Recommendation System: A Matrix Factorization Approach

Awodumila Tobi [1]

Abstract

We approach the recommender system with matrix factorization over the movielen 25 million dataset. By recreating a suitable data structure from the dataset we explore three models. The first is a model that solely depends on bias, the second incorporates latent vectors for users and items, and finally the third extends the item latent vector by incorporating a latent feature vector. We evaluated our models and observed that the best recommendations were from models with a latent width of $k = 20$.

1. Introduction

Every day, businesses answer the question of what to produce. and for whom to produce? Similar users and consumers of products want to either buy a product or are looking for product that matches their taste or is similar to a product they have already consumed. Users mostly depend on recommendations from friends or from other users who may have used a similar produce what they have used previously, while businesses after they have produced their product, needs to find the right customers who are willing to pay or who are buying a product that has a joint or competitive demand for the business's produce.

In a market, the dynamics of purchase are influenced by advertisements, which serve as a means to suggest a product to a user. With the emergence of the internet and e-commerce, big tech companies are leveraging different recommendation algorithms to recommend new or similar products to their users/ customers. This has helped improve sales and has also given users a better experience with respect to receiving services they are interested in easier.

Recommendation systems have existed for a long time whether it be the offline pear recommendation or electronically through various algorithms like collaborative filtering, content filtering, and context filtering. Tech companies like Google, Meta, Netflix, Amazon, Windows, etc. all implement a form of recommender system as a service or a way to sell their products and introduce new / less-known products.

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

Collaborative Filtering: is a recommender system approach that predicts a user's preference based on past preference extracted from the historic usage of a product or service. It works by taking the similarity of user preference behavior to anticipate the future interaction between users and items. The model is based on user historical behavior, such as previously purchased things, ratings for those items, and judgments made by other users that are similar to their own. The theory goes that if a group of individuals have previously made similar decisions and purchases—for example, choosing a movie—there's a good chance they'll agree on more choices in the future (nvi). The model will recommend items liked by a user to other users who have similar preferences to this user.

Content Filtering:is a recommender system model that recommends based on the item feature that a given user likes. The model will recommend to a user an item that has similar features to an item the user has previously used or interacted with, in contrast to collaborative filtering, which finds similarities between users. For example, if a user has seen a Marvel movie like Justice League, the model may recommend Batman, Superman, or a similar Marvel to them. (nvi)

Context Filtering The recommender model incorporates context information about a user and bases it on this. This method predicts the likelihood of the subsequent action based on a series of contextual user actions and the present context. (nvi)

2. Data

We would use the movielen dataset for this exploration of the recommender system. The dataset is generated on the Movielens website, which is a website where people express preferences for movies, which were originally made available in 1998. The format of these preferences is

```
<user, item, rating, timestamp>
```

tuples, each of which is the outcome of an individual rating a movie at a specific period (0–5). The MovieLens website, a recommender system that requests movie ratings from its customers in exchange for individualized movie recommendations, was used to register these preferences. (Harper & Konstan, 2015) The dataset is available on the Grouplen website (mov). The site has a variety of movielen datasets. MovieLens 25M Dataset contains 25M movie ratings. Stable benchmark dataset. 25 million ratings and one million tag applications were applied to 62,000 movies by 162,000 users. Includes tag genome data with 15 million relevance scores across 1,129 tags, Released 12/2019. MovieLens 100K Dataset contains 100K movie ratings. Stable benchmark dataset. 100,000 ratings from 1000 users on 1700 movies Released 4/1998. MovieLens 1M Dataset contains 1M movie ratings. Stable benchmark dataset. 1 million ratings from 6,000 users on 4000 movies. Released 2/2003. MovieLens 10M Dataset contains 10M movie ratings. Stable benchmark dataset. 10 million ratings and 100,000 tag applications were applied to 10,000 movies by 72,000 users. Released 1/2009. The MovieLens 20M Dataset contains 20M movie ratings. Stable benchmark dataset. 20 million ratings and 465,000 tag applications were applied to 27,000 movies by 138,000 users. Includes tag genome data with 12 million relevance scores across 1,100 tags. Released 4/2015; updated 10/2016 to update links.csv and add tag genome data. (mov) For our exploration, we would be using the 100,000 dataset for development and the 25 million dataset for training and prediction in our model.

2.1. Data Analysis

The movielen dataset contains seven csv files, of which we are using two for our model, and we analyze these two to understand the distribution of the dataset. We worked with the Movielen 25M Dataset, and we are using the ratings.csv and movies.csv data. The movie data contains movieId, title, and genre as attributes, while the rating data has userId, movieId, timestamp, and rating. The movie title is observed to carry the year of release; therefore, we added the year feature to the movie data set. The oldest movie was released in 1874, and the new movies were released in 2019. We have movies from 135 years between 1874 and 2019.

We have a total of 62423 movies, out of which 410 have no release year indicated in the data. The movie data contains 19 genres, namely: action, adventure, animation, children, comedy, crime, documentary, drama, fantasy, film-noir, horror, IMAX, musical, mystery, romance, sci-fi, thriller, war, and western. About 5062 do not have a genre indicated, 75% have 2 genres, and the maximum number of genres a movie had was 10. Movies in the action genre were rated 7446918 times; it is the most-rated genre, while the IMAX genre is the least-rated genre with 27.

