DATS-SHU 240 Final Project

The Movie Recommendation System

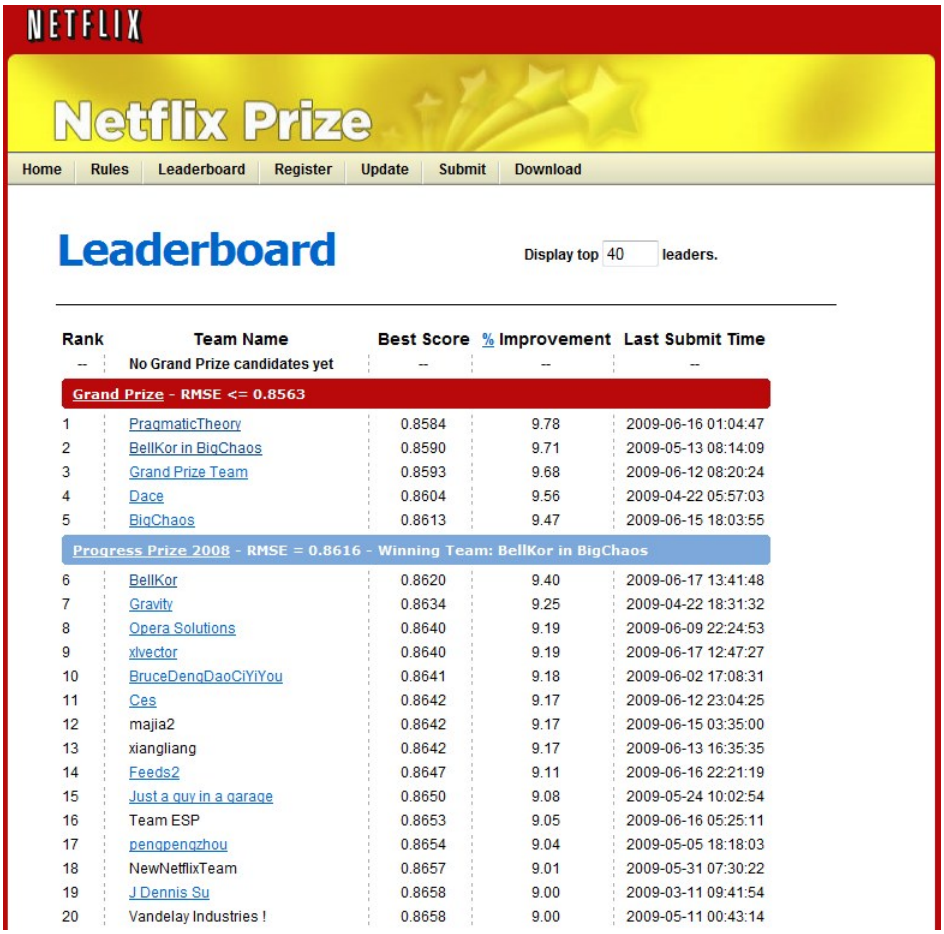
Jialin Wang (wj716), Pinyu Chen (pc2753))

May 2023

1 Introduction

Matrix completion is a method to fill in missing entries in a matrix. This method is widely applied in many data settings, including the famous Netflix movie recommendation system. In the system, platform users give their ratings for movies they have already watched, and the system needs to predict the ratings for the movies they have not watched and then recommend some potential preferences for the customers. There was once a well-known online competition called Netflix Prize from 2006 to 2009, which challenges the competitors to predict user ratings for movies with the best collaborative filtering algorithm, based simply on previous ratings without any other information about the movies or users 1. On September 18, 2009, Netflix announced team "BellKor's Pragmatic Chaos" as the prize winner (a Test RMSE of 0.8567), and the prize was awarded to the team in a ceremony on September 21, 2009 [3].

This is how movie recommendation works in common online platforms. And such systems can fulfill our recreational satisfaction just through mathematical calculation. Interesting as the problem seems, there are numerous challenges that need to be overcome. For instance, it is difficult to transform the matrix completion problem into a convex optimization problem without much deviation. Additionally, the data matrix of user ratings is very sparse, and only around 1% of the ratings are present. So an



NETFLIX

Netflix Prize

Home | Rules | Leaderboard | Register | Update | Submit | Download

Leaderboard

Display top 40 leaders.

