Name: Yiqun Peng

1. Data preprocessing

```python
""" Read file """
origin_file = []
n = 0
p = 0
with open('ratings.csv') as myFile:
    lines =  csv.reader(myFile)
    start = True
    for line in lines:
        if(start):
            start = False
        else:
            uid = int(line[0])
            mid = int(line[1])
            if(uid>n):
                n = uid
            if(mid>p):
                p = mid
            insert = [uid, mid, float(line[2])]
            origin_file.append(insert)
```

2. Generate matrix M

```python
""" Generate matrix M """
M = np.zeros((n+1, p+1))
# print(M.shape)
omega = []
for record in origin_file:
    M[record[0]][record[1]] = record[2]
    omega.append([record[0],record[1],record[2]])
print(M)
print("Shape: ", M.shape)
print()
# print(omega)
```

3. Divide the training set and the test set

```python
omega_test_index = np.array(getRandomIndex(len(omega)-1, round(0.1*len(omega)
omrga_train_index = np.delete(np.arange(len(omega)-1), omega_test_index)
M_test = np.zeros((n+1, p+1))
M_train = np.zeros((n + 1, p + 1))
size = 0
for i in range(0,len(omega)):
    if i in omega_test_index:
        M_test[omega[i][0]][omega[i][1]] = omega[i][2]
    if i in omrga_train_index:
        M_train[omega[i][0]][omega[i][1]] = omega[i][2]
print("Number of nozero entries in test: ", np.count_nonzero(M_test))
print("Number of nozero entries in training: ", np.count_nonzero(M_train))
```

4. Derive the gradient

$$\frac{\partial F(U,V)}{\partial U} = \left[\frac{F(U,V)}{\partial u_1}, \frac{F(U,V)}{\partial u_2}, \dots, \frac{F(U,V)}{\partial u_n}\right]$$

$\because$ For $$\frac{\partial F(U,V)}{\partial u_1} = \frac{1}{2}\sum_{(1,j)}(M_{1j} - u_1 v_j^T)^2 + \cancel{\frac{1}{2}\sum_{\substack{(i,j)\\ i\neq 1}}(M_{ij} - u_i v_j)^2} + \frac{1}{2}\lambda\|u_1\|_2^2 + \cancel{\frac{1}{2}\lambda\sum_{i=2}^{M}\|u_i\|_2^2} + \cancel{\frac{\lambda}{2}\|V\|_F^2}$$

$$= \lambda u_1 + \sum_{(1,j)}(M_{1j} - u_1 v_j^T)\cdot(-v_j)$$

$\therefore$ For $$\frac{\partial F(U,V)}{\partial u_n} = \lambda u_n + \sum_{(n,j)}(M_{nj} - u_n v_j^T)\cdot(-v_j)$$

In the same way, we get:

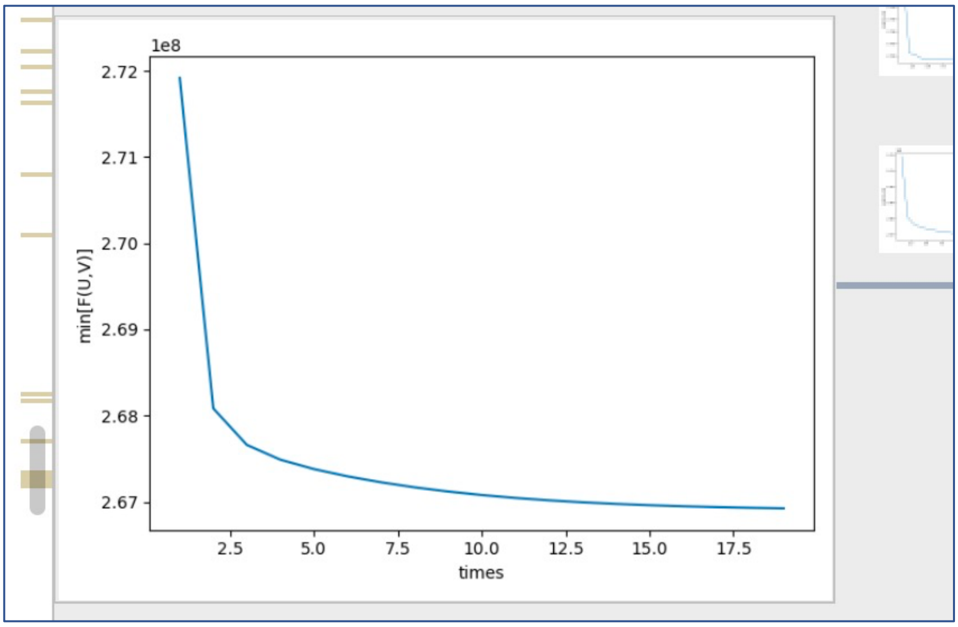
$$\frac{\partial F(U,V)}{\partial V} = \lambda v_n + \sum_{(i,n)}(M_{in} - u_i v_n^T)\cdot(-u_i)$$