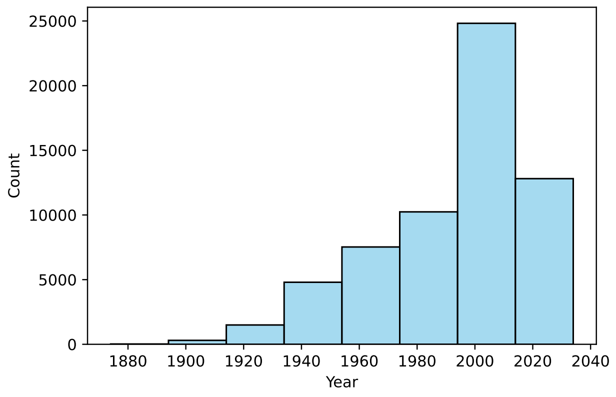


Figure 1. Distribution of Movie Release Years

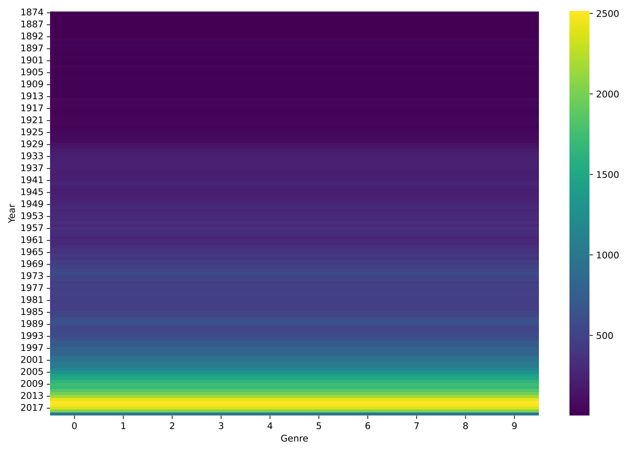


Figure 2. Number of Movies by Genre and Year

The rating data contains ratings the user gave to a movie, and the movie is rated 1–5 with a rating interval score of 0.5. We explore the timestamp feature by extracting the day of the week, date, hour, and when the rating was done. We found that ratings were given every hour of the day.

User 75% of users gave a rating of 4, on average users gave a rating of 3.5 while the highest rating given is 5. From the rating of the movie, we can infer the popularity of the movie. We want to show the popularity of the movie and the ratings the users gave follow the power law.

Power law is a relationship between two quantities that a proportional to each other in which relative change leads to a proportional change in the other quantity to a power of the change, and is independent of the initial value of the other quantity $y = ax^{-k}$. Given the relation $f(x) = ax^{-k}$ by scaling the argument x by a factor c, we obtain a

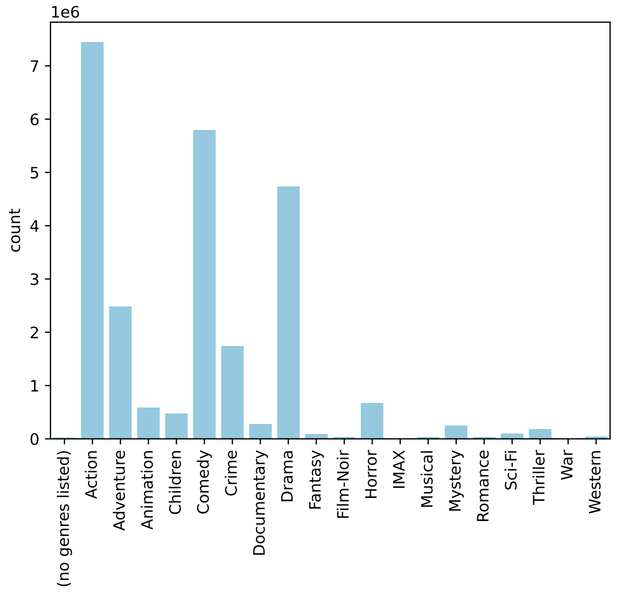


Figure 3. Distribution of Genre

proportional scale of the function. itself(pow);

$$f(cx) = a(cx)^{-k} = c^{-k} f(x) \propto f(x)$$

The power simple put is the nth most popular item or user is $\frac{1}{n}$ the popularity of the most popular item or user.

The rating data contains 25 million ratings from 162541 users. The UserId of the largest rating frequency is UserId: 72315. The top movie the user rated was Ecstasy (2011) which was rated twice: 2.5 on the first rating and 2.0 on the second. The user rated 32202 movies. The average rating is 3, this user rated a total of 9250 on Tuesday, which is the day with the highest frequency of ratings.

Overall, most movies were rated on Sunday with a frequency of 3876693. The time of day analysis shows that more movies were rated in the evening with a frequency of 13813235.

3. Methodology

In an attempt to map users to items using a matrix where the columns will represent all the ratings a movie item has received and the rows will represent all the ratings a user has given to all the movies in the data set, we encounter a problem of sparse mapping because not every user has given a rating to all the movie items. As a result of this sparse matrix, we could attempt to fill up the sparse portion of the matrix with zero, or the mean of the user's rating, but by

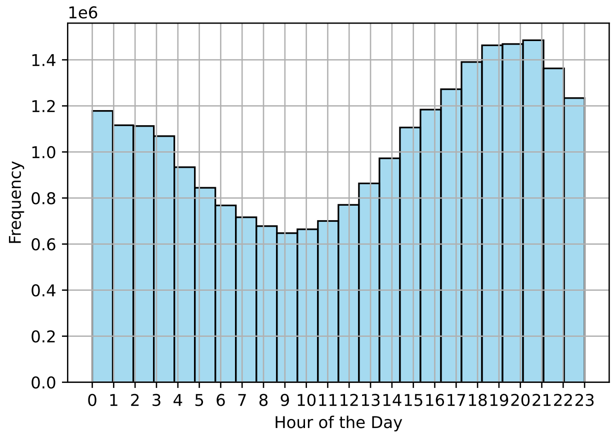


Figure 4. Distribution of Hour

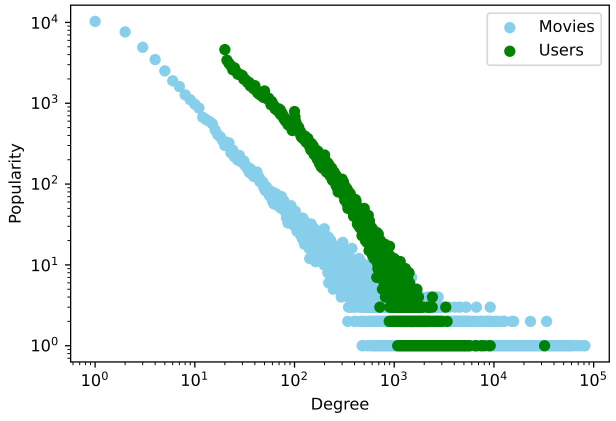


Figure 5. Distribution of Genre

doing so, we would succeed in encoding inaccurate data, leading to wrong predictions. Thus, a matrix method like Singular Value Decomposition can not used on our resulting sparse matrix, and so we need another way to fill up the missing data.

