IE 510 Project Report

Comparison of Optimization Methods on a Self-built MF Recommendation System

Jinghong Li(jl184)
Fang Li(fangli3)

supervised by Ruoyu Sun
May 2022

Contents

1 Introduction **3**

2 Establishment of Recommendation System **4**
2.1 Dataset . . . 4
2.1.1 Data Structure . . . 4
2.1.2 Train-Test Split . . . 4
2.2 Model . . . 4
2.2.1 Matrix Factorization . . . 4
2.2.2 Data Preprocessing . . . 5
2.2.3 Loss Function . . . 5
2.2.4 Regularization . . . 5
2.3 Training . . . 5

3 Methods & Results **6**
3.1 Gradient Descent . . . 6
3.1.1 Process . . . 6
3.1.2 Results . . . 6
3.2 Stochastic Gradient Descent . . . 8
3.2.1 Process . . . 8
3.2.2 Results . . . 8
3.3 Momentum . . . 9
3.3.1 Process . . . 9
3.3.2 Results . . . 10
3.4 BB method . . . 11
3.4.1 Process . . . 11
3.4.2 Results . . . 12
3.5 BFGS method . . . 13
3.5.1 Process . . . 13
3.5.2 Results . . . 13
3.6 CD . . . 14
3.6.1 Process . . . 15
3.6.2 Results . . . 15
3.7 CCD . . . 15
3.7.1 Process . . . 16
3.7.2 Results . . . 16
3.8 CCD++ . . . 17
3.8.1 Process . . . 17
3.8.2 Results . . . 17

4 Comparison **18**

5 Conclusion **19**

Abstract

Recommendation system is a growing technique for providing a better user experience for discovering new content on a platform. There are many ways to implement recommendation systems. This report investigated some of optimization methods on matrix factorization(MF) problem. Some of them have few past attempts on the MF(BB, BFGS), not to mention, compares with some state-of-the-art methods(SGD, CCD). This project compares the loss function, computation time, accuracy results, and tries to tune hyperparameters to achieve the best possible forecast. BB has been shown to perform outstandingly in our movie recommendation dataset. It helps this recommendation system to be much more efficient with nearly the same accuracy with the highest accuracy of all methods.

Keywords: Matrix Factorization; Optimization; Comparison; The Barzilai-Borwein method

1 Introduction

In a recommendation system, we want to learn a model from past incomplete rating data such that each user's preference over all items can be estimated with the model. Some of the most successful realizations of these models are based on matrix factorization. In its basic form, matrix factorization characterizes both items and user by vectors of factors inferred from item rating patterns. High correspondence between item and user factors leads to a recommendation. More recent work suggested modeling directly the observed ratings only, while avoiding over fitting through a regularized model.

Let $A \in R^{m \times n}$ be the rating matrix, where m and n are the numbers of users and items, respectively. The matrix factorization problem for recommendation systems is

$$\min_{U \in R^{m \times k}; V \in R^{n \times k}} \sum_{(i,j) \in \Omega} (A_{ij} - u_i^T v_j)^2 + \lambda(||U||_F^2 + ||V||_F^2) \tag{1}$$

where Ω is the set of indices for observed ratings, λ is the regularization parameter, and u_i^T and v_j^T are the i_{th} and j_{th} row vectors of the matrices U and V, respectively.

Over the years, a substantial amount of work has been dedicated to the design of fast and low complexity optimisation algorithms for MF. ALS and SGD are the two most popular methods. Simon[1] popularized a stochastic gradient descent optimization of Equation 1 wherein the algorithm loops through all ratings in the training set. Because both u_i and v_j are unknown, Equation 2 is not convex. If we fix one of the unknowns, the optimization problem becomes quadratic and can be solved optimally. ALS techniques rotate between fixing the u_i's and fixing the v_j's, ensures that each step decreases Equation2 until convergence[2]. However, they both have obvious pros and cons: ALS is easily parallelizable but has higher per-iteration computation cost than SGD; in contrast, SGD requires little computation per iteration, but its parallelization is a bit challenging[3].

Except for these mainstream methods, there are some different methods or methods developing from them. Coordinate descent(CD) has been proved to be an effective technique for matrix factorization, it is not only inherently parallel, but less sensitive to the choice of the learning rate. Compared to ALS, CD takes less time for a single iteration[4]. BB has a simple closed-form solution to solve a strongly convex quadratic minimization problem, and it was proved to converge under mild conditions[5]. Kim et al.[6] applied BFGS to the NLS subproblems, which can be further used to solve NMF problems. BFGS approximates the Hessian based on rank-one updates specified by gradient evaluations. Although BFGS converges, it requires a time-consuming line search and a complex Hessian updating procedure, so it is inefficient[7]. CCD++ algorithm choose blocks to be the columns of X, Y, it is shown to be much faster than ALS or CCD algorithm in some real data sets[8]. Although all these algorithms have shown to be useful in solving NMF problem, there is no paper put them all together for comparison in recommendation system application, this report aims at directly comparing the loss functions, computation time and accuracy results of these methods, and trying to tune hyperparameters to achieve the best possible forecast.

This report is organised as follows. The notation and general introduction of our recommendation dataset is introduced in Section 2. All algorithms we use and their experiment results are described in Section 3. In section 4, we present the comparison of all methods. And the conclusions are exposed in section 5.

May 13, 2022

2 Establishment of Recommendation System

2.1 Dataset

2.1.1 Data Structure

Dataset was achieved from Kaggle for this Netflix's movie recommendation system. Due to the the model and hardware limitation, this project choose to work on a given relatively small dataset containing more or less 10,000 examples instead of loading full dataset.

The dataset consist of the ratings provided by specific users for specific movies. Range of rates is from 0.5 to 5.0 stepped by 0.5. For the distribution of ratings, from 0.5 to 5.0, there are respectively 1101, 3326, 1687, 7271, 4449, 20064, 10538, 28750, 7723, 15095 examples correspondingly. In the view of users, this dataset guarantee each users have rated 1 movie at least and the average number of ratings per users is about 149.03.

