RECOMMENDATION SYSTEMS BASED ON MATRIX COMPLETION

JOY SONG[1]

1. INTRODUCTION

Nowadays recommendation systems are everywhere in various social media platforms. In general, a recommendation system utilizes user data to recommend items that the user may have an interest in. For example, a movie platform like Netflix will display videos that are related to what you have watched recently, a music app like Spotify will recommend new songs everyday based on your previous preferences, these are contributions of recommendation systems. Numerous methods have been developed for recommendation systems and they mainly fall into three categories: collaborative filtering, content-based filtering, and hybrid approaches. Collaborative filtering is a method assuming that similar users will have similar interests and thus recommend items that have been liked by similar users, content-based filtering explores items with similar content and recommends users similar items to what they have liked.

This project is focused on collaborative filtering approaches, most of which could be modeled as a matrix completion problem where the matrix is a user-item matrix composed of each user's rating for each item. *A Survey of Matrix Completion Methods for Recommendation Systems* (A. Ramlatchan, M. Yang, Q. Liu, M. Li, J. Wang and Y. Li) [3] gives a comprehensive summary to various matrix completion methods used in recommendation systems, this project is going to focus on the matrix factorization models and rank minimization models which are most related to what we have learned in MATH 510. The matrix factorization models are based on the idea of Singular Value Decomposition, which decompose the sparse data matrix into a user-feature matrix and an item-feature matrix which capture the latent relationship between users and items, then uses the dot product of these latent vectors to obtain the prediction at a certain entry. The rank minimization models are based on the assumption that our data matrix is of low rank since users of similar interests will have similar ratings, the algorithm aims to minimize the rank of the desired complete matrix under the restriction that it coincides with the original matrix.

This project first codes matrix completion solvers based on the matrix factorization models and rank minimization models mentioned above, then implements these solvers on artificial data several times to choose the best hyperparameters, finally it verifies the assumption that when there are more observed entries in a data matrix, which means when the original data matrix is less sparse, these matrix completion solvers will have better performances.

[1]BRIGHAM YOUNG UNIVERSITY
E-mail address: jsong@mathematics.byu.edu.
Date: August 21, 2020.

2. Matrix Completion

Now our problem of interest is, given a large and sparse data matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, how can we recover it and get a complete matrix? This section will explain the three main methods my project investigates.

2.1. **Matrix Factorization.**

Given a data matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ where the rows of $\boldsymbol{A}$ correspond to m users and the columns of $\boldsymbol{A}$ correspond to n items, the matrix factorization methods aim to find two matrices $\mathbf{U} = [\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_k] \in \mathbb{R}^{m\times k}, \mathbf{V} = [\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k] \in \mathbb{R}^{n\times k}$ such that $\boldsymbol{A} = \mathbf{U}\mathbf{V}^T$. Here $\mathbf{U}$ is regarded as a user-feature matrix composed of user-latent vectors $\boldsymbol{u}_i \in \mathbb{R}^m, 1 \leqslant i \leqslant k$ and $\mathbf{V}$ is regarded as an item-feature matrix composed of item-latent vectors $\boldsymbol{v}_j \in \mathbb{R}^m, 1 \leqslant j \leqslant k$, so that the (i, j) entry of the data matrix $\boldsymbol{A}$ is regarded as the i$^{\text{th}}$ user's score for the j$^{\text{th}}$ item, which satisfies $\boldsymbol{A}_{ij} = \boldsymbol{u}_i \cdot \boldsymbol{v}_j$. This means the dot product between the i$^{\text{th}}$ user-latent vector $\boldsymbol{u}_i$ and the j$^{\text{th}}$ item-latent vector $\boldsymbol{v}_j$ will give us the estimation of the score $\boldsymbol{A}_{ij}$.

2.1.1. *Direct SVD.*

A straightforward idea to achieve the decomposition $\boldsymbol{A} = \mathbf{U}\mathbf{V}^T$ is to directly use singular value decomposition [4]. We first fill in the missing entries of the data matrix $\boldsymbol{A}$ based on some prior knowledge, here we use the average score for each item (average over rows for each column) to fill in the missing entries and obtain a complete matrix $\tilde{\boldsymbol{A}}$; then we apply singular value decomposition $\tilde{\boldsymbol{A}} = \tilde{\mathbf{U}}\Sigma\tilde{\mathbf{V}}^T$; since larger singular values are thought to correspond to more important latent features, we truncate the matrices $\tilde{\mathbf{U}}, \tilde{\mathbf{V}}$ to be $\hat{\mathbf{U}}, \hat{\mathbf{V}}$, which are composed of the first k columns in $\tilde{\mathbf{U}}, \tilde{\mathbf{V}}$ respectively. Now we could define the user-feature matrix $\mathbf{U} = \hat{\mathbf{U}}\Sigma^{1/2} \in \mathbb{R}^{m\times k}$, and the item-feature matrix $\mathbf{V} = \hat{\mathbf{V}}\Sigma^{1/2} \in \mathbb{R}^{n\times k}$, which will give us the recovered complete matrix

$$\hat{\boldsymbol{A}} = \mathbf{U}\mathbf{V}^T = \hat{\mathbf{U}}\Sigma^{1/2}(\hat{\mathbf{V}}\Sigma^{1/2})^T = \hat{\mathbf{U}}\Sigma\hat{\mathbf{V}}^T = \tilde{\mathbf{U}}_{1:k}\Sigma\tilde{\mathbf{V}}_{1:k}^T$$