To achieve a non-sparse matrix for the user-item rating using matrix factorization, Given R, the matrix of user-item ratings represents the approximate rating a user would give to every movie item. We could get a two factor matrix U, V of R with latent dimension k such that ; $R = UV^T$

Matrix factorization is the foundation for some of the most effective applications of latent component models. Matrix factorization, in its most basic version, uses vectors of factors deduced from item rating patterns to characterize both items and users. A recommendation is the result of a correlation between item and user factors. These techniques have increased in popularity recently because they combine high

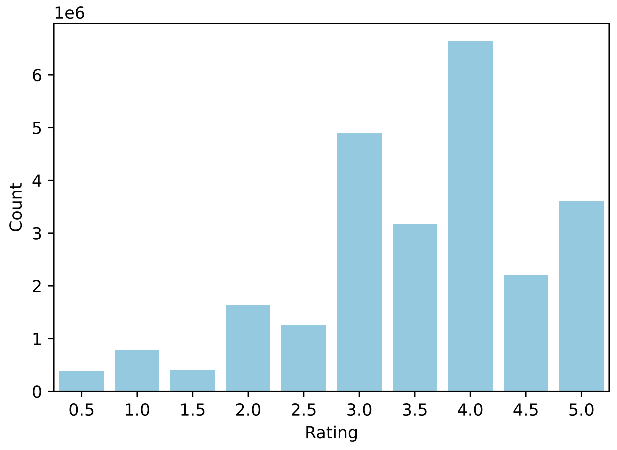


Figure 6. Distribution of Ratings

predicted accuracy with good scalability. Furthermore, they have versatility for modeling a variety of real-life scenarios. (Koren et al., 2009).

Another advantage of using matrix factorization is its ability to incorporate more information into the model. Where there is no explicit feedback, the recommender system can infer user preferences for implicit feedback, which is obtained for user interactions in a given environment such as browsing, searches, clicks, purchase histories, and mouse hovers. Implicit feedback indicates the presence or absence of an event, thus resulting in a densely populated matrix.

3.1. User Item Data Structure Transformation

We obtain from the data a structure that aids us in computing at a fast rate instead of depending on the structure of the raw data in a table. We created a data structure using a list of tuples, this is derived from the data. We end up with data like:

```
data_structure = [
[(movie_idx,rating),(movie_idx,rating)
   ,(movie_index,rating)],
   ...
]
 movie = [movieId1,movieId2,...]
 users = [userId1,userId2,...]
```

To achieve this, we first got all the unique user IDs and movie IDs present in the ratings data as two separate index lists. Using this list, we created two dictionaries that map the ids to the index two lists of length M according to the number of elements in the list of user ids, respectively. We loop through the entire dataset, and for each data entry, we get the index of your userId and movieId. We use the

Algorithm 1 Data Structure

```
Input: data
user_to_idx , movie_to_index = unique user ID, unique movie ID
user_to_rating = empty maps for user and movie indices to rating
for each rating in data entry do
  Map user and movie IDs to indices
  Store ratings in user-to-rating and movie-to-rating maps
end for
```

useId_idx to locate the list for the user in the list of empty lists we created with a length of M. We then append to the list the tuple containing the movie_idx and the rating given at that data point.

3.2. Data Split

To train our model, we need to have a way to evaluate it on a new dataset that was not part of the training set; hence, we have to split the whole 25 million datasets into test and train sets. We took 20% for the test and the remaining 20 percent for training.

During data analysis, it was observed that the userId was grouped/ordered, so, all users have their rating together in successive sequences on the table. This implies that splitting the data without the top means losing some users in the training set. We also need to keep the data structure we proposed in the previous section the same for both the training set and the test set. Hence, we came up with the idea of splitting before converting the resultant split to our desired structure.

To split, first obtain all unique userId and movieId in the whole data and preserve them. We shuffled the data, allowing the rows to remove any form of sorting that exited in the data and introduced randomness in the data set. After the shuffling, we split at the top and assigned the top 20% percent to the test and the bottom 80% to the train. We applied the data structure to the test and train sets using the same unique list of userId and movieId gotten before the split.

3.3. Model

Our model is a collaborative filter using the matrix factorization method, which maps both the user and items to a joint latent factor space of dimensionality k such that the inner product of user-item factors is an interaction in k space. Given that $U_m \in \mathbb{R}^k$ is the vector associated with user m and $V_n \in \mathbb{R}^k$ is the vector associated with item n, measure the degree to which item n is possessed by users. Let r_{mn} be the actual rating given by user m to item n and the inner

Algorithm 2 Split

```
Input: data
user_to_idx , movie_to_index = unique user ID, unique movie ID
shuffled_data = shuffle data
test = 20% of top data i.e shuffled_data[:20%]
train = 80% of bottom data i.e shuffled_data[20% :]
apply user_movie_structure test using user_to_idx and movie_to_index
apply user_movie_structure train using user_to_idx and movie_to_index
```

product of the latent factor leads to the estimate.

$$\hat{r}_{mn} = U_m^T V_n$$

(Koren et al., 2009)

3.4. Alternating Least Squares (ALS)

Alternating Least Squares is one of the tools used for matrix factorization aside from stochastic gradient descent (Koren et al., 2009).

Given R the ALS factorises R into two factors U and V such that $R \approx U^T V$. The latent factors are given to the algorithms. In other to find the user and item matrix, we find the U_m and V_n the minimum and the minimum of the error on r_{mn}

$$arg \min_{U,V} \sum_n \sum_m (r_{mn} - u_m^T v_m)^2 + \lambda \left(\sum_m u_m^T u_m + \sum_n v_n^T v_n \right)$$

(Zhou et al., 2008)

3.4.1. Model on Bais Only

We assume a hierarchical model to produce the likelihood of r_{mn}. We also assume the most basic form of the likelihood, which depends on $b_m^{(u)}$ and $b_n^{(i)}$. We defined the likelihood as follows:

$$p(r_{mn}|b_m^{(u)}, b_n^{(i)}) = \mathcal{N}(r_{mn}; b_m^{(u)} + b_n^{(i)}, \lambda^{-1})$$

where

- u_m - user m latent vector
- v_n - item n latent vector
- λ - regularisation factor the inverse of the standard deviation of r_{mn}
- r_{mn} - rating for item n by user m.