2.1.2 Train-Test Split

Considering the size of the utlized dataset , the entire dataset was divided into two parts containing train set and test set in the ratio of 8 to 2. So, train set has 80003 examples and test set has 20001 examples. Each dataset is a dataframe contains 'index', 'userId', 'movieId' and 'rating'. In train set, there are 671 users and 8401 movies counted in toall; in test set, there are 670 users and 4863 movies counted in total.

2.2 Model

2.2.1 Matrix Factorization

Based on the goal of this recommendation system which is to predict the possible ratings of movies that users did not ever seen before for each user. Under this circumstance, it is obvious that there are many empty cells containing NULL in the data matrix. So the K-nearest neighbors method cannot be useful for this task. Instead, matrix factorization method is chosen to fulfill the empty entries.

For one recommendation system, there are two such entities as users and movies. This project assumed there are m users and n items. The goal of matrix factorization is to establish an m-many, n-dimensional utility matrix(m×n) which is made up of the rating for each user-movie pair. Due to the lack of user-movie pairs, the utility is obvious to be a sparse matrix initially. Under the goal of this recommendation system, the task becomes finding the similarities between users and movies. For example, simply saying, if user A and user B gave the same high rating to 'Iron Man', it can be said that these two users have similar performance on the choice of movies. Moreover, user B also gave a high rating to 'Avengers: Endgame', so it can be predicted that user A might also like 'Avengers: Endgame'.

The most important part of matrix factorization is decomposition of the utility matrix into two tall and skinny matrices whose dot product is the rating prediction of original data matrix. The equation (2) of decomposition is:

$$P = UV^T \tag{2}$$

where P is m×n matrix, U is m×k matrix and V is n×k matrix. U represents users in low-dimensional space and V represents movies in low-dimensional space. The mechanism can be viewed directly in figure (1).

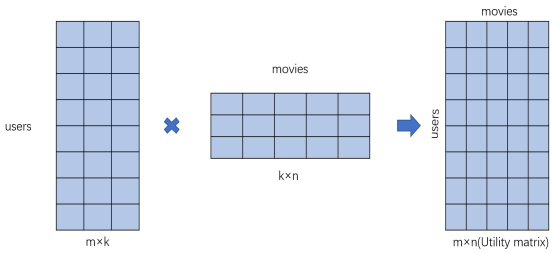

Figure 1: Matrix Factorization

2.2.2 Data Preprocessing

For convenience during coding, this project transformed the the real IDs to new continuous IDs by encoding. In addition, sparse matrix was created to lessen the computation time through avoiding many multiplications between 0.

2.2.3 Loss Function

Under the goal of finding the optimal U and V and going on to do prediction, this project tried to minimized the expectation of the mean squared error between the original data matrix and predicted data matrix. So this project defined the cost function as the equation (3) below. 1/N was added compared with the general cost function so that the average predicted error for each entry can be seen directly.

$$\min_{U \in R^{m \times k}; V \in R^{n \times k}} \frac{1}{N} \sum_{(i,j) \in \Omega} (P_{ij} - u_i^T v_j)^2 \tag{3}$$

where N is the number of non-empty entries in the utility matrix, Ω is the set of indices for observed ratings, and u_i^T and v_j^T are the i_{th} and j_{th} row vectors of the matrices U and V respectively.

2.2.4 Regularization

In this dataset, the number of users is less than the number of movies, which might result in build an overfitting model. To eliminate the probability of overfitting as much as possible. A regularization term was added. So, the new cost function can be written as equation (4) below:

$$\min_{U \in R^{m \times k}; V \in R^{n \times k}} \frac{1}{N} \sum_{(i,j) \in \Omega} (P_{ij} - u_i^T v_j)^2 + \lambda(||U||_F^2 + ||V||_F^2) \tag{4}$$

where the new added λ is the regularization parameter.

2.3 Training

In training, this project initialized two random matrix U and V and set the regularization parameter λ to 0.000002 which contributed to the best performance of the model after tuning . Then train the model to get the optimal solution of U and V.

In the next part, this project implemented such different methods as GD, SGD, Momentum, BB, BFGD, CCD, CCD+ methods in training process and continued to compare the performance of different methods and analyze the reasons of the difference of the performance of each method.

3 Methods & Results

3.1 Gradient Descent

3.1.1 Process

Gradient Descent(GD) is one of the most popular way to solve machine learning problem. This is the most general way to solve a unconstrained problem without any particular assumptions. But the requirement to the capacity of hardware of this method is tremendously high. In each iteration, the gradient is computed considering all the examples in the training set.The equations (5), (6) and (7) below showed how the gradient was computed.

$$\delta = R - UV^T \tag{5}$$

$$w_{user} = -\frac{2}{N} \times (\delta \times V) + 2 \times \lambda \times U \tag{6}$$

$$w_{movie} = -\frac{2}{N} \times (\delta^T \times U) + 2 \times \lambda \times V \tag{7}$$

Above three equations show how the gradient is computed. Each iteration, the whole matrices U and V are implemented into gradients. δ is the error between the real ratings and predicted ratings. N is the number of examples in the training set. λ is the regularization parameter.

The algorithm below is the process of GD method.

Algorithm 1 GD method for MF

Input: Initialize: randomly generate($U \in R^{m \times K}, V \in Y^{n \times K}$), regularization parameter λ , stepsize α and max_iteration

for i in max_iteration **do**:

$w_{user} = -\frac{2}{N} \times (\delta \times V) + 2 \times \lambda \times U$

$w_{movie} = -\frac{2}{N} \times (\delta^T \times U) + 2 \times \lambda \times V$

$U = U - \alpha \times w_{user}$

$V = V - \alpha \times w_{movie}$

end for

This project try different K according to each loss by material research and trials among K = 1, 2, 3, 4, 5, 6.

3.1.2 Results

Figure (2) shows the changes of the value of cost function corresponding to the increase of the number of iterations.

