Name: Hongda Li Class: CSE 546 SPRING 2021 HW4B

B1: Matrix Completion and Recommendation System

Note: Yi Yun Dong, a student in 546 is my collaborator for this problem. Type of collaborations:

1. Me doing the math first, and thinking about the problem and then read Yiyun Dong's code to speed up the implementaitons of the algorithm. No direct copy is involved, I write my own code, but based on my understanding of the problem and Yiyun's code.

2. We discussed the math together and look for mistakes in each other's argument.

B1.a

Using the average of each user to predict new enries causes the MSE to just be the variance of the matrix.

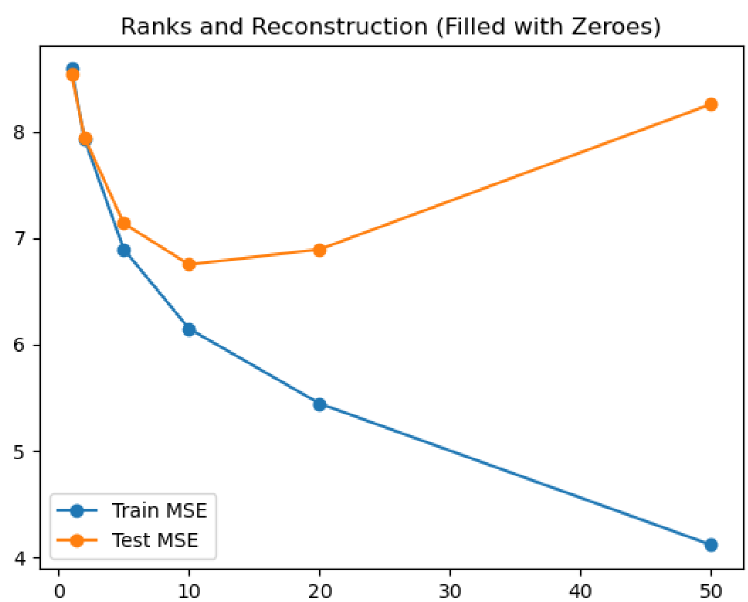
1. The average rating of movies are computed by: $\frac{\text{total rating of i'th movie}}{\text{number of user rated that movie}}$, and if that movie is not rated by any user, replace the average estimation to be the averate of all rated movies.

2. Then we just use the $\epsilon(\hat{R})_{\text{test}}$ to compute the error.

The error I have at the end for $\epsilon(\hat{R})_{\text{Test}}$ is: 1.0551305913743587. Refer to my code in section B1.code

B1.b

This is the graph I have.



Refer to my code in section B1.code

B1.c

Create the Masking matrix as the following:

$$M_{i,j} = \begin{cases} 1 & (i,j,R_{i,j}) \in \text{Train} \\ 0 & \text{else} \end{cases} \tag{B1.c.1}$$

Here we also assume that $u_i, v_i \in \mathbb{R}^d$

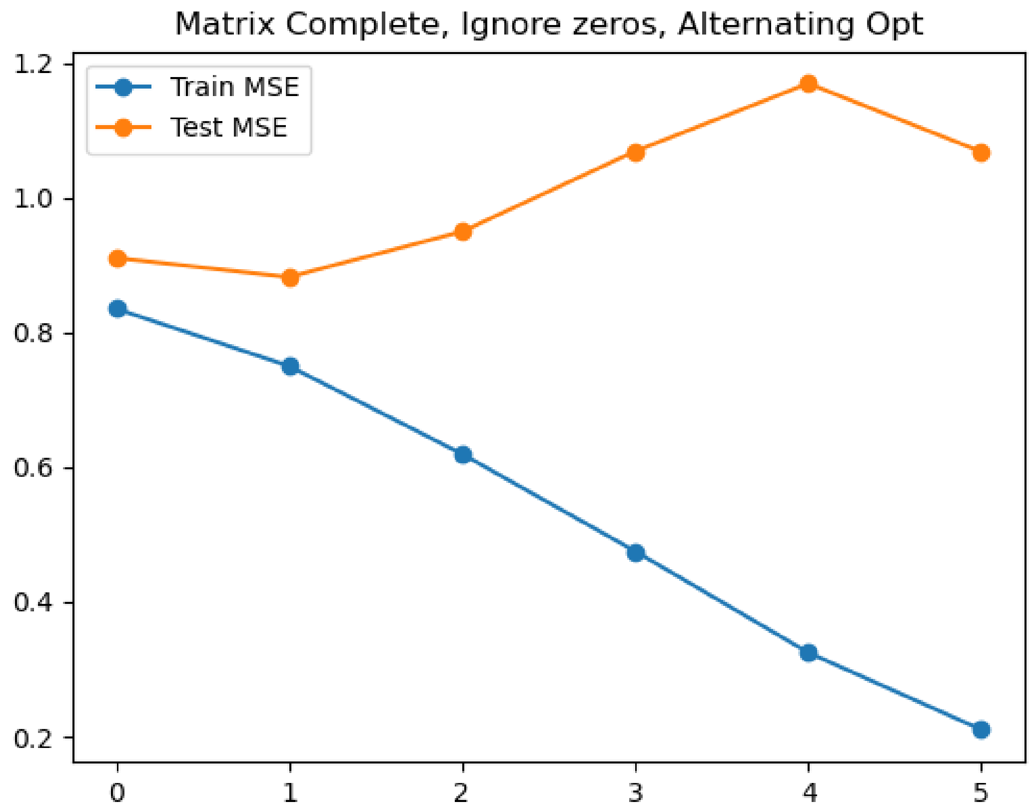
This matrix will mask out elements instances where given user j didn't rate movie i. Then, we can say the following with the cost function:

$$\begin{aligned}
\nabla_{u_k}\left[\sum_{(i,j)\in\text{Train}}(u_i^T v_j - R_{i,j})^2 + \lambda\sum_{i=1}^{d}\|u_i\|_2^2\right] &= \mathbf{0} \\
\nabla_{u_k}\left[\sum_{j\in\text{train}}(u_k^T v_j - R_{k,j})^2 + \lambda\|u_k\|_2^2\right] &= \mathbf{0} \\
\nabla_{u_k}\left[\sum_{j=1}^{n}(M_{k,j}u_k^T v_j - R_{k,j})^2\right] + 2\lambda u_k &= \mathbf{0} \\
\left(\sum_{j=1}^{n}(M_{k,j}u_k^T v_j - R_{k,j})M_{k,j}v_j\right) + 2\lambda u_k &= \mathbf{0} \\
\left(\sum_{j=1}^{n}M_{k,j}^2 v_j v_j^T u_k - M_{k,j}R_{k,j}v_j\right) + \lambda u_k &= \mathbf{0} \\
\left(\sum_{j=1}^{n}M_{k,j}v_j v_j^T + \lambda I\right)u_k &= \sum_{j=1}^{n}M_{k,j}R_{k,j}v_j \\
u_k &= \left(\sum_{j=1}^{n}M_{k,j}v_j v_j^T + \lambda I\right)^{-1}\sum_{j=1}^{n}M_{k,j}R_{k,j}v_j
\end{aligned} \tag{B1.c.2}$$

And this is the closed form solution of solving for one of the vector in the matrix U, similarly we can get the closed form solution for one of the vector in V:

$$\begin{aligned}
\nabla_{v_k}\left[\sum_{(i,j)\in\text{Train}}(u_i^T v_j - R_{i,j})^2 + \lambda\sum_{i=1}^{d}\|u_i\|_2^2\right] &= \mathbf{0} \\
\nabla_{v_k}\left[\sum_{i=1}^{m}(M_{i,k}u_i^T v_k - Ri,j)^2 + \lambda\|v_k\|_2^2\right] &= 0 \\
\sum_{i=1}^{m}(M_{i,k}^2 u_i^T v_k u_i - R_{i,k}M_{i,k}u_i) + \lambda v_k &= 0 \\
\sum_{i=1}^{m}(M_{i,k}u_i u_i^T v_k) + \lambda I v_k &= \sum_{i=1}^{m}M_{i,k}R_{i,k}u_i \\
\left(\sum_{i=1}^{m}(M_{i,k}u_i u_i^T) + \lambda I\right)v_k &= \sum_{i=1}^{m}M_{i,k}R_{i,k}u_i \\
v_k &= \left(\sum_{i=1}^{m}(M_{i,k}u_i u_i^T) + \lambda I\right)^{-1}\sum_{i=1}^{m}R_{i,k}M_{i,k}u_i
\end{aligned} \tag{B1.c.3}$$

Implementing them as python code for the alternating optimization is not easy, and I think I did it. And this is the plot I have:



As we can seem it didn't generalize too well, but it seems like it works well for the training set. This is very interesting.

