Recommender system

Jadwiga Świerczyńska

1 Introduction

In this report we present few approaches to design a recommender system. We try to predict ratings with data of users who rated some of the movies. To be more specific, we are given a set of ratings in a form

`userId`	`movieId`	`rating`	`timestamp`
1	1	4.0	964982703
1	3	4.0	964981247
1	6	4.0	964982224
⋮	⋮	⋮	⋮
2	115713	3.5	1445714854
2	122882	5.0	1445715272
⋮	⋮	⋮	⋮

Ratings of the movies are numbers from 0 to 5. The above data means that user 1 gave movie 3 rating 4.0, user 2 gave movie 122882 rating 5.0, etc (the `timestamp` column is irrelevant for us). We have around 100000 ratings – around 600 users rated around 9000 movies. Therefore we have a lot of missing values and our goal is to design a recommender system which will be able to predict the ratings.

Of course it would be hard to evaluate our model when we do not know the future and the ratings that users will assign to the yet unrated movies. Hence we split the data into two disjoint sets: train set and test set. We will train our model on a train set and then evaluate how well it predicted the ratings from the test set – more details in the next sections.

In the further sections we present 4 methods of training the model – NMF (Non-Negative Matrix Factorization), SVD1 (Singular Value Decomposition), SVD2 (Iterated Singular Value Decomposition), SGD (Stochastic Gradient Descent) and the results obtained with each of them.

2 General analysis

Now we will reformulate our problem in order to present some mathematical ideas behind the solutions. Let's denote $n = 610$ – number of users and $d = 9724$ – number of movies. Without the lost of generality we will assume that movies are numbered from 1 to d. Now we can think of our data as a matrix $\mathbf{Z}$ of size $n \times d$, where $\mathbf{Z}[i, j]$ denotes the rating that user i assigned to movie j. Let's denote $\mathcal{I}$ – set of pairs (i, j) for which entries in $\mathbf{Z}$ are defined.

2.1 Model evaluation

We split $\mathcal{I}$ into two disjoint, nonempty sets: $\mathcal{I}_{train}$ and $\mathcal{I}_{test}$. Denote $\mathbf{Z}_{train}$ – matrix of size $n \times d$ such that $\mathbf{Z}_{train}[i,j] = \mathbf{Z}[i,j]$ for all $(i,j) \in \mathcal{I}_{train}$. Assume that our model approximated $\mathbf{Z}$ as $\mathbf{Z}'_{train}$ (matrix in which all entries are defined). Then the *quality* of the system is defined as **root-mean squared error (RMSE)**, i. e.:

$$q(\mathbf{Z}'_{train}, \mathcal{I}_{test}, \mathbf{Z}) = \sqrt{\frac{1}{|\mathcal{I}_{test}|} \sum_{(i,j)\in\mathcal{I}_{test}} (\mathbf{Z}'_{train}[i,j] - \mathbf{Z}[i,j])^2}.$$

Our goal is to minimize the RMSE.

2.2 Cross-validation

In order to find the best parameters of the model we need to split the data into test and train set many times and find those which give the best results. In this report we use a **10-fold cross-validation** method. The algorithm is as follows:

1. Split the data into 10 parts numbered from 1 to 10, such that each part contains around 10% of each user data (if a number of a particular user's entries is not divisible by 10, then split the remainder randomly between 10 parts).
2. For $i = 1, 2, \ldots, 10$ calculate q_i – quality of the model trained on data from all parts except i-th and tested on the data from the i-th part.
3. Calculate mean quality $q = \frac{q_1+q_2+\ldots+q_{10}}{10}$. This is the approximation of the quality of the model with given parameters.
4. Choose the hyperparameters that give the best mean quality.

2.3 Initial values

Some of the algorithms used in our approach need the matrix $\mathbf{Z}_{train}$ to have all entries defined in order to work. First solutions that come to mind are to fill undefined entries with 0, mean rating of the particular user, mean rating of the particular movie, mean rating of all the movies and users.

However we will present more sophisticated approach that gives better results.

Intuitively we would like to combine the approach with assigning mean of particular movie and mean of particular user. Assume that there are users u, u', v; users u' and v rated movie m whereas u did not. If we know that u and u' gave similar ratings to the movies they both watched and on the other hand u and v gave completely different ratings to the movies they both watched, then we would like to trust more user u' when calculating the initial rating for user u and movie m.

We would like to somehow measure this *similarity* and when we will try to calculate the initial rating of the user u and movie m we will take into the account ratings of K users most similar to u (we will refer to them as the *neighbourhood* of u).

For users x and y we will denote set of movies from train set rated by both x and y as $M_{x,y}$. Furthermore we will denote mean rating of user x of all movies from train set as $\overline{r}_x$. We will use

Pearson correlation similarity of users x and y, such that $|M_{x,y}| \geq 5$, defined as

$$simil(x,y) = \frac{\sum_{m \in M_{x,y}} (\mathbf{Z}_{train}[x,m] - \overline{r}_x)(\mathbf{Z}_{train}[y,m] - \overline{r}_y)}{\sqrt{\sum_{m \in M_{x,y}} (\mathbf{Z}_{train}[x,m] - \overline{r}_x)^2} \sqrt{\sum_{m \in M_{x,y}} (\mathbf{Z}_{train}[y,m] - \overline{r}_y)^2}}.$$

If $|M_{x,y}| < 5$, the similarity is undefined.

