

ENGR-UH 2017
NUMERICAL METHODS

Recommendation Systems

Submitted by:

Valentina Juarez Ortiz

Gelila Kebede

Hiba Assamaouat

FALL 2023



Contents

1	Introduction	1
2	Mathematical Formulation	4
2.1	Lagrangian Method	7
3	Numerical Methods	7
3.1	Inputs and Outputs	8
3.2	Pseudocode	9
4	How the Method Solves the Problem	13
4.1	Discussion	16

1 Introduction

Handling large sets of data often involves the use of matrices, as they facilitate the visualization of relationships and patterns within that data. In this project, we are aiming to model a recommendation system.

A **recommendation system** is essentially an algorithm that provides personalized suggestions (usually advertised products recommended to purchase and/or consume) to users of the internet based on information on the users' feedback given.

The person behind the first recommender model was American computer scientist, Elaine Rich, who developed it in 1979. Her initial intention was to make a system that would recommend her a book that she might like by asking her a few questions and then categorizes her under a certain "stereotype".(4)

We can model our system with a large matrix for which the users are listed down as the rows of the matrix (each user represents a row), and the products are the columns of the matrix. The entry in a matrix, corresponding to the intersection between a user and a product, would be the rating of the product submitted by the user. The same way we, as users of many platforms, choose to leave a like on a YouTube video, or give a restaurant a 3-star rating, all based on our experience.

Recommendation systems are used everywhere now. They are most commonly used for marketing

strategies in order to make more profit by having the right product reach the right customer, hence increasing the chance of purchase. It is not really applied in disciplines other than marketing, however, marketing is everywhere nowadays. The way engineering is involved in this is in the development of the system (software engineering).

The problem we are facing with recommendation systems is that users have the choice to leave a review or feedback on a particular products. Naturally enough, users cannot come across all possible products figuring in the database, and in the case where a user is interacting with a product, there is a considerable chance that the user would not leave feedback for that item. Although it might not be the most relevant thing in the context of this report, it is interesting to not the reasons why a user might or might not leave a review. In a more advanced level of designing a recommendation system, this information would be important to consider.

It is interesting to observe that a person dissatisfied with a product is more likely to leave a review (negative review of course) than someone who leaves satisfied (3).

This means that the matrix we are trying to create is going to be full of gaps. It would be lacking lots of data, making the whole system of recommendation incomplete and ineffective (See the simplified example in Fig. 1)













				
 A		?	?	
 B	?	?		?
 C		?		?

Figure 1: Example of a simple matrix recommendation system for movies

In the case below (Fig. 2), we can see that users A, B, and C had similar reviews on movie 3. Assuming that it was not an objectively great movie, that there also were users who did not find it that good, we

could suppose that they have similar tastes in movies (maybe they like the same genre, or the same actor/director). These similarities between users allow the system to classify them under the same category or as the "same" user. This enables the possibility of extrapolating data, such as user A's missing review for movie 4. Therefore, the objective of this project is to find a way to fill in those gaps by somehow extrapolating data using similar user trends.










	 1	 2	 3	 4
 A		?		?
 B	?	?		
 C	?	?		

Figure 2: Example of a simple matrix recommendation system where users have similar taste.

Not everyone rates the same way. Some leave more reviews than others, some not at all. Therefore we will conduct two types of experiments: the first set of simulations will be on a group of individuals with the same probabilities of rating, let us call them the "unbiased users"; then we will try to make it more realistic by making some users more active than others, with different probabilities of leaving a review, let us call them the "biased users".

We aim to demonstrate solving matrix completion problems using numerical methods. As it can be expected, recommender systems have to deal with very large amounts of data. This can only happen by filtering the relevant information, like those products which each user has interacted with previously. Recommender systems are present in our everyday lives, applications include platforms like Netflix, YouTube and X, which use these systems to suggest content that the user is likely to engage with (6). In terms of engineering applications, recommendation systems technology has found applications in software engineering, data and knowledge engineering and persuasion technology. (7)

A Note on Cookies: Both recommender systems and cookies are user tracking systems to deliver personalized and targeted advertising. Cookies fulfill pretty much the same function as recommender systems, except that they facilitate the a faster retrieval of data, and in larger amounts, as they do not rely much on the users' will to provide the data, at least not directly. They could be seen as intrusive.

