Matrix Factorization Movie Recommendation System

Tlotlo Oepeng [1]

Abstract

Recommender systems assist customers in discovering new content, goods, and services that meet their needs. In this report, we investigate the popular matrix factorization technique and use it to build a movie recommender system implemented with an alternating least squares algorithm (ALS). We focus on the algorithms, implementation of mathematical foundations, optimising and analysing our implementation.

1. Introduction

Thanks to broadband internet, everyone now have access to almost infinite digital content. While the volume of available content continues to grow exponentially, The irony is, it has become increasingly difficult to discover content that is engaging, entertaining, or even relevant among the plethora of options. Perhaps this phenomenon can be attributed to information overload (Bawden & Robinson, 2008). The problem issue isn't lack of options; rather, it is challenging to find content that fits each person's particular interests and preferences while simultaneously searching through mountains of data. The way to get around this problem is to make a recommender system, which has emerged as a solution by providing users with personalised content suggestions based on their preferences, helping them navigate through massive datasets (Ricci et al., 2010; Jannach et al., 2010). This is even becoming a field as evidenced by the the abundance dedicated of periodicals.

There are various types of recommendation systems, each with distinct methodologies. One such are non-personalized recommendations are the first category of recommender systems. They are designed with all users in mind, regardless of their preferences. These techniques all have the same message, which is that identity is irrelevant. Amazon and Google are good example as we see the same thing as someone else performing the same search. The recommendations are periodically good, albeit not being tailored, as they are popular and widely available.

Recommendations based on similar item attributes are known as content-based recommendations. For instance, if one item, like movie A, is similar to another item, such as movie B, the assumption is that someone who likes movie A might also like movie B. This assumption is made by comparing the characteristics of the items. The advantage of using item attributes is that it offers recommendations for any items that have attribute data, even if its new. This works by creating profiles for items using their attributes, which can then be compared them using mathematical methods. For instance, platforms like Pandora use this approach to recommend music tracks based on a user's listening history and the characteristics of songs. Although effective in certain contexts, content-based systems are limited by their reliance on item attributes, often struggling to offer diverse recommendations when user profiles are sparse or when the system encounters new items (the "cold-start" problem) (Lops et al., 2010)

One of the most successful methods in large-scale recommender systems is collaborative filtering, which focuses on using user-item interactions rather than item attributes. In particular, matrix factorization techniques have become the dominant approach in collaborative filtering due to their ability to handle sparse data and provide accurate recommendations. These techniques gained prominence during the Netflix Prize competition, where the goal was to predict user ratings for unseen movies, and matrix factorization methods significantly outperformed traditional techniques such as nearest-neighbour models (Koren et al., 2009). Collaborative filtering techniques can be broadly categorised into two main approaches: memory-based and model-based methods. Memory-based methods, such as user-based and item-based nearest-neighbour algorithms, directly compute similarities between users or items based on their historical interactions, making recommendations by identifying neighbours with similar preferences (Sarwar et al., 2001). While simple to implement, memory-based methods suffer from scalability issues as they require computing similarity scores for potentially millions of users or items. Model-based approaches, on the other hand, learn latent factors

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

from user-item interaction data using techniques such as matrix factorization (Koren et al., 2009), Singular Value Decomposition (SVD) (Paterek, 2007), or more recent methods like neural collaborative filtering (He et al., 2017). These models generalise well to sparse datasets by capturing underlying patterns in user behaviour and item features, leading to improved accuracy and scalability. Collaborative filtering, particularly in its model-based forms, has seen widespread adoption in various real-world applications, including movie recommendation (Netflix), e-commerce (Amazon), and music streaming (Spotify). However, despite its successes, collaborative filtering remains susceptible to the cold-start problem, where it struggles to provide accurate recommendations for new users or items that lack sufficient interaction history.

Matrix factorization works by decomposing a large user-item matrix into lower-dimensional matrices representing latent features of users and items. This enables the system to predict missing ratings by computing the inner product of the corresponding latent factors. Among the most popular matrix factorization algorithms is Alternating Least Squares (ALS), which iteratively optimises user and item latent vectors to minimise the error between predicted and actual ratings (Zhou et al., 2008). ALS is particularly suited for large-scale systems due to its scalability and ability to parallelise computations.