Let N_x be a neighbourhood of user x (K users for which $simil(x, \cdot)$ has the greatest value). Assume that x didn't rate movie m and let $N_{x,m}$ be a subset of N_x of those users who rated movie m. If $N_{x,m}$ is non-empty, then the initial rating of user x for movie m is equal to

$$\max \left(\overline{r}_x + \frac{1}{k} \sum_{y \in N_{x,m}} simil(x,y)(\mathbf{Z}_{train}[y,m] - \overline{r}_y),\ 0 \right),$$

where $k = \sum_{y \in N_{x,m}} |simil(x,y)|$. The maximum above guarantees that all of the ratings are non-negative. If $N_{x,m}$ is empty, then the initial rating is equal to $\overline{r}_x$.

To evaluate which way of filling the missing values is better, we can measure the RMSE without using any additional algorithm. The results are below.

method	**RMSE**
zeros	3.6534
user mean	0.9409
movie mean	0.9704
Pearson correlation coefficient, $K = 50$	0.9776
Pearson correlation coefficient, $K = 150$	0.9007
Pearson correlation coefficient, $K = 200$	0.8869

We can see that zeros are the worst method as one would expect. However it's worth noticing that user mean gives better results than movie mean – and it can explained by the fact that particular user's movie rating is correlated more with this particular user's other ratings then ratings of this movie of all the other users.

It seems that Pearson correlation coefficient with $K = 200$ should give us the best results. On the other hand result for $K = 150$ is similar and what's more – greater value of K slows down the computations. Moreover experiments for SVD and NMF showed that after applying these algorithms the results for $K = 150$ and $K = 200$ were similar (more details in the next sections).

3 Non-Negative Matrix Factorization

Now we can present our first algorithm which was used to train the model.

Non-Negative Matrix Factorization

Given a matrix $\mathbf{Z}$ of size $n \times d$ with non-negative entries and a hyperparameter $r \leq d$ we try to find non-negative matrices $\mathbf{W}$ and $\mathbf{H}$ of sizes $n \times r$ and $r \times d$ respectively, such that

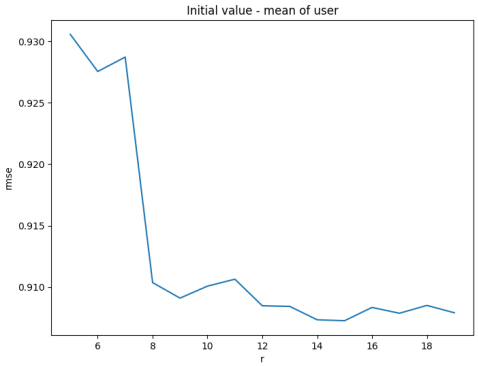
$$||\mathbf{Z} - \mathbf{WH}|| = \min_{\mathbf{W},\mathbf{H} \geq 0} ||\mathbf{Z} - \mathbf{WH}||.$$

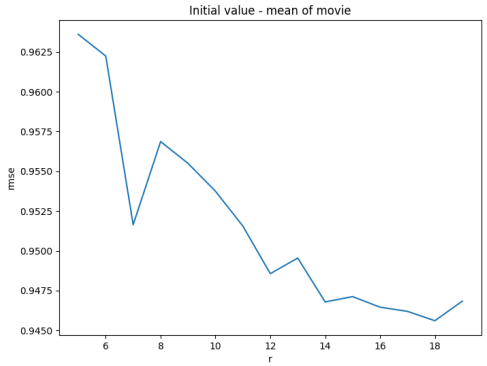
Then we can approximate $\mathbf{Z}$ as

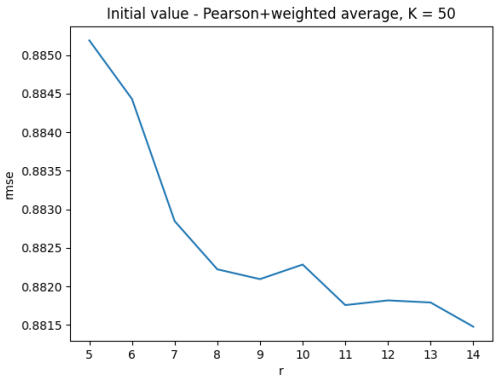
$$\mathbf{Z} \approx \mathbf{WH}.$$

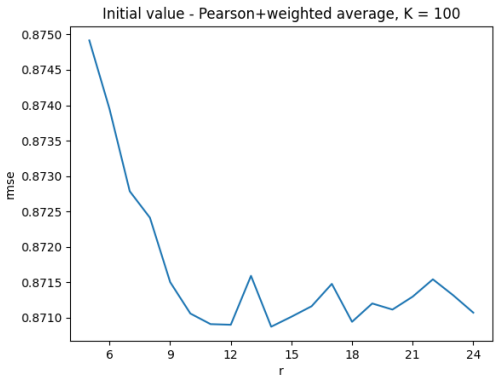
As a consequence of the above if we fill the matrix $\mathbf{Z}_{train}$ with some initial values and then perform NMF, we obtain matrix $\mathbf{Z}'_{train}$ which approximates entries from $\mathcal{I}_{test}$.

