CS 541 Homework 4

Stephen Szemis

December 5, 2020

Part 1: Data Set
temp

Part 2: Learning

1. **Derive the gradients** We start with out original equation

$$\min_{U,V} F(U,V) := \frac{1}{2} \sum_{(i,j)\in\Omega_1} (M_{ij} - u_i v_j^T)^2 + \frac{\lambda}{2}\big(\parallel U \parallel_F^2 + \parallel V \parallel_F^2 \big)$$

We will look at each part of our derivative individually, as it can be complex to try to derive the answer in one big step. Importantly the derivative of sums is the sum of derivatives. So we can look at each element of our Summation, and try to see what the final partial might be.

$$z_{ij} = \sum_{(i,j)\in\Omega_1} (M_{ij} - u_i v_j^T)^2 \text{ for some i and j}$$

We use the chain rule to get the "partial" solution...

$$\frac{\partial z_{ij}}{\partial U} = -2(M_{ij} - u_i v_j^T)V$$

$$\frac{\partial z_{ij}}{\partial V} = -2(M_{ij} - u_i v_j^T)U$$

The other part of our original equation is easier to solve for.

$$y = \frac{\lambda}{2}\big(\parallel U \parallel_F^2 + \parallel V \parallel_F^2 \big)$$

Using the rules for the derivative of a Frobenius norm and chain rule we find...

$$\frac{\partial y}{\partial V} = \lambda U$$

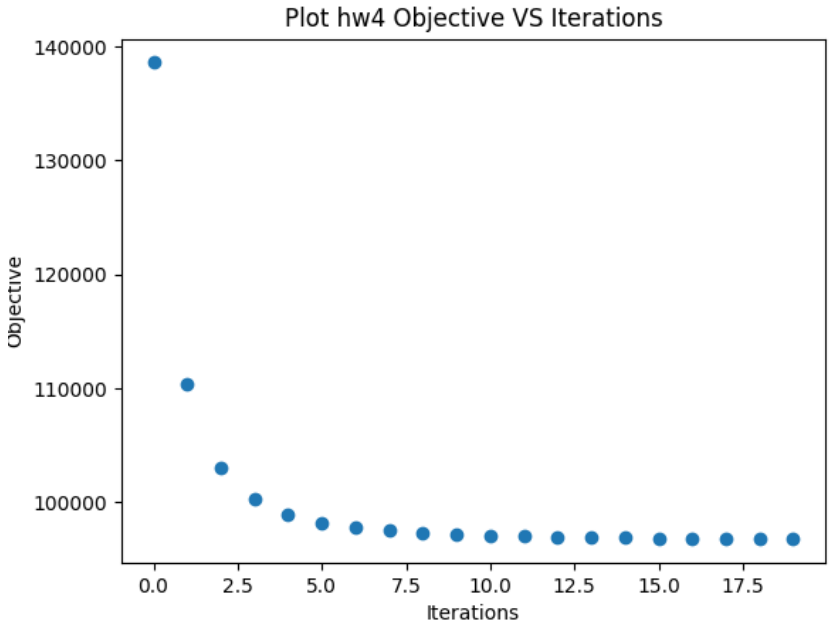
$$\frac{\partial y}{\partial U} = \lambda V$$

Finally putting it all together we get.

$$\frac{\partial F(U,V)}{\partial U} = V \sum_{(i,j)\in\Omega_1} (M_{ij} - u_i v_j^T) + \lambda U$$

$$\frac{\partial F(U,V)}{\partial U} = U \sum_{(i,j)\in\Omega_1} (M_{ij} - u_i v_j^T) + \lambda V$$

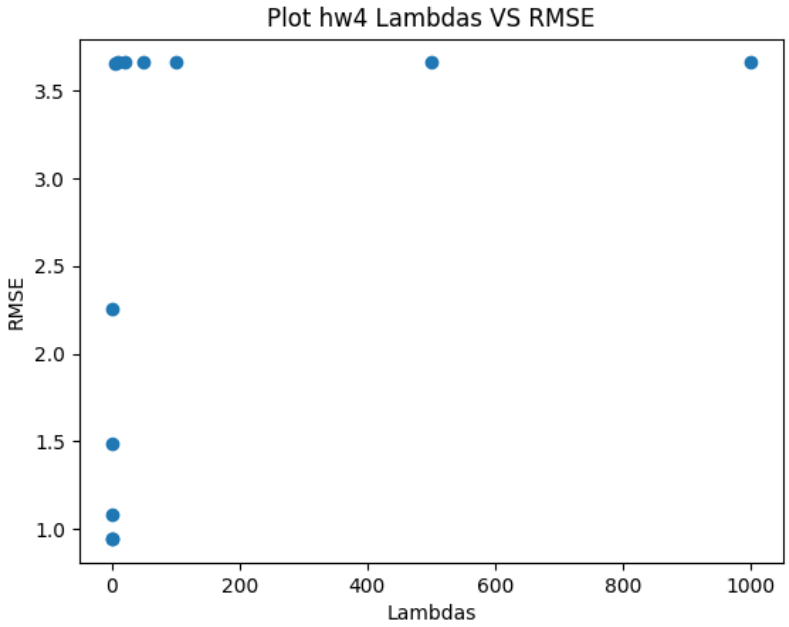
2. Lambda is 1 You can explore the code on the final pages of this report to see how this is implemented. Our update step is obviously just using the gradients above and updating each "row" of our U and V matrixes. The learning rate was a serious challenge for this entire assignment for me. I had a lot of trouble with overflow errors and had to more or less choose learning rates to stop this. Clearly I was finding some bad behavior from my code. The learning rates in my final version run and give "fine" results, but I have a feeling I'm missing something about the relation between lambda and the learning rate. My graphs for part 2 are below.