In this work, we implement a movie recommender system using matrix factorization with the ALS algorithm. We train the model on the MovieLens dataset, a widely used benchmark in recommender system research, and evaluate its performance by optimising hyperparameters and measuring prediction accuracy using Root Mean Square Error (RMSE). Our study aims to demonstrate the effectiveness of matrix factorization in producing high-quality recommendations while balancing accuracy and computational efficiency.

2. Matrix Factorization with Alternating Least Square(ALS)

Matrix factorization is a widely used method in recommendation systems for predicting user preferences based on historical data. The goal is to factorise a large user-item interaction matrix (such as movie ratings) into two smaller matrices that represent latent features of users and items. These latent features can then be used to predict missing entries in the matrix, such as a user's unknown rating for a particular movie. The underlying idea is simple: each user and each item (movie) is represented as a vector in a shared feature space. For example, a user might have a vector that encodes their preferences across several dimensions (like their tendency to enjoy action or comedy movies), and each item will have a vector representing its characteristics in those same dimensions. By finding the dot product of these two vectors, we can predict how much the user will like the item.

2.1. L is for Learning

In the Alternating Least Squares (ALS) approach, we start with an incomplete matrix R (the user-item interaction matrix), where each entry r_{ij} represents the rating user i has given to item j, if available. The objective is to predict the missing values in R by approximating it as the product of two lower-dimensional matrices: a user matrix U (where each row represents a user) and an item matrix V (where each row represents an item). This gives:

$$R \approx U^\intercal V \tag{1}$$

Each user i is represented by a latent vector u_i, and each item j is represented by a latent vector v_j. The predicted rating $\hat{r}_{ij}$ is the dot product of u_i and v_j:

$$\hat{r}_{ij} = u_i^\intercal v_j \tag{2}$$

To *learn* these latent vectors, the ALS algorithm minimises the difference between the observed ratings r_{ij} and the predicted ratings $\hat{r}_{ij}$. The model is trained by solving the following optimisation problem:

$$\min_{U,V} \sum_{(i,j)\in\mathcal{R}} \left(r_{ij} - u_i^T v_j\right)^2 + \lambda \left(\|u_i\|^2 + \|v_j\|^2\right) \tag{3}$$

This objective function seeks to minimise the error between the observed and predicted ratings, while also including a regularisation term (controlled by λ) to prevent overfitting by discouraging overly complex latent vectors.

If we adopt a Bayesian framework for modelling the observed ratings r_{ij}. We assume that each rating r_{ij} is generated from an underlying latent factor model with Gaussian noise ϵ_{ij}. Specifically, for each user i and item j, the rating can be expressed as:

$$r_{ij} = u_i^T v_j + \epsilon_{ij} \tag{4}$$

the likelihood of observing the rating r_{ij} given the latent factors u_i and v_j is given by:

$$p(r_{ij}|u_i, v_j) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(r_{ij} - u_i^T v_j)^2}{2\sigma^2}\right) \tag{5}$$

Assuming the errors are independent and identically distributed (i.i.d.), the joint likelihood of observing all ratings R can be expressed as:

$$p(R|U,V) = \prod_i \prod_{j \in \Omega(i)} p(r_{ij}|u_i, v_j) \tag{6}$$

specifying prior distributions for the latent factors U and V as being drawn from Gaussian distributions:

$$p(U) = \prod_{i=1}^{M} p(u_i) \quad \text{where } p(u_i) \sim \mathcal{N}(0, \sigma_u^2)$$
$$p(V) = \prod_{j=1}^{N} p(v_j) \quad \text{where } p(v_j) \sim \mathcal{N}(0, \sigma_v^2)$$

the posterior distribution for the latent factors given the observed ratings can be expressed as:

$$p(U,V|R) \propto p(R|U,V)p(U)p(V) \tag{7}$$

it is far simpler to work with logs

$$\log p(U,V|R) = \log p(R|U,V) + \log p(U) + \log p(V) \tag{8}$$

$$\begin{aligned}\log p(U,V|R) &= \sum_i \sum_{j \in \Omega(i)} \log p(r_{ij}|u_i, v_j) + \sum_{i=1}^{M} \log p(u_i) \\ &\cdots + \sum_{j=1}^{N} \log p(v_j) \\ &= \sum_i \sum_{j \in \Omega(i)} \Big(- \frac{(r_{ij} - u_i^T v_j)^2}{2\sigma^2} - \frac{1}{2\sigma_u^2}\|u_i\|^2 \\ &\cdots - \frac{1}{2\sigma_v^2}\|v_j\|^2 + \text{const} \Big)\end{aligned}$$