2 Mathematical Formulation

Let us now model the situation with an actual matrix R , whose rows and columns represent users and products respectively. Given that a normal user has probably not interacted with most of the vast available products, as mentioned in the introduction, we must expect the matrix to have missing entries, zero-entries ("0"). These values are what we are trying to fill in. This is mathematically equivalent to finding a new matrix that matches the already known ratings while minimizing the rank of the matrix.

Let R represent the rating matrix,

$$R = \begin{bmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,M-1} & r_{1,M} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,M-1} & r_{2,M} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ r_{N-1,1} & r_{N-1,2} & \cdots & r_{N-1,M-1} & r_{N-1,M} \\ r_{N,1} & r_{N,2} & \cdots & r_{N,M-1} & r_{N,M} \end{bmatrix} \quad (1)$$

Where $r_{n,m} \in \{-1, 0, +1\}$ corresponds to the rating given by user n to product m . A "+1" indicates a positive rating, a "-1" indicates a negative feedback, and "0" is the absence of feedback. We could have used any other set of values such as $r_{n,m} \in [1, 5]$ to mimic the star ratings, etc. But for the sake of simplicity, we will work with the positive-negative-null reviews (for example the like and dislike button on YouTube).

As previously mentioned, most of these values will be 0s. The objective of the Recommender System is to predict whether the 0s are likely to become -1s or 1s, therefore indicating whether the user is likely to interact well with the product or not.

The way to do so is to set up a mathematical model that seeks to find another matrix, X , that matches R on the non-zero entries, i.e. the existing ratings. Following, we aim to fill in the 0 entries with numbers that indicate the likelihood of a +1 rating. This is done by minimizing the rank of matrix X .

Rank

The rank of a matrix X , denoted $\text{rank}(X)$, is the maximum number of linearly independent rows or columns in the matrix

By minimizing the rank, we can maximize the similarity and find as much redundancy as possible to make recommendations. In other words, the more we repeat patterns in X , the higher the linear dependence in X and the lower the **rank** of X .

$$\underset{X \in \mathbb{R}^{N \times M}}{\text{minimize}} \text{rank}(X) = \underset{X \in \mathbb{R}^{N \times M}}{\text{minimize}} \|\vec{\sigma}(x)\|_0 \quad (2)$$

and the problem then becomes:

$$\text{rank}(X) : (X - R)_{n,m}^2 = 0, \text{ for } (n, m) - \text{pairs for which } R_{n,m} \neq 0 \quad (3)$$

It is important to note that the problem is an optimization problem, and therefore the matrix we obtain must be differentiable and convex, which is not the case.

Therefore **convexification** is required.

Convexification

Convexification, or convex relaxation, is the transformation of a non-convex function to a convex function by approximation in order to facilitate the solving of an optimization problem.

According to Candes and Recht (2008) (1), for sparse matrices we can replace the rank, $\text{rank}(X)$, with $\|X\|_*$, the nuclear norm of X , and get the same optimal solution. Hence, a good approximation of the problem becomes:

$$\underset{X \in \mathbb{R}^{N \times M}}{\text{minimize}} \|X\|_* : \sum_{(i,j) \in \Omega} (X_{i,j} - R_{i,j})^2 = 0 \text{ for } (n, m) - \text{pairs for which } R_{n,m} \neq 0 \quad (4)$$

However, while $\|X\|_*$ is convex, it is not differentiable. Indeed, the nuclear norm involves the summation of singular values. This means we will be performing a Singular Value Decomposition (SVD), not differentiable in our case, which is problematic as we wish to solve a minimization problem which, of course, involves the use of gradients (differentiation).