Rank	Team Name	Best Score	% Improvement	Last Submit Time
--	No Grand Prize candidates yet	--	--	--
Grand Prize - RMSE <= 0.8563				
1	PragmaticTheory	0.8584	9.78	2009-06-16 01:04:47
2	BellKor in BigChaos	0.8590	9.71	2009-05-13 08:14:09
3	Grand Prize Team	0.8593	9.68	2009-06-12 08:20:24
4	Dace	0.8604	9.56	2009-04-22 05:57:03
5	BigChaos	0.8613	9.47	2009-06-15 18:03:55
Progress Prize 2008 - RMSE = 0.8616 - Winning Team: BellKor in BigChaos				
6	BellKor	0.8620	9.40	2009-06-17 13:41:48
7	Gravity	0.8634	9.25	2009-04-22 18:31:32
8	Opera Solutions	0.8640	9.19	2009-06-09 22:24:53
9	xlvector	0.8640	9.19	2009-06-17 12:47:27
10	BruceDengDaoCiYiYou	0.8641	9.18	2009-06-02 17:08:31
11	Ces	0.8642	9.17	2009-06-12 23:04:25
12	majia2	0.8642	9.17	2009-06-15 03:35:00
13	xiangliang	0.8642	9.17	2009-06-13 16:35:35
14	Feeds2	0.8647	9.11	2009-06-16 22:21:19
15	Just a guy in a garage	0.8650	9.08	2009-05-24 10:02:54
16	Team ESP	0.8653	9.05	2009-06-16 05:25:11
17	pengpengzhou	0.8654	9.04	2009-05-05 18:18:03
18	NewNetflixTeam	0.8657	9.01	2009-05-31 07:30:22
19	J Dennis Su	0.8658	9.00	2009-03-11 09:41:54
20	Vandelay Industries !	0.8658	9.00	2009-05-11 00:43:14

Figure 1: Netflix Prize

efficient algorithm should be figured out to complete the missing entries, thus solving the optimization problem.

1.1 Literature Review

The paper "Exact Matrix Completion via Convex Optimization" by Candes, E. J. and Recht, B. proposes a convex optimization approach for matrix completion, whose goal is to recover a low-rank matrix from a subset of its entries. The nuclear norm is the sum of the singular values of the matrix. The authors proved that under certain conditions, the proposed approach based on minimizing the nuclear norm of the matrix is exact, meaning that it recovers the low-rank matrix as desired.

The paper "A Singular Value Thresholding Algorithm for Matrix Completion" by Cai, J.-F., Candes, E. J., and Shen, Z. proposes a novel algorithm for minimizing nuclear norm and uses it to recover a low-rank matrix. The proposed algorithm, called Singular Value Thresholding (SVT), is based on the idea of thresholding the singular values of a matrix. The algorithm starts with an initial guess of the low-rank matrix and then iteratively updates it by thresholding its singular values.

Overall, these papers present a simple and effective algorithm for matrix completion that has theoretical guarantees and performs well in practice. The algorithm has since been widely adopted in various applications and has paved the way for further research in the field of matrix completion.

1.2 Contributions

We finished the literature review, during which we get to know that in order to fill a sparse matrix, we can approximate it with a low-rank matrix, and then recover a filled matrix using the low-rank matrix. Also, we get to know that approximating a sparse matrix with the objective of minimizing rank in NP-hard. Under certain conditions, the objective can be converted to minimizing nuclear norms. We use the SVT algorithm

proposed in paper "A Singular Value Thresholding Algorithm for Matrix Completion". Based on this algo and using the Netflix Prize Dataset, we write Python code to get the final result of movie recommendation.

2 Problem Formulation

2.1 Model Formulation

According to Candès et al. [2], the common way to deal with the matrix completion problem is to impose a rank constraint on the sparse matrix. The reason is that many real-world matrices have low rank, meaning that they can be approximated by a much smaller number of basis vectors than the total number of entries in the matrix. By imposing a rank constraint on the sparse matrix, we can approximate the partially observed matrix using fewer parameters than the original matrix. This can lead to more efficient algorithms for completing the matrix. Additionally, imposing a rank constraint can also help to grasp the characteristics of a matrix and avoid overfitting, which can occur when a model tries to fit the training data too closely.