B1.d

Nope I didn't do it. Maybe I will do it in the future, after the class ends, as a legit resume project or something... You know.. something like that

B1.e

Bleh...

B1.code

file: "matrix_completion.py"

```
import torch
import torch.nn as nn
import numpy as np
from scipy import sparse
from scipy.sparse.linalg import svds
import matplotlib.pyplot as plt
from math import isnan
import csv
zeros = np.zeros
pinv = np.linalg.pinv
DATA_PATH = "./data/ml-100k/u.data"
```

```
def MoiveAvgRating(sparseDataMatrix:sparse.coo_matrix):
    DenseMatrix = sparseDataMatrix.toarray().astype(np.float)
    # number of rated movie
    RatedMovies = np.sum(np.array(DenseMatrix != 0, dtype=np.float),
                         axis=1, keepdims=True)
    TotalRatings = np.sum(DenseMatrix, axis=1, keepdims=True)
    AverageRatings = TotalRatings /RatedMovies  # average rating for each movie
    GlobalAvg = np.sum(TotalRatings.reshape(-1))/np.sum(RatedMovies.reshape(-1))
    # take care of movie nobody viewed.
    return np.nan_to_num(AverageRatings, nan=GlobalAvg)


if "DATA" not in dir():
    DATA = []
    with open(DATA_PATH) as csvfile:
        spamreader = csv.reader(csvfile, delimiter='\t')
        for row in spamreader:
            DATA.append([int(row[0]) - 1, int(row[1]) - 1, int(row[2])])
    DATA = np.array(DATA)
    NUM_OBSERVATIONS = len(DATA)           # num_observations = 100,000
    NUM_USERS = max(DATA[:, 0]) + 1        # num_users = 943, indexed 0,...,942
    NUM_ITEMS = max(DATA[:, 1]) + 1        # num_items = 1682 indexed 0,...,1681
    np.random.seed(1)
    NUM_TRAIN = int(0.8*NUM_OBSERVATIONS)
    perm = np.random.permutation(DATA.shape[0])
    TRAIN = DATA[perm[0:NUM_TRAIN], :]
    TEST = DATA[perm[NUM_TRAIN::], :]
    del perm
    TRAIN_SPR = sparse.coo_matrix(
        (TRAIN[:, 2], (TRAIN[:,1], TRAIN[:, 0])), (NUM_ITEMS, NUM_USERS)
    )
    TEST_SPR = sparse.coo_matrix(
        (TEST[:, 2], (TEST[:,1], TEST[:, 0])), (NUM_ITEMS, NUM_USERS)
    )
    TRAIN_AVG = MoiveAvgRating(TRAIN_SPR)
    print("DATA HAS BEEN LOADED. ")


# ========================== List of Helper Functions ==========================

def Ts():
    from datetime import datetime
    SysTime = datetime.now()
    TimeStamp = SysTime.strftime("%H-%M-%S")
    return TimeStamp


def mkdir(dir):
    from pathlib import Path
    Path(dir).mkdir(parents=True, exist_ok=True)


def log(fname:str, content:str, dir):
    mkdir(dir)
    TimeStamp = Ts()
    with open(f"{dir}{TimeStamp}-{fname}.txt","w+") as f:
        f.write(content)

# ==============================================================================

def Epsilon(approx, train=True):
    Sparse = TRAIN_SPR if train else TEST_SPR
    DiffSum = 0
    for Idx, (II, JJ, Rating) in enumerate(
            zip(Sparse.row, Sparse.col, Sparse.data)
    ):
```

```
        DiffSum += (Rating - approx[II, JJ]) ** 2
    return DiffSum / Idx


class AlternatingMinimization:

    def __init__(this,
                 dataMatrix:np.ndarray,
                 d:int,
                 sigma,
                 regularizer,
                 tol=1e-2):
        assert dataMatrix.ndim == 2
        m, n = dataMatrix.shape
        this.m, this.n = m, n
        this.R = dataMatrix
        this.Sigma = sigma
        this.Lambda = regularizer
        this.Tol = tol
        this.Rank = d
        this.V = sigma*np.random.randn(d, n)
        this.U = sigma*np.random.randn(d, m)
        this.I = np.eye(d)
