Ukrainian Catholic University

Faculty of Applied Sciences

IT & Business Analytics Programme

Comparison of Matrix Factorization Techniques for Collaborative Filtering-Based Recommendation Systems

Mathematical Methods of Machine Learning Project

Authors:

Anastasiia Havryliv

Yuliia Maksymiuk

14 May 2023

1 Problem statement and justification

The modern world generates loads of data; sometimes, finding what you are looking for is difficult. It is even more challenging to find something that you will like. That is where recommendation systems step up because why waste your time digging through endless data sources to see something like that movie you watched yesterday and liked? This algorithm saves time and gives results that will leave you satisfied. At the same time, the only information it needs is knowledge of past behaviors of yourself and users with similar tastes, paired with data about items, movies, or books you are looking for.

Collaborative filtering can be based on two main principles: user-based and item-based. In the first one, we identify other users that have similar tastes to the person for whom we are trying to find some recommendations. In contrast, in the second option, we try to identify items that are similar to the ones that the user previously liked.

As for the ways to solve this problem and implement the algorithm, matrix factorization is often considered among the other options. The matrix that will be later constructed from smaller, factorized ones will represent user-item interaction and show us the ratings given to items by users.

The main aim of this project is to explore the use of matrix factorization techniques for collaborative filtering in recommendation systems. We will compare different approaches to matrix factorization and evaluate their performance on the relevant data using some of the error metrics.

2 Detailed description of the theoretical background

As mentioned previously, the objective of a Recommendation System is to recommend relevant items for users, based on their preference. Preference and relevance are subjective and generally inferred by items users have consumed previously. There are two popular approaches used in recommender systems to suggest items to the users(1):

1. **Collaborative Filtering:** This method makes automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating). The underlying assumption of the collaborative filtering approach is (2):

   if: person A has the same opinion as person B on a set of items, A is more likely to have B's opinion for a given item than that of a randomly chosen person.

   Collaborative filters do not require item metadata like their content-based counterparts.

2. **Content-Based Filtering:** To model a user's preferences, this method only takes into account the attributes and description of the products they have already consumed. In other words, these algorithms attempt to suggest products that are similar to those that a user has previously liked (or is currently looking at). In particular, various candidate items are compared with items previously rated by the user, and the best-matching items are recommended.(3) This system uses item metadata, such as genre, director, description, actors, etc. for movies to make these recommendations.

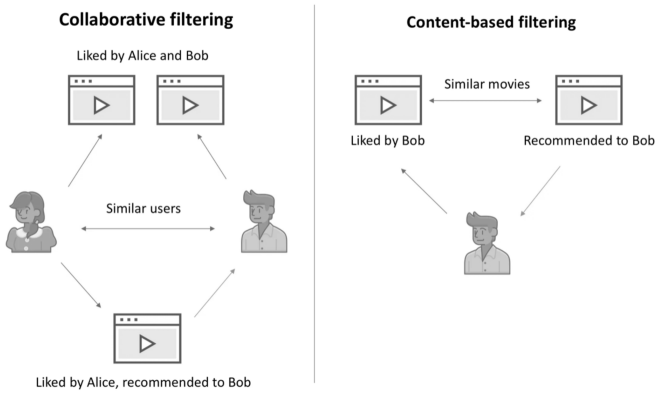

Figure 1: Collaborative vs. Content-Based Filtering

Our project focuses on the collaborative filtering approach since it looks at what users like and recommends similar things without needing detailed information. It helps us provide personalized recommendations, discover new items users might like, and works well even with limited data about new users or items.

To achieve this, we will analyze a collaborative filtering technique known as Matrix factorization. This technique involves decomposing the user-item preference matrix into lower-dimensional factor matrices, thereby identifying hidden features and interactions. Matrix factorization offers several methods, including **Singular Value Decomposition** (*SVD*), **Non-negative Matrix Factorization** (*NMF*), and **Alternating Least Squares** (*ALS*). By identifying hidden features and interactions, these methods allow us to spot patterns and relationships within the data, helping us provide customized user recommendations based on their past preferences.

3 Description of the solution methods

As we approach the task of collaborating filtering through matrix factorization, using previously mentioned methods, such as SVD, NMF, and ALS, is justified because those are some of the most popular choices for this task. Theoretical background was explained in the previous block, so here we will focus more on how these algorithms help with collaborative filtering and our task.