Maximising the log-posterior is equivalent to minimising the negative log-posterior. Thus, we define the objective function and for convenience, rename the constants (σ's) to be more appealling

$$\begin{aligned}L(U,V) = \frac{\lambda}{2} \sum_i \sum_{j \in \Omega(i)} (r_{ij} - u_i^T v_j)^2 \\ + \frac{\tau}{2} \sum_i \|u_i\|^2 + \frac{\tau}{2} \sum_j \|v_j\|^2\end{aligned} \tag{9}$$

this is a comprehensive Bayesian perspective model is not too far of from the starting one which is interesting since ratings are categorical one would have assumed this should not work but it does and so we leave it up to the reader to comprehend for himself why this be.

2.2. Algorithm - ALS

As the name suggests, learning happens through an alternating process. Instead of trying to update both user and item vectors at the same time (which would be computationally difficult), ALS alternates between fixing one set of vectors and solving for the other:

Fix V and Optimize U:

To find the optimal U, we take the derivative with respect to u_i and set it to zero:

$$\frac{\partial L(U,V)}{\partial u_i} = -\lambda \sum_{j \in \Omega(i)} (r_{ij} - u_i^T v_j) v_j + \tau u_i = 0 \tag{10}$$

$$\begin{aligned}\lambda \sum_{j \in \Omega(i)} (r_{ij} - u_i^T v_j) v_j &= \tau u_i \\ \tau u_i &= \lambda \sum_{j \in \Omega(i)} (r_{ij} v_j - u_i^T v_j v_j) \\ u_i &= \frac{1}{\tau} \left(\lambda \sum_{j \in \Omega(i)} (r_{ij} v_j) - \lambda \sum_{j \in \Omega(i)} (u_i^T v_j) v_j \right)\end{aligned}$$

and the final update rule

$$u_i = \left(\lambda \sum_{j \in \Omega(i)} v_j v_j^T + \tau I \right)^{-1} \left(\lambda \sum_{j \in \Omega(i)} r_{ij} v_j \right) \tag{11}$$

Fix U and Optimize V:

Now, we fix U and optimize for V in a similar fashion

$$\min_V L(U,V) = \frac{\lambda}{2} \sum_j \sum_{i \in \Omega^{-1}(j)} (r_{ij} - u_i^T v_j)^2 + \frac{\tau}{2} \sum_j \|v_j\|^2 \tag{12}$$

we take the derivative with respect to v_j and set it to zero:

$$\frac{\partial L(U,V)}{\partial v_j} = -\lambda \sum_{i \in \Omega^{-1}(j)} (r_{ij} - u_i^T v_j) u_i + \tau v_j = 0$$

$$\gamma v_j = \lambda \sum_{i \in \Omega^{-1}(j)} (r_{ij} u_i - v_j^T u_i u_i)$$

$$v_j = \frac{1}{\gamma} \left(\lambda \sum_{i \in \Omega^{-1}(j)} (r_{ij} u_i) - \lambda \sum_{i \in \Omega^{-1}(j)} (v_j^T u_i) u_i \right)$$

hence the update Rule for V

$$v_j = \left(\lambda \sum_{i \in \Omega^{-1}(j)} u_i u_i^T + \gamma I \right)^{-1} \left(\lambda \sum_{i \in \Omega^{-1}(j)} r_{ij} u_i \right) \tag{13}$$

Algorithm 1 Alternating Least Squares (ALS)

Initialize user latent factors $\mathbf{U}$ and item latent factors $\mathbf{V}$.
repeat
  for each user m **do**
    Fix $\mathbf{U}$, update $\mathbf{V}$, b_j.
  end for
  for each item n **do**
    Fix $\mathbf{V}$, update $\mathbf{U}$, b_m.
  end for
until maximum number of iterations is reached

2.3. Incorporating biases

Biases are crucial in addressing systematic tendencies in user ratings and item popularity, allowing models to adapt to the inherent differences in user-item interactions. In recommendation systems, biases consist of user-specific and item-specific intercepts that account for variations in ratings that cannot be attributed solely to latent factors. Certain users may consistently rate items higher or lower than average, while some items may naturally garner higher ratings due to their perceived quality or popularity.