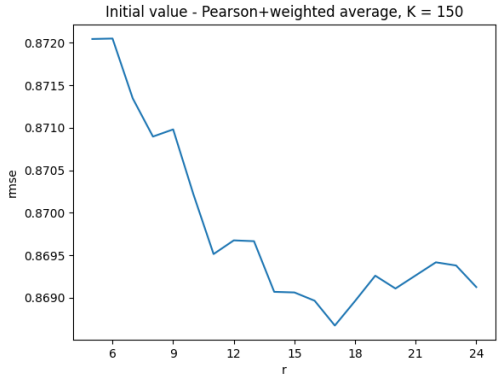
Below we present results obtained with cross-validation for different initial values and values of hyperparameter r.











As we can see from the plots above the best results were obtained for $K = 150$ and $r = 17$, which gave approximately 0.8687 RMSE.

Remarks.

- We tested also other correlation coefficients (Spearman, Kendall Tau), but the obtained results were worse than those for Pearson.
- Increasing the size of the neighbourhood to $K = 200$ does not give noticeably better results.

4 Singular Value Decomposition

Singular Value Decomposition

Theorem. Let $\mathbf{Z}$ be a matrix of size $n \times d$ (assume $n \geq d$). Then $\mathbf{Z}$ can be decomposed to matrices $\mathbf{U}, \mathbf{\Lambda}^{\frac{1}{2}}, \mathbf{V}^T$, such that

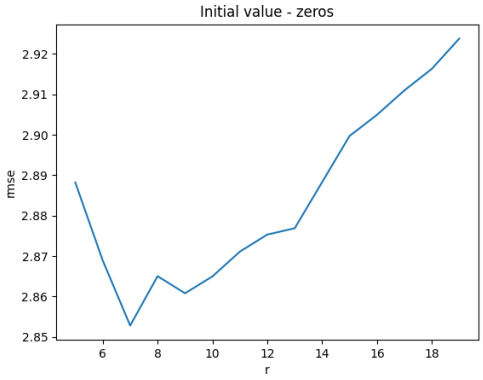
- $\mathbf{Z} = \mathbf{U}\mathbf{\Lambda}^{\frac{1}{2}}\mathbf{V}^T$,
- $\mathbf{U}$ is $n \times d$ matrix with orthonormal columns,
- $\mathbf{V}$ is $d \times d$ matrix with orthonormal columns,
- $\mathbf{\Lambda}$ is $d \times d$ diagonal matrix with non-negative entries.

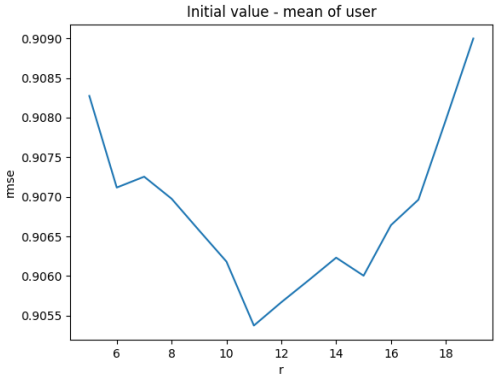
Now we can define **truncated SVD**. Let $\mathbf{Z}$ be a matrix of size $n \times d$ and $\mathbf{U}, \mathbf{\Lambda}^{\frac{1}{2}}, \mathbf{V}^T$ be its SVD. Define $\mathbf{U}_r$ – matrix containing first r columns of $\mathbf{U}$, $\mathbf{V}_r$ – matrix containing first r columns of $\mathbf{V}$ and $\mathbf{\Lambda}_r$ – matrix containing first r columns and r rows of $\mathbf{\Lambda}$. Then we can approximate $\mathbf{Z}$ as

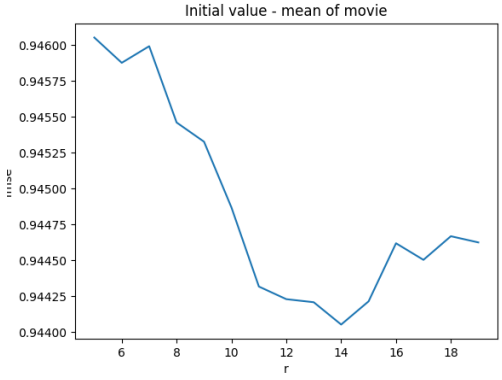
$$\mathbf{Z} \approx \mathbf{Z}_r = \mathbf{U}_r\mathbf{\Lambda}_r^{\frac{1}{2}}\mathbf{V}_r^T.$$

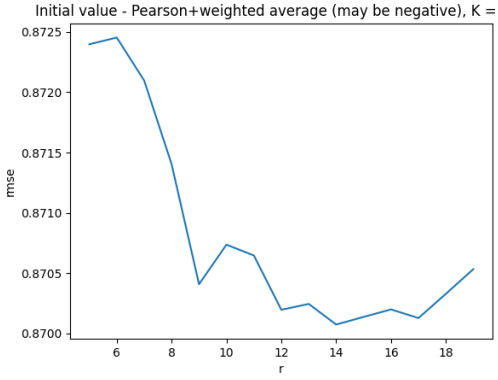
Similarly as in NMF if we fill the matrix $\mathbf{Z}_{train}$ with some initial values and then perform truncated SVD, we obtain matrix $\mathbf{Z}'_{train}$ which approximates entries from $\mathcal{I}_{test}$.

