Movie Recommender System

Kamil Zaborniak, Tomasz Gładki

April 2024

Contents

1 Introduction

This is the first project from course Methods of Classification and Dimensionality Reduction about the system of recommending movies. The purpose of this project is to predict movie ratings of different users, using following mathematical methods: NMF, SVD, SVD2 and SGD.

2 Data

2.1 Origins and source

We are given large data set of movie ratings from website MovieLens. It describes 5-star rating from movie recomendation system. It contains 100836 ratings across 9742 movies by 610 users between March 29, 1996 and September 24, 2018. Users were selected at random for inclusion. All selected users had rated at least 20 movies. No demographic information is included. Each user is represented by an id, and no other information is provided.

2.2 Format of data

It has following format:

Table 1: Format of ratings data

User Id	Movie Id	Rating	Timestamp
1	1	4.0	964982703
1	3	4.0	964981247
1	6	4.0	964982224
1	47	5.0	964982815
1	50	5.0	964982931
...	...	...	...
330	1	4.0	1285904910
...	...	...	...

but in our concern will only be ratings of each movie by each user, so we will consider matrix Z with ratings from users (rows) to each film (columns), so that

$$Z[0,0] = 4.0, \quad Z[0,2] = 4.0, \quad Z[0,5] = 4.0, \quad Z[0,46] = 5.0, \quad \text{etc.}$$

We have to keep in mind that in Python indexing starts from 0, so $Z[0,0]$ means rating of first user to first movie. This matrix will look like this:

$$Z = \begin{pmatrix} 4.0 & & 4.0 & & & 4.0 & \dots \\ & & & & & & \dots \\ 4.0 & & & & & & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 2.5 & & 2.0 & & & & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \end{pmatrix}.$$

As we can see there occurs large problem - many spaces are left blank, we will consider this problem in next subsection.

2.3 Problem of blank spaces

We are considering following ideas of filling this spaces to solve this problem:

1. replace with zeros,
2. replace with mean of ratings for this user (mean by row),
3. replace with mean of ratings for this movie (mean by column),
4. replace with weighted mean of ratings by each user and movie.

We assume that the first method will not be a good prediction, second and third may be close, but the fourth one is expected to have the best results. Weighted mean depends on choosing of parameter α:

$$\bar{\mathbf{Z}} = \alpha \cdot \bar{\mathbf{Z}}_{\text{movie}} + (1 - \alpha) \cdot \bar{\mathbf{Z}}_{\text{user}},$$

we will briefly discuss choosing of this parameter later.

3 Mathematic algorithms

In this section we will consider 4 mathemathic algorithms: NMF, SVD, SVD2, SGD and way to compare them. We are spliting our data set into 2 groups: 90% of them will be used as a training set (Z) for each method and rest will be a test set T.

3.1 Quality of algorithms - RMSE

Let $\mathcal{T}$ denote a set of pairs (u, m) with ratings present in a test set - then the rating will be denoted as $T[u, m]$.

After training on Z our algorithm computes Z', a matrix containing elements $Z'[u.m]$ for $(u.m) \in \mathcal{T}$. Then the quality is computed as **root-mean square error**:

$$\text{RMSE} = \sqrt{\frac{1}{|\mathcal{T}|} \sum_{(u,m)\in\mathcal{T}} (Z'[u, m] - T[u, m])^2}.$$

We will compare algorithms based on value of RMSE, the best will be the one with the lowest RMSE.

3.2 NMF

Non-negative Matrix Factorization is a method that approximates Z of size $n \times d$ as

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

where $\mathbf{W}$ is of size $n \times r$ and $\mathbf{H}$ is of size $r \times d$ (r is parameter) are two non-negative matrices. This method is useful, when we consider $r << d$, so that it actually allows us to compress our data and use less memory - which is very important issue in machine learning. Our aim is to minimize the following difference:

$$\min_{W,H} ||\mathbf{Z} - \mathbf{WH}||$$

in this case by RMSE.