For example, consider (Koren et al., 2009)'s illustration: suppose the overall average rating for movies is 3.7 stars. If a particular movie, like *Titanic*, is perceived as superior and receives a bias of 0.5 stars, while a user named Joe tends to rate movies 0.3 stars lower than average, we can estimate Joe's rating for*Titanic* using the $\hat{r}_{ij} = \text{Average Rating} + b_j + b_m$ gives $\hat{r}_{\text{Joe, Titanic}} = 3.7 + 0.5 - 0.3 = 3.9$: This calculation highlights how biases can refine rating predictions by accounting for individual user behaviours and item characteristics.

Moreover, biases serve as a fundamental mechanism for addressing the cold-start problem, where new users or items lack sufficient interaction data to yield reliable recommendations (Schein et al., 2002). By incorporating user bias b_m and item bias b_j, the prediction of the rating is modified from $u_i^T v_j$ Eq (**equation**) to:

$$\hat{r}_{ij} = u_i^T v_j + b_m + b_n + \epsilon \tag{14}$$

To derive the update rules for the biases, we adjust our loss function to include the bias terms and their regulisers

$$L(U, V, b) = -\frac{\lambda}{2} \sum_i \sum_{j \in \Omega(i)} (r_{ij} - (u_i^T v_j + b_i + b_j))^2 - \frac{\gamma}{2} \sum_i b_i^2 - \frac{\gamma}{2} \sum_j b_j^2 + \cdots \tag{15}$$

Here, the first term represents the squared error for predictions, while the second and third terms regularise the user and item biases, respectively.

To find the optimal biases, we compute the derivatives of the loss function with respect to b_i and b_j, setting these derivatives to zero.

Updating User Bias b_i

$$\frac{\partial L}{\partial b_i} = -\lambda \sum_{j \in \Omega(i)} (r_{ij} - (u_i^T v_j + b_i + b_j)) \cdot (-1) - \gamma |\Omega(i)| b_i \tag{16}$$

$$\lambda \sum_{j \in \Omega(i)} (r_{ij} - (u_i^T v_j + b_i + b_j)) - \gamma |\Omega(i)| b_i = 0$$

$$\gamma |\Omega(i)| b_i = \lambda \sum_{j \in \Omega(i)} (r_{ij} - (u_i^T v_j + b_j)) - \lambda \sum_{j \in \Omega(i)} b_i$$

$$\gamma |\Omega(i)| b_i + \lambda b_i = \lambda \sum_{j \in \Omega(i)} (r_{ij} - (u_i^T v_j + b_j))$$

Thus, the update rule for b_i becomes:

$$b_i = \frac{\lambda \sum_{j \in \Omega(i)} (r_{ij} - (u_i^T v_j + b_j))}{\gamma |\Omega(i)| + \lambda} \tag{17}$$

Updating Item Bias b_j

Similarly, we derive the update rule for b_j by taking the derivative with respect to b_j:

$$\frac{\partial L}{\partial b_j} = -\lambda \sum_{i \in \Omega^{-1}(j)} (r_{ij} - (u_i^T v_j + b_i + b_j)) \cdot (-1) - \gamma |\Omega^{-1}(i)| b_j \tag{18}$$

Setting this to zero gives us:

$$0 = \lambda \sum_{i \in \Omega^{-1}(j)} (r_{ij} - (u_i^T v_j + b_i + b_j)) - \gamma |\Omega^{-1}(i)| b_j$$

This simplifies to:

$$\gamma|\Omega^{-1}(i)|b_j = \lambda \sum_{i \in \Omega^{-1}(j)} (r_{ij} - (u_i^T v_j + b_i + b_j))$$

Therefore, the update rule for b_j is:

$$b_j = \frac{\lambda \sum_{i \in \Omega^{-1}(j)} (r_{ij} - (u_i^T v_j + b_i))}{\gamma|\Omega^{-1}(j)| + \lambda} \tag{19}$$

3. Dataset

3.1. Overview of the Dataset

The dataset used in this project is the popular MovieLens dataset, which is the go-to for recommender system research. Curated by the GroupLens research group at the University of Minnesota, the dataset contains user ratings of movies, alongside various metadata about the movies themselves. We used two versions of the dataset: the **100k dataset**, which includes 100,000 ratings from 943 users on 1,682 movies, and the **25M dataset**, which contains 25 million ratings from 162,541 users on 62,423 movies. The ratings in the 25M dataset span from January 9, 1995, to November 21, 2019 (Harper & Konstan, 2015).