we initialize $b_m^{(u)}$ and $b_n^{(i)}$ to be 0 vectors and regulariser of γ.

$$\begin{tikzcd}
b_m^{(u)} \arrow[dr] & & b_n^{(i)} \arrow[dl] \\
& \boxed{r_{mn}} &
\end{tikzcd}$$

We express the loss $\mathcal{L}$ for our model with only bia as;

$$\mathcal{L} = -\frac{\lambda}{2} \sum_n \sum_m (r_{mn} - (b_m^{(u)} + b_n^{(i)})^2 - \frac{\gamma}{2} \sum_m b_m^{(u)2} - \frac{\gamma}{2} \sum_n b_n^{(i)2}$$

Using this, we obtain the bias update by minimizing the loss function with respect to bais $b_m^{(u)}$ and $b_n^{(i)}$ respectively, by computing the derivative and equating it to 0 to update the minimum in an alternating sequence until the best bias vectors are obtained until the end of the sent number of epochs.

The derivatives for the updates are as follows;

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial b_3^{(u)}} &= \frac{\partial}{\partial b_3^{(u)}} \Big(-\frac{\lambda}{2} \sum_n \sum_m (r_{mn} - (b_m^{(u)} + b_n^{(i)})^2 \\
&\quad - \frac{\gamma}{2} \sum_m b_m^{(u)2} - \frac{\gamma}{2} \sum_n b_n^{(i)2}\Big) \\
0 &= -\lambda \sum_{n \in \Omega(m)} (r_{mn} - b_n^{(i)}) - \lambda \sum_{n \in \Omega(m)} b_3^{(u)} - \gamma b_3^{(u)} \\
\lambda \sum_{n \in \Omega(m)} b_3^{(u)} + \gamma b_3^{(u)} &= -\lambda \sum_{n \in \omega(m)} (r_{mn} - b_n^{(i)}) \\
(\lambda |\Omega(m)| + \gamma) b_3^{(u)} &= -\lambda \sum_{n \in \Omega(m)} (r_{mn} - b_n^{(i)}) \\
b_3^{(u)} &= \frac{-\lambda \sum_{n \in \Omega(m)} (r_{mn} - b_n^{(i)})}{\lambda |\Omega(m)| + \gamma}
\end{aligned}$$

We do a similar derivation for the $b_n^{(i)}$ to obtain it update as ;

$$b_3^{(i)} = \frac{-\lambda \sum_{m \in \Omega(n)} (r_{mn} - b_m^{(u)})}{\lambda |\Omega(m)| + \gamma}$$

3.4.2. Algorithm

We implement the algorithm to update the biases. We start initializing the biases to 0, and the biases are N and M in size, corresponding to the number of items and users, respectively. We loop over the training data structure for users, At each iteration, we iterate over all the ratings given, i.e., the list of tuples, and compute the sum of $r_{mn} - b_n^{(i)}$ of each tuple. At the end of the tuple loops, we update the user bias by subtracting the product of the sum and α the regularizing parameter or the learning rate. Then we

Algorithm 3 Bias Update

```
for $m = 1$ to $M$ do
    Initialize bias: $bias = 0$
    Initialize item count: $item_count = 0$
    for each $(n, r)$ in $U[m]$ do
    Update bias: $bias += \lambda \times (r -$
    end for
    Calculate updated bias: $bias = \frac{bias}{(\lambda \times itemcount) + \gamma}$
Update user bias:
end for
for $n = 1$ to $N$ do
    Initialize bias: $bias = 0$
    Initialize user count: $user_count$ = 0
    for each $(m, r)$ in $V[n]$ do
        Update bias: $bias += \lambda \times (r -$ user biases$[m])$
        Increment user count: $user_count += 1$
    end for
    Calculate updated bias: $bias = \frac{bias}{(\lambda \times usercount) + \gamma}$
Update item bias:
end for
```

implement a similar update on the item bias by looping over the item data structure and repeating the whole process over the number of epochs.

Using this, we obtain the bias update by minimizing the loss function with respect to bais $b_m^{(u)}$ and $b_n^{(i)}$ respectively, by computing the derivative and equating it to 0 to update the minimum in an alternating sequence until the best bias vectors are obtained until the end of the sent number of epochs.

The above model is not enough to predict user and item ratings because it is too simple and does not contain important attributes of the system we aim to model. We introduce the personalization component for both users and items.

3.4.3. Model with Latent Vectors

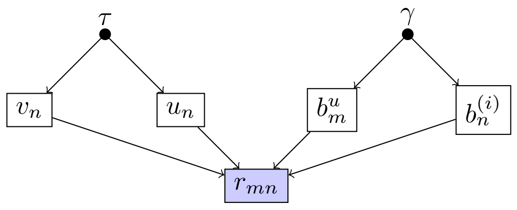
We modify the previous model to incorporate latent vectors for users and items. The likelihood can be redefined to include u_m and v_n;

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

where

- $u_m^{(u)}$ is the user vector with a k latent.
- $v_n^{(i)}$ is the item vector with a k latent.
- τ regularization term for u_m and v_n

we initialize u_m and v_n as $(M \times k)$ and $(N \times k)$ matrix with ; $u_m \sim \mathcal{N}(0, \frac{1}{\sqrt{k}})$ and $v_n \sim \mathcal{N}(0, \frac{1}{\sqrt{k}})$



By incorporating latent user matrix b_n and item matrix The loss function become;

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

By minimizing the new loss to obtain the updated for u_n and $u_m^{(i)}$ by computing the derivative of the loss with respect to $u_n^{(u)}$ and $u_m^{(i)}$ respectively.

$$u_3 = (\lambda \sum_{n\in\Omega(3)} v_n^T v_n - \tau I)^{-1} - \lambda \sum_{n\in\Omega(3)} (r_{3n} - (b_3^{(u)} + b_n^{(i)})).v_n$$

with a similar simplification we obtain

$$v_3 = (\lambda \sum_{n\in\Omega(3)} u_m^T u_m - \tau I)^{-1} - \lambda \sum_{n\in\Omega(3)} (r_{3n} - (b_3^{(u)} + b_n^{(i)})).u_m$$