Therefore, we will decompose our matrix into $X = UP^T$. We call this "matrix factorization" or "matrix completion". The matrix X is not expected to not be symmetric nor squared therefore we need to use a decomposition that works for any real matrix, namely we will use the SVD, where $X = U\Sigma V^T$ where U and V . Where U and V are orthogonal matrices and the diagonal entries of Σ are called the singular values of X . An important feature of the SVD is that it relates to the rank of a matrix such that:

$\text{Rank}(X) =$ the number of non-zero elements among $\sigma(X) = \{\Sigma_{ii}\}$ (the singular values of X) (5)

Hence, minimizing the $\text{rank}(X)$ is equivalent to minimizing the 0-norm of $\sigma(X)$ denoted as $\|\vec{\sigma}(X)\|_0$. So we are left with the following expressions

$$\text{rank}(X) = \|\vec{\sigma}\|_0 = \# \text{ of non-zeros in } \vec{\sigma} \quad (6)$$

$$\underset{X \in \mathbb{R}^{N \times M}}{\text{minimize}} \text{rank}(X) = \underset{X \in \mathbb{R}^{N \times M}}{\text{minimize}} \|\vec{\sigma}(x)\|_0 \quad (7)$$

In our matrix decomposition where $X = UP^T$ we can interpret $U \in \mathbb{R}^{N \times r}$ as the user-feature matrix and $P \in \mathbb{R}^{M \times r}$ as the product-feature matrix.

Features

The features we are talking about here are some hidden characteristics of the products and the users. Product features can represent a movie genre, a price, a brand, or even a simple keyword. User features could be age, gender, device type, etc.

Therefore, $r \ll \min N, M$ and corresponds to the number of hidden features. From the compact form of the SVD we know that $X = U\Sigma V^T$. If we define

$$\begin{aligned} C &= U_r \Sigma^{1/2} \\ P &= V_r \Sigma^{1/2} \end{aligned} \quad (8)$$

then

$$X = CP^T \quad (9)$$

$$\underset{C \in \mathbb{R}^{N \times r}, P \in \mathbb{R}^{M \times r}}{\text{minimize}} \frac{1}{2} (\|C\|_F^2 + \|P\|_F^2) : \sum_{(i,j) \in \Omega} (X_{i,j} - R_{i,j})^2 = 0 \quad (10)$$

Where $\|C\|_F$ and $\|P\|_F$ are the Frobenius norms of C and P , respectively. The Frobenius norm of

a matrix X is defined as such:

$$\|X\|_F = \sum_i \sum_j X_{i,j}^2$$

(For error calculation. more on that in later part). Either take the nuclear norm, or the Frobenius norm

$$\|C\|_F^2 = \text{trace}(CC^T) = \text{trace}(U_r \tilde{\Sigma} U_r^T) \quad (11)$$

Note that the trace of a matrix is the sum of the entries in its diagonal.

2.1 Lagrangian Method

The Lagrangian Method is a method of minimization or maximization where the target function is under constraints of another function. This is done through the use of a Lagrange multiplier (Lagrangian λ) to combine target and constrained functions into one function through the following formulation:

$$\mathcal{L}(x, \lambda) = f(x) - \lambda \cdot g(x)$$

From this function, the extrema are found by taking the derivative of the Lagrangian with respect to each of the variables in the target and constraining formulas. The derivatives are set equal to zero, as standard when finding the minimum or maximum of a function. The system of equations produced by setting all the derivatives to zero is then solved simultaneously to find the values of the variables at the extrema.

While it is extremely useful for optimization, the Lagrangian is limited in use for this system. Because the recommend system is not convex or continuous, it cannot yield a global optimum. Lagrangian relies on differentiation, so it wouldn't be able to optimize the discrete values of a sparse recommender matrix. It also cannot guarantee that the results are global extrema in non-convex problems like this one. Finally, it only finds stationary points and cannot determine if the point is a maximum, minimum, or saddle point has been found, further limiting its use.

