SC505 Linear Algebra and Optimization
Autumn 2020

Project Title: Matrix Completion Algorithm — Date: December 10, 2020
Authors: Aakash T Dhedhi — Vijaykumar R Visavaliya

1 Introduction

A partial matrix is a rectangular array in which some entries are specified, while the remaining unspecified entries are free to be chosen from an indicated set (such as the real numbers or complex numbers). A completion of a partial matrix is a specific choice of values for the unspecified entries resulting in a conventional matrix.*Matrix completion* is the task of filling in the missing entries of a partial matrix. A wide range of datasets are naturally organized in matrix form. Consider the example of a movie-rental service collecting its customers' ratings on movies. This dataset can be represented as a movie-ratings matrix where each customer is represented by a row and each movie is represented by a column. Each entry of this matrix represents the preference of the associated customer for a particular movie. One example is the movie-ratings matrix, as appears in the Netflix problem [7]. Similarly, the frequencies of words used in a collection of documents can be represented as a term-document matrix, where each entry corresponds to the number of times the associated term appears in the indicated document.

A matrix completion problem asks whether a given partial matrix has a completion of a desired type; for example the low rank matrix completion asks for which entries the marix will be converted into full rank matrix.
The matrix completion problems can be classified into three categories [5]:

1. Hermitian problems
2. Rank problems
3. Eigenvalue and singular value completion problems

[8] In matrix completion problem, Given a list of m users $\{u_1, u_2, \dots, u_m\}$ and n items $\{i_1, i_2, \dots, i_m\}$, the preferences of users toward the items can be represented as an incomplete $m \times n$ matrix A, where each entry either represents a certain rating or is unknown. The ratings in A can be explicit indications, such as scores given by the users in scales $1 - 5$ or ordinal favorability (e.g., strongly agree, agree, neutral, disagree, and strongly disagree). These ratings can also be implicit indications, e.g., item purchases, website/store visits, or link click-throughs. It is generally assumed a user rates a specific item only once. As a result, recommendations can be made by filling out the unknown entries and then ranking them according to the predicted values.
Denoting Λ as the complete set of N entries in A with known ratings, the general matrix completion problem is defined as finding a matrix R such that

$$R_{ui} = A_{ui},$$

for all entries $(u, i) \in \Lambda$. In addition, we denote $\bar{\Lambda}$ as the complement set to Λ, and $P_\Lambda(A)$ as an orthogonal projector onto Λ which is an $m \times n$ matrix with the known elements of A preserved and the unknown elements as 0s [1]. However, since the number of known entries is less than the overall number of entries, there exist infinitely many solutions. Nevertheless, it is commonly believed that only a few latent factors [10] influence how much a user likes an item. For example, studies show that the attributes of actor/actress, director, and decade contribute most to a user's preference to a movie. This relatively small number of influence factors compared to the total number of users or items in the rating matrix A provides a guiding framework to fill in the missing values and to select the correct

complete matrix. This corresponds to the low-rank assumption in matrix completion, i.e., the rating matrix A is low-rank or approximately low-rank.
Some applications [6] where the underlying matrix is modeled as a low-rank matrix are

1. Recommendation System
2. Phase Retrival
3. Localization of IoT newroks
4. Image compression and restoration
5. Massive multiple-input multiple-output (MIMO)
6. Milimeter wave (mmWave) channel estimation
7. Topological intereference Management
8. Mobile edge caching in fog radio access netwroks (Fog-RAN)

In the next section, we discuss one LRMC with the rank information in detail in section (3). The implementation of an algorithm is discussed in section (4) and finally, section (5) concludes this report. Low rank matrix completion algorithms (LRMC) can be classified into two categories:

1. LRMC without the rank information
2. LRMC with the rank information

2 Related work

3 Low rank Matrix Completion

In many applications such as localization in IoT networks, recommendation system, and image restoration, we encounter the situation where the rank of a desired matrix is known in advance. The low-rank matrix is expressed in terms of degree of freedom, in a matrix is much smaller than the total number of entries. Therefore, even though the number of observed entries is small, we still have a good chance to recover the whole matrix. There are a variety of scenarios where the number of observed entries of a matrix is tiny. In the recommendation systems, for example, users are recommended to submit the feedback in a form of rating number, e.g., 1 to 5 for the purchased product. However, users often do not want to leave a feedback and thus the rating matrix will have many missing entries.