The bias updates is also computed as follows;

$$\frac{\partial \mathcal{L}}{\partial b_3^{(u)}} = \frac{\partial}{\partial b_3^{(u)}}(-\frac{\lambda}{2}\sum_n\sum_m(r_{3n} - (u_m^T v_n - b_m^{(u)} + b_n^{(i)})^2 \; - \frac{\gamma}{2}\sum_m b_m^{(u)2} - \frac{\gamma}{2}\sum_n b_n^{(i)2})$$

$$0 = -\lambda \sum_{n\in\Omega(3)} (r_{3n} - u_m^T v_n - b_n^{(i)}) - \lambda \sum_{n\in\Omega(m)} b_3^{(u)} - \gamma b_3^{(u)}$$

$$\lambda \sum_{n\in\Omega(3)} b_3^{(u)} + \gamma b_3^{(u)} = -\lambda \sum_{n\in\omega(3)} (r_{3n} - u_3^T v_n - b_n^{(i)})$$

$$(\lambda|\Omega(3)| + \gamma) b_3^{(u)} = -\lambda \sum_{n\in\Omega(3)} (r_{3n} - u_3^T v_n - b_n^{(i)})$$

$$b_3^{(u)} = \frac{-\lambda \sum_{n\in\Omega(3)}(r_{3n} - u_3^T v_n - b_n^{(i)})}{\lambda|\Omega(3)| + \gamma}$$

Algorithm 4 Alternating Bias and Latent Vector Update

```
Get the lengths of $U$ and $V$: $M, N = \text{len}(U), \text{len}(V)$
for $m = 1$ to $M$ do
    for each $(n, r)$ in $U[m]$ do
        Compute user bias update: $bias\_update+ = (r - \mathbf{u}[m]^T\mathbf{v}[n] - b[n]^{(u)})$
        Update sum of latent vectors: $latent\_sum+ = \mathbf{v}[n]$
        Compute $b+ = (r - (b_m^{(u)} + b_n^{(i)}))\mathbf{v}_n$
    end for
    Update user bias: user_biases$[m] = \frac{-\lambda \cdot bias\_update}{(\lambda \cdot |U[m]| + \gamma)}$
    for each $(n, r)$ in $U[m]$ do
        Update matrix $A$: $A+ = \mathbf{v}_n^T\mathbf{v}_n$
    end for
    Update matrix $A$: $A = \lambda \cdot A - \gamma I$
    Update user latent vector: user_vec$[m] = A^{-1} \cdot b$
end for
for $n = 1$ to $N$ do
    Update item bias: item_biases$[n] = \frac{-\lambda \cdot bias\_update}{(\lambda \cdot |V[n]| + \gamma)}$
    Update item latent vector: item_vec$[n] = A^{-1} \cdot b$
end for
```

and with similar algebra;

$$b_3^{(i)} = \frac{-\lambda \sum_{m \in \Omega(3)} (r_{m3} - u_m^T v_3 - b_m^{(u)})}{\lambda |\Omega(3)| + \gamma}$$

3.4.4. Algorithm

The bias and latent vectors are updated in alternation by first looping over the data from user ratings. For each iteration, iterate over the ratings the user gave and compute the bias update using $\frac{-\lambda \sum_{m\in\Omega(n)}(r_{mn} - u_m^T v_n - b_m^{(u)})}{\lambda|\Omega(m)|+\gamma}$ and in the same user data structure, iterate again over the ratings of the user and update the user latent vector using $(\lambda \sum_{n\in\Omega(m)} v_n^T v_n - \tau I)^{-1} - \lambda \sum_{n\in\Omega(m)}$ $(r_{mn} - (b_m^{(u)} + b_n^{(i)})).v_n$. The item data structure is looped over, and for each item, the item bias is updated with the formula: $\frac{-\lambda \sum_{m\in\Omega(n)}(r_{mn} - u_m^T v_n - b_m^{(u)})}{\lambda|\Omega(m)|+\gamma}$ After that, the item latent vector is updated with $(\lambda \sum_{n\in\Omega(m)} u_m^T u_m - \tau I)^{-1} - \lambda \sum_{n\in\Omega(m)} (r_{mn} - (b_m^{(u)} + b_n^{(i)})).u_m$.

3.4.5. Model with Feature vector

Features are details about a system that provides information about the state of the system in the case of our model we incorporate features to the item vector by adding features to the mean of the item latent vector. The modified loss with feature vector F is given as;

$$\mathcal{L} = -\frac{\lambda}{2} \sum_n \sum_m (r_{mn} - (u_m^T v_n - b_m^{(u)} + b_n^{(i)})^2 - \frac{\tau}{2} \sum_m u_m^T u_m - \frac{\tau}{2} \sum_m (v_n - \frac{1}{\sqrt{f_i}} \sum_{i \in features} f_i)^T (v_n - \frac{1}{\sqrt{f_i}} \sum_{i \in features} f_i) - \frac{\tau}{2} \sum_{i \in features} f_i^T f_i - \frac{\gamma}{2} \sum_n b_m^{(u)2} - \frac{\gamma}{2} \sum_n b_n^{(i)2}$$

where;

- f_i is the i genre

The optimal for item vector v_n by taking the first partial derivative with respect to v_n

$$v_n = (\sum_m u_m^T u_m -)^{-1} (\lambda \sum_m r_{mn} - \frac{\tau}{\sqrt{f_n}} \sum_{i \in features} f_i)$$

similar the update for the f_i by partial derivative of the loss function with feature vector;

$$f_w = (\tau \sum_{n \in car} \frac{1}{\sqrt{f_n}} + \tau I)^{-1} + \tau (\sum_n \frac{1}{\sqrt{f_n}} (v_n - \sum_{i \in feature_n ot_w} f_i))$$

where:

- **w** is a feature representing the wth genre.
- **feature_not_w** is the set of feature the are not the wth genre.