        this.M = np.array(dataMatrix != 0, dtype=np.float)

    def UOpt(this):
        L = this.Lambda
        I = this.I
        R = this.R
        V = this.V
        U = this.U
        M = this.M
        for K in range(this.m):
            U[:, K:K+1] = pinv(V@(M[K:K + 1,:].T*V.T) + L*I)@(V@R[K:K+1, :].T)

    def VOpt(this):
        L = this.Lambda
        I = this.I
        R = this.R
        V = this.V
        U = this.U
        M = this.M
        for K in range(this.n):
            V[:, K:K+1] = pinv(U@(M[:, K:K+1]*U.T) + L*I)@(U@R[:, K:K+1])

    def TrainLoss(this):
        return this.Loss()

    def TestLoss(this):
        return this.Loss(False)

    def Loss(this, train=True):
        Approx = this.U.T@this.V
        return Epsilon(Approx, train=train)


def PartA():
    DiffSum = 0
    for Idx, (II, _, Rating) in enumerate(
            zip(TEST_SPR.row, TEST_SPR.col, TEST_SPR.data)
        ):
        DiffSum += (Rating - TRAIN_AVG[II, 0])**2
    return DiffSum/Idx


def PartB(ranks=[1, 2, 5, 10, 20, 50]):
    Ranks = sorted(ranks + [0])
    RTilde = TRAIN_SPR.asfptype()  # Filled with zeros.
```

```
    U, Sigma, VTransposed = svds(RTilde, k=942)
    U, Sigma, VTransposed = U[:, ::-1], Sigma[::-1], VTransposed[::-1]
    Approximation = np.zeros(RTilde.shape)
    MSETrain, MSETest = [], []

    for RankStart, RankEnd in zip(Ranks[: -1], Ranks[1:]):
        Approximation += U[:, RankStart: RankEnd]\
            @\
            np.diag(Sigma[RankStart: RankEnd])\
            @\
            VTransposed[RankStart: RankEnd]
        MSETrain.append(Epsilon(Approximation, True))
        MSETest.append(Epsilon(Approximation, False))
    return ranks, MSETrain, MSETest

def MatrixComplete(d, sigma, regularizer):
    Instance = AlternatingMinimization(
        TRAIN_SPR.asfptype().toarray(),
        d=d,
        sigma=sigma,
        regularizer=regularizer
    )
    for II in range(100):
        Loss = Instance.Loss()
        Instance.UOpt()
        Instance.VOpt()
        print(Loss)
        if Loss - Instance.Loss() < 1e-2:
            TestLoss = Instance.TestLoss()
            break
    return Loss, TestLoss


def main():
    FolderPath = "./B1"
    mkdir(FolderPath)
    def ParA():
        PartAError = PartA()
        with open(f"{FolderPath}/part-a.txt", "w+") as f:
            f.write(f"For␣part␣(a),␣the␣error␣on␣test␣set␣is:␣{PartAError}")
        print(f"ParA␣Done")
    # ParA()
    def ParB():
        Ranks, TrainErr, TestErr = PartB()
        print(f"Train␣Errors␣{TrainErr,␣TestErr}")
        plt.plot(Ranks, TrainErr, "-o")
        plt.plot(Ranks, TestErr, "-o")
        plt.legend(["Train␣MSE", "Test␣MSE"])
        plt.title("Ranks␣and␣Reconstruction␣(Filled␣with␣Zeroes)")
        plt.savefig(f"{FolderPath}/{Ts()}-b1-b.png")
        plt.show()
        plt.clf()
    # ParB()
    def ParC():
        Ranks = [1, 2, 5, 10, 20, 50]
        TrainLosses, TestLosses = [], []
        for Rank in Ranks:
            TrainLoss, TestLoss = MatrixComplete(Rank, 1, Rank/10)
            TrainLosses.append(TrainLoss)
            TestLosses.append(TestLoss)
        plt.plot(TrainLosses, "-o")
        plt.plot(TestLosses, "-o")
        plt.legend(["Train␣MSE", "Test␣MSE"])
        plt.title("Matrix␣Complete,␣Ignore␣zeros,␣Alternating␣Opt")
        plt.savefig(f"{FolderPath}/{Ts()}-b1-c.png")
        plt.show()
        plt.clf()
    ParC()
```

```python
if __name__ == "__main__":
    import os
    print(os.getcwd())
    print(os.curdir)
    main()
```