3 Numerical Methods

In order to solve our mathematical problem, we will resort to a numerical method called the Alternating Least Squares (ALS) algorithm. The ALS method is an iterative algorithm that we use as a solution to the minimization problem described in the previous part. Also known as collaborative filtering, we

use it in order to factorize the matrix R into U and P , such that $R = U^T P$. Recalling the nuclear norm minimization problem leads to the following equivalence:

$$\begin{aligned} & \text{minimize}_{X \in \mathbb{R}^{N \times M}} \quad \text{rank}(X) + \mu \sum_{(n,m) \in \Omega} (X - R)_{n,m}^2 \\ & \text{minimize}_{U \in \mathbb{R}^{N \times r}, P \in \mathbb{R}^{M \times r}} \quad \sum_{(n,m) \in \Omega} (UP^T - R)_{n,m}^2 + \frac{\lambda}{2} (\|U\|_F^2 + \|P\|_F^2) \end{aligned} \quad (12)$$

We proceed to take initial guesses for the matrices U and P (U^0 and P^0), and update the matrices one at a time by fixing the other. For instance, P^k is fixed while solving for U^{k+1} and so on. The update happens row by row by the algorithm below:

for each user $n = 1, \dots, N$

$$U_n^{[k+1]} := \left(\sum_{m \text{ observed}} P_m^{[k]} \left(P_m^{[k]} \right)^T + \lambda I_{M \times M} \right)^{-1} \sum_{m \text{ observed}} R_{n,m} P_m^{[k]T} \quad (13)$$

for each product $m = 1, \dots, M$

$$P_m^{[k+2]} := \left(\sum_{n \text{ observed}} U_m^{[k+1]} \left(U_m^{[k+1]} \right)^T + \lambda I_{M \times M} \right)^{-1} \sum_{n \text{ observed}} R_{n,m} U_m^{[k+1]T} \quad (14)$$

The iterations will continue until the relative error stops improving or a significantly enough small value for the error is reached. The method of alternating least squares allows us to write the solution to the problem in a closed form.

3.1 Inputs and Outputs

Inputs:

- Number of Users, N
- Number of Products, M
- Probability that the User has rated the Product, p_{rating}
- Number of latent features, r
- Regularization parameter, λ

Outputs:

- Recommendation Matrix, X
- Error of Recommendation System, $error_{\text{history}}$

3.2 Pseudocode

The pseudocode for the Unbiased User Recommender system is shown below.

Start

1. Clear all existing variables and close all figures
2. Set parameters:
 - $N = 1000$ (number of users)
 - $M = 1200$ (number of items)
 - $r = 10$ (number of latent features)
 - $k = 100$ (number of iterations)
 - $\lambda = 0.02$ (regularization parameter)
 - $p_{\text{rating_exist}} = 0.02$ (probability that a product is rated)
3. Generate a true ratings matrix (X_{true}) of size $N \times M$ with values -1, 0, or 1
4. Create a binary matrix (ω) of size $N \times M$, where each element is 1 with probability $p_{\text{rating_exist}}$, indicating if a rating exists
5. Initialize observed ratings matrix (R) as the element-wise product of X_{true} and ω
6. Initialize user features matrix (U) as an identity matrix of size $N \times r$
7. Initialize item features matrix (P) as an identity matrix of size $M \times r$
8. Set lambda scaled identity matrix (lam_I) as lambda times an identity matrix of size $r \times r$
9. Initialize an array (error_history) of size $k+1$ to store error values, and calculate initial error
10. Perform ALS algorithm:

- For each iteration up to k:
 - a. Update U matrix:
 - i. For each user n:
 - Find indices of rated items
 - Initialize two zero matrices (US1 and US2) of size $r \times r$ and $r \times 1$, respectively
 - For each rated item:
 - * Update US1 and US2 based on item features (P) and observed ratings (R)
 - Update the feature vector for user n in U
 - b. Update P matrix:
 - i. For each item m:
 - Find indices of users who rated the item
 - Initialize two zero matrices (PS1 and PS2) of size $r \times r$ and $r \times 1$, respectively
 - For each user who rated the item:
 - * Update PS1 and PS2 based on user features (U) and observed ratings (R)
 - Update the feature vector for item m in P
 - c. Calculate and store the error for the current iteration
- 11. Calculate the final recommendation matrix (X) as the product of U and P'
- 12. Display error after k iterations
- 13. Plot the error convergence over iterations

End

The pseudocode for the Biased User Recommender system is shown below.