2.1.2. *Optimization Model.*

Although singular value decomposition is straightforward, it requires the latent vectors to be orthogonal, which is not necessary. A more general and probably more accurate approach is to construct the matrix factorization model as an optimization problem that minimizes the difference between $\boldsymbol{A}$ and $\mathbf{U}\mathbf{V}^T$. Let Ω denote the set of known entries in $\boldsymbol{A}$, then the loss function is formed as

$$E = \frac{1}{2}\sum_{\substack{i=1\\(i,j)\in\Omega}}^{m}\sum_{j=1}^{n}(A_{ij} - \mathbf{u}_i\mathbf{v}_j^T)^2 = \frac{1}{2}\sum_{\substack{i=1\\(i,j)\in\Omega}}^{m}\sum_{j=1}^{n}(A_{ij} - \sum_{r=1}^{k}\mathbf{u}_{ir}\mathbf{v}_{jr})^2 = \frac{1}{2}\sum_{\substack{i=1\\(i,j)\in\Omega}}^{m}\sum_{j=1}^{n}e_{ij}^2.$$

Minimizing E could be achieved by gradient descent method where the gradients:

$$\frac{\partial E}{\partial U_{pq}} = -\sum_{\substack{j=1\\(p,j)\in\Omega}}^{n} e_{pj}v_{jq}, \qquad \frac{\partial E}{\partial V_{pq}} = -\sum_{\substack{j=1\\(i,p)\in\Omega}}^{n} e_{ip}u_{iq}.$$

With the appropriate step size (learning rate) α, we simultaneously update the two matrices in each iteration:

$$\mathbf{U} \leftarrow \mathbf{U} - \alpha\nabla_u, \qquad \mathbf{V} \leftarrow \mathbf{V} - \alpha\nabla_v.$$

2.2. **Rank Minimization.** It is a widely used assumption that the data matrix is of low rank since we could expect similar users to have similar preferences. The original construction of the rank minimization model is to minimize the rank of the data matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, which is computationally infeasible, so a classical approach is to use the nuclear norm $\|\boldsymbol{A}\|_*$ as an approximation to the rank of the matrix and thus turn this infeasible problem into a convex optimization problem which is feasible. And it has been proved that we are able to achieve perfect recovery of low rank matrices if enough entries are observed [2].

The nuclear norm of a matrix $\mathbf{X}$ is defined as

$$\|\mathbf{X}\|_* = \sum_{i=1}^{r} \sigma_i(\mathbf{X})$$

where $\sigma_i(\mathbf{X})$ are the singular values of $\mathbf{X}$. Now the rank minimization method could be constructed as the following nulcear norm minimization model

$$\begin{aligned} &\min \quad \operatorname{rank}(\mathbf{X}) \quad \text{s.t.} \quad \mathbf{X}_{ij} = \mathbf{A}_{ij}, \quad \forall (i,j) \in \Omega \\ &\min \quad \|\mathbf{X}\|_* \qquad \text{s.t.} \quad \mathscr{P}_\Omega(\mathbf{X}) = \mathscr{P}_\Omega(\mathbf{A}) \end{aligned}$$

where Ω is the set of known entries in the data matrix $\boldsymbol{A}$ and $\mathscr{P}_\Omega$ is an operator that extracts the entries in the set Ω. This is a convex minimization problem and could be solved using standard methods [5].

3. Experiments

The outline of the numerical experiments in this project is to first code matrix completion solvers based on the three different models mentioned in section 2: direct singular value decomposition, matrix factorization and nuclear norm minimization; then implement them on one set of artificial data several times to choose the optimal hyperparameters involved in the algorithms; finally explore different factors that might influence matrix completion error, as well as compare the accuracy and efficiency of different algorithms.

In order to measure the performances of our algorithms, every time we first randomly generate a 50×10 matrix $\boldsymbol{M}$ as the ground-truth complete matrix, and the matrix is also of a certain fixed rank; then we set a probability p of each entry in $\boldsymbol{M}$ to be observed and sample on $\boldsymbol{M}$ to obtain an artificial incomplete data matrix $\boldsymbol{A}$. Now we could apply different matrix completion solvers to the incomplete data matrix $\boldsymbol{A}$ and obtain the recovered complete matrix $\mathbf{X}$. By this mean, we are able to measure the performances of these algorithms by computing the errors between the complete matrix $\boldsymbol{M}$ and recovered matrix $\mathbf{X}$.

Choosing the hyperparameters for these algorithms is subtle but very important. In the nuclear norm minimization algorithm, we fix the over-relaxation parameter in the augmented Lagrangian function to be 0.1 and everything works out fine, while the parameters in the matrix factorization algorithms turn out to be much more sensitive to the input matrices. Both direct SVD model and matrix factorization model need us to choose an appropriate k which is the length of the latent vectors where $\mathbf{U} \in \mathbb{R}^{m\times k}, \mathbf{V} \in \mathbb{R}^{n\times k}$. If k is too small, the error is expected to be very large since only a few features have been captured in $\mathbf{U}$ and $\mathbf{V}$, then $\mathbf{U}\mathbf{V}^T$ can not represent the whole complete matrix; if k is too large (up to $\min\{m,n\}$), $\mathbf{U},\mathbf{V}$ might contain some unnecessary information since in real life the data matrix is expected to be sparse, and it will unnecessarily slow down our algorithm. In most of the experiments, for data matrices of the