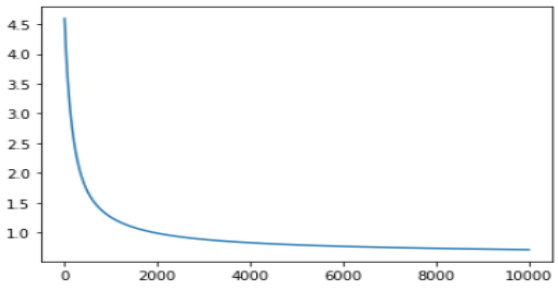

Figure 2: Loss-Iter of GD

By 10,000 iterations, GD method needs 8 minutes to get the result that the value of the loss of training set achieves the lowest loss which is 0.7096907688530887 when K = 5, meanwhile, the value of the loss of testing set achieves 1.613277685682581. The figure (3) below shows the training loss and test loss corresponding to each K.

K	train set loss	test set loss
1	1.323936601	2.632157214
2	0.880291809	1.868536044
3	0.772292982	1.674828081
4	7.3033E+15	1.578795958
5	0.709690769	1.613277686
6	0.696870785	1.557849771

Figure 3: Loss corresponding to different K

It is easy to find that when K = 6, the lowest training loss and test loss can we get. But when K = 6, the computing time is about 9.6 min which is larger than the one when K = 5. But comparing to K = 5, the loss only decrease 0.01, but need relatively more time. This kind of situation also works for K = 7,8,... So, this project can assume that when K = 5, the model has the best performance. In this way, this project also use K = 5 in the following SGD and Momentum method which have no restriction on K.

Figure (4) below shows a 20-example sample randomly exacted containing the real ratings and the predicted ratings.

	userId	movieId	rating	prediction
50	45	50	4.0	2.494754
51	46	51	4.5	4.046038
52	47	52	4.0	4.182935
53	48	53	2.0	2.598049
54	20	54	4.5	3.601484
55	11	55	3.0	3.220449
56	49	56	3.0	2.800421
57	50	57	3.0	2.363324
58	51	58	3.0	4.496389
59	52	59	4.0	3.807040
60	53	60	4.0	3.165757
61	54	61	4.0	4.155712
62	55	62	4.0	2.490038
63	56	14	5.0	2.761534
64	57	63	2.0	3.025502
65	58	12	3.5	3.868080
66	3	64	4.0	2.895188
67	59	65	2.5	3.681963
68	60	66	5.0	3.086789
69	61	67	4.0	2.441844

Figure 4: Sample of real and predicted ratings

In figure (4), most of the predicted ratings are within the ranges of ± 1.5 from the corresponding real ratings. It can be said that the predicted ratings are meaningful for the recommendations.

3.2 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) method is another version of gradient descent method. Comparing to considering all examples in training set per iteration which might result in high cost in each iteration, SGD trains the model and implement gradient descent considering only one example per iteration. In this way, the computation cost in each iteration can be less than before, but it might trigger more iterations to converge. In addition, the accuracy might be not as good as GD and the curve of loss with regard to iterations might not be so smooth like before. The algorithm below is the process of SGD method.

3.2.1 Process

Algorithm 2 SGD method for MF

Input: Initialize: randomly generate($U \in R^{m\times 5}, V \in Y^{n\times 5}$), λ, max_iteration, stepsize α

for k = 1,2,3,...,max_iteration **do**:

Pick $i \in [m]$ and $j \in [n]$ uniformly at random

Update vectors x_i and yi as follows:

$x_i^+ \leftarrow x_i - \alpha \times (x_i^T \times y_i - R_{ij}) \times y_j$

$y_i^+ \leftarrow y_i - \alpha \times (x_i^T \times y_i - R_{ij}) \times x_j$

$x_i \leftarrow x_i^+$

$y_i \leftarrow y_i^+$

end for

3.2.2 Results

Figure (5) shows that the changes of the value of cost function corresponding to the increase of the number of iterations.

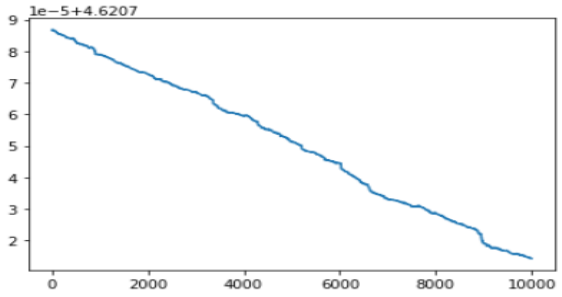

Figure 5: Loss-Iter of SGD

By 10,000 iterations, SGD method needs 4 minutes to get the result that the value of the loss of training set achieves 4.620714217478643 and the value of the loss of testing set achieves 4.785656452175485. For SGD, this project found that the curve is not smooth and the value of loss decreases slowly. It used 10,000 iterations to reach 4.620714217478643. When this project tried to double the max iterations, the curve seems to decrease indeed and converge, but really slowly. So it can be assumed that the cost of each iteration in SGD is low, but the speed of converge is also low. So SGD really need tremendously large iterations to the final optimal solutions. In addition, the loss of both training set and testing set are higher than the ones in GD correspondingly. Figure (5) below shows a 20-example sample randomly exacted containing the real ratings and the predicted ratings.

	userId	movieId	rating	prediction
50	45	50	4.0	0.580033
51	46	51	4.5	1.374709
52	47	52	4.0	2.622560
53	48	53	2.0	2.127750
54	20	54	4.5	0.739548
55	11	55	3.0	2.297206
56	49	56	3.0	1.258105
57	50	57	3.0	1.290341
58	51	58	3.0	3.312845
59	52	59	4.0	2.267828
60	53	60	4.0	1.677466
61	54	61	4.0	1.619668
62	55	62	4.0	1.402626
63	56	14	5.0	1.188032
64	57	63	2.0	2.275963
65	58	12	3.5	1.212493
66	3	64	4.0	1.573764
67	59	65	2.5	2.617584
68	60	66	5.0	2.378034
69	61	67	4.0	2.142566

Figure 6: Sample of real and predicted ratings

In figure (6), it can be found that the accuracy of predicted ratings is low. Most of the predicted ratings are within the ranges of ± 2 from the corresponding real ratings. So the accuracy is not satisfying and the results are not as meaningful as before. This project will discuss the reasons behind it later in following parts.

