Recommender Systems using Matrix Factorization
Data Science Assignment 1

Abdelkader Chihab Benamara
Aymen Djelid
Mohamed Ali Benrekia

October 24, 2021

Overview

---

1 Introduction

Many customers are inundated with the choices of products. For example, Spotify and GrooveShark provide a huge number of songs for online audiences, So does, Amazon and Fnac. An important problem is how to let users easily find items meeting their needs. Therefore Recommender systems have been constructed for such a purpose.

These systems relies only on Collaborative filtering which analyzes past user behavior regarding the choice or rating of items and aim to estimate what item to recommend for the user. This approach is the application of Matrix Factorization in the context of recommender systems.

In this report, we approach the matrix factorization problem as a machine learning task where we detail about the problem statement, then we propose two solutions to solve the problem which are: Stochastic gradient descent and Alternating least squares. After that, we highlight the important parts of our implementation and discuss the results of the conducted experimentation on a real-world dataset.

2 Problem Statement

Online services apply collaborative filtering in order to analyze correlation between users and items to identify new user-item interdependencies. Imagine a recommendation system for movies, we can represent the reviews of users in a form of a matrix where the rows correspond to the users and the columns to the movies, which results in a sparse matrix since not all the users are going to watch every single movie and rate it.

Matrix factorization is a way to generate low rank matrix when multiplying two different matrices, and thanks to that we can assimilate missed reviews, derived by examining the associations between the users and the items. Hence, we can predict if a user is likely to give a movie high rating or low rating, and given that information the system recommend the movie to that user.

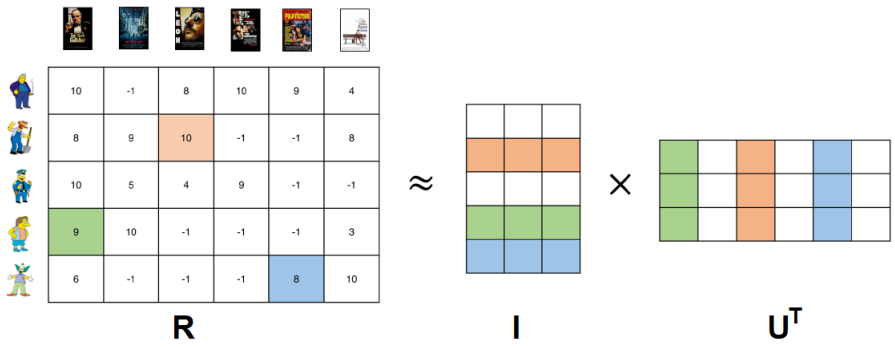

Figure 1: Illustration of a Matrix factorization problem.

We summarize the core idea of matrix factorization in the figure 1, Let R be a *(m,n)* matrix, we can approximate it by applying a dot product between matrix I of dimensionality *(m,k)* and a matrix U of dimension *(n,k)*. Formally, we can write it as :

$$R \approx I \times U^T$$

Our goal is to find the matrices U and I which estimates as close as possible the matrix R, in other words, we want to minimize a function C:

$$\min_{I,U} \left\| R - IU^\top \right\|_{\mathcal{F}}^2$$

where $\|.\|_F$ is matrix norm of an $m \times n$ matrix A defined as:

$$\|A\|_F^2 = \mathrm{Tr}\left(A^T A\right)$$

Since we want to approach the problem as a machine learning task we incorporate the regularization term to our function. This will help avoiding overfitting and generalize to unseen data. We can, then, formulate our function to minimize as :

$$\min_{I,U} \left\| R - IU^\top \right\|_{\mathcal{F}}^2 + \lambda \|I\|_{\mathcal{F}}^2 + \mu \|U\|_{\mathcal{F}}^2$$

where λ and μ are regularization parameters.

3 Solutions

To solve this minimization problem we opted for stochastic gradient descent (SGD) and alternating least squares (ALS).

3.1 Stochastic gradient descent:

Since we aim for any local minimum of the function C previously presented, we can use stochastic gradient descent (SGD) to achieve this goal. SGD is a generic method for continuous optimization, so it can be, and is very commonly, applied to non-convex functions.