Start

1. Clear all existing variables and close all figures
2. Set parameters:

- $N = 1000$ (number of users)
 - $M = 1200$ (number of items)
 - $r = 10$ (number of latent features)
 - $k = 500$ (number of iterations)
 - $\lambda = 0.02$ (regularization parameter)
 - $p_{\text{rating}_1} = 0.10$ (probability that a product is rated for first user group)
 - $p_{\text{rating}_2} = 0.05$ (probability that a product is rated for second user group)
 - $\text{biased_Split} = 0.05 * N$ (index for biased sampling split)
3. Generate a true ratings matrix (X_{true}) of size $N \times M$ with values -1, 0, or 1
 4. Initialize a binary matrix (ω) for rating existence:
 - For the first ' biased_Split ' users, set ratings with probability p_{rating_1}
 - For the remaining users, set ratings with probability p_{rating_2}
 5. Initialize observed ratings matrix (R) as the element-wise product of X_{true} and ω
 6. Initialize user features matrix (U) and item features matrix (P) as identity matrices of size $N \times r$
 7. Set λ scaled identity matrix (lam_I) as λ times an identity matrix of size $r \times r$
 8. Initialize an array (error_history) to store error values and calculate the initial error
 9. Perform ALS algorithm:
 - For each iteration up to k :
 - a. Update U matrix:
 - i. For each user n :
 - Find indices of rated items
 - Initialize matrices (US_1 and US_2) for updates
 - Update these matrices based on item features (P) and observed ratings (R)
 - Update the feature vector for user n in U
 - b. Update P matrix:
 - i. For each item m :
 - Find indices of users who rated the item

- Initialize matrices (PS1 and PS2) for updates
- Update these matrices based on user features (U) and observed ratings (R)
- Update the feature vector for item m in P

c. Calculate and store the error for the current iteration

10. Calculate the final recommendation matrix (X) as the product of U and P'

11. Display the final recommendation matrix and error after k iterations

12. Plot the error convergence over iterations

End

4 How the Method Solves the Problem

Thanks to the ALS algorithm we are able to obtain a constrained approximation of minimizing matrices in the decomposition of X . By fixing U and P consecutively we obtain a quadratic form of the function which can then be solved directly. The matrix also becomes more accurate to the real matrix listed as the number of iterations increases. This is true because the algorithm shrinks the error in each step.

The error is initially:

$$Error(U, P) = \frac{1}{2}(\|U\|_F^2 + \|P\|_F^2) + \lambda \sum_{(n,m)} (UP^T - R)_{n,m}^2$$

After initializing U^0 and P^0 and updating U and P by

$$U^k = \operatorname{argmin} Error(U, P^{k-1}) \text{ and } P^k = \operatorname{argmin} Error(U^k, P)$$

We can tell that

$$Error(U^k, P^k) < Error(U^k, P^{k-1}) < Error(U^{k-1}, P^{k-1})$$

Therefore, this showcases how the error shrinks in every iteration until it converges and stops decreasing, meaning we have reached an optimal solution.

For the unbiased matrix, the error of the recommend matrix is at 1241.67 at 10 iterations.

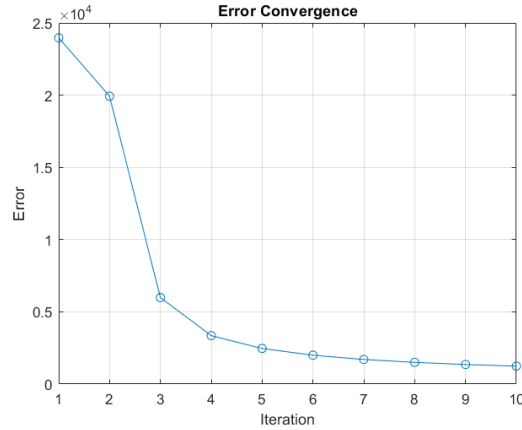


Figure 3: Error from the Unbiased Recommender after 10 iterations

The error comes down to 423.4 at 100 iterations.