Given a data matrix $\mathbf{Z}$ whose entries are indexed by the subset $\Omega \subset \{1, ..., m\} \times \{1, ..., n\}$, and Ω is a random subset of cardinality s, we derive the approach to find the lowest rank approximating matrix $\hat{\mathbf{Z}}$. The matrix $\hat{\mathbf{Z}}$ interpolates the observed entries of data matrix $\mathbf{Z}$, and the expression is [2]:

$$\text{minimize} \quad \text{rank}(\mathbf{M}) \tag{1}$$

$$\text{subject to} \quad m_{ij} = z_{ij} \quad \text{for } \forall\ (i,j) \in \Omega \tag{2}$$

$$a \leq m_{ij} \leq b \quad \text{for } \forall\ (i,j) \notin \Omega \tag{3}$$

where $a,\ b$ are the lower and upper constraints of the predicted ratings (1 and 5 usually). However, the optimization problem is computationally intractable (NP-hard) and is not applicable for even moderately large matrices, not to mention the Netflix problem whose dataset has 17770 columns (movies) and 48189 rows (users).

The nuclear norm is the sum of the singular values of a matrix, which is also called the trace norm. Candès and Recht's result proves that under suitable conditions, the rank minimization program above is equivalent to the following program because they have the same unique solution [2]:

$$\text{minimize} \quad \|\mathbf{M}\|_* \tag{4}$$

$$\text{subject to} \quad m_{ij} = z_{ij} \quad \text{for } \forall\ (i,j) \in \Omega \tag{5}$$

$$a \leq m_{ij} \leq b \quad \text{for } \forall\ (i,j) \notin \Omega \tag{6}$$

The rationale behind the matrix completion using minimum nuclear norm rather than rank minimization is that the problem turns into a convex one. To be specific, with the nuclear norm - the sum of singular values, it turns into a semi-definite program, which belongs to the convex programs. Thus, special solvers can be applied to find the optimal solution to the data matrix.

2.2 Algorithm Outline

Since minimizing the nuclear norm is a productive way to complete sparse matrices and recover a low-rank matrix subject to restrictions, it is of great interest to develop an algorithm to solve the program. A numerical method being researched is the singular value thresholding algorithm, which usually deals with the nuclear norm minimization problem of the following form:

$$\text{minimize} \quad \|\mathbf{X}\|_* \quad \text{subject to} \quad \mathcal{A}(\mathbf{X}) = \mathbf{b} \tag{7}$$

where $\mathcal{A}$ is a linear operator acting on matrices of size $m \times n$ and $\mathbf{b} \in \mathbb{R}^m$. This method applies to matrices of large size and will produce solutions with low rank. To fit for the sparse matrix completion setting, we introduce the projection operator

$\mathcal{P}_\Omega : \mathbb{R}^{m\times n} \mapsto \mathbb{R}^{m\times n}$ as follows [1]:

$$[\mathcal{P}_\Omega(\mathbf{Z})]_{ij} = \begin{cases} z_{ij} & \text{if } (i,j) \in \Omega \\ 0 & \text{if } (i,j) \in \Omega \end{cases} \tag{8}$$

As a result, our program can be expressed as:

$$\text{minimize} \quad \|\mathbf{X}\|_* \quad \text{subject to} \quad \mathcal{P}_\Omega(\mathbf{X}) = \mathcal{P}_\Omega(\mathbf{M}) \tag{9}$$

where $\mathbf{X} \in \mathbb{R}^{m\times n}$ is an optimization variable. We fix $\tau \geq 0$ and a sequence $\{\delta_k\}_{k\geq 1}$ of scalar step sizes. Then we start with $\mathbf{Y}^0 = 0 \in \mathbb{R}^{m\times n}$, and iteratively the algorithm defines [1]:

$$\begin{cases} \mathbf{X}^k = \text{shrink}(\mathbf{Y}^{k-1}, \tau), \\ \mathbf{Y}^k = \mathbf{Y}^{k-1} + \delta_k \mathcal{P}_\Omega(\mathbf{M} - \mathbf{X}^k) \end{cases} \tag{10}$$

And $\text{shrink}(\mathbf{Y}, \tau)$ is a nonlinear function employing a soft-thresholding rule at level τ to the singular values of the input matrix. Procedures for deriving the program will be discussed in the Appendix. Here, according to Cai, Candès, and Shen, there are two vital remarks about this program worth mentioning [1]:

1. *Sparsity.* For $k \geq 0$, $\mathbf{Y}^k$ vanishes outside of Ω and is sparse. As a result, the shrink function can be evaluated quickly.