Let's assume we have considerable data with users and their reviews for different movies. For our task, we will store all of this data in the matrix, where rows and columns are users and movies, respectively. And the entries in this matrix, for example, U_{ij}, is the rating review that the i-th user has given to the j-th film. From this user-item "correlation" matrix, we can easily observe the similarities between users. In fact, matrix factorization uses similarities in what items different users prefer to provide recommendations for them.

Matrix factorization involves decomposing a matrix of size $m \times n$ into **two lower-rank matrices** of sizes $m \times k$ and $k \times n$, respectively. This factorization enables us to perform complex matrix operations and, importantly, by multiplying the factorized matrices, we can reconstruct the original matrix, as can be seen in the picture below. It is worth noting that while matrix factorization can be extended to more than two entities, such cases fall under the realm of tensor factorization.

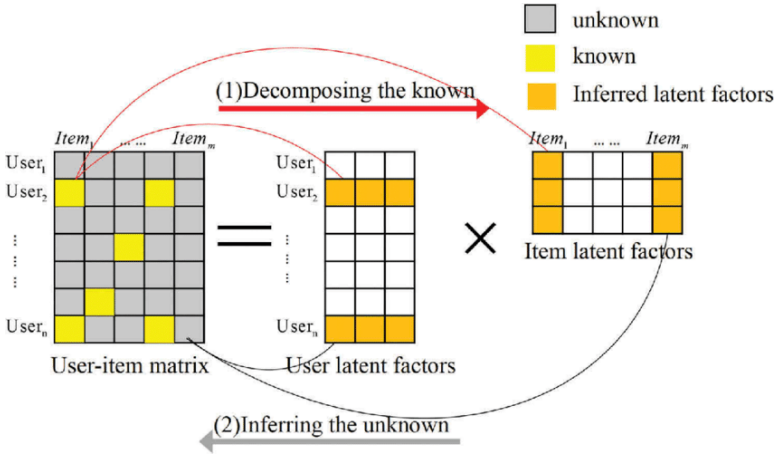

Figure 2: Matrix Factorization

So why is matrix factorization so valuable for our case? It is because having a real-world system comes with loads of data and can be hard or even impossible to deal with and process all at once. Factorization is expressing a large matrix through the product of its small factors, which are a lot easier to deal with.

3.1 SVD

Singular Value Decomposition (SVD) is a matrix factorization technique that decomposes the user-item preference matrix into singular vectors and singular values. It aims to capture the latent features and interactions between users and items. The SVD factorization of the user-item matrix can be represented as(4):

$$A = U\Sigma V^T \quad , \tag{1}$$

where:

- A is an $n \times m$ user-item preference matrix, which provides a structured representation of users' preferences for different items.
- U is an orthogonal $n \times n$ matrix of left singular vectors, representing the user latent factors. These factors capture underlying patterns or characteristics in users' preferences, allowing for interpretability.
- Σ is an $n \times m$ diagonal matrix of singular values, representing the importance of each latent factor. This matrix has all zeros except for its diagonal entries, which are nonnegative real numbers. The singular values indicate the significance of each latent factor in explaining the variations in the user-item preferences.

  if: σ_{ij} is the i, j entry of Σ, then $\sigma_{ij} = 0$ unless $i = j$ and $\sigma_{ii} = \sigma_i \geq 0$. The singular values provide insight into the relative contribution of each latent factor to the overall user-item preference matrix.
- V^T is a transpose of an orthogonal $m \times m$ matrix V. It is the matrix of right singular vectors, representing the item latent factors. These factors capture underlying characteristics or features of the items, offering interpretability in terms of the item properties that influence user preferences.

The concept behind SVD is straightforward: by focusing on the most significant latent factors (those with the largest singular values), we can decrease the dimensionality of the original matrix, leading to faster and simpler calculations. The recommendation process involves reducing the dimensionality of user-item matrix by selecting **top k** singular values and their corresponding singular vectors. The reconstructed matrix, obtained by multiplying the selected singular vectors and values, provides recommendations based on the estimated ratings.

In addition, this technique has **no restriction to the shape or properties** of the matrix to be decomposed: either square or rectangular matrices, degenerate or non-singular. This broad applicability makes SVD such a valuable and widely-used method of processing matrices.