When there is no restriction on the rank of a matrix, the problem to recover unknown entries of a matrix from partial observed entries is ill-posed. This is because any value can be assigned to unknown entries, which in turn means that there are infinite number of matrices that agree with the observed entries. As an example, let us consider the following 2×2 matrix with one unknown entry, say x

$$M = \begin{bmatrix} 1 & 5 \\ 2 & x \end{bmatrix} \tag{1}$$

If M is a full rank matrix, then $x \in \mathbb{R} \setminus \{10\}$ i.e., if the rank of M is two, then any value except 10 can be assigned to x. Whereas, if M is a low rank matrix (here the rank is 1), two columns are lineaarly dependent and hence $x = 10$. In this exmple, it is obvious to get x but for the higher dimensional matrix it may not be obviuos. So in that situation, low-rank constraint plays a pivotal role in recovering unknown entries of the matrix.

3.1 Basics of Low Rank Matrix Completion

we discuss the basic principle to recover a low-rank matrix from partial observations. Basically, the desired low-rank matrix M can be recovered by solving the rank minimization problem

$$\begin{aligned} &\min_X rank(X) \\ &subject\ to\ x_{ij} = m_{ij}, (i,j) \in \Omega \end{aligned} \tag{2}$$

where Ω is the index set of observed entries (e.g., $\{(1;1);(1;2);(2;1)\}$ in the example in (1)). One can alternatively express the problem using the sampling operator P_Ω. The sampling operation P_Ω of a matrix A is defined as

$$[P_\Omega(A)]_{ij} = \begin{cases} a_{ij} & if (i,j) \in \Omega \\ 0 & otherwise. \end{cases}$$

Using this operator, the problem (3) can be equivalently formulated as

$$\begin{aligned} &\min_X rank(X) \\ &subject\ to\ P_\Omega(X) = P_\Omega(M) \end{aligned} \tag{3}$$

In order to solve the rank minimization problem (3), we first assume that $rank(M) = 1$. Then any two columns of M are linearly dependent and thus we have the system of expressions

$$m_i = \alpha_{i,j} m_j$$

for some $\alpha_{i,j} \in \mathbb{R}$. If the system has no solution for the rank-one assumption, then we move to the next assumption of $rank(M) = 2$. In this case, we solve the new system of expressions

$$m_i = \alpha_{i,j} m_j + \alpha_{i,k} m_k$$

This procedure is repeated until the solution is found. Clearly, the combinatorial search strategy would not be feasible for most practical scenarios since it has an exponential complexity in the problem size. For example, when M is an $n \times n$ matrix, it can be shown that the number of the system expressions to be solved is $O(n2^n)$.

As a cost-effective alternative, various low-rank matrix completion (LRMC) algorithms have been proposed over the years.

3.2 LRMC Algorithm Using Rank Information

In many applications such as localization in IoT networks, recommendation system, and image restoration, we encounter the situation where the rank of a desired matrix is known in advance. In this situation, the LRMC problem can be formulated as a Frobenius norm minimization (FNM) problem:

$$\begin{aligned} &\min_X \frac{1}{2} \|P_\Omega(M) - P_\Omega(X)\|_F^2 \\ &subject\ to\ rank(X) \leq r. \end{aligned} \tag{4}$$

The FNM problem has two main advantages:

1. the problem is well-posed in the noisy scenario

2. the cost function is differentiable so that various gradient-based optimization techniques (e.g., gradient descent, conjugate gradient, Newton methods, and manifold optimization) can be used to solve the problem.

It has been shown that under suitable conditions of the sampling ratio $p = \frac{|\Omega|}{n_1 n_2}$ and the largest coherence μ_0 of M, the gradient-based algorithms globally converges to M with high probability [2]. Well-known FNM-based LRMC techniques include greedy techniques [4], alternating projection techniques [3], and optimization over Riemannian manifold [11].

3.3 Alternating Minimization Technique

Many of LRMC algorithms require the computation of (partial) Singular Value Decomposition (SVD) to obtain the singular values and vectors (expressed as $O(rn^2)$). As an effort to further reduce the computational burden of SVD, alternating minimization techniques have been proposed [3], [9], [12].