3.3 Momentum

Considering GD method is going too fast on the new direction, researchers started to take the information of the old direction into account many years ago. The reason behind it is not far to seek that GD method spend a lot of time on the direction not aiming at the global optimum, whereas Momentum method focus more on the direction aiming at the global optimum. Figure (7) below shows the conceptual fact behind it.

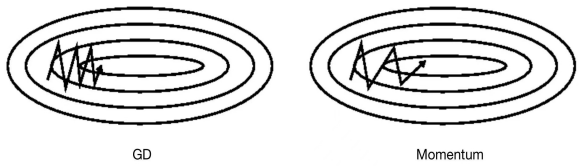

Figure 7: Momentum

3.3.1 Process

In Momentum method, it can be considered as changing speed by gradient. The equation (8) and (9) below shows how Momentum method works. Difference of x is velocity and difference of v is acceleration.

$$V^k = \beta \times V^{k-1} - \alpha \times \bigtriangledown f(x) \tag{8}$$

$$x^{k+1} = x^k - v^k \tag{9}$$

The algorithm below is the process of Momentum method.

Algorithm 3 Momentum method for MF

Input: Initialize: randomly generate($U \in R^{m\times 5}, V \in Y^{n\times 5}$), λ, max_iteration, Momentum parameters α and β.
Calculate the w_user and w_movie of the first iteration
let v_user and v_movie = w_user and w_movie
for k = 1,2,3,...,max_iteration **do**:
    Calculate the w_user and w_movie of each iteration
    $v_{user} = \beta \times v_{user} + (1 - \beta) \times w_{user}$
    $v_{movie} = \beta \times v_{movie} + (1 - \beta) \times w_{movie}$
    $U = U - \alpha \times w_{user}$
    $V = V - \alpha \times w_{movie}$
end for

3.3.2 Results

Figure (8) shows that the changes of the value of cost function corresponding to the increase of the number of iteration.

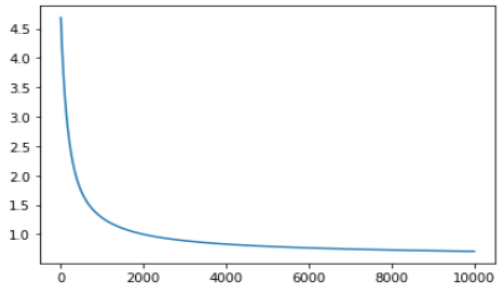

Figure 8: Loss-Iter of Momentum

By 10,000 iterations, Momentum method needs 9 minutes to get the result that the value of the loss of training set achieves the lowest loss which is 0.706413800829807 when K = 5, meanwhile, the value of the loss of testing set achieves 1.5624844307772927. Comparing to GD, Momentum method needs one more minutes to computing the results, but less losses in both training set loss and test set loss. In the 4000th iteration, the training set loss is slightly smaller the the one in GD method. So, it can be concluded that the convergence speed in Momentum method is slightly faster than the one in GD on this dataset. Figure (9) below shows a 20-example sample randomly exacted containing the real ratings and the predicted ratings.

	userId	movieId	rating	prediction
50	42	50	5.0	3.663856
51	43	51	4.0	4.695410
52	44	7	3.0	2.797099
53	45	52	2.0	2.905921
54	46	45	4.0	2.916859
55	47	53	0.5	4.095338
56	48	54	2.0	3.805382
57	49	55	4.0	4.079585
58	50	56	0.5	2.061450
59	51	57	3.0	4.483839
60	11	58	2.0	3.646179
61	52	59	3.5	4.191033
62	53	60	3.5	3.538337
63	54	61	4.0	3.896161
64	55	62	3.0	2.622600
65	39	63	3.0	4.053023
66	56	64	2.0	4.049407
67	57	65	4.0	2.421460
68	58	66	4.5	3.509622
69	59	67	4.0	2.834966

Figure 9: Sample of real and predicted ratings

In figure (9), most of the predicted ratings are within the ranges of ± 1.5 from the corresponding real ratings. It can be said that the predicted ratings are meaningful for the recommendations.

3.4 BB method

The Barzilai-Borwein(BB) method is a popular and efficient tool for solving large-scale unconstrained optimization problems. Its search direction is the same as gradient descent, but its stepsize rule is motivated by Newton's method but not involves any Hessian. The method often significantly improves the performance of a standard gradient method. In BB method, the stepsize of gradient-type iterative methods is replaced by the matrix $H_k = \alpha_k I$, this matrix is served as an approximation of the inverse Hessian matrix, following the quasi-Newton approach, the stepsize is calculated by forcing either H_k^{-1}(LBB method) or H_k(SBB method) to satisfy the secant equation in the least square sense.

3.4.1 Process

Algorithm 4 BB method for MF

Input: Initial A, randomly generate($U_0 \in R^{m\times 1}, Y_0 \in Y^{n\times 1}$), λ

The k-th iteration:

$U_k \leftarrow U_{k-1} - \alpha_U^{BB} \nabla f(U_{k-1})$

$V_k \leftarrow V_{k-1} - \alpha_k^{BB} \nabla f(V_{k-1})$

where the stepsize α_U^{BB} is chosen according to SBB:

$\alpha_U^{BB} = \alpha_U^{SBB} = \frac{y_u k-1^T s_u k-1}{y_u k-1^T s_u k-1}$

$\alpha_V^{BB} = \alpha_V^{SBB} = \frac{y_v k-1^T s_v k-1}{y_v k-1^T s_v k-1}$

Here $s_u k = U_k - U_{k-1}, y_u k = \nabla U_{k-1} - \nabla U_k$

$s_u k = U_k - U_{k-1}, y_u k = \nabla U_{k-1} - \nabla U_k$

3.4.2 Results

If BB runs too many times, the predictive ratings will appear lots of negative values, in our view it is a big drawback for BB in solving NMF recommendation problem. Besides, LBB stepsize has a bad performance, it will cause the loss increase, we need to use gradient descent first to run BB method with LBB stepsize. But SBB stepsize preforms normally.