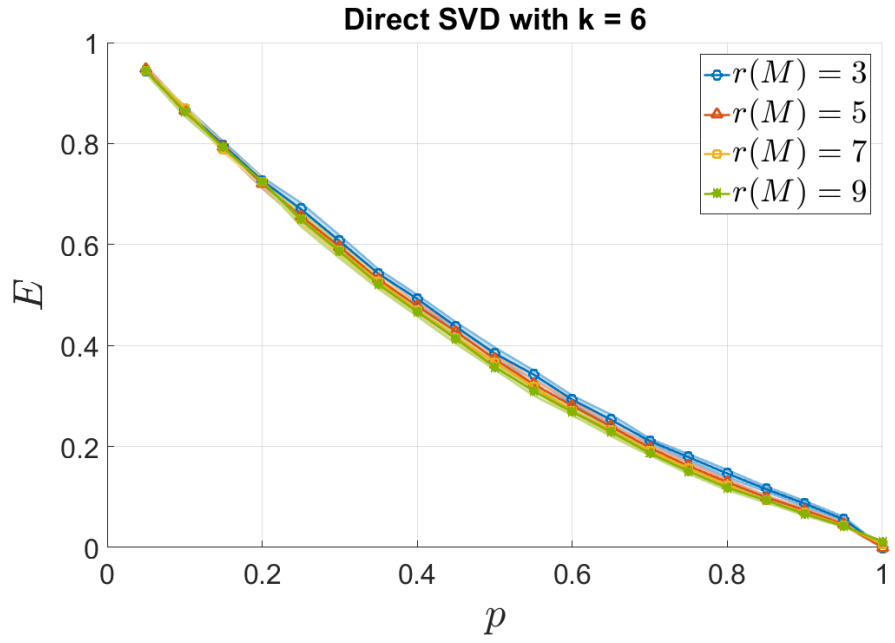

Figure 1. Error vs. Sparsity & Rank for direct singular value decomposition model, with $k = 6$

size 50×10, we choose $k = 8$, we also choose k based on the rank of the matrices in some situations. The last hyperparameter we need to care about is the step size (learning rate) α of the gradient descent method in the matrix factorization algorithm. For convenience, in our experiments we use constant step size instead of adaptive step size. If α is too small, it would take too long for the algorithm to find the local minimum; if α is too large, it is possible for the algorithm to bypass the local minimum every time and never find it. In most of the experiments, we use $\alpha = 0.0005$ and there are also circumstances when we need to adjust the value of α.

3.1. **Error vs. Sparsity & Rank.** For each of the three matrix completion models respectively, we mainly consider two factors that might influence the matrix completion error: one is the sampling probabilities which determine the sparsity of the data matrices $\boldsymbol{A}$, the other is the rank of the original complete matrices $\boldsymbol{M}$.

3.1.1. *Direct Singular Value Decomposition.* As shown in Figure 1 and Figure 2, as the sampling probabilities approach one, the errors produced by direct SVD are decreasing to zero, and the rank of the complete matrices $\boldsymbol{M}$ seem to have little impact on the errors. However, in Figure 1 when $k = 6$, we notice that matrices of rank seven and nine produce larger errors near $p = 1$; while in Figure 2 when $k = 9$, matrices of all ranks produce nearly zero errors. This could be explained by the construction of this algorithm, where we only choose the first k left singular vectors and the first k right singular vectors to form the user-feature matrix and item-feature matrix. When k is larger than the rank of the complete matrix $\boldsymbol{M}$, we are essentially doing a reduced singular value decomposition which is expected to achieve exact recovery on the set of known entries; when k is smaller than the rank of the complete matrix $\boldsymbol{M}$, it is possible for the algorithm to miss some important latent features and thus result in larger error.

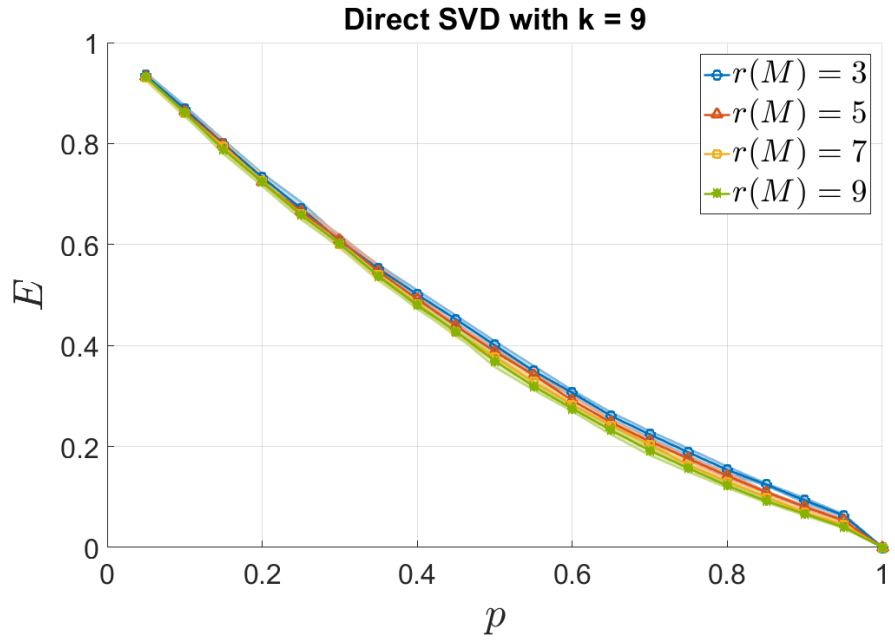

Figure 2. Error vs. Sparsity & Rank for direct singular value decomposition model, with $k = 9$