3.3 SVD

Another method in this project is Singular Value Decomposition, which is little different from NMF. This method aproximates Z as:

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

where $\mathbf{U}$ is orthonormal matrix $n \times d$ which columns are eigenvectors, $\mathbf{\Lambda}$ is diagonal matrix with eigenvalues of matrix $\mathbf{Z}$ and orthonormal matrix $\mathbf{V}^T$ which rows are eigenvectors of matrix $\mathbf{Z}$.

We again want to do some compression to use lowest possible memory. So we will take first r columns of $\mathbf{U}$, r columns and rows of $\mathbf{\Lambda}$ and first r rows of $\mathbf{V}^T$, all denoted by lower index r. In upper equation we denote $\mathbf{\Lambda}_r\mathbf{V}_r^T$ as $\mathbf{H}_r$ what makes it look simillar to NMF. It can be properly visualized on this graph:

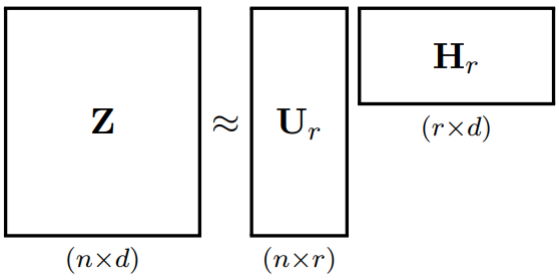

Figure 1: Graph of a Truncated SVD.

Our parameter that will be changed here will be r - we want to take it as small as possible, but also we don't want to loose possible accuracy of SVD approximation.

3.4 SVD2

This method uses SVD multiple times for better approximation. We will try to explain in by an example. We consider following 3×6 matrix $\mathbf{Z}$:

$$\begin{pmatrix} 3 & 0 & 4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 4 & 5 & 0 \\ 1 & 2 & 0 & 2 & 0 & 0 \end{pmatrix}.$$

Than we perform SVD on this matrix with $r = 2$ and get:

$$\mathbf{Z}' = \begin{pmatrix} 3.08 & 0.29 & 3.91 & 0.15 & -0.17 & 0.98 \\ 0.12 & 0.43 & -0.13 & 4.23 & 4.75 & -0.03 \\ 0.5 & 0.15 & 0.58 & 1.01 & 1.08 & 0.14 \end{pmatrix}.$$

After that we restore previous non-zero values to matrix $\mathbf{Z}'$:

$$\begin{pmatrix} 3 & 0.29 & 4 & 0.15 & -0.17 & 1 \\ 0.12 & 0.43 & -0.13 & 4 & 5 & -0.03 \\ 1 & 2 & 0.58 & 2 & 1.08 & 0.14 \end{pmatrix}$$

and than we peform SVD again and after that, we perform again this entire procedure.

In this procedure we control not only the number of components r, but also the number of iterations of this algorithm, however we also want to keep it on minimal level, to use as low memory as possible.

3.5 SGD

The stochastic gradient descent (SGD) method is one of the most popular optimization algorithms used in machine learning. This is a stochastic approximation to gradient descent optimization because it replaces the actual gradient (calculated from the entire data set) with its estimate (calculated from a randomly selected subset of the data). For a given matrix $\mathbf{Z}$ size of $n \times d$ we want to find matrices $\mathbf{W}$ size of $n \times r$ and $\mathbf{H}$ size of $r \times d$, that satisfy:

$$\arg_{W,H} \min \sum_{(i,j):z_{i,j} \neq '?'} (z_{i,j} - w_i^T h_j)^2$$

or (for $\lambda > 0$)

$$\arg_{W,H} \min \sum_{(i,j):z_{i,j} \neq '?'} (z_{i,j} - w_i^T h_j)^2 + \lambda(\|w_i\|^2 + \|h_j\|^2),$$

where h_j is j-th column of $\mathbf{H}$ and w_i is a row of $\mathbf{W}$. To calculate the gradient of a given function $f(w_i, h_j) = \sum_{(i,j):z_{i,j} \neq '?'} (z_{i,j} - w_i^T h_j)^2 + \lambda(\|w_i\|^2 + \|h_j\|^2)$ with respect to the variables w_i and h_i, we first calculate the partial derivatives with respect to them: $\nabla_{w_i} = \frac{\partial f}{\partial w_i}$ and $\nabla_{h_i} = \frac{\partial f}{\partial h_i}$. After obtaining the gradients using the algorithm, we replace:

$$w_i \longleftarrow w_i - \gamma \nabla_{w_i},$$

$$h_i \longleftarrow h_i - \gamma \nabla_{h_i}.$$

To improve SGD, we can use decay, which means that during each iteration we overwrite the γ value.