To evaluate the performance of this approach for our project, we decided to compare following different computational implementations of SVD:

- **Our SVD**: developed based on the *power method*.
- Python **Scipy SVD**: this implementation is part of the SciPy library, a popular scientific computing library in Python. It utilizes the SVD algorithm provided by SciPy, which is an efficient implementation of SVD using numerical linear algebra techniques.
- Python **Surprise SVD**: Part of the Surprise library, specifically designed for recommender systems, it employs SVD to factorize the user-item preference matrix and make predictions. We will compare with parameter *biased* set to both True and False.
- Python Surprise **SVDpp (SVD++)**: An extension of Surprise's SVD implementation, it incorporates implicit feedback and additional factors to enhance recommendation accuracy.

By comparing these implementations, we aimed to gain insights into their respective strengths and weaknesses and determine the most suitable approach for our task.

3.1.1 Our SVD - Power Method approach

The power method is great in providing accurate approximations for the extreme eigenvalues of a matrix, specifically, the eigenvalues with the largest and smallest magnitudes, represented as λ_1 and λ_n, along with their corresponding eigenvectors. As known, we can obtain singular values of a matrix by computing the eigenvalue of a matrix transpose by itself. Let us suppose that the eigenvalues of $B = A^T A$ are ordered the following way:

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| ... \geq |\lambda_n|$$

The idea is simple: we first select a random vector x and compute Bx. As noted previously, the matrix

$B = A^T A = V\Sigma^2 V^T$ because U and V are orthogonal matrices. As a result, we have:

$$B = \sum_{i=1}^{n} \sigma_i^2 v_i v_i^T$$
$$\implies B^k = \sum_{i=1}^{n} \sigma_i^{2k} v_i v_i^T$$
$$\implies B^k x = \sum_{i=1}^{n} \sigma_i^{2k} v_i v_i^T x$$

As we know, $\sigma_1 > \sigma_2 > \sigma_3 ... > \sigma_n$, this means that the first term will dominate as we take the limit on $k \longrightarrow \infty$. The summation will thus converge to the first term, giving us $B^k \longrightarrow \sigma_1^{2k} v_1 v_1^T$. After normalizing the resulting vector will give us v_1, the first singular vector of A. Using $\sigma_1 = \|Av_1\|$ and $u_1 = \frac{1}{\sigma_1}\|Av_1\|$ will give us the first singular value and left singular vector of A.
Since we have computed (σ_1, u_1, v_1) we can extend this algorithm to a full SVD of the matrix A by repeating the above procedure for:

$$A - \sigma_1 u_1 v_1^T = \sum_{i=2}^{n} \sigma_i u_i v_i^T$$

and of which the singular vector and vectors will be (σ_2, u_2, v_2). We continue this process for k (set it by ourselves) times to get SVD of a matrix A.

3.1.2 Python Surprise SVD

The main framework of the (basic) SVD model is characterizing a_{ij} by the inner product of a user-related s-vector u_i and an item-related s-vector v_j or, in other words:

$$\hat{a_{ij}} = u_i \cdot v_j^T$$

To find all vectors for users and items to minimize the following optimization problem:

$$min_{u_i, v_j} \sum_{a_{ij} \in A} (a_{ij} - u_i v_j^T)^2 + \lambda(\|u_i\|^2 + \|v_j^T\|^2) \tag{1}$$

where a_{ij} is the real rating, $u_i v_j^T$ is the predicted value, $\lambda > 0$ is a regularization constant to avoiding overfitting, and the term in brackets in used for penalizing errors. The minimization is performed by a very straightforward stochastic gradient descent:

$$u_i \longleftarrow u_i + \gamma((a_{ij} - u_i v_j^T)v_j - \lambda u_i)$$
$$v_j \longleftarrow v_j + \gamma((a_{ij} - u_i v_j^T)u_i - \lambda v_j)$$

These steps are performed over all the ratings of the data and repeated n_{epochs} (set by ourselves) times.