3.1.2. *Matrix Factorization.* As shown in Figure 3 and Figure 4, in general, as the sampling probabilities approach one, the errors produced by matrix factorization are decreasing to zero, and the rank of the complete matrices $\boldsymbol{M}$ seem to have little impact on the errors. However, in Figure 3 when $k = 6$ and $\alpha = 0.0005$, we notice that matrices of rank seven produce larger errors near $p = 1$, and the errors produced by matrices of rank nine even increase near $p = 1$; while in Figure 4 when $k = 6$ and $\alpha = 0.0001$, errors produced by matrices of all ranks decrease as the sampling probabilities approach one, but the errors in the end are much larger than those in Figure 3. A possible explanation for this is that the appropriate step size α is very sensitive to our input matrices. From Figure 4 we could conclude that $\alpha = 0.0001$ is not large enough for the algorithm to converge; And the increase of errors in Figure 3 is probably because $\alpha = 0.0005$ is not small enough for matrices with few missing entries since when $p \approx 1$, our starting point is really close to the optimal solution, and a large step size could even increase the error.

3.1.3. *Nuclear Norm Minimization.* Figure 5 shows that for nuclear norm minimization models we also have the errors decrease to zero as the sampling probabilities approach one. Besides, we notice that complete matrices of lower rank will produce fewer errors, this is consistent with the construction of the model, which aims to minimize the rank of the complete matrix.

3.2. **SVD vs. MF vs. NNM.** Now in order to compare the three different matrix completion methods, we fix the rank of our 50×10 matrices to be $r(\boldsymbol{M}) = 5$, choose $k = 8$ for both of direct SVD and matrix factorization, and the step size for matrix factorization is chosen to be $\alpha = 0.0005$. The results are shown in Figure 6, still, all of the errors decrease to zero as the sampling probabilities approach one, we could also see that on our synthetic data, matrix factorization outperforms the other two methods, and direct SVD produces much larger errors than the other two methods.

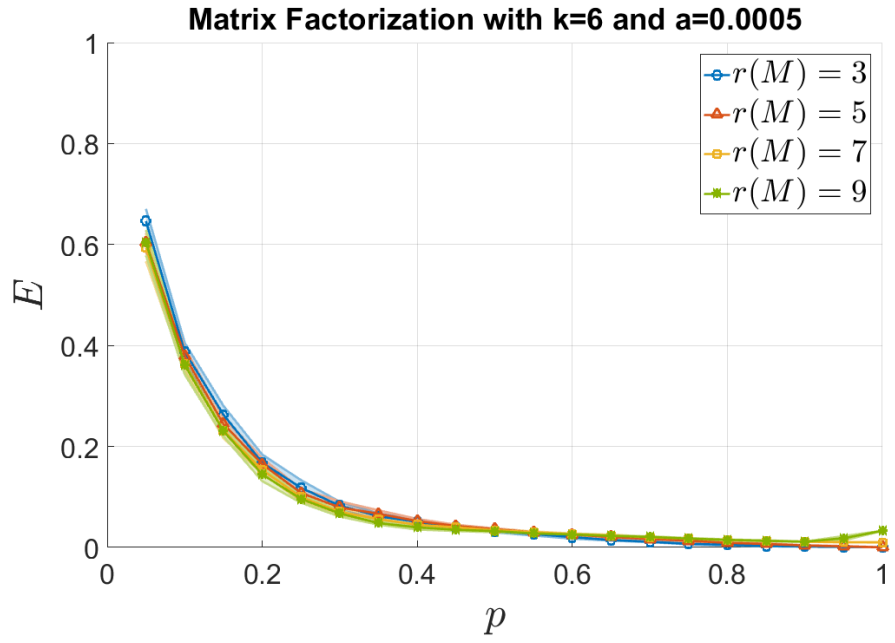

Figure 3. Error vs. Sparsity & Rank for matrix factorization model, with $k = 6$ and $\alpha = 0.0005$

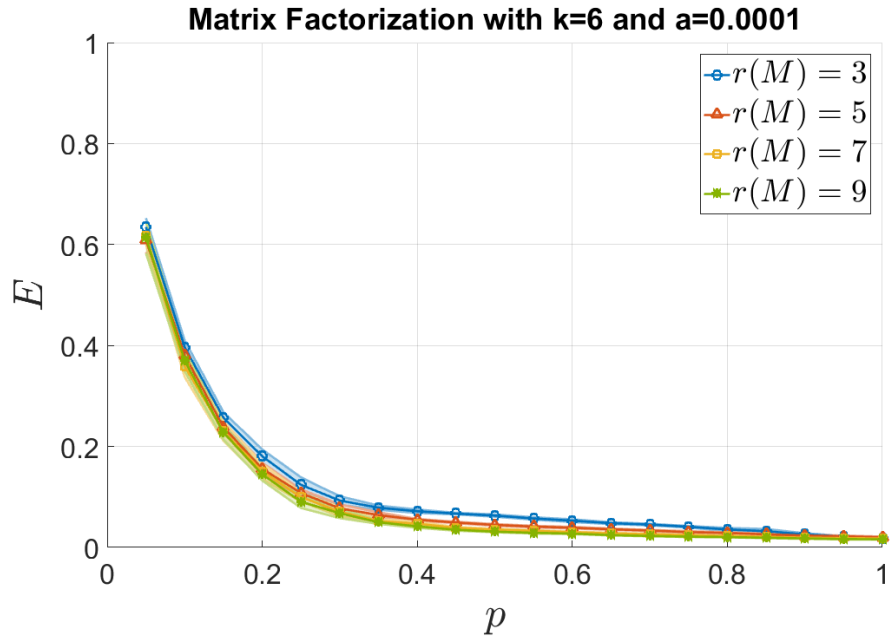

Figure 4. Error vs. Sparsity & Rank for matrix factorization model, with $k = 9$ and $\alpha = 0.0001$