4 Results

In this section we will consider received results and discuss the optimal parameters.

4.1 NMF

First we will consider what parameter α is the best for filling matrix $\mathbf{Z}$ for this method.

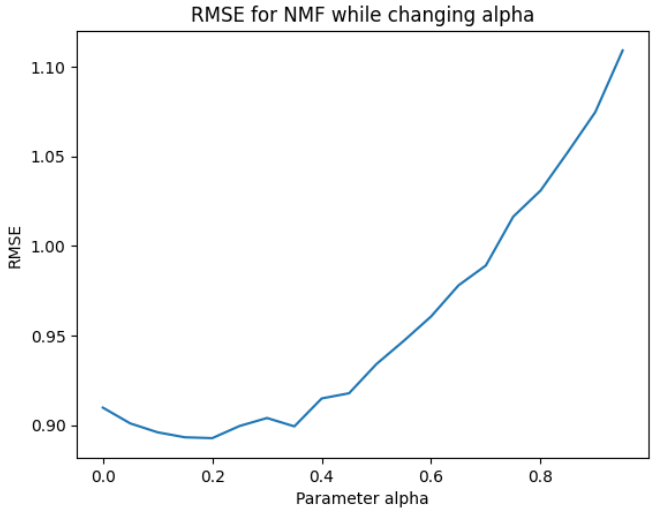

Figure 2: Illustrating change of RMSE dependent on parameter from weighted average.

We can see that RMSE is minimalized near $\alpha = 0.2$, which we choose as the best for our model. Value of RMSE for $\alpha = 0$ corresponds to mean based on rows - other rathings of specific user, while RMSE for $\alpha = 1$ corredponds to mean based on columns - rathings of specific movie by different users. It is clear that usage of mean for users brings better results than mean for movies.

For the best α, weighted mean presents this way:

$$\bar{\mathbf{Z}} = 0.2 \cdot \bar{\mathbf{Z}}_{\text{movie}} + 0.8 \cdot \bar{\mathbf{Z}}_{\text{user}},$$

which means that users rating of each movie is based more on his ratings of other films, than rating of movie by other users.

Now we shall consider the parameter of NMF - what we can choose in this algorithm is paremeter r being resposible for data compression. We will also show it on a plot:

We can see that there are some fluctuations for specific r, however we choose value of 15 as the best. Choice of smaller r is exposed for higher errors, while

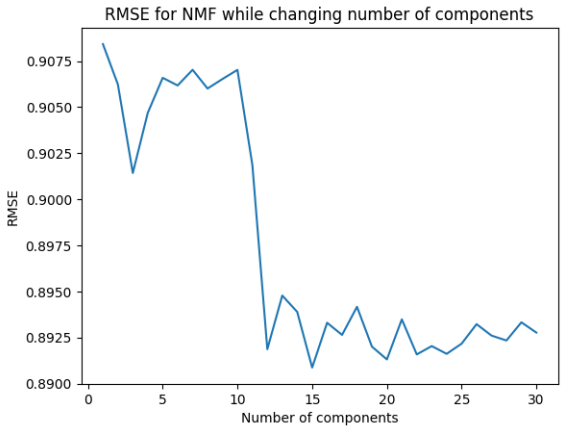

Figure 3: Illustrating change of RMSE dependent on parameter r for number of dimentions taken.

choice of bigger r does not bring much better errors, while also lowering compression of data. With parameter $r = 15$ we are considering matrices 610×15 and 15×9742 instead of 610×9742, this matrices have this many numbers: 9150 and 146130 against 5942620 from $\mathbf{Z}$ matrix. That is great compression in comparison to this large matrix.