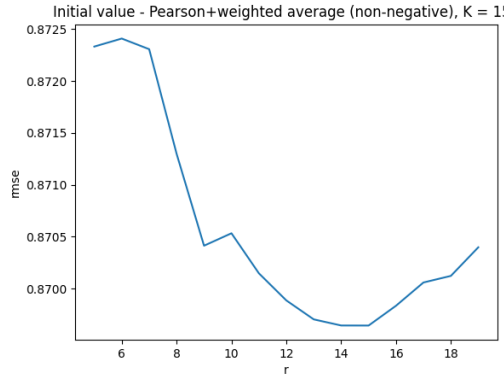
Note that in SVD we do not require the entries in the matrix to be non-negative. Therefore we do not have to apply $\max(\cdot, 0)$ when calculating initial values using weighted averages with Pearson coefficient. Below we present results obtained for different initial values and hyperparameters r.

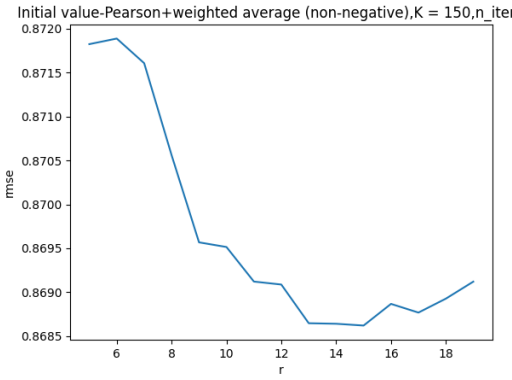

Initial value - zeros
rmse
r

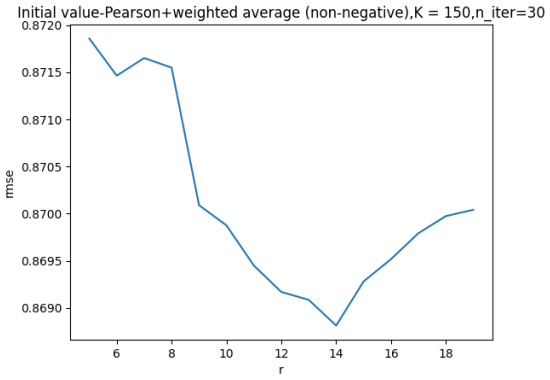

Initial value - mean of user
rmse
r


Initial value - mean of movie
rmse
r


Initial value - Pearson+weighted average (may be negative), K = 150
rmse
r


Initial value - Pearson+weighted average (non-negative), K = 150
rmse
r


Initial value-Pearson+weighted average (non-negative),K = 150,n_iter=10
rmse
r



We can observe that (quite surprisingly) applying $\max(\cdot, 0)$ when calculating initial values gives us slightly better results. The last plot presents results for the SVD with hyperparameter `n_iter` $= 30$ (number of iterations for randomized SVD solver from `sklearn.decomposition.TruncatedSVD`).

We can see that the best results were obtained for initial values with Pearson correlation coefficient, $K = 150$, `n_iter` $= 10$, $r = 14$, which gave us 0.8686 RMSE.

Remarks.

- Increasing `n_iter` to 30 or 100 does not give better results.
- Changing the size of the neighbourhood to $K = 50$ or $K = 200$ does not give better results.

5 Iterated Singular Value Decomposition

Iterated Singular Value Decomposition

Let $\mathbf{Z}$ be a matrix of size $n \times d$ (assume $n \geq d$) with some undefined entries and let $r < d$. Denote $\mathbf{Z}_0$ – matrix $\mathbf{Z}$ in which we somehow filled the undefined entries. Recall from truncated SVD described in the section above that $\mathbf{Z}_0$ can be approximated as $(\mathbf{Z}_0)_r = \mathbf{U}_r \mathbf{\Lambda}_r^{\frac{1}{2}} \mathbf{V}_r^T$. Denote

$$\mathbf{Z}_{k+1}[i,j] = \begin{cases} \mathbf{Z}[i,j] & \text{if } \mathbf{Z}[i,j] \text{ is defined} \\ \mathbf{Z}_k[i,j] & \text{otherwise} \end{cases}$$