However, for this project we decided also to test this same SVD function, but also try setting *biased* parameter to True. **What is the effect** of this parameter? The inclusion of **biased=True** in models takes into account the inherent biases that individuals have (ex. high ratings giving overall bias, genre bias, popularity bias, etc). By incorporating bias of both users and items, models can provide a more comprehensive perspective on the data.
After setting **biased=True** the prediction $\hat{a_{ij}}$ becomes the following:

$$\hat{a_{ij}} = \mu + b_i + b_j + u_i \cdot v_j^T,$$

where b_i - user's bias (shows whether a user tends to rate movies higher or lower compared to the average of everyone else), b_j - bias for item (represents the difference between the average rating for a specific movie and the overall average rating across all movies), μ - global bias calculated as a global mean of all ratings.
As a result, (1) is a bit changed, so now we minimize the following optimization problem:

$$min_{u_i, v_j} \sum_{a_{ij} \in A} (a_{ij} - \hat{a_{ij}})^2 + \lambda(b_i^2 + b_j^2 + \|u_i\|^2 + \|v_j^T\|^2)$$

where all other components are the same as in (1), we just added item and user bias to the penalizing error. The minimization is also performed by stochastic gradient descent the following way:

$$
\begin{aligned}
u_i &\longleftarrow u_i + \gamma((a_{ij} - u_i v_j^T)v_j - \lambda u_i) \\
v_j &\longleftarrow v_j + \gamma((a_{ij} - u_i v_j^T)u_i - \lambda v_j) \\
b_i &\longleftarrow b_i + \gamma((a_{ij} - u_i v_j^T) - \lambda b_i) \\
b_j &\longleftarrow b_j + \gamma((a_{ij} - u_i v_j^T) - \lambda b_j)
\end{aligned}
$$

These steps are also performed over all the ratings of the data and repeated n_{epochs} times.

3.1.3 Python Surprise SVDpp (SVD++)

The SVD++ algorithm is just an extension of SVD taking into account implicit ratings. The most basic criteria for implicit feedback is whether a user gives a rating to a specific item, regardless of the rating score. Implicit ratings are indications of user preferences that are derived from implicit feedback rather than explicit ratings. Explicit ratings are direct assessments provided by users (e.g., giving a movie a rating of 5 stars), while implicit ratings are derived from users' behavior or interactions with items (e.g., the number of times a user watched a movie or clicked on an item).

In SVD++, implicit ratings are typically represented as binary indicators, where a value of 1 indicates a positive interaction or preference, and a value of 0 indicates no interaction or preference.
The prediction $\hat{a_{ij}}$ becomes the following:

$$
\hat{a_{ij}} = \mu + b_i + b_j + u_i \cdot (v_j^T + \|I_i\|^{-\frac{1}{2}} \sum_{p \in I_i} y_p),
$$

where y_p terms are a new set of item factors that capture implicit ratings. Similar to Singular Value Decomposition (SVD), the parameters are obtained by utilizing Stochastic Gradient Descent (SGD) on the objective of minimizing the regularized squared error of its respective optimization problem which we will not focus in this paper.

3.2 NMF

Non-negative Matrix Factorization (NMF) is another matrix factorization technique that decomposes the user-item preference matrix into non-negative factors. NMF assumes that both the user and item factors are non-negative, allowing for interpretability and capturing positive interactions.
The NMF factorization of the user-item matrix can be represented as:

$$
A = W \times H \quad , \tag{2}
$$

where:

- A is an $n \times m$ user-item preference matrix.
- W is an $n \times k$ matrix of non-negative user factors.
- H is an $k \times m$ matrix of non-negative item factors.

The objective of NMF is to find W and H that **minimize the reconstruction error** between A and $W \times H$ since it is impossible to reconstruct the initial matrix precisely and we want to be as "close" as possible to it. For this, the Frobenius norm is used to compute the dissimilarity as follows(6):

$$
M = V - W \times H, \quad \|M\|_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{m} |a_{ij}|^2}
$$

The advantage of NMF in collaborative filtering is that since we apply nonnegativity to nonnegative matrices, prediction errors are reduced compared to methods such as SVD. As a result, we always have interpretable and sparse decompositions. NMF reveals hidden features and patterns in the data, which allows for personalized recommendations based on the factors studied.

This algorithm iterates through every rating in the user-item matrix and if the rating is known, it computes

the error and updates the elements of P and Q using gradient descent. This updates are also controlled by hyperparameters such as learning rate and regularization parameter.