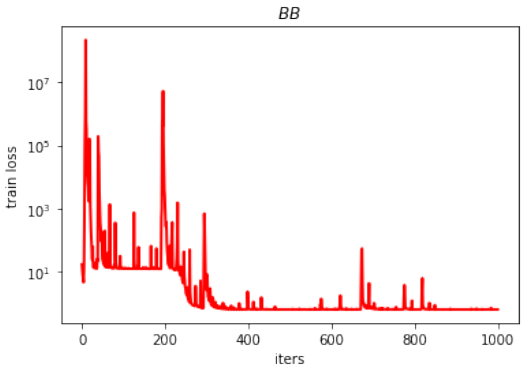

Figure 10: Train Loss-Iter of BB

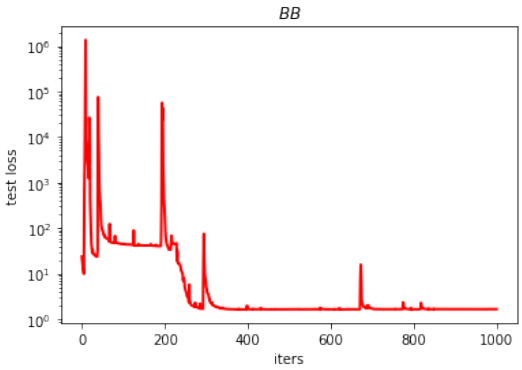

Figure 11: Test Loss-Iter of BB

Although BB may cause negative predictive ratings, overall, BB is still a quite good tool in our MF for recommendation system problem, by 1000 iterations, BB method needs 28s to get the result that the value of the loss of training set achieves 0.6477360575774878 and the value of the loss of testing set achieves 1.663580077620589. It converges faster than GD and also has a shorter iteration time, last but not least, BB don't need to tune the stepsize. Figure (12) below shows a 20-example sample randomly exacted containing the real ratings and the predicted ratings.

	userId	movieId	rating	prediction
20	20	20	3.0	2.788585
21	21	21	4.0	3.810079
22	22	22	4.0	3.386450
23	23	23	4.0	3.548053
24	24	24	5.0	3.918966
25	25	25	5.0	3.770060
26	26	26	4.0	3.961072
27	27	27	4.0	3.757345
28	28	28	1.5	3.460039
29	29	29	4.0	3.455279
30	30	30	3.0	2.376203
31	31	31	3.5	3.669353
32	32	32	5.0	2.984772
33	33	33	4.0	3.600299
34	34	34	4.0	3.691466
35	35	35	1.0	3.720505
36	36	36	2.5	3.459556
37	37	37	4.0	3.236753
38	38	38	4.0	5.284153
39	39	39	2.0	4.115156

Figure 12: Sample of real and predicted ratings

3.5 BFGS method

The BFGS (Broyden-Fletcher-Goldfarb-Shanno) algorithm is an iterative method for solving unconstrained nonlinear optimization problems. It does so by gradually improving an approximation to the Hessian matrix of the loss function, obtained only from gradient evaluations (or approximate gradient evaluations) via a generalized secant method. BFGS with proper stepsize converges to optima for strongly convex problems, but we know the objective function of NMF are jointly non-convex with respect to both factor matrices. Fortunately, NMF is convex with respect to each factor matrix, i.e., the sub-problems for updating individual factor matrix are convex, and thus it can be solved by recursively updating both factor matrices in the framework of ALS. We set d_0 as 1.

3.5.1 Process

Algorithm 5 BFGS method for MF

Input: Initial A, randomly generate($U_0 \in R^{m\times 1}, V_0 \in R^{n\times 1}$), λ, $H_u = I, H_v = I$
  for outeriter = 1,2,... **do**
    for inneriter = 1,2,... **do**
      $U_{k+2} \leftarrow U_{k+1} - \alpha_u^{(k+1)} H_u^{k+1} \nabla f(U_{k+1})$
      H is update as:
        $H_u^{k+1} = (I - \rho_u^k s_u^k y_u^{kT}) H_u^{k+1} (I - \rho_u^k y_u^k y_s^{kT}) + \rho_u^k s_u^k s_u^{kT}$
      Here $\rho_u^k = 1/(s_u^k)^T y_u^k$
      $\alpha_{u(k+1)}$ is determined by Wolfe rule($c_1 = 10^{-4}, c_2 = 0.9$).
    end for
    for inneriter = 1,2,... **do**
      $V_{k+2} \leftarrow V_{k+1} - \alpha_v^{(k+1)} H_v^{k+1} \nabla f(U_{k+1})$
      H is update as:
        $H_v^{k+1} = (I - \rho_v^k s_v^k (y_v^k)^T) H_v^{k+1} (I - \rho_v^k y_v^k (y_s^k)^T) + \rho_v^k s_v^k (s_v^k)^T$
      Here $\rho_v^k = 1/(s_v^k)^T y_v^k$
      $\alpha_{v(k+1)}$ is determined by Wolfe rule($c_1 = 10^{-4}, c_2 = 0.9$).
    end for
  end for

Wolfe rule is formed by two conditions:

$$f(x_t + \alpha_t d_t) \leq (x) + c_1 \alpha_t d_t^T \nabla f(x_t) \tag{10}$$

$$\nabla f(x_t + \alpha_t d_t)^T d_t \geq c_2 \nabla f(x_t)^T d_t \tag{11}$$

3.5.2 Results

We first tried set inneriter as 5, then train loss normally converged in 3 iterations, but it would continue to drop after switching to other factor matrix. So the better strategy should be set a small innneriter and a suitable and large enough outeriter value.