In this approach a low rank matrix $M \in \mathbb{R}^{n_1 \times n_2}$ of rank r can be factorized into tall and fat matrices, i.e., $M = XY$ where $X \in \mathbb{R}^{n_1 \times r}$ and $Y \in \mathbb{R}^{r \times n_2}$ $(r \ll n_1,\ n_2)$. That means to find out X and Y minimizing the residual (the difference between the original matrix and the estimate of it) on the sampling space. In other words, X and Y are recovered by solving

$$\min_{X,Y} \frac{1}{2} \|P_\Omega(M) - P_\Omega(XY)\|_F^2 \tag{5}$$

Power factorization, a simple alternating minimization algorithm, finds out the solution to (8) by updating X and Y alternately as [3]

$$X_{i+1} = \arg\min_X \|P_\Omega(M) - P_\Omega(XY_i)\|_F^2, \tag{6}$$

$$Y_{i+1} = \arg\min_Y \|P_\Omega(M) - P_\Omega(X_{i+1}Y)\|_F^2, \tag{7}$$

Alternating steepest descent (ASD) is another alternating method to find out the solution [9]. The key idea of ASD is to update X and Y by applying the steepest gradient descent method to the objective function

$$f(X,Y) = \min_{X,Y} \frac{1}{2} \|P_\Omega(M) - P_\Omega(XY)\|_F^2 \tag{8}$$

in (8). Specifically, ASD first computes the gradient of $f(X,Y)$ with respect to X and then updates X along the steepest gradient descent direction:

$$X_{i+1} = X_i - t_{x_i} \nabla f_{Y_i}(X_i) \tag{9}$$

where the gradient descent direction $\nabla f_{Y_i}(X_i)$ and stepsize t_{x_i} are given by

$$\nabla f_{Y_i}(X_i) = -(P_\Omega(M) - P_\Omega(X_i Y_i))Y_i^T \tag{10}$$

$$t_{x_i} = \frac{\|\nabla f_{Y_i}(X_i)\|_F^2}{\|P_\Omega(\nabla f_{Y_i}(X_i) Y_i)\|_F^2} \tag{11}$$

After updating X, ASD updates Y in a similar way:

$$Y_{i+1} = Y_i - t_{y_i} \nabla f_{X_{i+1}}(Y_i) \tag{12}$$

where

$$\nabla f_{X_{i+1}}(Y_i) = -X_{i+1}^T(P_\Omega(M) - P_\Omega(X_{i+1}Y_i)) \tag{13}$$

$$t_{y_i} = \frac{\|\nabla f_{X_{i+1}}(Y_i)\|_F^2}{\|P_\Omega(X_{i+1}\nabla f_{X_{i+1}}(Y_i))\|_F^2} \tag{14}$$

The low-rank matrix fitting (LMaFit) algorithm finds out the solution in a different way by solving [12]

$$\arg\min_{X,Y,Z}\{\|XY - Z\|_F^2 : P_\Omega(Z) = P_\Omega(M)\}. \tag{15}$$

With the arbitrary input of $X_0]in\mathbb{R}^{n_1\times r}$ and $Y_0 \in \mathbb{R}^{r\times n_2}$ and $Z_0 = P_\Omega(M)$, the variables X, Y and Z are updated in the $i-$th iteration as

$$X_{i+1} = \arg\min_X\{\|XY_i - Z_i\|_F^2 = Z_iY^\dagger \tag{16}$$

$$Y_{i+1} = \arg\min_Y\{\|X_iY - Z_i\|_F^2 = X_{i+1}^\dagger Z_i \tag{17}$$

$$Z_{i+1} = X_{i+1}Y_{i+1} + P_\Omega(M - X_{i+1}Y_{i+1}) \tag{18}$$

where $x^\dagger$ is Moore-Penrose pseudoinverse of matrix X.

It has been shown that alternating minimization techniques are simple to implement.

