end for
  for $i = 0$ **to** $num_f - 1$ **do**
    update feature $f[i:]$ with equation (14)
  end for
until convergence

6.2. Feature Embeddings

The parameters of our model, including item genres information, have been optimized using $K = 2$ as the dimension of the latent space on the full 25 million ratings. The **the figure 16** represents the **2D** embedding of the learned representations for *"Children's"* and *"Horror"* movies genres.

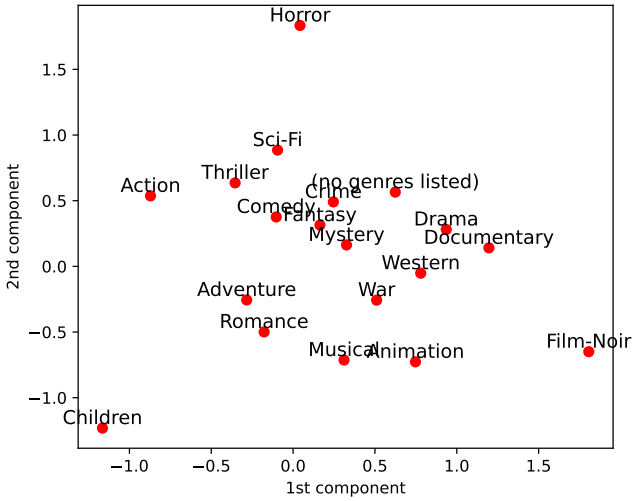


Figure 16. 2D feature embeddings

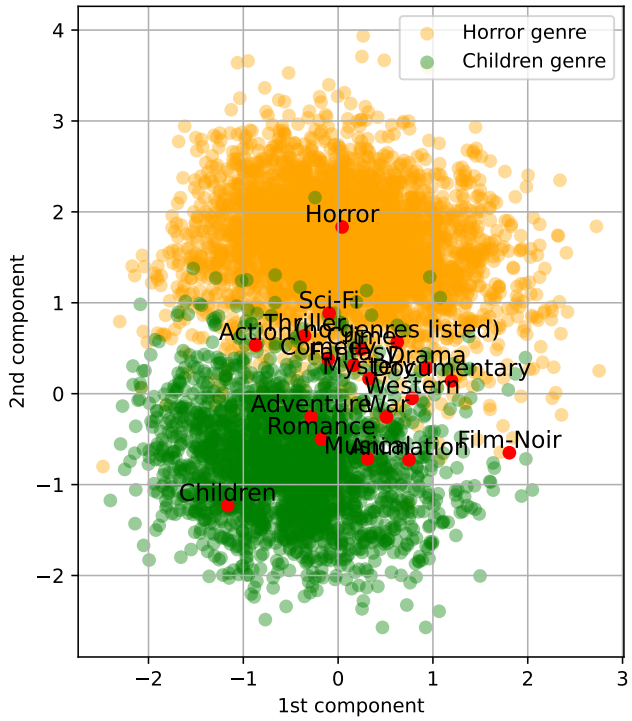


Figure 17. 2D item + feature embeddings

We can notice how better the embedding are, compared to the one with no featured information include (figure 10).

6.3. Recommendations

We have considered the hyperparameters used earlier ($K = 10,\ \lambda = 1,\ \gamma = 0.1,\ \tau = 1$) to optimize model parameters, including feature vectors. Then a *"dummy user"* that gave the "*Children's* genre movie named **"Toy Story"** five stars have been created. Below are the results of the first twenty movies recommended by our recommender prototype.

title	genres
Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
Babe (1995)	Children|Drama
Lion King, The (1994)	Adventure|Animation|Children|Drama|Musical|IMAX
Aladdin (1992)	Adventure|Animation|Children|Comedy|Musical
White and the Seven Dwarfs (1937)	Animation|Children|Drama|Fantasy|Musical
Beauty and the Beast (1991)	Animation|Children|Fantasy|Musical|Romance|IMAX
Pinocchio (1940)	Animation|Children|Fantasy|Musical
Cinderella (1950)	Animation|Children|Fantasy|Musical|Romance
Dumbo (1941)	Animation|Children|Drama|Musical
Little Mermaid, The (1989)	Animation|Children|Comedy|Musical|Romance
Babe: Pig in the City (1998)	Adventure|Children|Drama
Tarzan (1999)	Adventure|Animation|Children|Drama
Toy Story 2 (1999)	Adventure|Animation|Children|Comedy|Fantasy
Monsters, Inc. (2001)	Adventure|Animation|Children|Comedy|Fantasy
Finding Nemo (2003)	Adventure|Animation|Children|Comedy
Incredibles, The (2004)	Action|Adventure|Animation|Children|Comedy
ures of Winnie the Pooh, The (1977)	Animation|Children|Musical
Toy Story 3 (2010)	Adventure|Animation|Children|Comedy|Fantasy|IMAX
nie the Pooh and Tigger Too (1974)	Animation|Children
Inside Out (2015)	Adventure|Animation|Children|Comedy|Drama|Fantasy

Figure 18. Top twenty recommended movies

7. Conclusion

Recommendation systems are a fascinating subject, but they can quickly become overly complicated, especially when implemented on a large scale with millions of users and items. Matrix factorization provides a straightforward and convenient method for making recommendations when ratings are available. However, it has its pitfalls, one of which we have already encountered in our implementation: the "cold start problem." To tackle this issue, added additional information about items. Generally, it is challenging to obtain feedback from users, as most users only rate items when they either really like them or absolutely hate them. In such cases, we often need to find ways to measure implicit feedback and employ negative sampling techniques to create a reasonable training set.

Find the source code: Github

References

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

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

Lü, L., Medo, M., Yeung, C. H., Zhang, Y.-C., Zhang, Z.-K., and Zhou, T. Recommender systems. *Physics reports*, 519(1):1–49, 2012.

Schafer, J. B., Frankowski, D., Herlocker, J., and Sen, S. Collaborative filtering recommender systems. In *The adaptive web: methods and strategies of web personalization*, pp. 291–324. Springer, 2007a.

Schafer, J. B., Frankowski, D., Herlocker, J., and Sen, S. Collaborative filtering recommender systems. In *The adaptive web: methods and strategies of web personalization*, pp. 291–324. Springer, 2007b.