In our case, we calculate the partial derivatives of the function C, formulated as :

$$\frac{\partial C}{\partial U}(I, U) = -2R^\top I + 2UI^\top I + 2\mu U$$

$$\frac{\partial C}{\partial l}(I, U) = -2RU + 2IU^\top U + 2\lambda I$$

First, we initialize the two matrices U and I randomly and, at each step t we update these two matrices according to these equations:

$$I_t = I_{t-1} - \eta_{t-1} \frac{\partial C}{\partial l}(I_{t-1}, U_{t-1})$$
$$U_t = U_{t-1} - \xi_{t-1} \frac{\partial C}{\partial U}(I_{t-1}, U_{t-1})$$

where, η and ξ are the learning rates used to reach a local minimum. This popular method combines flexibility of implementation and a relatively fast training time.

Some users tend to never give a movie a better rating or some movies are widely known to be good movies, for that matter, we can incorporate bias in calculating the estimation of the matrix of rating R :

$$r_{ui} \approx \nu + b_i + b_u + \mathbf{y}_i \cdot \mathbf{x}_u^\top$$

where ν is global bias for the matrix of ratings, b_i and b_u are item bias and user bias respectively.

3.2 Alternating least squares:

Since we want to solve an equation for two variables i.e matrices U and I, our function C is not convex. However, if we alternate between fixing one of the unknowns and solving the other, the optimization problem becomes quadratic and can be solved optimally. This is exactly the basic idea of Alternating Least Squares (ALS). This technique is guaranteed to converge because each step is guaranteed to decrease the loss.

In our case, we set the partial derivatives of the function C to zero and calculate the gradient with respect to U while fixing the I and vice-versa:

$$\frac{\partial C}{\partial U}(I,U) = -2R^\top I + 2UI^\top I + 2\mu U = 0$$

$$\frac{\partial C}{\partial l}(I,U) = -2RU + 2IU^\top U + 2\lambda I = 0$$

Thus, we can get the formulas to update the U and I matrices at each step t presented as follow:

$$I_t = RU_{t-1}\left(U_{t-1}^\top U_{t-1} + \lambda\mathbb{I}\right)^{-1}$$
$$U_t = R^\top I_{t-1}\left(I_{t-1}^\top I_{t-1} + \mu\mathbb{I}\right)^{-1}$$

Where $\mathbb{I}$ represent the identity matrix. In contrast to SGD method which is faster and easier to implement, ALS takes more running time, but, if we can use parallelization in the process of solving the problem, it is preferred to use ALS.

3.3 Evaluation metric:

To evaluate how good our approaches fit the data, we measure the error between predicted values represented as matrix $\hat{R}$ and actual values represented as matrix R, so we use Root Mean Square Error (RMSE) as an error metric for both approaches. RMSE calculates the root mean square error between the ground truth and the predicted values, formally represented as:

$$\text{RMSE}(R,\hat{R},T) = \sqrt{\frac{\sum_{(i,u)\in T}\left(R_{\text{iu}} - \hat{R}_{iu}\right)^2}{|T|}}$$

Where, R is the original rating matrix, $\hat{R}$ is the estimated rating matrix and T is the test set.

4 Experimentation & Results

In this section we will discuss what we implemented in terms of code and what experiments we were able to do in our project. The following figure explain the pipeline of this implementation.

The first step in our implementation was to collect and prepare the data that we got from the Movielens Dataset with above 100K ratings in a tuple format **(user,item,rating)**, we splitted this dataset into 95% for training and 5% for testing.
Note. A validation set is given as an option on the train method for the two models so that we could have the history of the loss on the validation set (a float validation_split is charged of this split).
But as we said in the initial setup we need our data in a format of a matrix $R \in \mathbb{R}^{m\times n}$, so to do this we created a function that transforms this tuple set (X_train) into a matrix R (see