As you can see, we get a very nice move towards a minimum.

Part 3: Evalutation

The graph is below, as discuss in the last problem, I had issue with overflow errors and therefore I don't think my data is a very good reflection of how lambda changes behavior. Even so, this is what I ended up with.



As you can see, lambda's that are lower have a much better error rate, but this might be due to the number of iterations being low, or my learning rate being wrong.

My code is below:

```
import numpy as np
import random as rand
import matplotlib.pyplot as plt
import csv
from numpy.linalg import matrix_rank

# Author: Stephen Szemis
# Date: December 5, 2020
# I pledge my honor that I have abided by the Steven's Honor system.
np.seterr(all='raise')
n = 0
p = 0
mov_index = []
dim = 5
```

```
# Part 1
with open('ml-latest-small/ratings.csv', 'r') as temp:
    ratings_csv = csv.DictReader(temp)
    ratings = list(ratings_csv)
    for row in ratings:
        if int(row['userId']) > n:
            n = int(row['userId'])
        if int(row['movieId']) not in mov_index:
            mov_index.append(int(row['movieId']))
    p = len(mov_index)

    # Create M matrix
    M = np.zeros((n,p))
    # Create test
    M_test = np.zeros((n,p))

    for row in ratings:
        if rand.random() < 0.9:
            t = mov_index.index(int(row['movieId']))
            M[int(row['userId']) - 1][t] = float(row['rating'])
        else:
            t = mov_index.index(int(row['movieId']))
            M_test[int(row['userId']) - 1][t] = float(row['rating'])

print('Number of users:', n)
print('Number of movies:', p)

def objective(lbda, U, V):
    omega = np.transpose(np.nonzero(M))
    summation = 0
    for i, j in omega:
        summation += (M[i][j] - np.dot(U[i], np.transpose(V[j]))) ** 2
    summation /= 2
    return summation + ((lbda / 2) * ((np.linalg.norm(U) ** 2) + (np.linalg.norm(V

def train(lr, lbda, num):
    U = np.random.rand(n, dim)
    V = np.random.rand(p, dim)
```

```
    obj = []
    users, items = M.nonzero()
    for epoch in range(num):
        obj.append(objective(lbda, U, V))
        print('Iteration: ' + str(epoch) + ' Objective: ' + str(obj[-1]))
        for i, j in zip(users, items):
            error = M[i, j] - np.dot(U[i], V[j].T)
            U[i] = U[i] + (lr * ((error * V[j]) - (lbda * U[i])))
            V[j] = V[j]  + (lr * ((error * U[i]) - (lbda * V[j])))
    return obj, U, V

def rmse(X):
    omega = np.transpose(np.nonzero(M_test))
    error = 0
    for i, j in omega:
        error += (M_test[i][j] - X[i][j]) ** 2
    error = np.sqrt((1 / len(omega)) * error)
    return error

def evaluate():
    lbda = 1

    # Part 2-3
    objective, U, V = train(0.1, 1, 20)

    # Graph data
    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.scatter(range(len(objective)), objective)#, s=10, c='b', marker='s')

    # Produce Graph
    ax.set_title("Plot hw4 Objective VS Iterations")
    ax.set_xlabel('Iterations')
    ax.set_ylabel('Objective')
    fig.savefig('hw4_part2.png')

    # Part 3-1
```

```
    X = np.dot(U, V.T)
    rmse1 = rmse(X)
    print('RMSE of lambda 1:', rmse1)

    lambdas = [0.001, 0.01, 0.1, 0.5, 2, 5, 10, 20, 50, 100, 500, 1000]
    R = []
    for l in lambdas:
        if l < 1:
            _, U, V = train(0.1 * l, l, 20)
        else:
            _, U, V = train(0.2/l, l, 20)
        X = np.dot(U, V.T)
        r = rmse(X)
        print('RMSE of lambda ' + str(l) + ' : ' + str(r))
        R.append(r)

    fig = plt.figure()
    ax = fig.add_subplot(111)
    ax.scatter(lambdas, R)#, s=10, c='b', marker='s')

    # Produce Graph
    ax.set_title("Plot hw4 Lambdas VS RMSE")
    ax.set_xlabel('Lambdas')
    ax.set_ylabel('RMSE')
    fig.savefig('hw4_part3.png')

    # print('RMSE of lambda 1:', rmse1)

evaluate()
```