4 Implementation/Experiments

```
import numpy

def matrix_factorization(R, P, Q, K, steps=5000, alpha=0.0002, beta=0.02):
        Q = Q.T
        for step in range(steps):
                for i in range(len(R)):
                        for j in range(len(R[i])):
                                if R[i][j] > 0:
                                        eij = R[i][j] - numpy.dot(P[i,:],Q[:,j]
                                        for k in range(K):

                                                P[i][k] = P[i][k] + alpha *
                                                        (2 * eij * Q[k][j] -
                                                        beta * P[i][k])
                                                Q[k][j] = Q[k][j] + alpha *
                                                        (2 * eij * P[i][k] -
                                                        beta * Q[k][j])

                eR = numpy.dot(P,Q)
                e = 0
                for i in range(len(R)):
                        for j in range(len(R[i])):
                                if R[i][j] > 0:
                                        e = e + pow(R[i][j]
                                             − numpy.dot(P[i,:],Q[:,j]), 2)
                                             for k in range(K):
                                                     e = e + (beta/2) *
                                                            (pow(P[i][k],2) +
                                                            pow(Q[k][j],2))
                    if e < 0.001:
                            break
        return P, Q.T

R = [
[5,3,0,1],
[4,0,0,1],
[1,1,0,5],
[1,0,0,4],
[0,1,5,4],
]

R = numpy.array(R)

N = len(R)
M = len(R[0])
K = 2

P = numpy.random.rand(N,K)
Q = numpy.random.rand(M,K)

nP, nQ = matrix_factorization(R, P, Q, K)
nR = numpy.dot(nP, nQ.T)

#Printing Decomposed Matrices
print(P)
print(Q)


#Printing the final Output
QT=Q.T
final = [[0 for x in range(M)] for y in range(N)]

for i in range(len(P)):
        for j in range(len(QT[0])):
                for k in range(len(QT)):

# resulted matrix
                        final[i][j] += P[i][k] * QT[k][j]

print(final)
```

5 Discussion & Conclusion

Here we discussed one LRMC algorithm but the reader might observed that there are many algorithms for LRMC techniques so the naural question one might ask is what algorithm should one choose? Obviously, the choice should go on the complexity of the algorithm which we have not discussed here. But the reader may use the SDP-based NNM technique when the accuracy of a recovered matrix is critical.

When the dimension of a low-rank matrix increases and thus computational complexity increases significantly, so we want an algorithm such that the complexity scales has linear relation with the problem size.

References

[1] Emmanuel J Candès and Terence Tao. The power of convex relaxation: Near-optimal matrix completion. *IEEE Transactions on Information Theory*, 56(5):2053–2080, 2010.

[2] Rong Ge, Chi Jin, and Yi Zheng. No spurious local minima in nonconvex low rank problems: A unified geometric analysis. *arXiv preprint arXiv:1704.00708*, 2017.

[3] Justin P Haldar and Diego Hernando. Rank-constrained solutions to linear matrix equations using powerfactorization. *IEEE Signal Processing Letters*, 16(7):584–587, 2009.

[4] Kiryung Lee and Yoram Bresler. Admira: Atomic decomposition for minimum rank approximation. *IEEE Transactions on Information Theory*, 56(9):4402–4416, 2010.

[5] Sang-gu Lee and Han-guk Seol. A survey on the matrix completion problem. *Trends in Mathematics*, 4(1):38–43, 2001.

[6] Luong Trung Nguyen, Junhan Kim, and Byonghyo Shim. Low-rank matrix completion: A contemporary survey. *IEEE Access*, 7:94215–94237, 2019.

[7] Sewoong Oh. *Matrix completion: Fundamental limits and efficient algorithms*. PhD thesis, Stanford University, 2011.

[8] Andy Ramlatchan, Mengyun Yang, Quan Liu, Min Li, Jianxin Wang, and Yaohang Li. A survey of matrix completion methods for recommendation systems. *Big Data Mining and Analytics*, 1(4):308–323, 2018.

[9] Jared Tanner and Ke Wei. Low rank matrix completion by alternating steepest descent methods. *Applied and Computational Harmonic Analysis*, 40(2):417–429, 2016.

[10] Madeleine Udell, Corinne Horn, Reza Zadeh, and Stephen Boyd. Generalized low rank models. *arXiv preprint arXiv:1410.0342*, 2014.

[11] Bart Vandereycken. Low-rank matrix completion by riemannian optimization. *SIAM Journal on Optimization*, 23(2):1214–1236, 2013.

[12] Zaiwen Wen, Wotao Yin, and Yin Zhang. Solving a low-rank factorization model for matrix completion by a nonlinear successive over-relaxation algorithm. *Mathematical Programming Computation*, 4(4):333–361, 2012.