# Machine Learning Project II Report

Joaquim Campos
joaquim.campos@epfl.ch

Simão Sarmento
simao.moraessarmento@epfl.ch

Rodrigo Bernardo
rodrigo.moreirabernardo@epfl.ch

***Abstract*—In this project we implemented a Matrix Factorization technique for recommendation systems. In the final implementation we used a modified version of ALS with biases, for which we provide the derivation for the update of one of the biases, as an example.**

## I. Introduction

This report describes an implementation of a recommender system based on Matrix Factorization. This problem is of high commercial relevance since, after training a model, we could predict the rating that a certain user $u$ would give to a list of non-rated movies, sort the result by descending order and recommend the movie with the highest predicted rating.

We start by describing the data from which the model is built in Section II. Further on, we describe in Section III the baseline predictors we use for the task. Then, two different implementations of Matrix Factorization are explored: one using SGD with biases, described in Section IV, and another one using a modified ALS with biases, described in Section V. The later gave rise to our final implementation and a theoretical derivation for one of the biases is detailed, since we have not seen it detailed elsewhere. The derivation for the other bias is analogous and the matrix updates only suffer minor changes compared to the version implemented in the lab.

In Section VI we discuss the inclusion of implicit ratings in our ALS algorithm and in the SVD (NSVD/SVD++) and discuss the computational difficulties we faced while implementing it.

Results are described in section VII.

## II. Data Analysis

The data we will be working with consists in a list of ratings given by a certain user $u$ to an item $i$. More concretely, we are dealing with 1176952 different entries in total, where each entry corresponds to a rating. From the data, we infer that there are ten thousand distinct items, along with a thousand different users. Moreover, we verify that the minimum number of items per user is eight. On the other hand, the minimum number of users per item is three.

We can notice that some users rated thousands of movies, while others rated less than ten. On the other hand, some movies are very popular and have hundreds of ratings while others have less than five ratings. This asymmetry is one of the difficulties when dealing with recommendation systems. The problem of prediction for users or items with few ratings is called the "cold start" problem. Unlike in the lab, we did not discard users and movies with few ratings. We could have done this and then predict the entries for the users/movies that were removed by other means like K-means (no pun intended) or using baseline predictors, but from our research, we found using the whole data for matrix factorization would still be better.

For the data splitting, we found that a ratio of 10% test data was an adequate compromise between overfitting and amount of training data. As a preprocessing step, we tried transforming the data by applying $X_{dn} = \sqrt{6 - X_{dn}}$, which according to [1] would "make the model better match the data" (matrix factorization is in fact a Gaussian model), but we did not get better results.

## III. Baseline Predictors

We use the notation $A_{d:}$ to represent the row $d$ of the rectangular matrix $A$ and $A_{:n}$ to represent the column $n$ of $A$.

We use the following models, that use the mean to predict, as baselines:

- Global Mean: $\hat{X} := \frac{1}{|\Omega|} \sum_{(d,n)\in\Omega} X_{dn}$;
- User Mean: $\hat{X}_n := \frac{1}{|\Omega_{:n}|} \sum_{d\in\Omega_{:n}} X_{dn}$;
- Movie Mean: $\hat{X}_d := \frac{1}{|\Omega_{d:}|} \sum_{n\in\Omega_{d:}} X_{dn}$;

Where $\Omega$ is the set of non-zero rating indices $(d, n)$ in the training data matrix, $\Omega_{:n}$ is the set of movies rated by the $n$-th user, and $\Omega_{d:}$ is the set of users who have rated the $d$-th movie.

These were used as a baseline term of comparison with the other methods. We can observe that the best predictor was the user mean. This shows that for the same movie, there is more variation of ratings around the mean than for the different item ratings given by each user. This is to expect since, for example, some users give conservative ratings almost independently of the movie they are rating. We could also use a baseline predictor that depends in the global mean and on the deviations of user $n$ and item $d$ from the average.

$$b_{dn} = \mu + \beta_n + \gamma_d \tag{1}$$

$\gamma_d$ and $\beta_n$ are called the biases of item $d$ and user $n$, respectively. We use these in the methods described next.