The MovieLens 100k dataset is used for initial training and optimisation, while the larger 25M dataset is employed for final model evaluation and validation. This combination allows us to explore how the model performs on both a smaller, more manageable dataset, and a larger, more realistic dataset representative of real-world applications. Each entry in the MovieLens dataset corresponds to a user rating a movie, with key attributes including:

- User ID: A unique identifier for each user.
- Movie ID: A unique identifier for each movie.
- Rating: A score from 1 to 5, indicating how much the user liked the movie.
- Timestamp: The time when the user rated the movie.
- Genres: A list of genres associated with each movie.
- Tags: User-applied tags, which provide additional information or context for the movie.

The 100k dataset contains pre-processed train-test splits, while the 25M dataset requires additional processing for splitting into training and test sets. The rich metadata, including genres and tags, provides valuable context for recommendations, especially when exploring content-based approaches in addition to collaborative filtering.

3.2. Distribution and patterns in dataset

Here, we examine the data sets in depth and create visual representations of them. This will allow us to get a quick and visual overview of the data. We will also provide summaries of the data for the ml-25 set and 100k data set. Below are distributions of data for ratings and the degrees to which movies were rated or users rated movies.

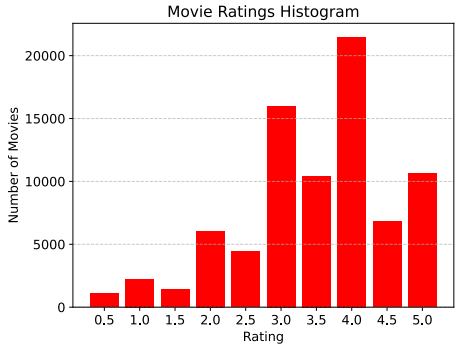

Figure 1. 100k data set rating distributions

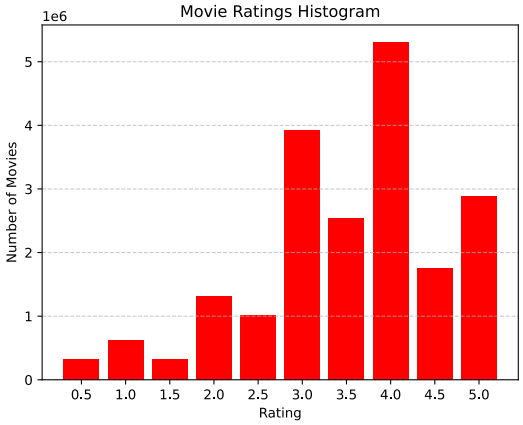

Figure 2. 25M dataset ratings distribution

The ratings in both datasets exhibit a slight skew towards higher values, with most ratings falling between 3 and 5 stars. The average rating in the 25M dataset is approximately 3.5 stars, indicating that users tend to rate movies positively. Figure [1] shows the distribution of ratings in the 100k dataset, which follows a similar trend to the 25M dataset Figure [2].

Another notable pattern is the distribution of user activity. A small number of users account for a disproportionate share of the total ratings, with many users rating only a handful of movies, while a minority of power users have rated thou-

sands. This pattern aligns with Zipf's Law (Newman, 2005), which suggests that a few items or users are responsible for most of the interactions in a system. Figure 2 illustrates the long-tail distribution of user ratings in the 25M dataset.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(r - \hat{r}_{pred})^2} \tag{20}$$