3.3. **GD vs. SGD.** For matrix factorization model where we need to minimize the loss function

$$E = \frac{1}{2}\sum_{\substack{i=1\\(i,j)\in\Omega}}^{m}\sum_{j=1}^{n}(A_{ij} - \sum_{r=1}^{k}\mathbf{u}_{ir}\mathbf{v}_{jr})^2 = \frac{1}{2}\sum_{\substack{i=1\\(i,j)\in\Omega}}^{m}\sum_{j=1}^{n}e_{ij}^2,$$

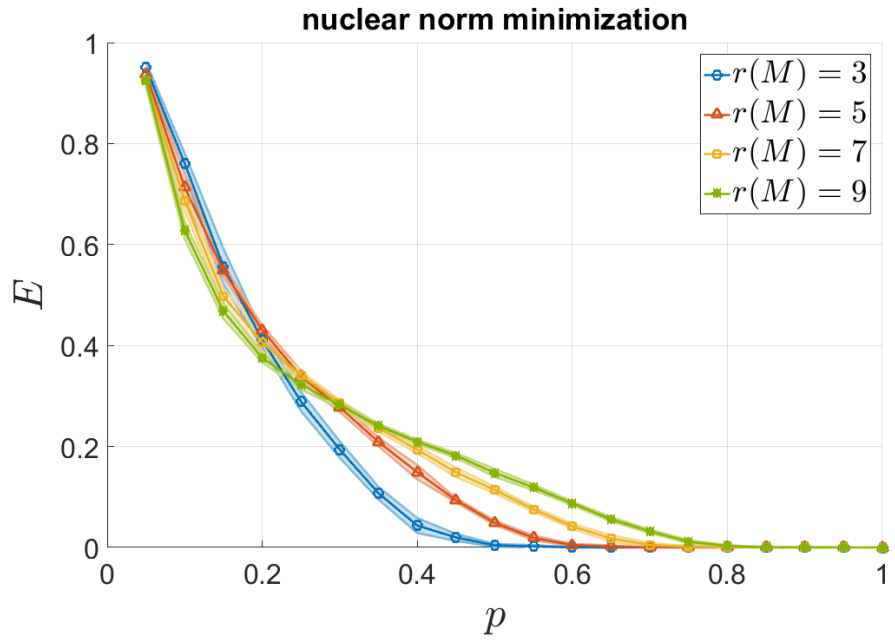

Figure 5. Error vs. Sparsity & Rank for nuclear norm minimization model

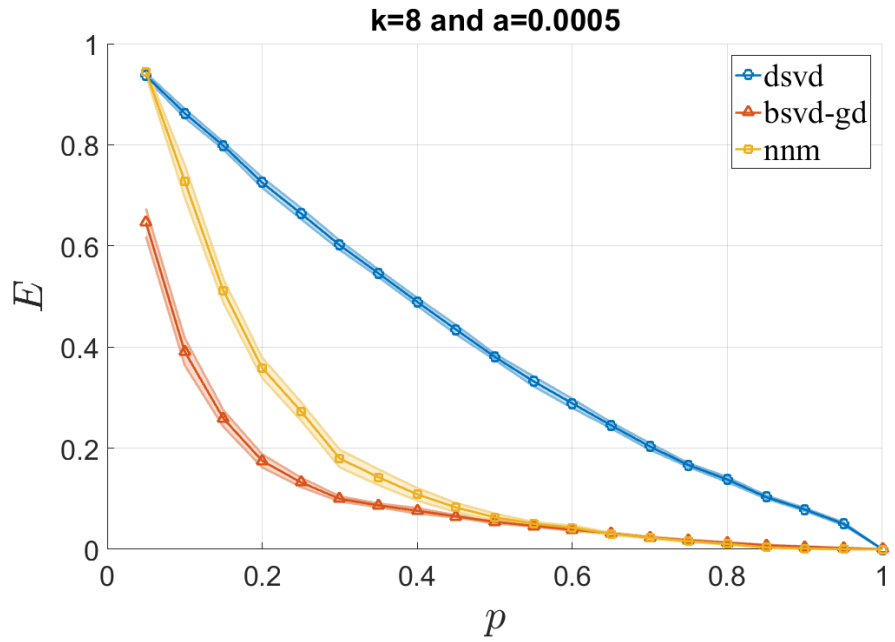

Figure 6. compare three different matrix completion methods with $k = 8$ and $\alpha = 0.0005$

besides gradient descent (GD) method, we could also choose a much faster stochastic gradient descent (SGD) method. In each iteration of gradient descent, the algorithm needs to compute the entire gradient matrix (e_{ij}) and update all the $(mk + nk)$ entries in $\mathbf{U}$ and $\mathbf{V}$, which is shown in Figure 7; while in each iteration of stochastic gradient descent, it randomly chooses some e_{ij} as an approximation to the gradient and update $2k$ entries in $\boldsymbol{u}_i$ and $\boldsymbol{v}_j$, which is shown in Figure