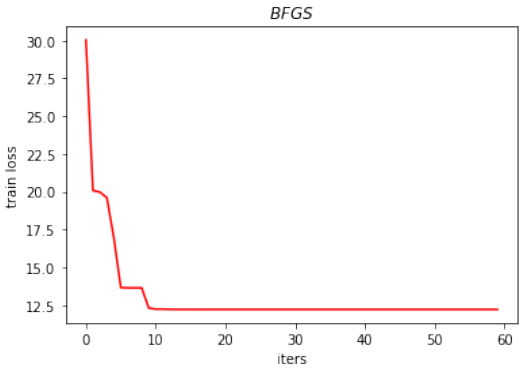

Figure 13: Train Loss-Iter of BFGS

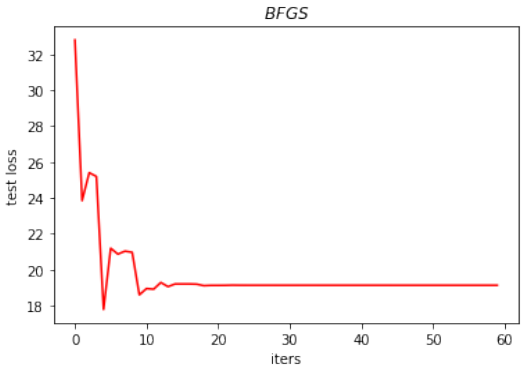

Figure 14: Test Loss-Iter of BFGS

By $10 \times (3 \times 2)$,i.e.,60 iterations, BFGS method needs 32 minutes to get the result that the value of the loss of training set achieves 12.209201178667028 and the value of the loss of testing set achieves 19.12554699667965.

BFGS converged in a relatively high loss value comparing to other algorithms, but we find it can be combined with GD to solve this MF problem. For our rank-1 MF, after running 500 iterations of GD, the train loss function would converge, then we swithched to use BFGS method, figure (15) shows the downtrend of loss function. This means that BFGS may have higher predictive potential only from the perspective of the final forecast results, however, we should notice that BFGS has much more per-iteration time. Therefore, only using BFGS to solve the MF problem is not recommended in terms of efficiency, combination of BFGS and GD might be a feasible solution.

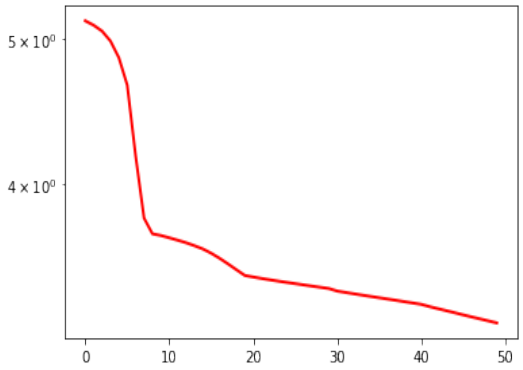

Figure 15: Train Loss-Iter of BFGS+GD

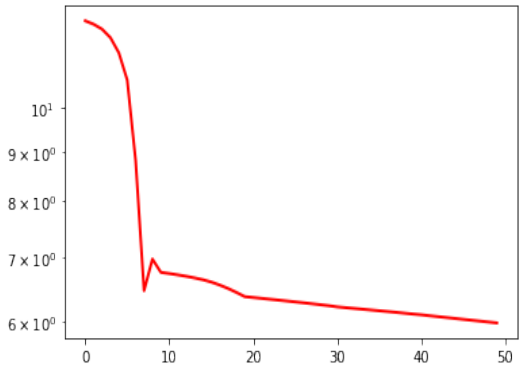

Figure 16: Test Loss-Iter of BFGS+GD

3.6 CD

CD(Coordinate descent), a classical optimization approach, has been used for many other large-scale problems, but its application to matrix factorization for recommender systems has not been explored thoroughly. At each iteration, CD algorithm determines a coordinate or coordinate block via a coordinate selection rule, then exactly or inexactly minimizes over the corresponding coordinate hyperplane while fixing all other coordinates or coordinate blocks. In his part, we tried to realized CD for MF in the slide to see its performance in MF.

3.6.1 Process

Algorithm 6 CD Algorithm

Input: Initial A, randomly generate($U_0 \in R^{m\times 1}, V_0 \in V^{n\times 1}$), λ, and k

for iter = 1,2,... **do**

for k = 1,2,...,m **do**

$$u_k \leftarrow \frac{\sum_{j=1}^{n} A_{kj} v_j}{\sum_{j=1}^{n} v_j^2}$$

end for

for t = 1,2,...,n **do**

$$v_t \leftarrow \frac{\sum_{i=1}^{m} A_{it} u_i}{\sum_{t=1}^{n} u_i^2}$$

end for

end for

3.6.2 Results

By 50 iterations, CD method needs 65 minutes to get the result that the value of the loss of training set achieves 10.212501168687028 and the value of the loss of testing set achieves 12.37389028667965.

Objective function f, i.e., Equation 2, is non-smooth, it poses a significant challenge to CD, our coordinate descent iteration get stuck at a non-stationary point which still has a high loss (10.189212386679253), as shown in Figure (17). In addition, per-iteration time of CD is almost twice of BFGS's, which are 78s and 32.83s. We speculate that the reason is, BFGS use gradient to update the whole factor matrix, but CD has to update u_k and v_t sequentially. Maybe if we want to use CD in MF for recommendation problem, it is necessary to update u_k and v_t simultaneously (Jacobi method) and use distributed computing.