for $k = 0, 1, \ldots$. We can approximate $\mathbf{Z}$ as

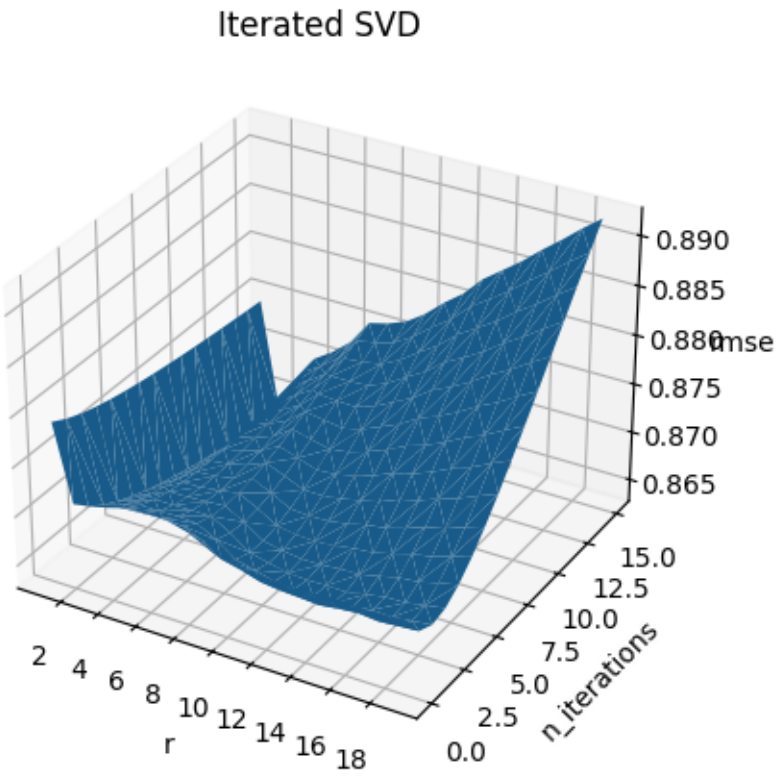
$$\mathbf{Z} \approx \mathbf{Z}_{n_{iterations}}.$$

In other words we iterate SVD to obtain new initial values for undefined entries in $\mathbf{Z}$.

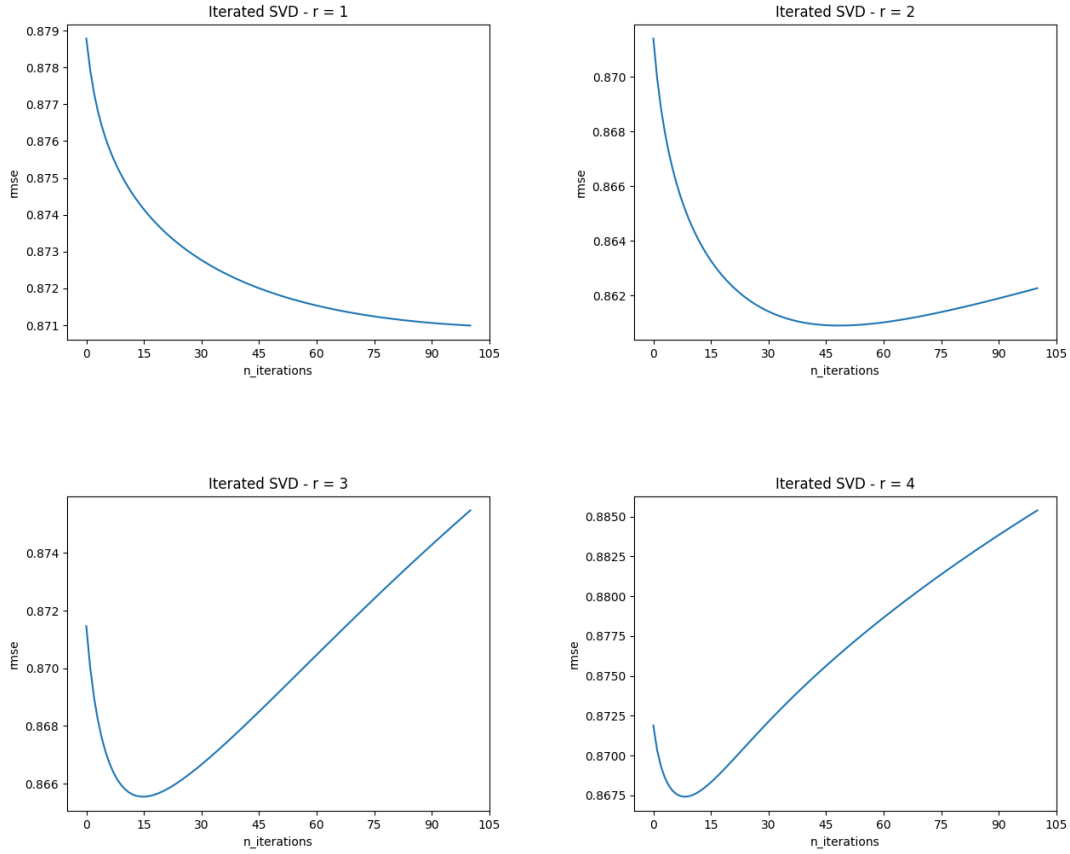
Similarly as before if we fill the matrix $\mathbf{Z}_{train}$ with some initial values and then perform iterated SVD, we obtain matrix $\mathbf{Z}'_{train}$ which approximates entries from $\mathcal{I}_{test}$.