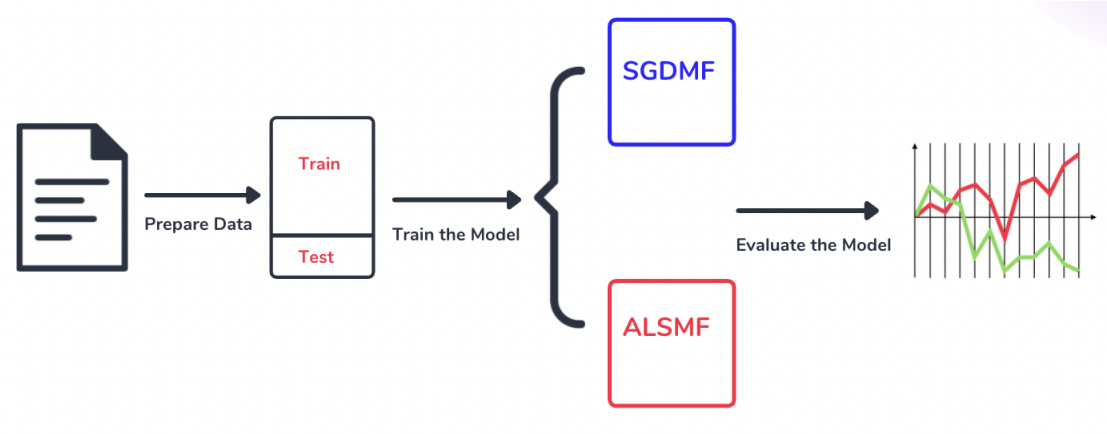

Figure 2: Architecture of the project (Pipeline)

more).
Now we will discuss what we got in our experiments of the two models (SGDMF & ALSMF) and some comparaison between this two approaches.

4.1 Training results :

So in this section we will show our experiments about the first approach of stochastic gradient descent , in the first time we trained this model without any biases and then we decided to add biases and compare the two results that we got.

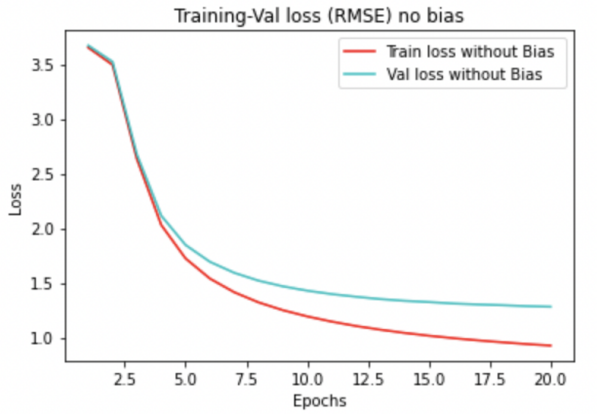

Figure 3: [SGDMF] without biases

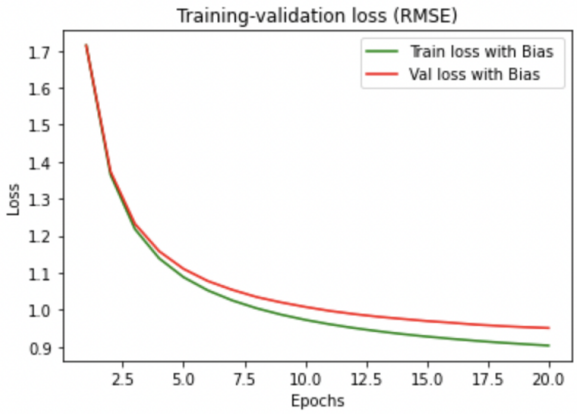

Figure 4: [SGDMF] with biases

In the graphs above we have the training and validation losses (RMSE) metric for the SGDMF model when we don't use the biases and when we don't, and as we can see the loss take up to 10 epochs to start decreasing in a better way than its starting in the case when we don't use biases.
In the other hand we see clearly the advantage of using biases on the graph at our right, the first loss we've got when using biases we were able to get it after more than 7 epochs when we don't use biases.
To study the impact of the bias on the SGDMF model we plotted the validation loss of both approaches and also we measured the cost in terms of memory occupation and running time, the following figure and table summarize the outcomes. We notice that in the case of using

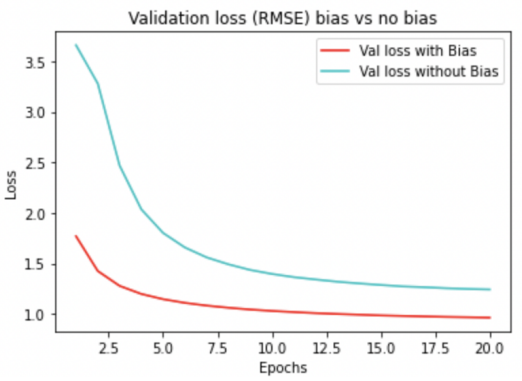

Figure 5: [SGDMF] without biases vs with biases

SGDMF execution time & space per epoch		
	Time per epoch (sec)	**Space per epoch (GB)**
Without biases	54.29 (sec)	2.1 (GB)
With biases	10.82 (sec)	3.4 (GB)

Table 1: [SGDMF] without biases vs with biases

bias, the loss function converges quicker than without using bias which is a natural outcome regarding the sparsity of the matrix R with/(out) bias. Also adding tha bias helps the model improving the precision of the result.
It is important to mention that using bias takes less running time in comparison to not using it , yet it takes more memory space per epoch.