2. *Low-rank property.* The matrix $\mathbf{X}^k$ has a low rank and therefore, the algorithm only needs minimum storage to save the principal components in memory. Thus, the method is powerful in solving large-size matrix completion problems.

3 Experiments

The official Netflix movie rating dataset contains over 100 million ratings from 480 thousand randomly chosen, anonymous Netflix customers over 17000 movies. The ratings are on a scale from 1 to 5 (integer) to reflect their personal preference for a specific

Algorithm 1: Singular Value Thresholding (SVT) Algorithm

Input: sampled set Ω and sampled entries $\mathcal{P}_\Omega(\boldsymbol{M})$, step size δ, tolerance ϵ, parameter τ, increment ℓ, and maximum iteration count $k_{\max}$
Output: $\boldsymbol{X}^{\text{opt}}$
Description: Recover a low-rank matrix $\boldsymbol{M}$ from a subset of sampled entries

Set $\boldsymbol{Y}^0 = k_0 \delta \, \mathcal{P}_\Omega(\boldsymbol{M})$ (k_0 is defined in (5.3))
Set $r_0 = 0$
for $k = 1$ **to** $k_{\max}$
  Set $s_k = r_{k-1} + 1$
  repeat
    Compute $[\boldsymbol{U}^{k-1}, \boldsymbol{\Sigma}^{k-1}, \boldsymbol{V}^{k-1}]_{s_k}$
    Set $s_k = s_k + \ell$
  until $\sigma^{k-1}_{s_k - \ell} \leq \tau$
  Set $r_k = \max\{j : \sigma_j^{k-1} > \tau\}$
  Set $\boldsymbol{X}^k = \sum_{j=1}^{r_k} (\sigma_j^{k-1} - \tau) \boldsymbol{u}_j^{k-1} \boldsymbol{v}_j^{k-1}$

  if $\|\mathcal{P}_\Omega(\boldsymbol{X}^k - \boldsymbol{M})\|_F / \|\mathcal{P}_\Omega \boldsymbol{M}\|_F \leq \epsilon$ **then break**

  Set $Y_{ij}^k = \begin{cases} 0 & \text{if } (i,j) \notin \Omega, \\ Y_{ij}^{k-1} + \delta(M_{ij} - X_{ij}^k) & \text{if } (i,j) \in \Omega \end{cases}$
end *for* k
Set $\boldsymbol{X}^{\text{opt}} = \boldsymbol{X}^k$

Figure 2: Algorithm for Singualr Value Thresholding

movie. To boost simplicity, here we extract the first 20000 rows of the data to mimic the movie recommendation system using the singular value thresholding algorithm. According to Candès et al. [2], the number of non-zero entries must be larger than $Cn^{1.2}rlog(n)$, for some positive numerical constant C in an $n \times n$ matrices of rank r, then the matrix could be recovered by solving a simple convex optimization program. As a result, we drop columns (movies) which has more than 80 percent missing entries to fulfill such threshold. After basic data processing, the rating matrix we are going to deal with is of size 110×202, which represents 110 users and 202 movies.

We then use Python to deal with the convex optimization problem. The algorithm we follow is the singular value thresholding in Figure 2 [1]. Within 1000 iterations, we generate the recovered matrix with a minimum nuclear norm of around 430. The incomplete and complete matrices after conducting the optimization are shown in Figure 3 and Figure 4. Figure 4 is the completed matrix. After sorting and scaling the rating between 1 and 5 to fulfill the constraint, we conclude the five most recommended movies

```
[[3. 2. 5. ... 3. 3. 5.]
 [5. 4. 5. ... 5. 5. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [4. 3. 0. ... 3. 4. 4.]
 [0. 0. 0. ... 0. 0. 0.]]
```