Initial values in this section are weighted means with Pearson coefficients – as they gave the best results in single SVD ($K = 150$). We can see that now there are two hyperparameters that we can optimize: $n_{iterations}$ – number of iterations and r – rank of the approximation.

Below we can see the 3D plot for $r = 1, 2, \ldots, 19$ and $n_{iterations} = 0, 1, \ldots, 15$ (where 0 iterations means single SVD as described in the previous section).



It is easy to see that small values of r (like 2, 3, 4) and big values of $n_{iterations}$ (like 15 and more) should give us the best results. Hence we decided to investigate the behaviour of RMSE for $r = 1, 2, 3, 4$.



We can see that the best results were obtained for $r = 2$ and $n_{iterations} = 47$ which gave 0.8609 RMSE.

6 Stochastic Gradient Descent

Stochastic Gradient Descent

Let $\mathbf{Z}$ be a matrix of size $n \times d$. We would like to find matrices $\mathbf{W}$ and $\mathbf{H}$ of sizes $n \times r$ and $r \times d$ respectively such that

$$(\mathbf{W}, \mathbf{H}) = \underset{(\mathbf{W},\mathbf{H})}{\arg\min} \sum_{(i,j)\in\mathcal{I}_{train}} (\mathbf{Z}[i,j] - \mathbf{W}_i^T \mathbf{H}_j)^2$$

(here $\mathbf{W}_i^T$ is the i-th row of $\mathbf{W}$ and $\mathbf{H}_j$ is the j-th column of $\mathbf{H}$).
Then we can approximate $\mathbf{Z}$ as

$$\mathbf{Z} \approx \mathbf{W}\mathbf{H}.$$

The above gives us idea of the goal of SGD – we just have to find minimum of a function $f(w_{11}, \ldots, w_{nd}, h_{11}, \ldots, h_{rd})$ of $r(n + d)$ variables. One way to achieve it would be to calculate

the gradient of f and to take small steps in the direction opposite to the gradient. After many steps we would finally get an approximation of a local minimum of f. To be more specific, the algorithm for **gradient descent** is as follows (λ (learning rate) and n_{epochs} (number of epochs) are hyperparameters and $M = |\mathcal{I}_{train}|$):

1. Let $x \in \mathbb{R}^{r(n+d)}$ be some initial approximation of $(\mathbf{W}, \mathbf{H})$.
2. For $e := 0, 1, 2, \ldots, n_{epochs} - 1$:
   $x := x - \frac{\lambda}{M} \nabla f(x)$
3. Return x.

However calculating the gradient of f may be costly in time. Therefore we would like to increase the efficiency of this step.

Note that f is given in a form of a sum $f = \sum_{(i,j) \in \mathcal{I}_{train}} f_{(i,j)}$. Hence $\nabla f = \sum_{(i,j) \in I_{train}} \nabla f_{(i,j)}$. The main idea of improvement is to uniformly choose some $(i, j) \in \mathcal{I}_{train}$ and approximate $\nabla f(x)$ as $\nabla f_{(i,j)}(x)$. Then the expected value of a gradient will be $\frac{1}{M} \nabla f(x)$. Therefore algorithm for **stochastic gradient descent** is as follows:

1. Let $x \in \mathbb{R}^{r(n+d)}$ be some initial approximation of $(\mathbf{W}, \mathbf{H})$.
2. For $e := 0, 1, 2, \ldots, n_{epochs} - 1$:
   (a) $indices :=$ randomly shuffled $\mathcal{I}_{train}$
   (b) For $(i, j) \in indices$:
   $x := x - \lambda \nabla f_{(i,j)}(x)$
3. Return x.

However it still may be quite slow, because only one pair (i, j) contributes to the value of gradient at a time and therefore direction pointed by the gradient is inaccurate. The final improvement is to group indices (i, j) in *batches* and all indices from the batch will contribute to the value of a current gradient. The algorithm for **mini-batch stochastic gradient descent** is as follows ($batch\_size$ is a hyperparameter – size of a batch):

1. Let $x \in \mathbb{R}^{r(n+d)}$ be some initial approximation of $(\mathbf{W}, \mathbf{H})$.
2. For $e := 0, 1, 2, \ldots, n_{epochs} - 1$:
   (a) $indices :=$ randomly shuffled $\mathcal{I}_{train}$
   (b) For $batch$ in $indices$:
   i. $step := \sum_{(i,j) \in batch} \nabla f_{(i,j)}(x)$
   ii. $x := x - \frac{\lambda}{batch\_size} step$
3. Return x.

We obtained an algorithm in which we have 4 hyperparameters: r – number of columns in $\mathbf{W}$ and rows in $\mathbf{H}$, n_{epochs}, λ and $batch\_size$. We will try to find the best values of them assuming that they are independent (i. e. if we would like to find for example the best value of r, we would fix values of the other hyperparameters and choose the value of r that gives the best

result – as opposed to the approach where we check all of the combinations of the values of hyperparameters).