3.4.6. Algorithm

The update to the algorithm still uses the alternate least square by iterating over the user ratings data structure the user bias and user latent vector are updated in the same manner as in the previous model. We iterate over item ratings data structure the item bia is updated as in the previous section, however, the update of the item vector will be done with $(\sum_m u_m^T u_m -)^{-1} (\lambda \sum_m r_{mn} - \frac{\tau}{\sqrt{f_n}} \sum_{i \in features} f_i)$ and similar we iterate over all feature in the dataset and updated the feature vector according to the formulae $(\tau \sum_{n \in car} \frac{1}{\sqrt{f_n}} + \tau I)^{-1} + \tau (\sum_n \frac{1}{\sqrt{f_n}} (v_n - \sum_{i \in feature_n ot_w} f_i))$ t The genre of each movie is encoded as a one-hot vector was applied in computing the formula.

4. Result / Discussion

The results of our models are discussed with results obtained and performance, loss error, and Root Mean Squared error

Algorithm 5 Alternating Bias and Latent Vector Update with Feature vector

```
Get the lengths of $U$ and $V$
Initialize user biases, item biases, user vectors, item vectors, feature vectors,
for $m = 0$ to $M - 1$ do
    Update user bias Update user vectors
end for
for $n = 0$ to $N - 1$ do
    Update item bias
    Update item latent vector
end for
for $i = 0$ to  do
    =0)Calculate feature update: Update feature vector: feature_vec$[i] = \frac{\text{total\_right}}{\sum_{n \in \text{indices}} \frac{1}{\sqrt{\text{feature\_sums}[n]}} - 1}$
end for
```

are compared for all models.

Training

The training on each model is done by iterative update of all parameters for a set amount of epochs. At the end of each epoch, the loss is computed on the training data, and the Root Mean Squared Error (RMSE) is computed for both the training dataset and test dataset. The RMSE is given by;

$$RMSE = \sqrt{\frac{1}{|R|} \sum_{i \in R} r_{ui} - \hat{r_{ui}}}$$

where;

- **R** - Rating matrix
- r_{ui} - rating user u gave to item i
- $\hat{r_{ui}}$ - predicted rating of user u for item i

4.0.1. Model with bia only

The result obtain from the model with only bia shows that the loss converges quickly after 3 epochs ;

Metric	Value
Loss	38,477,725.28
Training RMSE	0.887993
Test RMSE	0.878385

Table 1. Bia only Model Loss

The Loss converging quickly shows it learns almost nothing hence we improve on this basic model.

4.0.2. Model with Latent Vector

The model with a latent vector allows for encoding more information in the latent dimension. By taking advantage

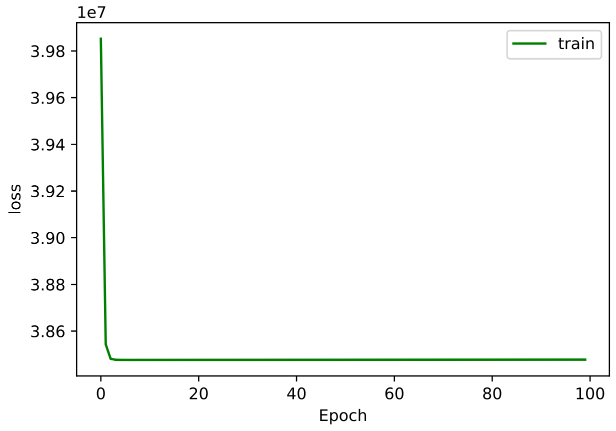


Figure 7. Training log likelihood on k=20 for Bia only Modle

of this we obtain a better model. The model needs to be checked for overfitting hence we experimented with different values k parameters. The results obtained are shown as follows;

K	Train Loss	RMSE Train	RMSE Test
2	31,017,352.5742	0.8179	0.7886
5	27,642,591.0044	0.8011	0.7448
10	24,172,736.9046	0.8023	0.6970
20	19,930,552.5733	0.8419	0.6338

Table 2. Performance Metrics for Different Values of K

As k increases, the training loss, train RMSE, and test RMSE decreases.

The model with latent dimension $k = 20$ converges after 40 epochs, $k = 10$ converges after 20 epochs. The model performs well with $k = 20$ and does not perform well on the test data compared to $k = 10$. We can say the model over-fits for $k = 20$.

4.0.3. Prediction

Predictions are obtained from the rank of the rating score.

$$score = U.V^T \times \alpha b_n^{(i)}$$

We tested what prediction and made recommendations for a user who has seen Lord of the Rings (2001). The top 20 recommendations are as shown in the table below;

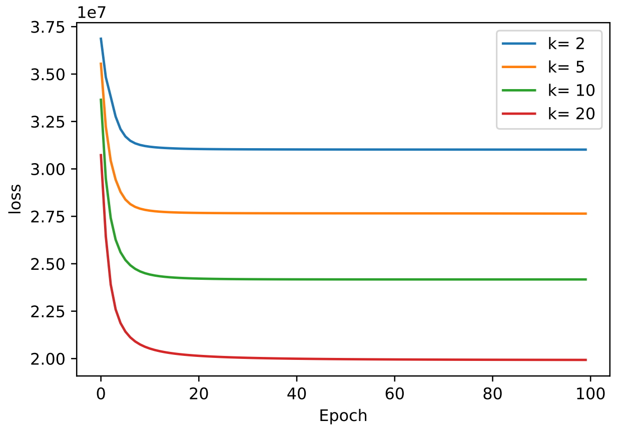


Figure 8. Training log likelihood on Latent vector model for different values of k.

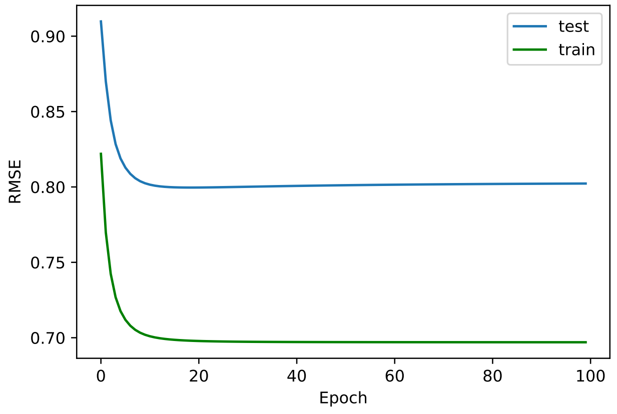


Figure 9. RSME for $k = 10$

Title
of the Rings: The Return of the King, The (2003)
Lord of the Rings: The Two Towers, The (2002)
Lord of the Rings: The Fellowship of the Ring, The (2001)
Hobbit: An Unexpected Journey, The (2012)
The Hobbit: The Battle of the Five Armies (2014)
Hobbit: The Desolation of Smaug, The (2013)
Harry Potter and the Deathly Hallows: Part 2 (2011)
Harry Potter and the Half-Blood Prince (2009)
Harry Potter and the Deathly Hallows: Part 1 (2010)
Harry Potter and the Order of the Phoenix (2007)
Harry Potter and the Goblet of Fire (2005)
Harry Potter and the Prisoner of Azkaban (2004)
Harry Potter and the Chamber of Secrets (2002)
Harry Potter and the Sorcerer's Stone (a.k.a. Harry Potter and the Philosopher's Stone) (2001)