Figure 3: Before matrix completion

```
[[1.17261942 0.32946645 2.18919403 ... 1.08226951 0.39657567 1.17456695]
 [1.93776479 1.85122593 2.07816775 ... 2.43431039 2.31464596 1.80313773]
 [2.78263925 2.30399386 2.83435544 ... 3.22205249 2.42373246 2.01156631]
 ...
 [2.87362269 2.56687371 3.16503645 ... 3.37072615 2.61324192 2.6522982 ]
 [2.00815022 1.60470261 2.02162615 ... 1.97177976 1.61194984 1.20115379]
 [1.6400041  1.50810879 2.08980106 ... 2.2598167  2.25464628 1.75524819]]
```

Figure 4: After matrix completion

for different users. For instance, among all the movies he or she has not watched before, user 6 will have a predicted 5.00 rating for the movie "Secondhand Lions", which is the highest predicted rating; secondly, the user will probably rate 4.26 for the movie "Hidalgo", which might be his or her second favorite; thirdly, 4.21 for the movie "Harry Potter and the Prisoner of Azkaban"; fourthly, 3.97 for the movie "The Princess Bride", and then 3.96 for the movie "Shrek (Full-screen)". And for user 10, he or she will have a predicted rating 5.00 for the movie "Secondhand Lions", which is the highest predicted rating; secondly, the user will probably rate 4.70 for the movie "Swordfish"; thirdly, a predicted rating 4.62 for the movie "The Passion of the Christ"; and then 4.43 for the movie "A Few Good Men", 4.38 for the movie "Rain Man". Consequently, we can follow such a structure to generate recommended movies for each user.

4 Conclusions and Discussions

With this project, we get to know how to solve a real-life problem, the movie recommendation, with the mathematical and programming knowledge we have. The first step is to convert it to a matrix completion problem. Then we explore existing papers to see how to solve a matrix completion problem. It consists of approximating the sparse matrix with a low-rank matrix and then recovering a filled matrix. This is already an optimization problem. But it's NP-hard and thus requires further thought. Under certain conditions, we convert the optimization problem to another optimization

problem, which is approximating the sparse matrix with a low-nuclear-norm matrix. By applying the SVT algorithm, we successfully write codes and solve the Netflix Prize movie recommendation challenge.

References

[1] Cai, J.-F., Candes, E. J., and Shen, Z. A singular value thresholding algorithm for matrix completion.

[2] Candes, E. J., and Recht, B. Exact matrix completion via convex optimization.

[3] Wikipedia contributors. Netflix prize — Wikipedia, the free encyclopedia, 2023. [Online; accessed 5-April-2023].

Appendix A Algorithm Detail Analysis

According to Cai et al. [1], the definition of the singular value decomposition (SVD) of a matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$ of rank r is:

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T, \quad \Sigma = \text{diag}(\{\sigma_i\}_{1 \leq i \leq r}) \tag{11}$$

And here $\mathbf{U}$ is a matrix of size $m \times r$ and $\mathbf{V}$ is of size $n \times r$. They both have orthogonal columns and are called left and right singular vectors respectively. Additionally, the singular values σ_i are positive, whose sum is the nuclear norm mentioned before. For each $\tau \geq 0$, we define the soft-thresholding operator $\mathcal{D}_\tau$ as [1]:

$$\mathcal{D}_\tau(\mathbf{X}) := \mathbf{U}\mathcal{D}_\tau(\Sigma)\mathbf{V}^T, \quad \mathcal{D}_\tau(\Sigma) = \text{diag}(\{(\sigma_i - \tau)_+\}), \tag{12}$$

where $p_+ = \max(0, p)$. In other words, this operator imposes a soft-thresholding constraint to the singular values of matrix $\mathbf{X}$, which effectively shrinks these values toward zero. When some of the singular values of $\mathbf{X}$ are smaller than the threshold τ, we force them to turn to zero. This procedure reduces the rank of a matrix quickly. And then the rank of $\mathcal{D}_\tau(\mathbf{X})$ may become lower than that of $\mathbf{X}$, leading to sparse outputs [1]. Additionally, for each $\tau \geq 0$ and $\mathbf{Y} \in \mathbb{R}^{m \times n}$, the singular value shrinkage operator follows:

$$\mathcal{D}_\tau(\mathbf{Y}) = \arg\min_{\mathbf{X}} \{\frac{1}{2}\|\mathbf{X} - \mathbf{Y}\|_F^2 + \tau\|\mathbf{X}\|_*\} \tag{13}$$