Algorithm 1 Non-Negative Matrix Factorization (NNMF)

Require: User-item rating matrix R, Number of latent features K, Number of iterations $Iterations$, Learning rate η, Regularization parameter λ
Initialize P with random non-negative values
Initialize Q with random non-negative values
for $iteration = 1$ to $Iterations$ **do**
  for $i = 1$ to num_users **do**
    for $j = 1$ to num_items **do**
      if $R[i, j] > 0$ **then**
        Compute the error: $e_{ij} = R[i, j] - P[i, :] \cdot Q[:, j]$
        for $k = 1$ to K **do**
          Update $P[i, k]$: $P[i, k] = P[i, k] + \eta \cdot (2 \cdot e_{ij} \cdot Q[k, j] - \lambda \cdot P[i, k])$
          Update $Q[k, j]$: $Q[k, j] = Q[k, j] + \eta \cdot (2 \cdot e_{ij} \cdot P[i, k] - \lambda \cdot Q[k, j])$
        end for
      end if
    end for
  end for
end for
convergence
return P, Q

3.3 ALS

Alternating Least Squares (ALS) is yet another method of matrix decomposition that through iterative optimization process tries to arrive closer and closer to a factorized representation of our original data. In other words, we try to calculate resulting matrices in a way that their product will approximate original data matrix as close as possible. This factorization can be represented as following:

$$R = U \times V \ , \tag{3}$$

where:

- R is an $n \times m$ user-item preference matrix.
- U is an $n \times k$ matrix of users and hidden features.
- V is an $k \times m$ matrix of items and hidden features.
- In U and V we have we have weights of how each user relates to each feature.

One of the main high-level ideas of this algorithm is that it run two processes in parallel. First, we fix user matrix and run minimizing objective function on item matrix, and after that we do the same conversely. And this alternating process continues unil the convergence will be reached.

Last but not least, ALS has the advantage of being able to work with large amounts of data, as it can be parallelized and distributed among computational units, which makes it suitable for real-life systems.

Our approach, as we can see in the pseudocode of the algorithm will therefore be to fix Y and optimize X, then fix X and optimize Y , and repeat until convergence. For our objective function, the alternating least squares algorithm is as follows (7):

Algorithm 2 Alternating Least Squares algorithm

Initialize X, Y
repeat
  for $u = 1, ..., n$ **do** ▷ Fix Y and optimize X
    $x_u = (\sum_{r_{ui} \in r_{u*}} y_i y_i^T + \lambda I_k)^{-1} \sum_{r_{ui} \in r_{u*}} r_{ui} y_i$
  end for
  for $i = 1, ..., m$ **do** ▷ Fix X and optimize Y
    $y_i = (\sum_{r_{ui} \in r_{*i}} x_u x_u^T + \lambda I_k)^{-1} \sum_{r_{ui} \in r_{*i}} r_{ui} x_u$
  end for
until convergence
return X, Y

However, this approach is prohibitively expensive for most real-world datasets, as this approach will cost $O(nmk)$ if we'd like to estimate (predict) every user-item pair. Updating each x_u will cost $O(n_u k^2 + k^3)$, where n_u - number of items rated by user u, and also updating each y_i will cost $O(n_i k^2 + k^3)$ with n_i being number of users that have rated item i.

4 Review of the alternative approaches, comparison

Overall, Collaborative Filtering can be further categorized into two distinct methods, that are most widely used: **k-Nearest Neighbors (kNN)** and **Matrix Factorization**: **k-Nearest Neighbors (kNN) Approach:** recommendations are made by identifying the k most similar users or items to the target user. These similar users' preferences are then used to make predictions. The similarity is typically calculated using metrics like: **cosine similarity** (5):

$$\cos(\mathbf{t}, \mathbf{e}) = \frac{\mathbf{t} \cdot \mathbf{e}}{\|\mathbf{t}\| \|\mathbf{e}\|} = \frac{\sum_{i=1}^{n} \mathbf{t}_i \mathbf{e}_i}{\sqrt{\sum_{i=1}^{n} (\mathbf{t}_i)^2} \sqrt{\sum_{i=1}^{n} (\mathbf{e}_i)^2}} \tag{4}$$

or **Euclidean distance**:

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2} \tag{5}$$

Generally kNN is intuitive and simple, however it can have scalability issues as the data grows. Additionally, kNN is prone to the "sparsity problem," where finding enough sufficiently similar neighbors becomes challenging in sparse data scenarios. Also, it can suffer from scalability, because as the amount of data grows, computational complexity of kNN increases, which would be inconviniently for large systems.

5 Data Overview

For the research purposes of our project and testing discussed approaches we will be using MovieLens100k Dataset, which contains 100,000 ratings from 1000 users on 1700 movies. It has ratings for movies only before 1998, so if we would want to build a system that is relevant to modern life, we would have to consider other options. However, for approaches and algorithms testing in educational purposes, this data will be more than enough.