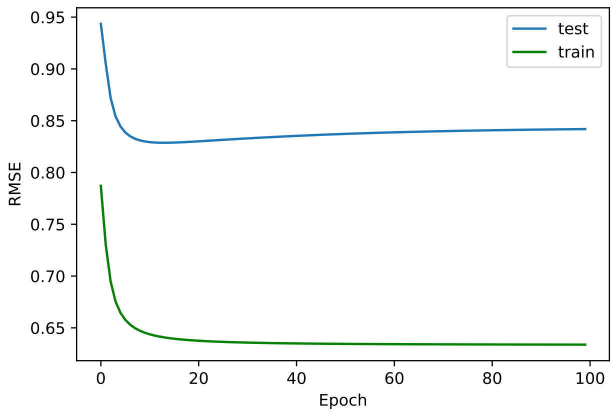


Figure 10. RSME comparison for $k = 20$

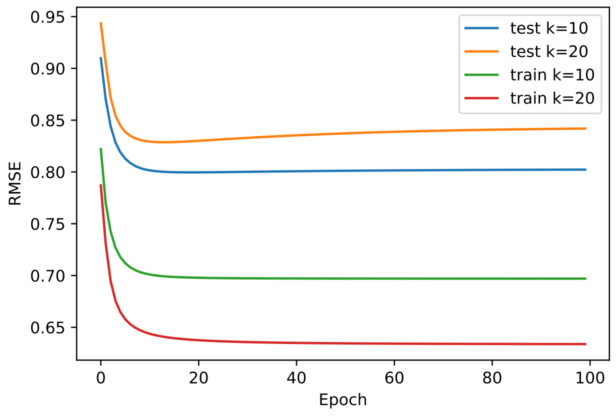


Figure 11. RSME comparison for $k = 10$ and $k = 20$

The recommendation makes sense because we can find the other Lord of the Rings, The Hobbit, and Harry Porter in the recommendation. All these movies have a similar storyline, and it is likely that if a user likes Lord of the Rings, they will be inclined to see the other parts and may also like Harry Potter and The Hobbit.

4.0.4. Polarity of Users and Items

Polarity represents the sensitivity of our ratings and the interpretation given to a user. A polarizing movie is a movie with a high variance in the rating score. The variance shows the degree deviation from the mean (how far a score lies from the mean). If an individual gets a high score, it means they really like the move, and if they get a low score, it means they really dislike the. movie. We check polarity by computing the Euclidean distance of the latent vector of items. The most polarizing movie is `Plan 9 from Outer Space (1959)`, taking popularity greater than

Title
Room, The (2003)
Adventures Of Sherlock Holmes And Dr. Watson: The Twentieth Century Approaches (1986)
The Adventures of Sherlock Holmes and Dr. Watson: The Hound of the Baskervilles (1981)
Night Watch (Nochnoy dozor) (2004)
Tyler Perry's Diary of a Mad Black Woman (2005)
Men with Brooms (2002)

Table 3. Top 20 recommendation for Lord of the rings(2001)

1000, while the least polarizing movie is `Escape from Alcatraz (1979).`

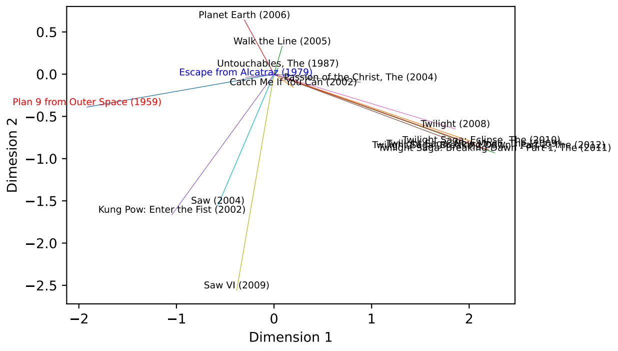


Figure 12. Top 20 Polarizing move and the 10 Least polarizing movies

4.0.5. Model with feature vector

The modified version of the model with feature vector enabled us to include genre as part of the model. The incorporation has a significant impact on Cold Start movies as it ideally pulls it into the feature space of movies with similar genres as the Cold Start movie instead of being at its origin.

K	Train Loss	RMSE Train	RMSE Test
2	31018487.2327	0.8179	0.7886
20	19939722.5306	0.8422	0.6339

Table 4. Training and Testing Performance Metrics

We show that the embedding is acceptable if models embed children's movies away from horror movies.

5. Conclusion

Matrix factorization has proven to be effective in recommender systems because it helps solve the problem of sparse