Algorithm *GD*(Ratings Matrix: R, Learning Rate: α)
begin
  Randomly initialize matrices U and V;
  $S = \{(i,j) : r_{ij} \text{ is observed}\}$;
  while not(convergence) **do**
  begin
    Compute each error $e_{ij} \in S$ as the observed entries of $R - UV^T$;
    for each user-component pair (i,q) **do** $u_{iq}^+ \Leftarrow u_{iq} + \alpha \cdot \sum_{j:(i,j)\in S} e_{ij} \cdot v_{jq}$;
    for each item-component pair (j,q) **do** $v_{jq}^+ \Leftarrow v_{jq} + \alpha \cdot \sum_{i:(i,j)\in S} e_{ij} \cdot u_{iq}$;
    for each user-component pair (i,q) **do** $u_{iq} \Leftarrow u_{iq}^+$;
    for each item-component pair (j,q) **do** $v_{jq} \Leftarrow v_{jq}^+$;
    Check convergence condition;
  end
end

Figure 7. algorithm for matrix factorization using gradient descent; *Recommendation Systems, Charu C. Aggarwal (2016), Chapter 3.6 Latent Factor Models* [1]

8. The operation count for each iteration in gradient descent is $2km_sn_s+2km_sn+2kmn_s+mk+nk$ and for each iteration in stochastic gradient descent is $6km_sn_s+2m_sn_s$, where m_s, n_s denote the number of known entries in rows and columns respectively. When the data matrix is sparse, we have $2km_sn_s+2km_sn+2kmn_s \geqslant 6km_sn_s$ and $mk+nk \geqslant 2m_sn_s$, which means SGD has less operation than GD and is thus faster; however when the data matrix is almost fully observed, we have $m \approx m_s, n \approx n_s$, then $6km_sn_s \approx 2km_sn_s + 2km_sn + 2kmn_s$ and $2m_sn_s$ could be even greater than $mk+nk$.

These theoretical results could be verified by the experiments shown in Figure 9, where the plot on the left shows that SGD produces much larger error than GD, and the plot on the right suggests that when p is small, that is, when the data matrix is sparse, SGD is much faster than GD; but as p approaches one, SGD could be even slower than GD.

4. Conclusions

We have implemented three matrix completion algorithms based on direct singular value decomposition, matrix factorization and nuclear norm minimization, and we investigate different factors that might influence the matrix completion error.

On our synthetic data matrices of size 50×10 and a certain fixed rank, it turns out that with appropriate hyperparameters, the errors produced by three different methods all decrease to zero as the number of known entries in the data matrices increase. For nuclear norm minimization algorithms, we also find out that matrices of lower rank would produce fewer errors, which is consistent with the rank minimization construction of this model. For direct SVD algorithms, choosing the right truncated size turns out to be very important to capture enough latent features as well as avoid unnecessary information. For matrix factorization algorithms, it is critical to choose the appropriate step size of gradient descent, such that the algorithm will neither bypass the local minimum nor spend too much time finding the local minimum. This hyperparameter turns out to be much more sensitive to the input matrices than others, the experiments suggest that we need to adjust the step size to the sparsity of the data matrices.

Algorithm *SGD*(Ratings Matrix: R, Learning Rate: α)
begin
  Randomly initialize matrices U and V;
  $S = \{(i, j) : r_{ij}$ is observed$\}$;
  while not(convergence) **do**
  begin
    Randomly shuffle observed entries in S;
    for each $(i, j) \in S$ in shuffled order **do**
    begin
      $e_{ij} \Leftarrow r_{ij} - \sum_{s=1}^{k} u_{is} v_{js}$;
      for each $q \in \{1 \ldots k\}$ **do** $u_{iq}^{+} \Leftarrow u_{iq} + \alpha \cdot e_{ij} \cdot v_{jq}$;
      for each $q \in \{1 \ldots k\}$ **do** $v_{jq}^{+} \Leftarrow v_{jq} + \alpha \cdot e_{ij} \cdot u_{iq}$;
      for each $q \in \{1 \ldots k\}$ **do** $u_{iq} = u_{iq}^{+}$ and $v_{jq} = v_{jq}^{+}$;
    end
    Check convergence condition;
  end
end

Figure 8. algorithm for matrix factorization using stochastic gradient descent; *Recommendation Systems, Charu C. Aggarwal (2016), Chapter 3.6 Latent Factor Models* [1]