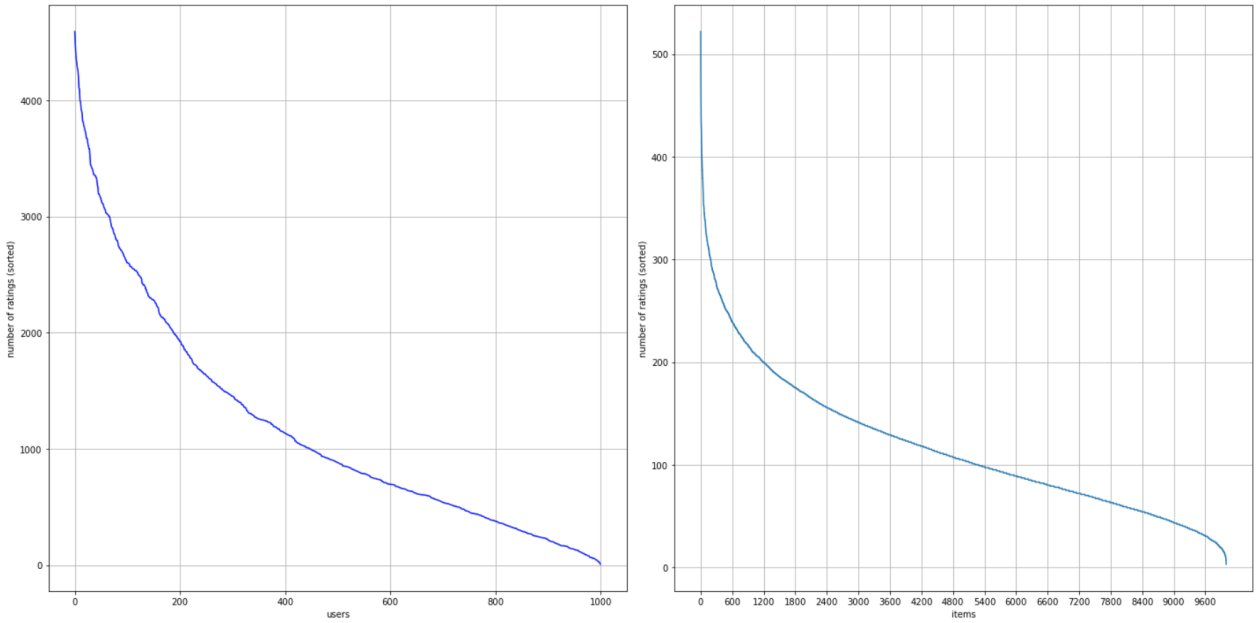


Figure 1. Distribution of ratings per user and per item

## IV. Matrix Factorization using Stochastic Gradient Descent (SGD) with Biases

We experimented with the most discussed method in class (SGD with matrix factorization). We slightly improved the computational speed by vectorizing the error calculation and making other minor changes. We then changed the loss function to include biases.

$$\hat{X}_{dn} = \mu + \beta_n + \gamma_d + W_{:d}^T Z_{:n} \tag{2}$$

In order to learn the model parameters $(\beta_n, \gamma_d, Z_{:n}, W_{:d})$ we minimize the regularized squared error

$$\begin{aligned} L(W, Z) = &\sum_{(d,n)\in\Omega} (X_{dn} - \mu - W_{:d}^T Z_{:n} - \beta_n - \gamma_d)^2 \\ &+ \sum_n (\lambda_{user} \left\| Z_{:n} \right\|^2 + \lambda_\beta \beta_n^2) \\ &+ \sum_d (\lambda_{item} \left\| W_{:d} \right\|^2 + \lambda_\gamma \gamma_d^2) \end{aligned} \tag{3}$$

with $\lambda_{item}, \lambda_{user}, \lambda_\gamma, \lambda_{beta}$ being the regularization parameters, which were learned using cross-validation (see Section VII for the values of the best parameters). Using different regularization parameters for different parameters allows for a more fine-grained optimization, though it also increases the computational complexity. However, in practice we found that using the same regularization for all the parameters, did not produce considerable differences.

The updates for the biases are calculated the same way as for the matrices updates (take the gradient, equal to zero). We get to the following expressions for the updates:

- $\beta_n \leftarrow \beta_n + l \cdot (e_{dn} - \lambda_\beta \cdot \beta_n)$
- $\gamma_d \leftarrow \gamma_d + l \cdot (e_{dn} - \lambda_\gamma \cdot \gamma_d)$
- $Z_{:n} \leftarrow Z_{:n} + l \cdot (e_{dn} \cdot W_{:d} - \lambda_{user} \cdot Z_{:n})$
- $W_{:d} \leftarrow W_{:d} + l \cdot (e_{dn} \cdot Z_{:d} - \lambda_{item} \cdot W_{:d})$

where the error is defined as $e_{dn} := X_{dn} - \hat{X}_{dn}$ and $l$ is the learning rate [2].

### A. The importance of biases

The matrix multiplication tries to capture the interactions between users and items that produce the different rating values. However, much of the observed variation in rating values is due to effects associated with either users or items, known as biases, independent of any of the interactions. For example, typical collaborative filtering data exhibits large systematic tendencies for some users to give higher ratings than others, and for some items to receive higher ratings than others. After all, some products are widely perceived as better or worse than others [3].