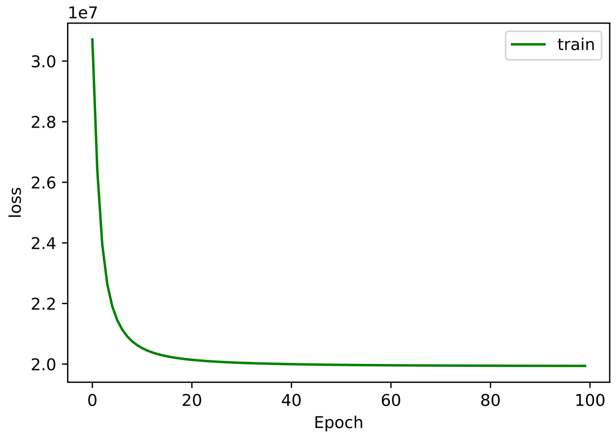


Figure 13. Log likelihood for k=20

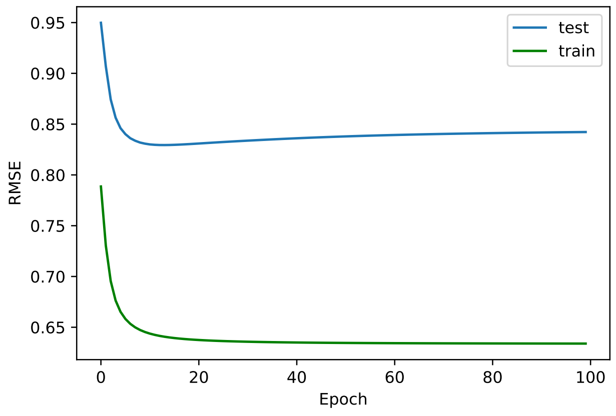


Figure 14. Feature RMSE

signals in observed data. It is also optimal for computation and memory management, as it is easier to save a latent vector than it would be to hold the whole user-item matrix in memory. In our report, we implemented matrix factorization for three degrees of complexity and observed that the model with the latent vector and feature vector would perform better. We used the latent vector to predict a new user and obtain a good result.

5.1. Further Research

To further this research, we should look into the implementation of recommendations for data from implicit feedback and experiments on the A/B test for all our models.

Code can be found on GitHub.
GitHub: https://github.com/TobiEbeneza/ML_AT_SCALE

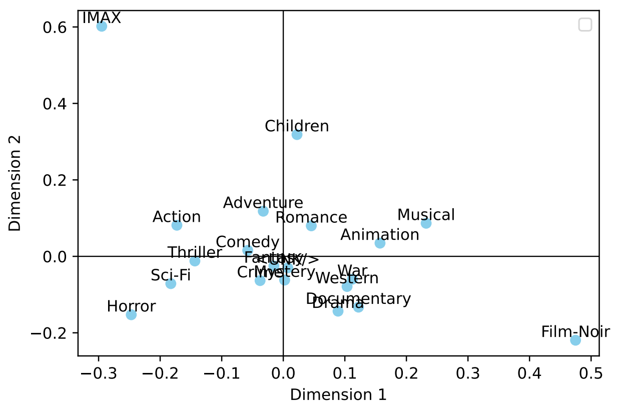


Figure 15. feature embedding for k=2

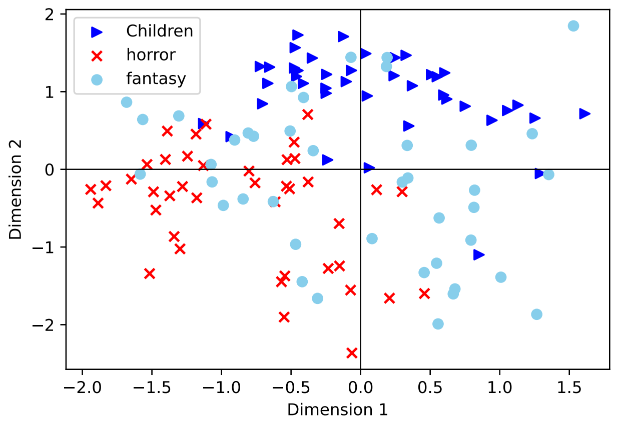


Figure 16. item embedding children, horror, and fantasy movies for k=2

References

Movielens dataset. https://grouplens.org/datasets/movielens/. Accessed: May 13, 2024.

Recommendation system. https://www.nvidia.com/en-us/glossary/recommendation-system/. Accessed: May 13, 2024.

Power law. https://en.wikipedia.org/wiki/Power_law. Accessed: May 13, 2024.

Harper, F. M. and Konstan, J. A. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, V (N):20, 2015. doi: XXXX.

Koren, Y., Bell, R., and Volinsky, C. Matrix factorization techniques for recommender systems. *IEEE Computer Society*, 2009.

Zhou, Y., Wilkinson, D., Schreiber, R., and Pan, R. Large-scale parallel collaborative filtering for the netflix prize. In Fleischer, R. and Xu, J. (eds.), *Algorithmic Aspects in Information and Management. AAIM 2008. Lecture Notes in Computer Science*, volume 5034, Berlin, Heidelberg, 2008. Springer. doi: 10.1007/978-3-540-68880-8_32. URL https://doi.org/10.1007/978-3-540-68880-8_32.

A. Appendix

A.1. Derivatives

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial u_3^{(u)}} &= \frac{\partial}{\partial u_3^{(u)}}(-\frac{\lambda}{2}\sum_n \sum_m (r_{mn} - (u_m^T v_n - b_m^{(u)} + b_n^{(i)})^2 - \\
&\frac{\tau}{2}\sum_m u_m^T u_m - \frac{\tau}{2}\sum_m v_n^T v_n - \frac{\gamma}{2}\sum_n b_m^{(u)2} - \\
&\frac{\gamma}{2}\sum_n b_n^{(i)2}) \\
&= -\lambda \sum_{n\in\Omega(3)} (r_{3n} - (u_3^T v_n - b_3^{(u)} + b_n^{(i)})).v_n - \\
&\tau u_3 \\
0 &= -\lambda \sum_{n\in\Omega(3)} (r_{3n} - (b_3^{(u)} + b_n^{(i)})).v_n \\
&-\lambda \sum_{n\in\Omega(3)} v_n^T v_n u_3 - \tau u_3 \\
0 &= -\lambda \sum_{n\in\Omega(3)} (r_{3n} - (b_3^{(u)} + b_n^{(i)})).v_n \\
&-(\lambda \sum_{n\in\Omega(3)} v_n^T v_n - \tau I)u_3 \\
(\lambda \sum_{n\in\Omega(3)} v_n^T v_n - \tau I)u_3 &= -\lambda \sum_{n\in\Omega(3)} (r_{3n} - (b_3^{(u)} + b_n^{(i)})).v_n \\
u_3 &= (\lambda \sum_{n\in\Omega(3)} v_n^T v_n - \tau I)^{-1} - \lambda \sum_{n\in\Omega(3)} (r_{3n} - (b_3^{(u)} + b_n^{(i)})).v_n
\end{aligned}$$