As a result, the original program (9) can be written as:

$$\text{minimize} \;\; \tau\|\mathbf{X}\|_* + \frac{1}{2}\|\mathbf{X}\|_F^2 \quad \text{subject to} \;\; \mathcal{P}_\Omega(\mathbf{X}) = \mathcal{P}_\Omega(\mathbf{M}) \tag{14}$$

The next thing is to derive the iteration (10). Here, we introduce the Lagrange multiplier. Firstly, we set $f_\tau(\mathbf{X}) = \tau\|\mathbf{X}\|_* + \frac{1}{2}\|\mathbf{X}\|_F^2$ for some fixed $\tau > 0$, thus the original

problem turns into [1]:

$$\text{minimize} \quad f_\tau(\mathbf{X}) \quad \text{subject to} \quad \mathcal{P}_\Omega(\mathbf{X}) = \mathcal{P}_\Omega(\mathbf{M}) \tag{15}$$

And the Lagrangian for this program is given by $\mathcal{L}(\mathbf{X}, \mathbf{Y}) = f_\tau(\mathbf{X}) + \langle \mathbf{Y}, \mathcal{P}_\Omega(\mathbf{M} - \mathbf{X}) \rangle$, where $\mathbf{Y} \in \mathbb{R}^{m \times n}$. Since strong duality holds, and $\mathbf{X}^*$ and $\mathbf{Y}^*$ are primal-dual optimal, obeying

$$\sup_{\mathbf{Y}} \inf_{\mathbf{X}} \mathcal{L}(\mathbf{X}, \mathbf{Y}) = \mathcal{L}(\mathbf{X}^*, \mathbf{Y}^*) = \inf_{\mathbf{X}} \sup_{\mathbf{Y}} \mathcal{L}(\mathbf{X}, \mathbf{Y}) \tag{16}$$

(The function $g_0(\mathbf{Y}) = \inf_{\mathbf{X}} \mathcal{L}(\mathbf{X}, \mathbf{Y})$ is called the dual function.) Then we apply Uzawa's algorithm to find a saddle point of the problem, the optimal point, to obtain an iterative procedure. We inductively define from $\mathbf{Y}_0 = 0$ that [1]:

$$\begin{cases} \mathcal{L}(\mathbf{X}^k, \mathbf{Y}^{k-1}) = \min_{\mathbf{X}} \mathcal{L}(\mathbf{X}, \mathbf{Y}^{k-1}) \\ \mathbf{Y}^k = \mathbf{Y}^{k-1} + \delta_k \mathcal{P}_\Omega(\mathbf{M} - \mathbf{X}^k), \end{cases} \tag{17}$$

where $\{\delta_k\}_{k \geq 1}$ is a sequence of positive scalar step sizes. Uzawa's algorithm is indeed a subgradient approach applied to the dual problem, with each step moving the iteration in the direction of the gradient or a subgradient. Then, we will acquire convergence. We observe that $\partial_{\mathbf{Y}} g_0(\mathbf{Y}) = \partial_{\mathbf{Y}} \mathcal{L}(\tilde{\mathbf{X}}, \mathbf{Y}) = \mathcal{P}_\Omega(\mathbf{M} - \tilde{\mathbf{X}})$ where $\tilde{\mathbf{X}}$ is the minimizer of the Lagrangian for the value of $\mathbf{Y}$. As a result, the gradient descent update for $\mathbf{Y}$ is of the form [1]:

$$\mathbf{Y}^k = \mathbf{Y}^{k-1} + \delta_k \partial_{\mathbf{Y}} g_0(\mathbf{Y}^{k-1}) = \mathbf{Y}^{k-1} + \delta_k \mathcal{P}_\Omega(\mathbf{M} - \mathbf{X}^k) \tag{18}$$