In order to examine the impact of the number of features (the factor K) on the precision of the predections , we trained SGDMF model using different values of K and we obtained the following graph :

We observe that for the early epochs (< 20) all the models undergo the same downtrend

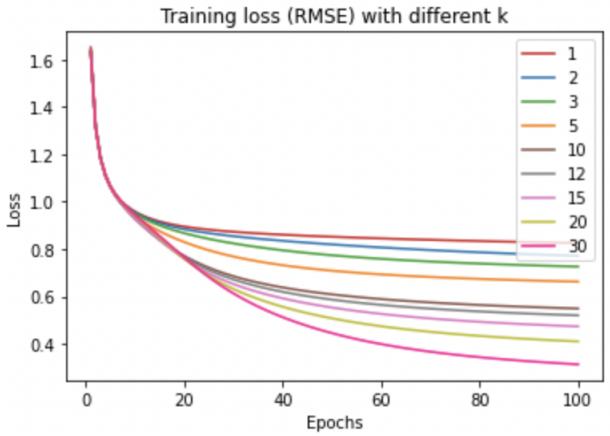

Figure 6: [SGDMF] with different K values

of the loss , after that they converge towards different values where bigger values of K gives better values of the loss. In contrast to running time, as K goes higher so does the training time.

We trained our ALSMF model and validate it against a validation set, and then we plot the train and validation loss in the graph below:

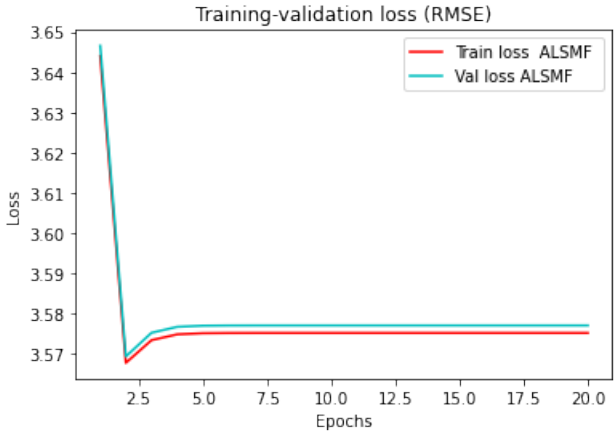

Figure 7: [ALSMF]

We can observe that the loss using ALS model converges towards a value between 3.57 and 3.58 after just 3 epochs and this is due to the fact that we fix one variable at a time so it is easier to the model to find a global minimum in short time.

4.2 ALSMF vs SGDMF :

In order to compare the performance of our two approaches ALSMF and SGDMF, we validate the models against the same validation set and plot the graph below:

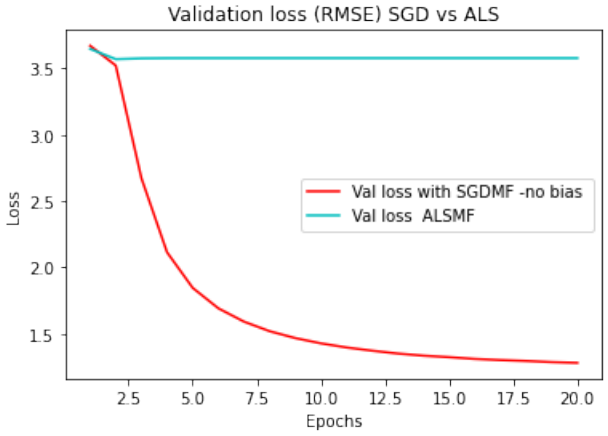

Figure 8: [SGDMF] vs [ALSMF]

We notice that SGDMF outperforms ALS model in term of precision of the results since it gives lower values of the loss metric, yet the ALSMF has the advantage of fast convergence. It is important to notice that, in term of running time, SGDMF is **6 times** faster than ALSMF.

5 Conclusion

In this report we addressed the problem of Matrix factorization, where we implemented two approaches to solve it which are : stochastic gradient descent and Alternating least squares. Through this study, we can conclude that stochastic gradient descent with biases is a good solution for this problem, since it gives better results and takes less running time.

Thanks to the added value of Matrix factorization to recommender systems, an important extensions are being studied such as the requirement that all the elements of the factor matrices U and I should be non-negative. In this case it is called non-negative matrix factorization (NMF). Other studies leverage Deep learning models to solve the problem.