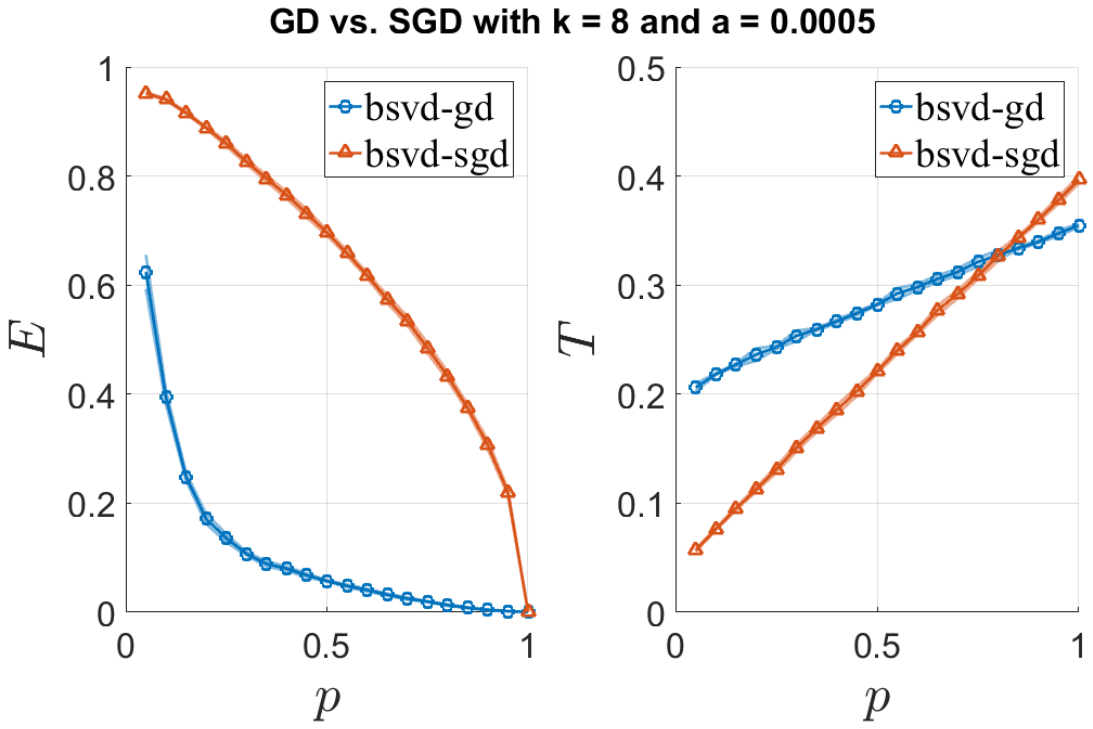

Figure 9. GD vs. SGD; the plot on the left shows the matrix completion errors, and the plot on the right shows the time each algorithm took

We compare these three matrix completion methods on the same data set and it shows that matrix factorization and nuclear norm minimization have much better performances than the direct SVD method, and matrix factorization outperforms nuclear norm minimization when

the data matrix is sparse. For matrix factorization model which involves minimizing the loss function, we also compare gradient descent method and stochastic gradient method, and we have verified that SGD is much faster than GD while it produces much larger error.

Some problems of interest for future work would be: for direct SVD algorithms, is there a way to decide the best truncated size? For matrix factorization algorithms, what if we use adaptive step size instead of constant step size?

REFERENCES

[1] Charu C. Aggarwal. *Recommendation Systems*. 2016.

[2] Emmanuel J. Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Foundations of Computational Mathematics*, 9:717–772, 2009.

[3] A. Ramlatchan, M. Yang, Q. Liu, M. Li, J. Wang, and Y. Li. A survey of matrix completion methods for recommendation systems. *Big Data Mining and Analytics*, 1(4):308–323, December 2018.

[4] Badrul Munir Sarwar, George Karypis, Joseph A. Konstan, and John Riedl. Application of dimensionality reduction in recommender system - a case study. 2000.

[5] Zehan Chao Zifeng Wang, Hui Jin and Deanna Needell. Matrix completion in structural observations and deterministic observations.

Appendices

Here we show how we choose the step size α of gradient descent for matrix factorization algorithms. When $\alpha = 1 \times 10^{-5}$, Figure 10 shows that the algorithm does not converge within 1000 iterations (we set the maximum number of iterations to be 1000), when the sampling probabilities approach zero, the algorithm still produce large errors; when $\alpha = 1 \times 10^{-4}$, Figure 11 shows that the errors are reduced but it still stay large when the sampling probabilities approach zero; so we try $\alpha = 1 \times 10^{-3}$, and the results in Figure 12 suggest that now the step size is too big since the errors start to increase. Therefore in the end we choose our step size to be $\alpha = 5 \times 10^{-4}$, which turns out to work well in most of our experiments.