Initial approximation of $(\mathbf{W}, \mathbf{H})$ are matrices with entries drawn uniformly from $\left[0, \frac{5}{r}\right)$. Note that

$$\frac{\partial f_{(i,j)}}{\partial w_{xy}} = \begin{cases} 0 & \text{if } i \neq x \\ -2\mathbf{H}[y,j](\mathbf{Z}_{train}[i,j] - \mathbf{W}_i^T \mathbf{H}_j) & \text{otherwise} \end{cases}$$

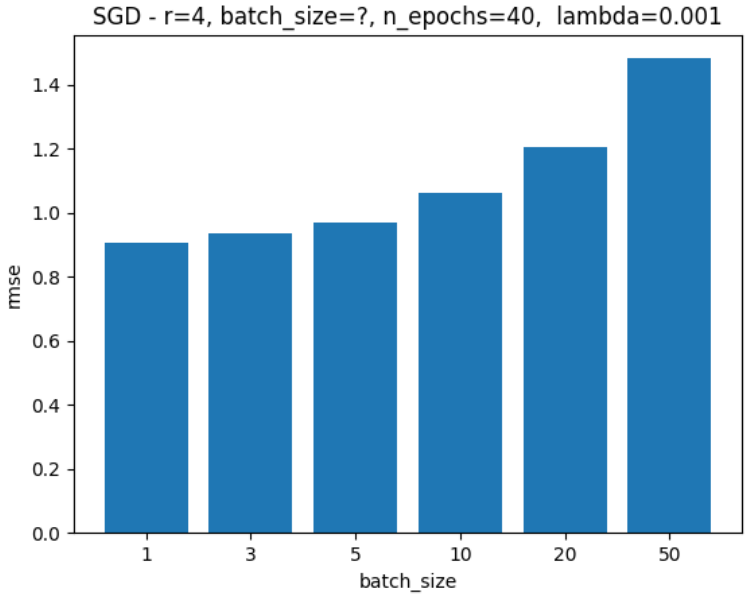
and

$$\frac{\partial f_{(i,j)}}{\partial h_{xy}} = \begin{cases} 0 & \text{if } j \neq y \\ -2\mathbf{W}[i,x](\mathbf{Z}_{train}[i,j] - \mathbf{W}_i^T \mathbf{H}_j) & \text{otherwise.} \end{cases}$$

Below we can see plots presenting results for different values of hyperparameters.

Value of *batch_size*

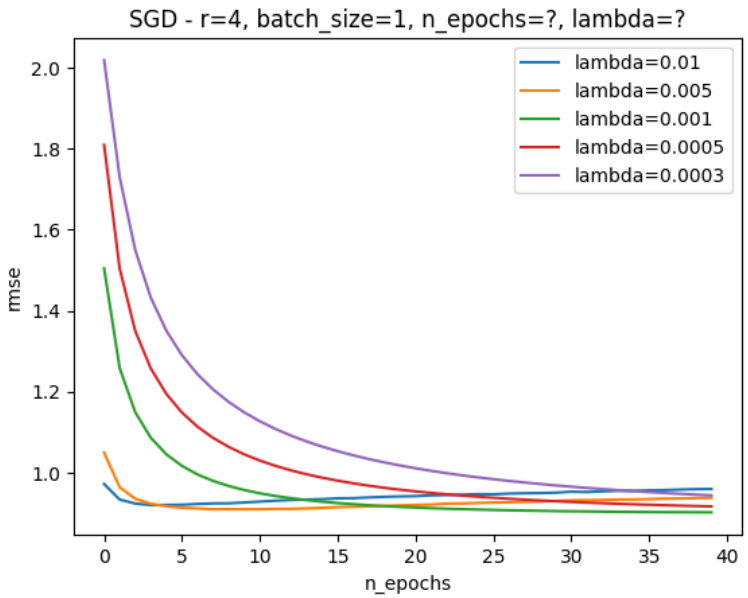
To find the best value of *batch_size*, we fix $n_{epochs} = 40$, $\lambda = 0.001$ and $r = 4$.



We can see that increasing the batch size gave us worse results (however the computations were significantly faster). Nevertheless we will set *batch_size* to 1.

Value of λ

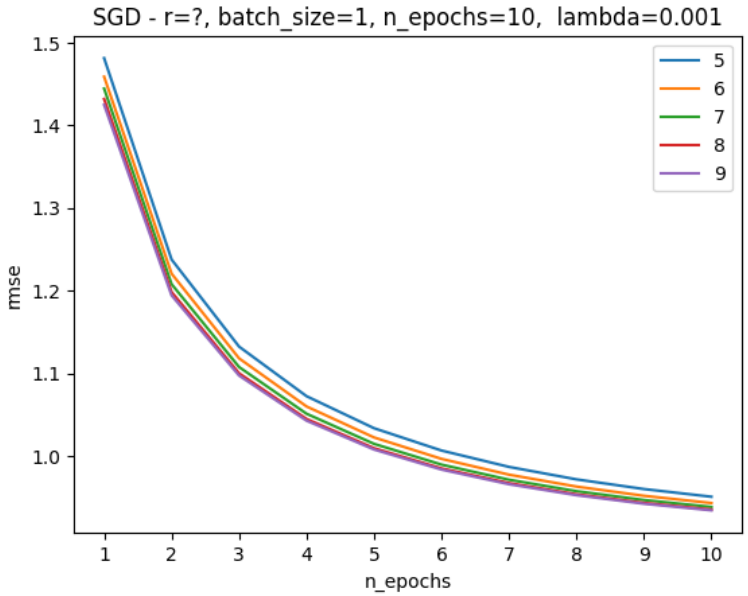
Here we set $r = 4$ and *batch_size* $= 1$. Below we can find a plot for different values of n_{epochs} and λ.