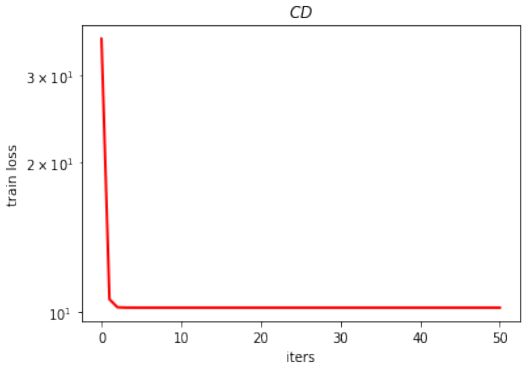

Figure 17: Train Loss-Iter of CD

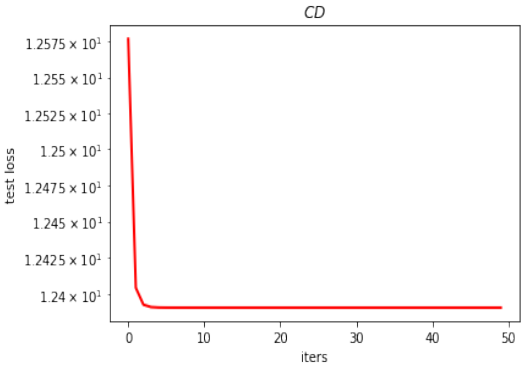

Figure 18: Test Loss-Iter of CD

3.7 CCD

CCD (cyclic coordinate descent) is a classic optimization method that has witnessed a resurgence of interest in signal processing, statistics, and machine learning. Reasons for this renewed interest include the simplicity, speed, and stability of the method, as well as its competitive performance on regularized smooth optimization problems. It is similar to ALS with respect to the update sequence. The only difference is the update rules, u_i is updated over all elements of u_i (i.e., u_1, . . . , u_k) with a finite

number of cycles. The entire update sequence of one iteration in CCD is

$$\overbrace{\underbrace{u_{11},...,u_{1k}}_{u_1},...,\underbrace{u_{m1},...,u_{mk}}_{u_m}}^{U},\overbrace{\underbrace{v_{11},...,v_{1k}}_{v_1},...,\underbrace{v_{n1},...,v_{nk}}_{v_n}}^{V} \tag{12}$$

3.7.1 Process

Algorithm 7 CCD Algorithm

Input: Initial R=A, randomly generate($U_0 \in R^{m\times k}, V_0 \in V^{n\times k}$), λ, and k
for iter = 1,2,... **do**
  for i = 1,2,...,m **do**
    for t = 1,2,...,k **do**
      $z^\star \leftarrow \frac{\sum_{j\in\Omega_i}(R_{ij}+u_{it}v_{jt})v_{jt}}{\lambda+\sum_{j\in\Omega_i}v_{jt}^2}$
      $R_{ij} \leftarrow R_{ij} - (z^\star - u_{it})v_{jt}, \forall j \in \Omega_j$
      $u_{it} \leftarrow z^\star$
    end for
  end for
  for j = 1,2,...,n **do**
    for t = 1,2,...,k **do**
      $s^\star \leftarrow \frac{\sum_{i\in\Omega_j}(R_{ij}+u_{it}v_{jt})u_{it}}{\lambda+\sum_{i\in\Omega_j}u_{it}^2}$
      $R_{ij} \leftarrow R_{ij} - (s^\star - v_{jt})u_{it}, \forall i \in \Omega_j$
      $v_{jt} \leftarrow s^\star$
    end for
  end for
end for

3.7.2 Results

By 30 iterations, CCD method needs 29 minutes to get the result that the value of the loss of training set achieves 10.680287823305443 and the value of the loss of testing set achieves 13.822838274363358.

We set k as 1,2,5, and when k=2, CCD has the best convergence effect, the train loss function declined from 11 to 2.102.In our experiment, the loss function usually only have a small drop in the first three iterations in CCD method, and the valid loss function instead shows an upward trend. Besides, The increase in the k-value will linearly increase per-iteration time. 1 minutes at k=1, 1.5 minutes at k=2 and 5 minutes at k=5.

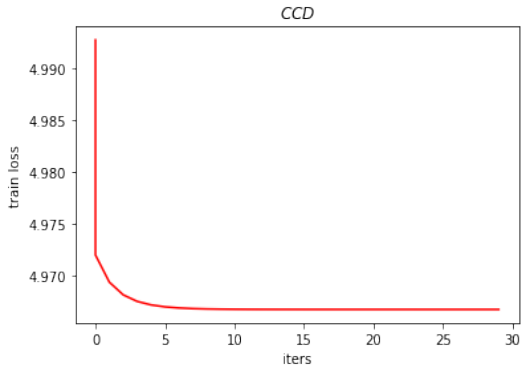

Figure 19: Train Loss-Iter of CCD

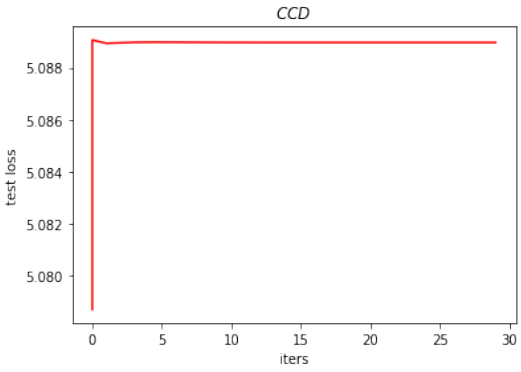

Figure 20: Test Loss-Iter of CCD

3.8 CCD++

CCD++ was developed as an reactive algorithm for parallel implementation on multicore processors.The only difference between CCD++ and CCD is their update sequences. However, such difference might affect the convergence. The feature-wise update sequence leads to faster convergence than other sequences on moderate-scale matrices.

3.8.1 Process

Algorithm 8 CCD++ Algorithm

Input: Initial R=A, randomly generate($U_0 \in R^{m\times k}, V_0 \in V^{n\times k}$), λ, T, and k
  for iter = 1,2,... **do**
    for t = 1,2,...,k **do**
      Get $(u^\star, v^\star)$ using T CCD iterations
      $R_{ij} \leftarrow \hat{R}_{ij} - u_i^\star v_j^\star, \qquad \forall (i,j) \in \Omega$
      $(\tilde{u}_t, \tilde{v}_t) \leftarrow (u^\star, v^\star)$
    end for
  end for

3.8.2 Results

By 100 iterations, CCD++ method needs 105 minutes to get the result that the value of the loss of training set achieves 11.680287823305443 and the value of the loss of testing set achieves 13.822838274363358.