4.2 SVD

For this method we will start again with choosing the optimal parameter α for weighted average. We will show it on a plot:

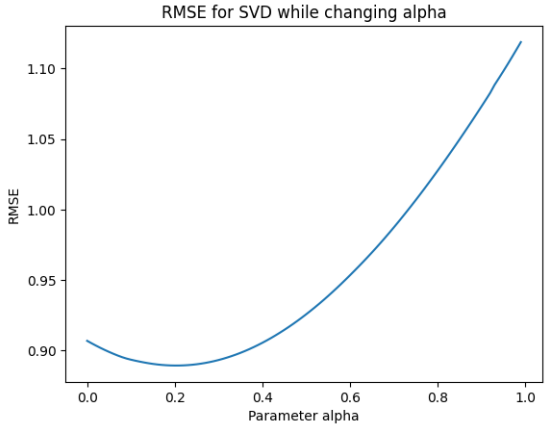

Figure 4: Illustrating change of RMSE dependent on parameter α from weighted average.

We can see that our best choice will be $\alpha = 0.2$ again, we are going to use this parameter in next methods, so we will not discuss or show it further.

Now we are going to consider the optimal choice of parameter r for this algorithm.

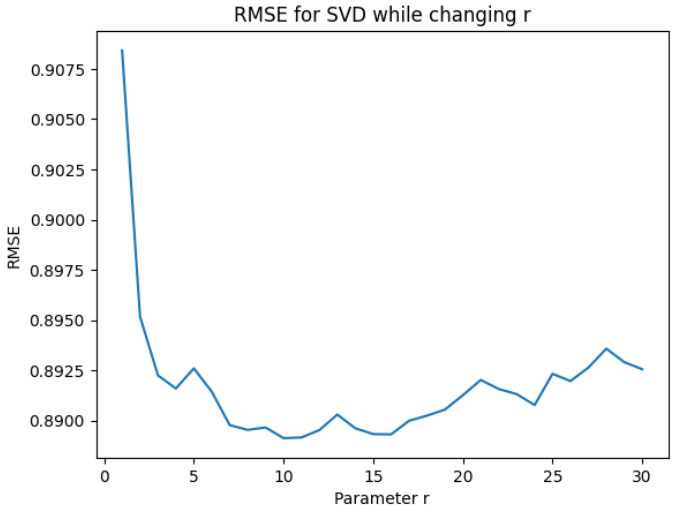

Figure 5: Illustrating change of RMSE dependent on parameter r for number of dimentions taken.

We can see that SVD algorithm quickly approximates value of $\mathbf{Z}$, faster than NMF - optimal value is $r = 10$, which is better than in previous algorithm. That mean we will not only have lower RMSE, but also obtain better compression of data, since $r_{\text{SVD}} < r_{\text{NMF}}$.

4.3 SVD2

For this method we will be considering changes of 2 parameters: number of components taken - r and number of iterations. First let's take a look at parameter r:

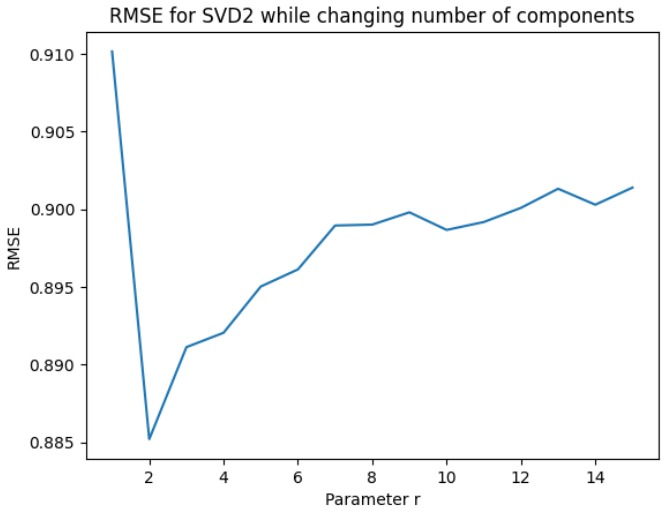

Figure 6: Illustrating change of RMSE dependent on parameter r for number of dimentions taken.