We can see that $\lambda = 0.001$ gives us the best RMSE.

Value of r

To find the best value of r, we fix $n_{epochs} = 10$, $batch\_size = 1$, $\lambda = 0.001$.



Let's take a closer look on what happens for $n_{epochs} = 10$.

r	**RMSE**
5	0.95102
6	0.94338
7	0.93865
8	0.93552
9	0.93458

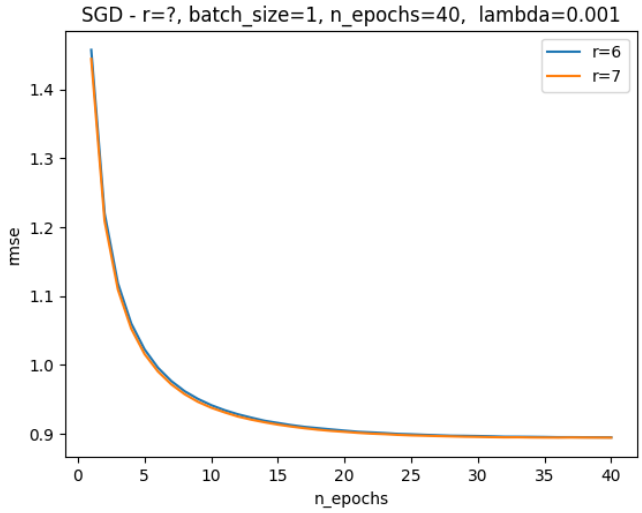
We can see that the best result were obtained for $r = 9$ which gave 0.93458 RMSE. However the results for $r = 7$, $r = 8$ and $r = 9$ are similar – therefore we conclude that greater values of r would not result in significantly better RMSE and on the other hand would significantly slow down the computations. Hence we will focus on $r = 6$ and $r = 7$ as the best values – sweet spots for both the time and RMSE.

Value of n_{epochs}

The last hyperparameter is n_{epochs}. We set $\lambda = 0.001$ and $batch\_size = 1$. However we check two values of r: $r = 6$ and $r = 7$.

Due to a very long time of computations we decided to slightly modify the 10-fold cross-validation algorithm. Instead of calculating the mean quality of all 10 tests corresponding to 10 parts of data, we pick randomly 3 parts and perform the tests only for them. The average quality is the average of the results of those 3 tests.

Below we can find a plot for different values of n_{epochs} and $r = 6$ and $r = 7$



We can see that they both result in similar RMSE in spite of the fact that (as we have seen in the experiment above) in the first epochs $r = 7$ gives better results. Indeed for $n_{epochs} = 40$ we have

r	**RMSE**
6	0.89481
7	0.89437

Therefore we fix $r = 6$ as the best value and $n_{epochs} = 40$.

Finally we obtain that hyperparameters in SGD should be set to $r = 6$, $n_{epochs} = 40$, $batch\_size = 1$, $\lambda = 0.001$.

Further improvement – SGD with *momentum*

SGD described above may sometimes get „stuck" – that is if we are near the minimum and take the gradient steps we may keep going back and forth and not getting any closer to the minimum. To help prevent this situation we will remember the update at each iteration and the update in the next iteration will be a linear combination of the current gradient and the previous update. The algorithm for **stochastic gradient descent with momentum** is as follows:

1. Let $x \in \mathbb{R}^{r(n+d)}$ be some initial approximation of $(\mathbf{W}, \mathbf{H})$.
2. $update := 0$
3. For $e := 0, 1, 2, \ldots, n_{epochs} - 1$:
   (a) $indices :=$ randomly shuffled $\mathcal{I}_{train}$
   (b) For $(i, j) \in indices$:
      i. $update := \alpha \cdot update - \lambda \nabla f_{(i,j)}(x)$
      ii. $x := x + update$
4. Return x.

In the above α is a **decay factor** which helps avoiding oscillations.

7 Summary

Final hyperparameters for particular algorithms are as follows

algorithm	**best hyperparameters**	**RMSE**
NMF	$r = 17$	0.8687
SVD1	`n_iter` $= 10$, $r = 14$	0.8686
SVD2	$r = 2$, $n_{iterations} = 47$	0.8609
SGD	$r = 6$, $n_{epochs} = 40$, $batch\_size = 1$, $\lambda = 0.001$	0.8948

The implementation of the model using these hyperparameters can be found in the file `recom_system_330498.py`. Moreover the script used to perform the tests can be found in `worker.py`.