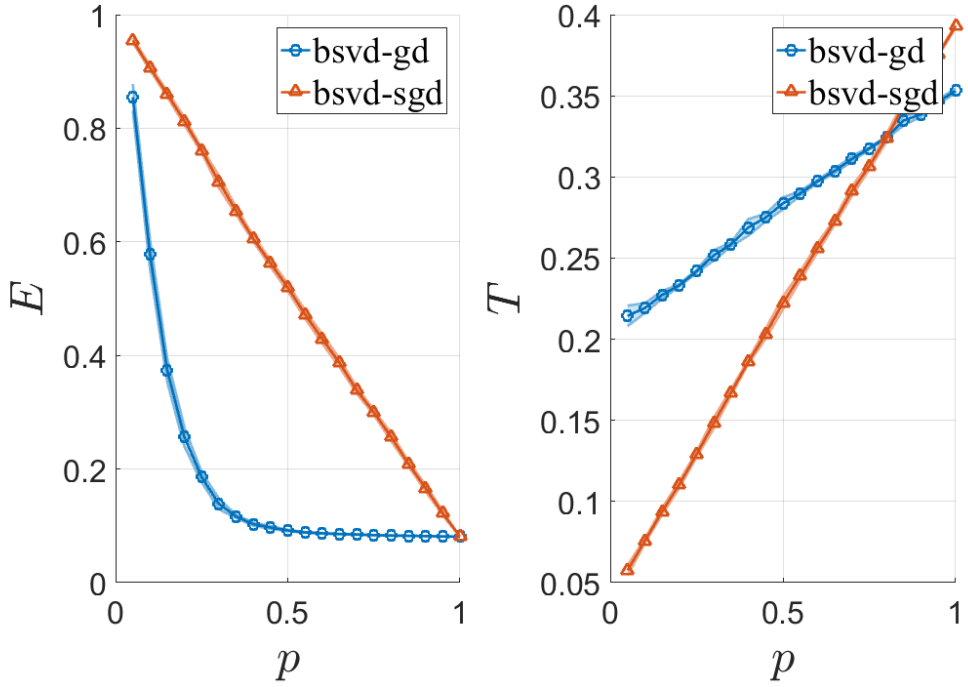

Figure 10. GD vs. SGD with $\alpha = 1 \times 10^{-5}$

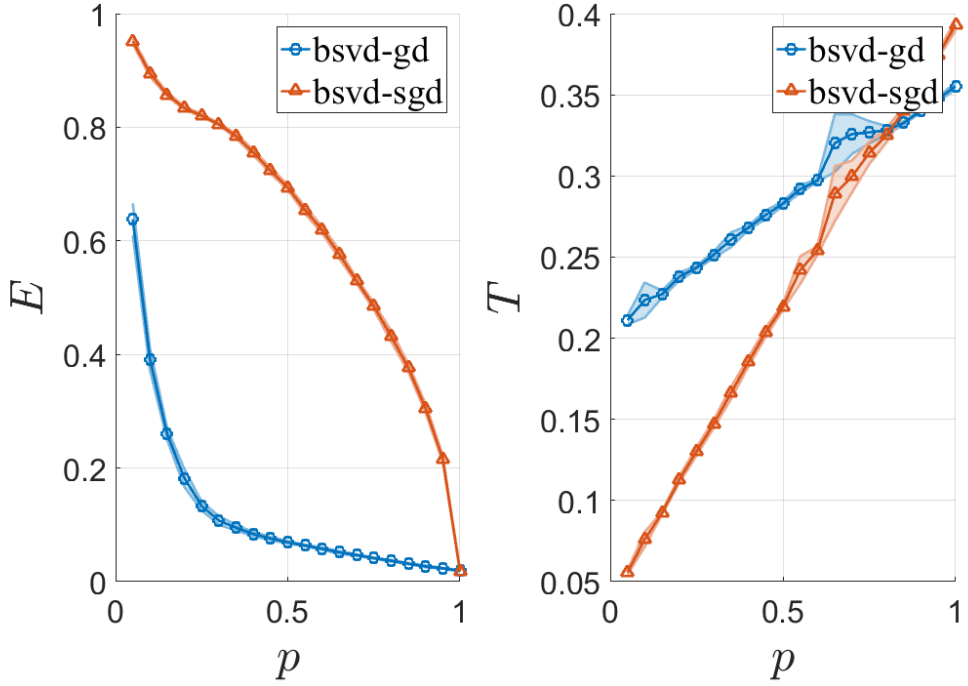

Figure 11. GD vs. SGD with $\alpha = 1 \times 10^{-4}$

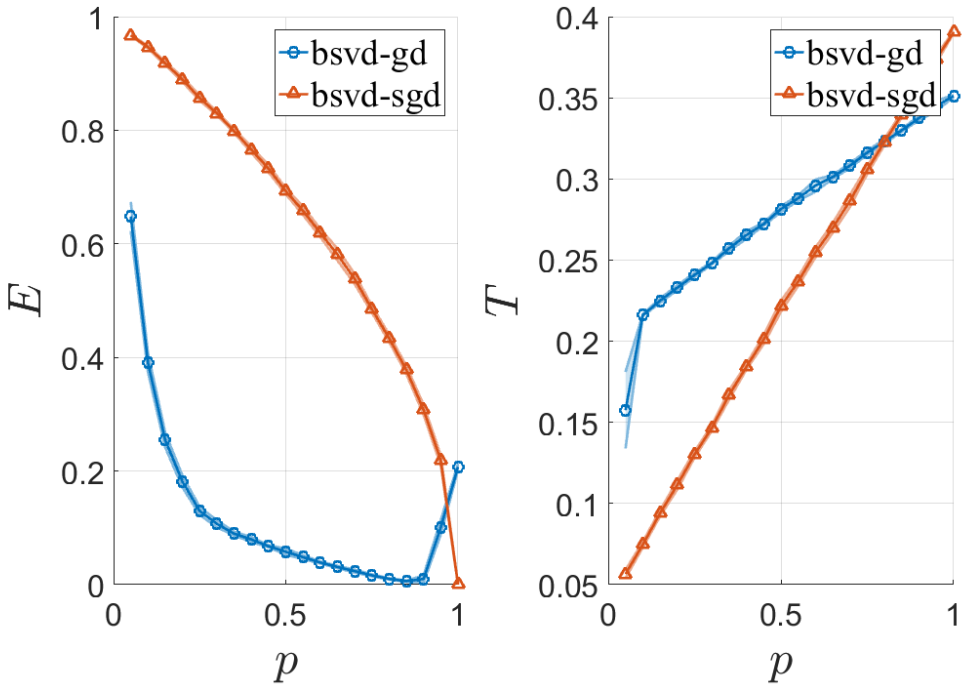

Figure 12. GD vs. SGD with $\alpha = 1 \times 10^{-3}$