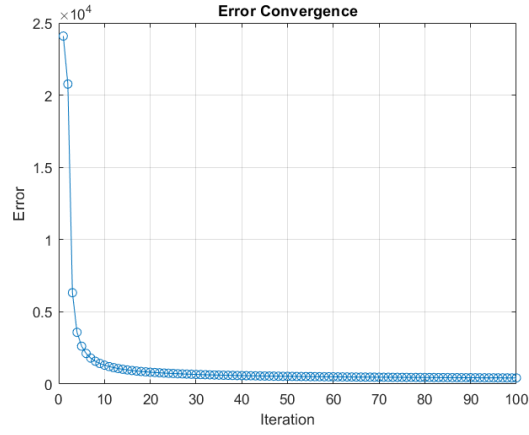


Figure 4: Error from the Unbiased Recommender after 100 iterations

And it reduces further to 258.5 at 1000 iterations. See Figure 5

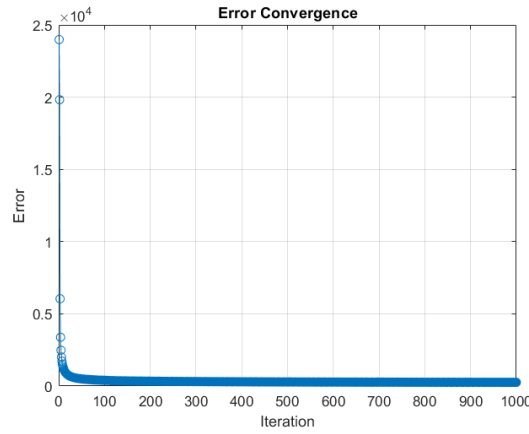


Figure 5: Error from the Unbiased Recommender after 1000 iterations

Though there are diminishing returns, increasing the number of iterations when building the recommendation makes it a more accurate tool. When applying this system, users can determine how much accuracy they need and how much computational bandwidth they can afford.

The biased system on the other hand is less accurate. With the same parameters, it has an error of 26763.9 at 100 iterations, which only reduces to 26140.0 at 500 iterations.

Because the sampling is skewed towards a very small minority of the population, it cannot accurately reflect the larger sampled population and further iterations cannot compensate for the inaccuracy.

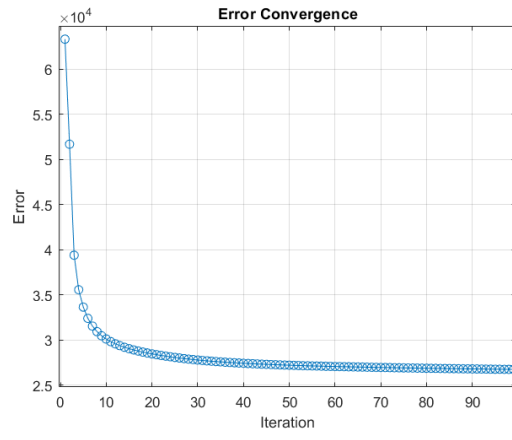


Figure 6: Error from the Biased Recommender after 100 iterations

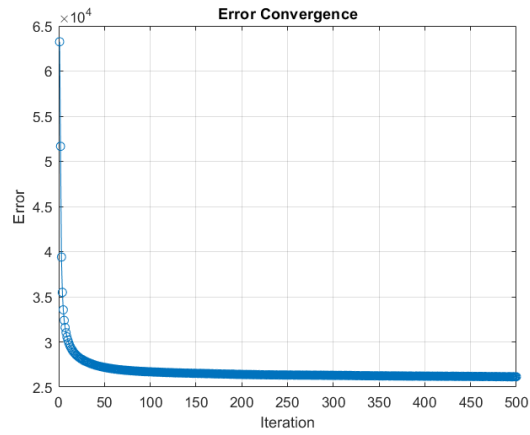


Figure 7: Error from the Biased Recommender after 500 iterations

When varying the number of features considered, there was an inverse correlation between the number of r values and the error at the end of 100 iterations.