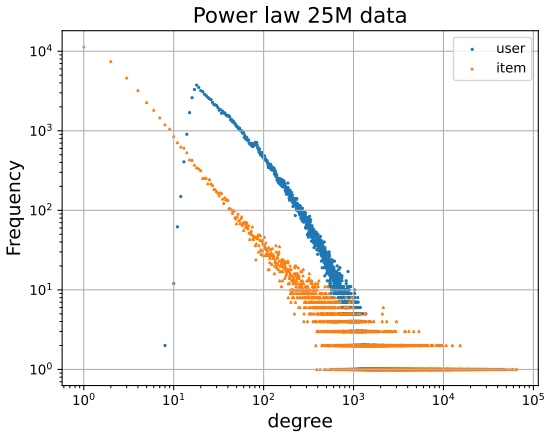

Figure 3. Frequency of user or items to their degree

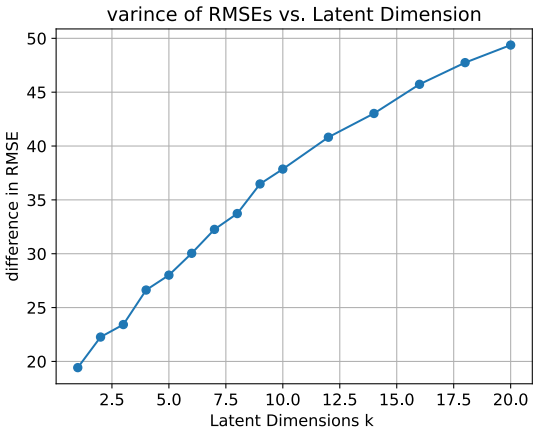

Figure 5. RMSEs variance with k on on 25M

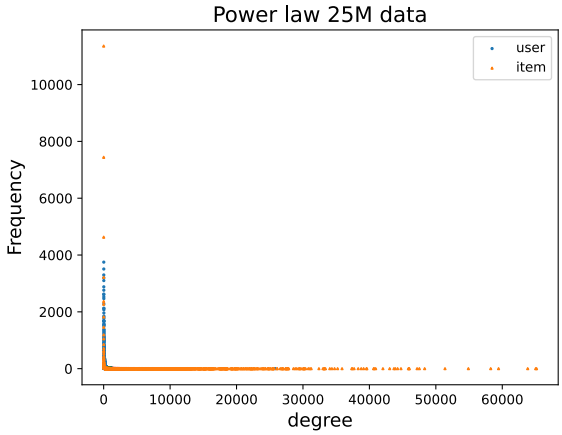

Figure 4. Logarithmic frequencies on log degree

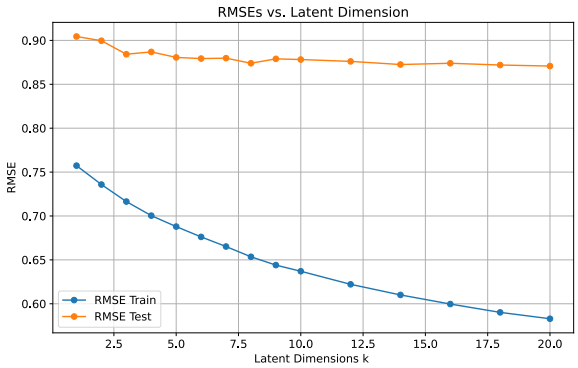

Figure 6. RMSE with k

4. Results

4.1. Optimisation

The models performance depend on choice of hyper parameters. Fourtunately, the effect of these can be measured using Root Mean Square Error (RMSE). which is how close estimates were to the actual values. The choice of hyper parameters is crucial for optimising the model. In order to find the most optimal hyper parameters latent vector k, expeiments were done to make to choose any λ, γ, β where the loss converges a pot of the percentage difference of the test and train RMSE made along with plot of rmse vs k.

The percentage difference in RMSE represent how far the test RMSE is from train. From Fig [5] , the gap shows the the distance grow linearly with k, at $k = 2$ its 22% of the test and with 20 latent dimensions it almost half. The shows that low latent dimensions $k < 10$ the model would be able to generalise well to new data. However choosing $k \approx 1$ is not necessarily ideal because from Fig [6] the RMSE decreases with higher k, because the model is able to fit better with higher k, Hence the trade off was made that $k = 5$ is the best dimension with the RMSE being low enough and the model not over-fitting. Find optimal λ, γ, β was implemented through a simple grid search and gave the

best hyper parameters as $\lambda = 0.01, \gamma = 0.01, \beta = 0.1$ as per Fig [9]. Optimisation was was attempted in the 100k