We firstly initialed U to 0 as the description of [8], however, it only increases the time required for convergence, but does not affect the final result, then we used randomly generated U_0, V_0. CCD++ could be slightly more efficient when $T > 1$ due to the benefit brought by the "delayed residual update." To verify it, we set $T = 1, 5$, it represents the iteration time of CCD we use, the more T we use, the better the approximation to subproblem. On the other hand, a large and fixed T might result in too much effort on a single subproblem.In our experiment, $T = 5$ has a better converge speed.

In multiple experiments, whether T is 1 or 5, the loss function value has significant drops in the first two update of $(\tilde{u}_t, \tilde{v}_t)$, while the function also has periodic fluctuations at the end. This fluctuation should be caused by a code bug, because we found that the predicted scores are almost the same, which is obviously wrong. For lack of python codes about the CCD algorithm online, as well as time limitation, we cannot continue to apply this method.

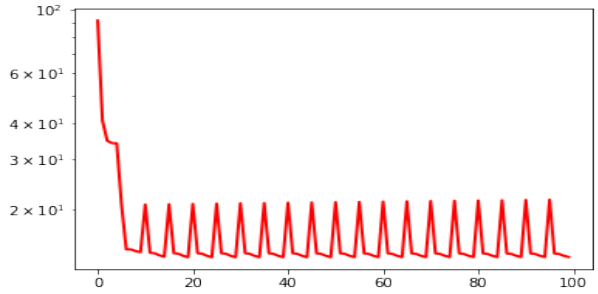

Figure 21: Loss-Iter of CCD++

4 Comparison

All the content in the Methods part has illustrated the performance of each method. Now, in this part, this project hands on the comparison of the performance of different methods. Figure (22) shows the performance of different methods.

Method	Training Set Loss	Test Set Loss	Running Time
GD	0.709690769	1.613277686	8 min
SGD	4.620714217	4.785656452	4 min
Momentum	0.706413801	1.562484431	9 min
BB	0.647736058	1.663580078	28 s
BFGS	12.20920118	19.125547	32 min
CD	10.21250117	12.37389029	65 min
CCD	10.68028782	13.82283827	29 min
CCD+	11.68028782	13.82283827	105 min

Figure 22: Comparison of the performance of different methods

From figure (22), we can see that different methods have different performance in losses and running time. For this recommendation system, the most important thing is the accuracy of the predicted ratings. This recommendation system needs the accuracy, in other words, the training set loss and test set loss to be as small as possible.

For BFGS, CD, CCD and CCD+ methods, the training set loss and test set loss are so large that the predicted ratings is far away from the real ratings. Moreover, their running times are also much larger than the other methods. So, these four methods are not meaningful for this recommendation system based on this dataset.

For GD and Momentum methods, the training set loss and test set loss is relatively low among all the methods, which can enable the predicted ratings to be within the range of ± 1.5 from the real ratings. But, their running times are both above 8 min which are larger than SGD and BB methods. So, these two methods are also not really meaningful indeed and cannot be utilized based on this dataset.

For SGD method, its training set loss and test set loss are also relatively higher than the GD, Momentum and BB methods, which turn out to be that the predicted ratings are slightly far away from the real ratings. In addition, the running time of SGD method is also larger than BB method. So, this method cannot be utilized in this recommendation system based on this dataset.

For BB method, its training set loss is the smallest among all the methods. Although its test set loss is a little bit larger than GD and Momentum, its running time is tremendously smaller than the other methods. BB method can achieve similar losses with other methods with losses in the same

level. Besides, the running time of BB method is the smallest among all the methods and greatly smaller than the ones in other methods. It can help this recommendation system to be much more efficient with nearly the same accuracy with the highest accuracy of all methods. So, BB method has the best performance among all the methods. It is the most meaningful method can be utilized in this recommendation system based on this dataset.

5 Conclusion

The results are mostly surprising. In our self-built recommendation system, BB method has not only the nearly lowest loss but also minimal computation time, and its actual predictive results are reasonable. As a rare method in MF problem, BB beats the baseline algorithms and some state-of-the-art algorithms. While there are some limitations in our project, not using GPU results in the number of tuning for the hyperparameters and the number of iterations of the algorithms are still not enough. Besides, some of codes need to be optimized to improve computation speed and prediction accuracy. For further research, it would be of relevance to see how the algorithms would behave if we would have used larger dataset, and moreover, the successful combination of BFGS and GD inspires us to try more blending between algorithms.

References

[1] Funk, Simon. "Monday, December 11, 2006." Netflix Update: Try This at Home, 2006, http://sifter.org/ simon/journal/20061211.html.

[2] Bell, R. M., Koren, Y. (2007, October). Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In Seventh IEEE international conference on data mining (ICDM 2007) (pp. 43-52). IEEE.

[3] Sun, R. (2015). Matrix completion via nonconvex factorization: Algorithms and theory (Doctoral dissertation, University of Minnesota).

[4] Yang, X., Fang, J., Chen, J., Wu, C., Tang, T., Lu, K. (2017, May). High performance coordinate descent matrix factorization for recommender systems. In Proceedings of the Computing Frontiers Conference (pp. 117-126).

[5] Huang, Y., Liu, H., Zhou, S. (2015). Quadratic regularization projected Barzilai–Borwein method for nonnegative matrix factorization. Data mining and knowledge discovery, 29(6), 1665-1684.

[6] Kim, D., Sra, S., Dhillon, I. S. (2007, April). Fast Newton-type methods for the least squares nonnegative matrix approximation problem. In Proceedings of the 2007 SIAM international conference on data mining (pp. 343-354). Society for Industrial and Applied Mathematics.

[7] Guan, N., Tao, D., Luo, Z., Yuan, B. (2012). NeNMF: An optimal gradient method for nonnegative matrix factorization. IEEE Transactions on Signal Processing, 60(6), 2882-2898.

[8] Yu, H. F., Hsieh, C. J., Si, S., Dhillon, I. (2012, December). Scalable coordinate descent approaches to parallel matrix factorization for recommender systems. In 2012 IEEE 12th international conference on data mining (pp. 765-774). IEEE.