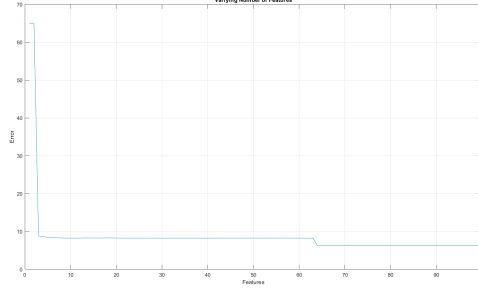


Figure 8: Error from different r values

This supports the idea that as the number of features in a model increases, there are more points of comparison available, which typically leads to a more accurate model.

4.1 Discussion

The unbiased recommender system, with its iterative approach, demonstrated convergence towards an accurate recommendation matrix. The results highlight the effectiveness of the ALS algorithm in minimizing error and refining the system with each iteration. The biased recommender system's limitations suggest caution in using skewed sampling approaches. While it might be computationally efficient, it compromises the accuracy and generalizability of the system, limiting its practical utility. Practically, the choice of the number of iterations and latent features depends on the trade-off between computational resources and the desired level of accuracy. Users can adjust these parameters based on their specific requirements.

The findings open avenues for future research, exploring novel algorithms or sampling strategies to enhance the accuracy of biased recommender systems. Further investigation into the interplay of latent features and error reduction can contribute to refining recommendation models. In conclusion, the unbiased recommender system, driven by the ALS algorithm, showcases promise in delivering accurate recommendations, while the biased system emphasizes the importance of representative sampling for robust and reliable results.

References

- [1] Candès, Emmanuel & Recht, Benjamin. (2008). Exact Matrix Completion via Convex Optimization. *Communications of the ACM*. 9. 717-772. 10.1007/s10208-009-9045-5.
- [2] Recommendation System, Nvidia <https://www.nvidia.com/en-us/glossary/data-science/recommendation-system/#:~:text=A%20recommendation%20system%20is%20an,demographic%20information%2C%20and%20other%20factors.>
- [3] Zendesk, Customer Service and Business Results: A Survey of Customer Service from Mid-Size Companies, Dimensional Research, 2013, http://cdn.zendesk.com/resources/whitepapers/Zendesk_WP_Customer_Service_and_Business_Results.pdf
- [4] Wikipedia contributors. (2023, November 27). Recommender system. In *Wikipedia, The Free Encyclopedia*. Retrieved 12:58, December 18, 2023, from https://en.wikipedia.org/w/index.php?title=Recommender_system&oldid=1187107803
- [5] Wikipedia contributors. (2023, December 17). Rank (linear algebra). In *Wikipedia, The Free Encyclopedia*. Retrieved 15:34, December 18, 2023, from [https://en.wikipedia.org/w/index.php?title=Rank_\(linear_algebra\)&oldid=1190320332](https://en.wikipedia.org/w/index.php?title=Rank_(linear_algebra)&oldid=1190320332)
- [6] J. Lu, D. Wu, M. Mao, W. Wang, and G. Zhang, *Recommender System Application Developments: A survey,* *Decision Systems e-Service Intelligence Lab, Centre for Quantum Computation Intelligent Systems Faculty of Engineering and Information Technology, University of Technology Sydney, Australia*. Retrieved from <https://www.uts.edu.au/sites/default/files/desi-publication-recommender%20system%20application%20developments%20a%20survey-accepted%20manuscript.pdf>. [Accessed:12-May-2022]
- [7] W. Wang, “Recommended system of application and development,” *AIP Conference Proceedings* 1955, 040180, Apr. 2018.