## V. Matrix Factorization using Alternating Least Squares (ALS) with Biases

The main advantage of ALS compared to SGD is that we can turn the non-convex optimization problem (3) into a quadratic one if we fix either $W$ or $Z$. ALS fixes each one of these alternately. When one is fixed, the other one is computed, and vice versa. Because of this, it is also very easy to parallelize, since we can do the $W$ and $Z$ update in parallel, but, by sequentially updating $W$ and $Z$, we can also use the recent update of the user features, for example, in the subsequent computation of the item features, as we did. We saw in the lab that these quadratic problem optimization (once you fix one of the matrices) leads to a much faster

convergence than in the case of the SGD and generally gives better results.

We started by optimizing the code from the lab to its fullest. We vectorized the error computation again and did some optimizations elsewhere. With this, we reduced the ALS computation time to a third of the code from the lab. Then, we tried adding the biases. We show our theoretical derivation next (because we haven't seen it anywhere else).

We optimized the item/user biases right after (in the same loop) as the corresponding, newly updated, item/user matrix, because they use the same data.

In pratice, we created a modified version of ALS bias. Using this new loss function from the derivation that we show in the next subsection we should also change accordingly the updates for the matrices to account for the biases. However, we found better results if we did not account for the biases in the matrix update and only added the biases at end. In any case, it was expected that this would not produce a dramatic difference since the sum of the biases, which is used for prediction, will be a number close to zero anyway, but we still can not fully explain why this tweaked version gives better results than the theoretically sound version.

### A. Bias update

Let $W \in \mathbb{M}_{K\times D}$, and fix user matrix $Z \in \mathbb{M}_{K\times N}$, user bias $\beta$ and item bias $\gamma$, where $D$ is the number of items and $N$ is the number of users.

$$\begin{aligned} L(W,Z) = &\sum_{(d,n)\in\Omega} (X_{dn} - W_{:d}^T Z_{:n} - \beta_n - \gamma_d)^2 \\ &+ \sum_n (\lambda_{user} \left\|Z_{:n}\right\|^2 + \lambda_\beta \beta_n^2) \\ &+ \sum_d (\lambda_{item} \left\|W_{:d}\right\|^2 + \lambda_\gamma \gamma_d^2) \end{aligned} \tag{4}$$

Deriving equation 4 with respect to $\beta_{n'}$ for some $n'$,

$$\begin{aligned} \frac{1}{2}\frac{\partial}{\partial \beta_{n'}} L(W,Z) = &\left[ - \sum_{d:(d,n')\in\Omega} X_{dn'} - W_{:d}^T Z_{:n'} - \beta_{n'} - \gamma_d \right] \\ &+ \lambda_\beta \beta_{n'} \end{aligned} \tag{5}$$

Let us write $S_{n'} := \{d : (d,n') \in \Omega\}$, $n'$ fixed. For convenience, since the user is fixed for this computation, we will write $S \equiv S_{n'}$. Also, let us write $y_{n'}[S] := X_{:n'}[S] - \gamma[S]$, where $X_{:n'}[S]$ is a vector consisting of the non-zero ratings for user $n'$, that is, only selecting the rated items for user $n'$.

$$\beta_{n'}(|S| + \lambda_\beta) = \mathbf{1}_{|S|}^T \cdot \left( y_{n'}[S] - \begin{bmatrix} W_{S_1}^T \\ \vdots \\ W_{S_{|S|}}^T \end{bmatrix} \cdot Z_{:n'} \right) \tag{6}$$

where $\mathbf{1}_{|S|}^T$ is a vector of ones of size $|S|$.

$$\beta_{n'} = \frac{\mathbf{1}_{|S|}^T}{(|S| + \lambda_\beta)} \cdot \left( y_{n'}[S] - \begin{bmatrix} W_{S_1}^T \\ \vdots \\ W_{S_{|S|}}^T \end{bmatrix} \cdot Z_{:n'} \right) \tag{7}$$

To calculate the final update on the bias vector $\beta = \begin{bmatrix}\beta_1 \dots \beta_n\end{bmatrix}^T$ at each iteration we use the equation 7 to calculate for each entry $n' \in \{1, \dots, N\}$.