5. Suppose λ= 1, Learning Rate = 0.05, Run GD.

```python
eta = 0.05
time = 20
objectives = []
times = []
for t in range(1, time):
    for i in range(0, n+1):
        for j in range(0, p+1):
            if M_train[i,j] > 0:
                first_in = M_train[i,j]-np.dot(u[i,:],np.transpose(v[j,:]))
                for rr in range(r):
                    first_u = first_in * (-1) * v[j][rr]
                    gradient_u = first_u + lamda * u[i][rr]
                    first_v = first_in * (-1) * u[i][rr]
                    gradient_v = first_v + lamda * v[j][rr]
                    u[i][rr] = u[i][rr] - eta * gradient_u
                    v[j][rr] = v[j][rr] - eta * gradient_v
```

6. Plot the objective value against the number of iterations

```
    uF = np.linalg.norm(u) ** 2
    vF = np.linalg.norm(v) ** 2
    print("lamda: ",lamda)
    nom = np.linalg.norm(M_train-np.dot(u, np.transpose(v))) ** 2
    print("nom: ",nom)
    min = 1/2 * nom ** 2 + lamda/2 * (uF + vF)
    print("min: ",min)
    times.append(t)
    objectives.append(min)
plt.figure()
plt.plot(times, objectives)
plt.xlabel('Times')
plt.ylabel('Min[F(U,V)]')
plt.show()
```



When the learning rate is 0.05 and λ is 1, it iterates about 20 times, using gradient descends to the minimum value and gradually converges.

7. Record the RMSE for the choice λ= 1.

```python
""" Evaluation """
RMSE = (np.linalg.norm(M_test - np.dot(u, np.transpose(v)))**2
        / omega_test_index.size) ** (1/2)
RMSEs.append(RMSE)
print("RMSE: ",RMSE)
```

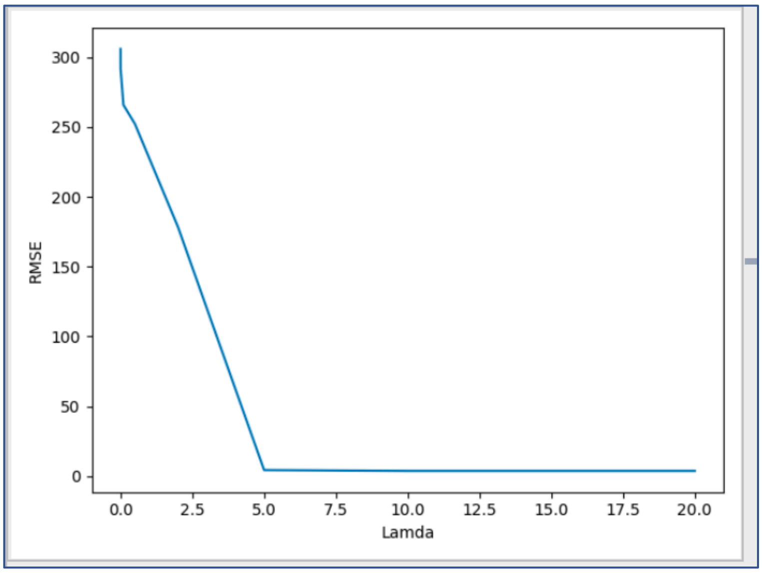
RMSE = 230.0960763102061

8. Pick λ from {10−6, 10−3, 0.1, 0.5, 2,5, 10, 20, 50, 100, 500, 1000}.

   For each value, learn and evaluate the model.

   RMSE = {305.96, 291.98, 265.86, 252.28, 178.47, 4.26, 3.65, 3.65, null, null, null}

   Plot RMSE against λ.



As λ becomes larger, the VALUE of RMSE decreases and becomes stable. However, when λ is greater than 50, the gradient descent reports an overflow warning and cannot successfully run out of the value of RMSE. Therefore, we cannot choose λ too large or too small.