Apart from that, we need to compute the minimizer of the Lagrangian (17), and note that:

$$\arg\min \ f_\tau(\mathbf{X}) + \langle \mathbf{Y}, \mathcal{P}_\Omega(\mathbf{M} - \mathbf{X}) \rangle = \arg\min \ \{ \frac{1}{2} \|\mathbf{X} - \mathcal{P}_\Omega \mathbf{Y}\|_F^2 + \tau \|\mathbf{X}\|_* \} \tag{19}$$

And from (13) we know that the minimizer is given by $\mathcal{D}_\tau(\mathcal{P}_\Omega(\mathbf{Y}))$ and also $\mathbf{Y}^k = \mathcal{P}_\Omega(\mathbf{Y}^k)$ for all $k \geq 0$, Uzawa's algorithm takes the form [1]:

$$\begin{cases} \mathbf{X}^k = \mathcal{D}_\tau(\mathbf{Y}^{k-1}, \tau), \\ \mathbf{Y}^k = \mathbf{Y}^{k-1} + \delta_k \mathcal{P}_\Omega(\mathbf{M} - \mathbf{X}^k) \end{cases} \tag{20}$$

This is exactly the update of (10) and we finish the derivation. For program (20), the shrinkage iteration is very simple to implement. At each iteration, we only need to compute an SVD and perform elementary matrix operations. The program could be coded easily with the help of a standard numerical linear algebra package.

Appendix B Code

Here are the codes for Algorithm for SVT, Data Processing and the main branch, in the next pages.

```python
import numpy as np
def svd_solver(A,mask,t=None,delta=None,epsilon=1e-3,max_iterations=1000):
  Y=np.zeros_like(A)

  #initialize the step
  if not t:
    t=5*np.sum(A.shape)/2
  if not delta:
    delta=1.2*np.prod(A.shape)/np.sum(mask)

  for i in range(max_iterations):
    #SVD decomposition
    U,S,V=np.linalg.svd(Y,full_matrices=False)
    #soft-thresholding operator
    S=np.maximum(S-t,0)
    #singular value shrinkage
    X=np.linalg.multi_dot([U,np.diag(S),V])
    #interation of Y
    Y+=delta*mask*(A-X)
    #error calculation
    # print(X.max())
    # print(X.min())
    error=np.linalg.norm(mask*(X-A))/np.linalg.norm(mask*A)
    if error<epsilon:
      break
  return X
```

Figure 5: Code for Algorithm for SVT

```python
from create_df import df
import pandas as pd

df = df.head(20000)
two_level_index_series = df.set_index(['movie','user'])['rating']
new_df = two_level_index_series.unstack(level=0)
df = new_df.dropna(axis=1, how='all')
percent_na = df.isna().sum() / len(df)
df = df.dropna(axis=1, thresh=len(df)*0.2)
df.to_pickle('dataframe.pkl')
```

Figure 6: Code for Data Processing

```python
import pandas as pd
import numpy as np

# load the dataframe from the pickle file
df = pd.read_pickle('dataframe.pkl')
df.to_excel('test.xlsx')
# print the dataframe
#print(df)

#add a mask
df = df.fillna(0)
df = df.astype(float)
data = df.iloc[:,:]
matrix = data.values
shape = matrix.shape
mask = np.zeros(shape)
for i in range(0,shape[0]):
    for j in range(0,shape[1]):
        if matrix[i,j]>0:
            mask[i,j]=1

from svd_solver import svd_solver
result=svd_solver(matrix,mask)
print('nuclear norm:', np.linalg.norm(result,'nuc'))

from new_rating import new_rating
A=new_rating(matrix,result,3)
rating = []
for each in A:
    rating.append(each[1])
from rescale import rescale_list
B=rescale_list(rating)
for j in range(len(B)):
    A[j][1]=B[j]
print('movie recommendation ranking:',A)
listA=[]
for each in A:
  listA.append(each[0])
print('five movie indice:',listA[:5])
for i in listA[:5]:
    print('movie name:', df.columns.values[i])
print(matrix)
print(result)
```

Figure 7: Code for the main branch

```
movie recommendation ranking: [[9, 5.0], [138, 4.261867231411915], [134, 4.2057045478169535], [80, 3.9697944739423603],
five movie indice: [9, 138, 134, 80, 163]
movie name: 1110
movie name: 12732
movie name: 12338
movie name: 7193
movie name: 14621
```

Figure 8: One of the running result