## VI. Matrix Factorization using Implicit Ratings

Implicit ratings methods is a name used for models that incorporate additional information that is not given explicitly by users (ratings) but that still provides useful insights into user preferences. For example, a retailer can use its customers' purchases or browsing history to learn their tendencies, in addition to the ratings those customers might supply [3]. In this case, the only type of implicit information we can use is the fact that a user chose to rate certain movies, independently of the rating that he gave to that movie. In a way, choosing to rate a movie can already give information on the user preference for that movie or non-preference for others. To incorporate implicit ratings, we implemented SVD++. To this end, a second set of items factors is added, relating each item $i$ to a factor $y_i \in \mathbb{R}^K$. Those new item factors are used to characterize users based on the set of items that they rated. The exact model is as follows:

$$\hat{X}_{dn} = \mu + \beta_n + \gamma_d + W_{:d}^T \left( Z_{:n} + |S_n|^{-\frac{1}{2}} \sum_{i\in X_{:n}[S_n]} y_i \right) \tag{8}$$

where the set $S_n$ is defined as previously. One large advantage is that we can include information from the sample submission. For each user $n$ we can not only see which items were rated in the training set, but also in the sample submission, and include them in the set $S_n$, even though these ratings are hidden from us. The model parameters are again computed minimizing the squared error function using the prediction above and regularizations.

- $\beta_n \leftarrow \beta_n + l \cdot (e_{dn} - \lambda_\beta \cdot \beta_n)$
- $\gamma_d \leftarrow \gamma_d + l \cdot (e_{dn} - \lambda_\gamma \cdot \gamma_d)$
- $Z_{:n} \leftarrow Z_{:n} + l \cdot (e_{dn} \cdot W_{:d} - \lambda_{user} \cdot Z_{:n})$
- $W_{:d} \leftarrow W_{:d} + l \cdot (e_{dn} \cdot (Z_{:d} + |S_n|^{-\frac{1}{2}} \sum_{i\in X_{:n}[S_n]} y_i) - \lambda_{item} \cdot W_{:d})$
- $\forall_{i\in S_n}$: $y_i \leftarrow y_i + l \cdot (e_{dn} \cdot |S_n|^{-\frac{1}{2}} \cdot W_{:d} - \lambda_y \cdot y_i)$

where again the error is defined as $e_{dn} := X_{dn} - \hat{X}_{dn}$ and $l$ is the learning rate [2]. We tried implementing this using the python "surprise" library. However, it was computationally very heavy and we were not able to run it until the end. We also tried implementing, but each iteration took too much time, so we do not include any results. This happens due to the inclusion of the implicit factors matrix $y$, where

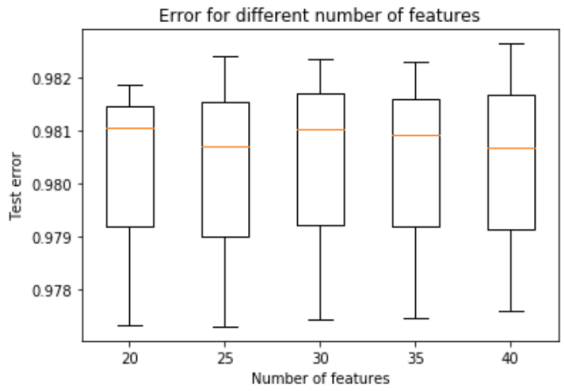


Figure 2. Test RMSE for different values of $K$

in each iteration we have to run a new cycle to do all the updates.

## VII. Results

| Global Mean | Item Mean | User Mean |
|---|---|---|
| 1.119 | 1.093 | 1.031 |

Table I
RMSE FOR THE BASELINE PREDICTORS

| SGD | SGD bias | ALS bias | Modified ALS Bias |
|---|---|---|---|
| 1.010 | 0.998 | 0.982 | 0.979 |

Table II
RMSE FOR DIFFERENT METHODS

| $K$ | $\lambda_{user}$ | $\lambda_{item}$ | $\lambda_\beta$ | $\lambda_\gamma$ | Stop Criterion |
|---|---|---|---|---|---|
| 25 | 0.1 | 0.1 | 0.1 | 0.1 | $0.5e{-4}$ |

Table III
BEST PARAMETERS FOR MODIFIED ALS BIAS

## VIII. Conclusion

We had difficulty in the implementation of more advanced methods, like SVD++, due to computational resources needed to run them. We found that SGD with biases included was not sufficiently good for our objectives. Our modified ALS with biases reached much better results with a relatively good computation time. We can also add that we could also have reached a lower RMSE if we had timing information, like timestamps (which the participants of the Netflix Prize competition had access to), since newer movies turn to be rated more liberaly.

## References

[1] Kevin P. Murphy, *Machine Learning: A Probabilistic Perspective*, 2012.

[2] Francesco Ricci, Lior Rokach, Bracha Shapira, Paul B. Kantor, *Recommender Systems Handbook*, Springer.

[3] Yehuda Koren, Robert Bell, Chris Volinsky, *Matrix Factorization Techniques for Recommender Systems*, IEEE, 2009.