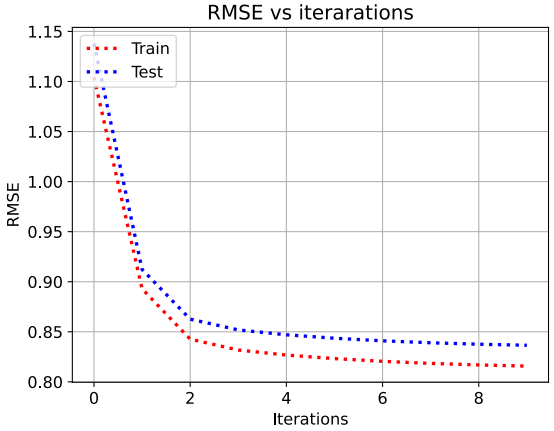

Figure 7. loss on 25M data, $\lambda = 0.01,\ \gamma = 0.01\ \tau = 0.1$

data but the data was found to be noisy and gave weird kinks Fig [8]

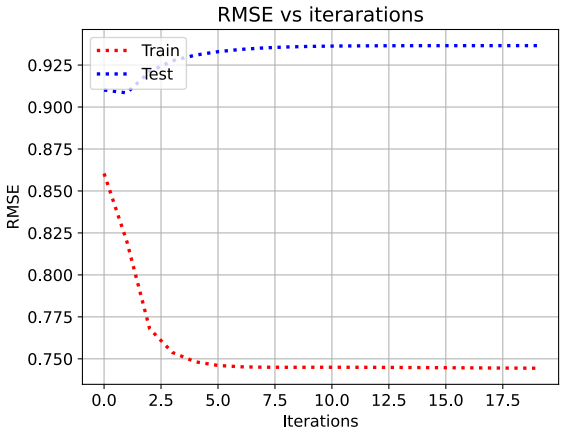

Figure 8. RMSEs on 100k $\lambda = 0.001,\ \gamma = 0.01\ \tau = 0.01$

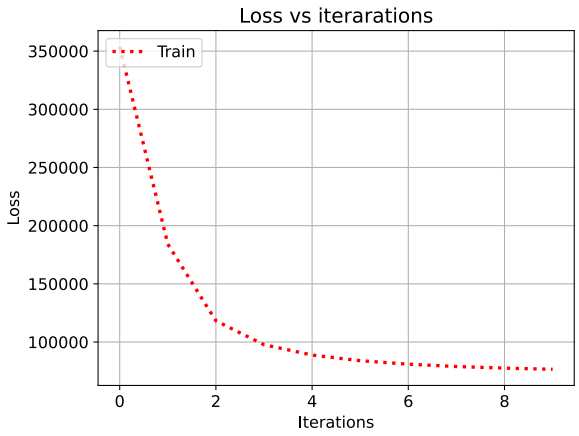

Figure 9. loss on 25M data, $\lambda = 0.01,\ \gamma = 0.01\ \tau = 0.1$

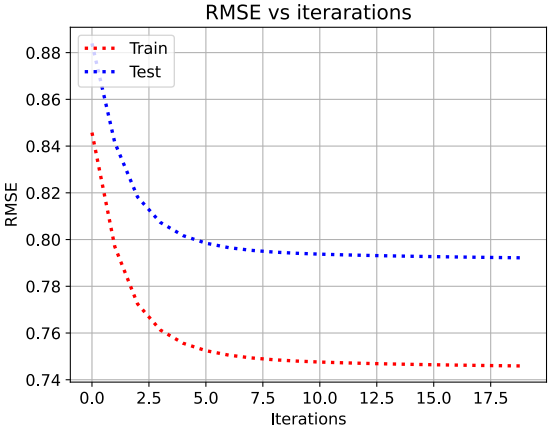

Figure 10. Test train RMSE's 25M data

4.2. Predictions and recommendations

In the context of testing recommendations for quality and relevance, consider a hypothetical scenario where we introduce a dummy user, from the CSV file, $u = 23$. This dummy user has rated "Lord of the Rings" with a 5-star rating. In this case, the vector representation of this dummy user will be identical to the vector representation of the item "Lord of the Rings." Consequently, the dot product of these two vectors will yield the top recommendations for this user. From the results, it's apparent that most of the top recommendations are from the "Lord of the Rings" series, with a few unexpected movies also appearing in top positions. This anomaly can be attributed to the high popularity of those movies, leading to their frequent recommendation to users. This is particularly notable in the case of "The Princess Bride (1987)," which tends to be recommended to almost any user.