We can clearly see, that the best one is $r = 2$, this was measured for 5 iterations, but now we will see what is the best parameter for number of iterations:

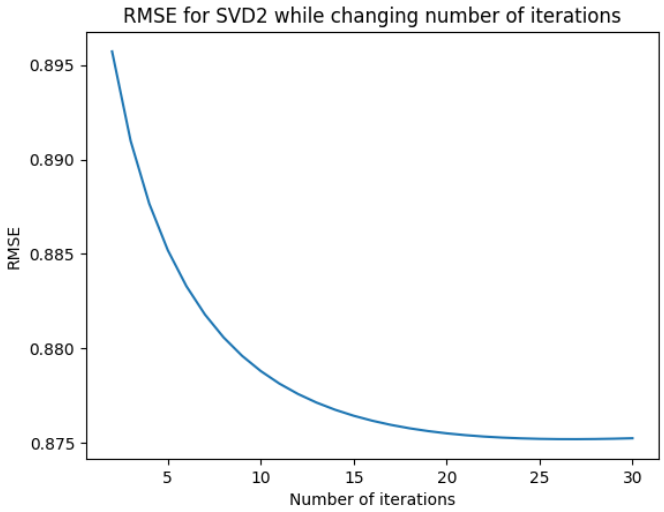

Figure 7: Illustrating change of RMSE dependent on number of done iterations.

We can see that the higher number of iterations the lower RMSE is comuted. However the more iterations, the longer is time of algorithm to work, so the optimal choice of iterations will be 25.

4.4 SGD

For this method we will need to find optimal values of number of factors and number of epochs. Let's take a look on the first one:

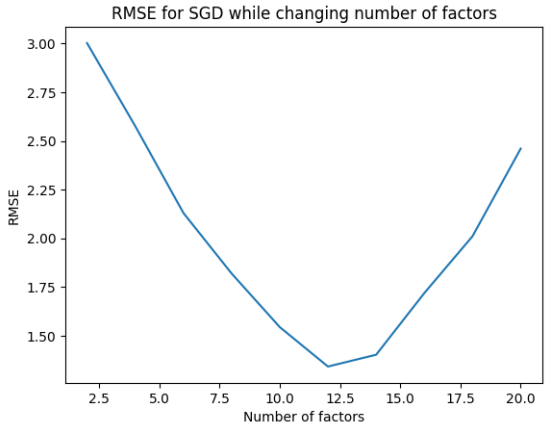

Figure 8: Illustrating change of RMSE dependent on number of factors taken.

We obtain best RMSE for 12, however it is high RMSE, now the second parameter:

Figure 9: Illustrating change of RMSE dependent on number of done epochs.

The best choice is epochs = 20, however again RMSE is much higher than it was in previous algorithms.

5 Conclusion

We will now compare all this methods in table:

Table 2: Format of ratings data

Variable	NMF	SVD	SVD2	SGD
RMSE	0.8928	0.8891	0.8852	1.3519
r	15	10	2	12
iterations	1	1	25	20

We can conclude:

- best RMSE - the lowest - is obtained for method SVD2, the second is SVD, than NMF and the worst score is from SGD method,
- SVD2 is method with best data compresion - $r = 2$ second best in this is SVD, than SGD and NMF,
- the fastest - the lowest iterations are both NMF and SVD - they do just one iteration of entire algorithm, while SVD2 and SGD needs high number of iterations.

So best algorithms for prediction would be SVD and SVD2.

The SVD will be great when we need fast algorithm. It also is a great tool to compress large data sets.

The SVD2 will be great when we need best prediction and lowest possible error. However this will cost a high number of iterations, which means longer time for computing algorithm.

Method NMF is also good, but is not better than two previous methods, while SGD is the worst algorithm here, not only has high RMSE, but also needs very long time to calculate the answers.