We will be using the user-movie ratings from this data to test all of the described above approaches and try to answer the question "Which of the matrix factorization algorithms works the best for recommendation systems?".

6 Results

Here we will discuss the results of approaches that we discussed throughout this report. To evaluate the performance of these approaches we will be using the RMSE (Root Mean Squared Error). It will be basically checking the objective function that we were trying to minimize in all of the algorithms.

$$RMSE(\hat{y}, y) = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}} \tag{6}$$

In the formula above we calculate the square root of average difference between predicted and actual ratings. And logically, lower RMSE indicates the better performance of the model.

As we said above, in the table below we will be able to see the comparison of RMSE's for various approaches, implemented by ourselves, as well as their library-ready versions: Singular Value Decomposition (SVD), SVD++, Non-Negative Matrix Factorization (NNMF) and Alternating Least Squares (ALS).

Approach	RMSE
Our SVD - Power Method Approach	2.3620
Python Scipy SVD	2.3620
Python Surprise SVD without bias	0.9641
Python Surprise SVD with bias	0.9454
Python Surprise SVDpp (SVD++)	0.9244
Our NNMF	3.5339
Python Sklearn NNMF	0.2685
Our ALS	0.3756

Table 1: Comparison of RMSE for different approaches

7 Conclusions

During this research project, we discovered different matrix factorization techniques for collaborative filtering - based recommendations systems and even taken a quick look at other approaches. We discovered by ourselves that all of those approaches need some improvement, as there is no perfect method to solve the problem of recommendation system based on collaborative filtering. Additionally, considering the time-dependent relationship between users and items would be beneficial, as users' preferences tend to evolve over time. These potential avenues for refinement highlight the potential for further advancements in collaborative filtering-based recommendation systems.

In addition to exploring various matrix factorization techniques and considering the inclusion of external information and time-dependent relationships, we also saw the impact of different regularization techniques on the performance of collaborative filtering-based recommendation systems. By systematically comparing and evaluating these techniques, we aimed to provide insights into the strengths and weaknesses of each approach, thereby assisting in the selection and implementation of the most suitable matrix factorization technique for specific recommendation system requirements.

8 References

1. CME 323: Distributed Algorithms and Optimization, Spring 2015 http://stanford.edu/ rezab/dao. Instructor: Reza Zadeh, Databricks and Stanford. Lecture 14, 5/13/2015. Yonathan Perez. Scribed by Haoming Li, Bangzheng He, Michael Lublin, 14 Matrix Completion via Alternating Least Square(ALS) 14.1 Introduction
2. Shah, Kunal & Salunke, Akshaykumar & Dongare, Saurabh & Antala, Kisandas. (2017). Recommender systems: An overview of different approaches to recommendations. 1-4. 10.1109/ICIIECS.2017.8276172.
3. Garima Gupta, Rahul Katarya. Recommendation analysis on Item-based and User- based Collaboration Filtering.
4. Mahajan, S., Pande, A., Pande, S., Sanghvi, D., & Shah, R. (2014). Mining Web Graphs for Recommendations. International Journal of Electronics Communication and Computer Engineering, 5(2), 351.
5. Yiqi Gua, Xi Yanga, Mengjiao Pengb, Guang Lin. August 10, 2019. Robust weighted SVD-type latent factor models for rating prediction. 4-6.
6. Hung-Hsuan Chen. October 3, 2017. Weighted-SVD: Matrix Factorization with Weights on the Latent Factors, 3-4.
7. Haoming Li, Bangzheng He, Michael Lublin, Yonathan Perez. Matrix Completion via Alternating Least Square(ALS). May 13, 2015. CME 323: Distributed Algorithms and Optimization, Spring 2015.

- **Github Repository**

[1] https://analyticsindiamag.com/singular-value-decomposition-svd-application-recommender-system/

[2] https://en.wikipedia.org/wiki/Collaborative_filtering

[3] https://en.wikipedia.org/wiki/Recommender_system

[4] https://en.wikipedia.org/wiki/Singular_value_decomposition

[5] https://towardsdatascience.com/various-implementations-of-collaborative-filtering-100385c6dfe0

[6] https://medium.com/logicai/non-negative-matrix-factorization-for-recommendation-systems-985ca8d5c16c

[7] https://stanford.edu/~rezab/classes/cme323/S15/notes/lec14.pdf

[8] https://surprise.readthedocs.io/en/stable/matrix_factorization.html#