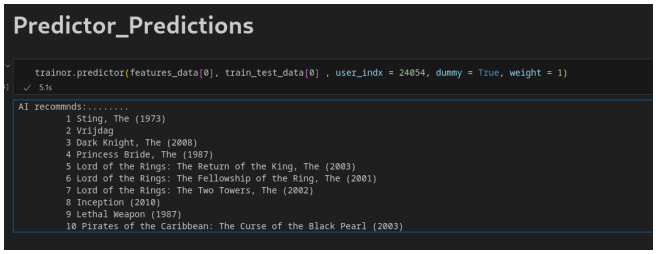

Figure 11. Test train RMSE's 25M data

The introduction of a weight term for the movie biases, set at 0.5, helped mitigate this effect to some extent. However, it was challenging to ensure that all "Lord of the Rings" series movies consistently appeared as top recommendations. Achieving this would likely require adding a dummy user who rated 2 lord of the rings movies and retraining the model, which can be computationally expensive. In toto, the recommendations make sense!

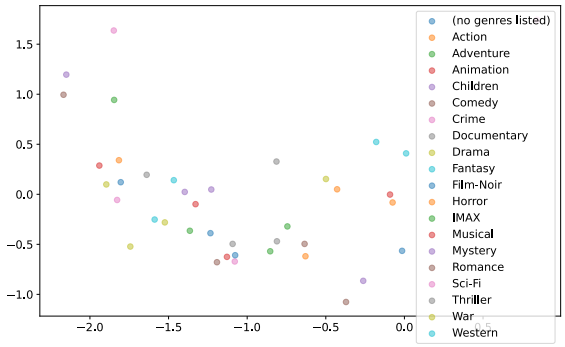

Figure 12. embeddings

The embedding in the recommendation system is a fundamental component that allows facilitates representing movies in a lower-dimensional vector space, facilitating the process of making personalized movie recommendations and testing the recommender. Each movie in the dataset is associated with a unique vector that captures its latent features. From the embedding movies with the same genre were chosen at random to observe their placement on the 3D spaces. overall movies of the same genre are typically not too far from each other, and also a few anomalies where some genres that are polar opposites are relatively close each other this can be attributed top the choice of plotting embedding, A movie might have 2 or more features.

References

Bawden, D. and Robinson, L. The dark side of information: overload, anxiety and other paradoxes and pathologies. *Journal of Information Science*, 35(2):180–191, November 2008. ISSN 0165-5515. doi: 10.1177/0165551508095781.

Harper, F. M. and Konstan, J. A. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):1–19, 2015.

He, X., Liao, L., Zhang, H., Nie, L., Hu, X., and Chua, T.-S. Neural Collaborative Filtering. *arXiv*, August 2017. doi: 10.48550/arXiv.1708.05031.

Jannach, D., Zanker, M., Felfernig, A., and Friedrich, G. *Recommender Systems: An Introduction*. Cambridge University Press, Cambridge, England, UK, September 2010. ISBN 978-0-52149336-9. doi: 10.1017/CBO9780511763113.

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

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Lops, P., de Gemmis, M., and Semeraro, G. Content-based Recommender Systems: State of the Art and Trends. In *Recommender Systems Handbook*, pp. 73–105. Springer, Boston, MA, Boston, MA, USA, October 2010. ISBN 978-0-387-85820-3. doi: 10.1007/978-0-387-85820-3_3.

Newman, M. E. Power laws, pareto distributions and zipf's law. *Contemporary physics*, 46(5):323–351, 2005.

Paterek, A. Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of KDD cup and workshop*, volume 2007, pp. 5–8, 2007.

Ricci, F., Rokach, L., and Shapira, B. Introduction to Recommender Systems Handbook. In *Recommender Systems Handbook*, pp. 1–35. Springer, Boston, MA, Boston, MA, USA, October 2010. ISBN 978-0-387-85820-3. doi: 10.1007/978-0-387-85820-3_1.

Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. Item-based Collaborative Filtering Recommendation Algorithms. *Proceedings of ACM World Wide Web Conference*, 1, August 2001. doi: 10.1145/371920.372071.

Schein, A. I., Popescul, A., Ungar, L. H., and Pennock, D. M. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 253–260, 2002.

Zhou, Y., Wilkinson, D., Schreiber, R., and Pan, R. Large-Scale Parallel Collaborative Filtering for the Netflix Prize. In *Algorithmic Aspects in Information and Management*, pp. 337–348. Springer, Berlin, Germany, 2008. ISBN 978-3-540-68880-8. doi: 10.1007/978-3-540-68880-8_32.