hw3

November 21, 2021

[1]:

```
from IPython.display import Image
from IPython.core.display import HTML
Image(url= "hw3.png")
```

[1]:

```
<IPython.core.display.Image object>
```

[2]:

```
import gzip
from collections import defaultdict
from sklearn import linear_model
import csv
```

[3]:

```
def readGz(path):
    for l in gzip.open(path, 'rt'):
        yield eval(l)

def readCSV(path):
    f = gzip.open(path, 'rt')
    c = csv.reader(f)
    header = next(c)
    #print(header)
    for l in c:
        d = dict(zip(header,l))
        yield d['user_id'],d['recipe_id'],d
```

[4]:

```
Image(url= "problem1.png")
```

[4]:

```
<IPython.core.display.Image object>
```

[5]:

```
data = []
user_per_recipe = defaultdict(set)
recipe_per_user = defaultdict(set)

for user,recipe,d in readCSV("trainInteractions.csv.gz"):
    data.append([user,recipe])
    recipe_per_user[user].add(recipe)
    user_per_recipe[recipe].add(user)

train = data[:400000]
```

```
valid = data[400000:]
```

[6]:

```
print(len(valid))
#print(ratings)
```

```
100000
```

[7]:

```
import random #this block takes so long to run
recipe_set = set(recipe_per_user.keys())
cnt = 0
valid_add = []
for user,recipe in valid :
    #print(user, recipe)
    if cnt%20000==0:
        print(cnt)
    cnt+=1
    false_set = recipe_set - recipe_per_user[user]
    recipe_false = random.sample(list(false_set), 1)
    #print(recipe_false)
    valid_add.append([user, recipe_false[0]])
valid += valid_add
print(len(valid))
```

```
0
20000
40000
60000
80000
200000
```

[13]:

```
### Would-cook baseline: just rank which recipes are popular and which are not,␣
 ↪and return '1' if a recipe is among the top-ranked

recipeCount = defaultdict(int)
totalCooked = 0

for user,recipe,_ in readCSV("trainInteractions.csv.gz"):
  recipeCount[recipe] += 1
  totalCooked += 1

mostPopular = [(recipeCount[x], x) for x in recipeCount]
mostPopular.sort()
mostPopular.reverse()
# high to low
#print(mostPopular)

return1 = set()
count = 0
```

```
for ic, i in mostPopular:
  count += ic
  return1.add(i)
  if count > totalCooked/2: break

y_pred = []
cnt=0
for user,recipe in valid :
    #print(recipe)
    if cnt%20000==0:
        print(cnt)
    cnt+=1
    if recipe in return1:
        y_pred.append(1)
    else :
        y_pred.append(0)

y_valid = [1]*100000+[0]*100000

TP = sum([(p and l) for (p,l) in zip(y_pred, y_valid)])
FP = sum([(p and not l) for (p,l) in zip(y_pred, y_valid)])
TN = sum([(not p and not l) for (p,l) in zip(y_pred, y_valid)])
FN = sum([(not p and l) for (p,l) in zip(y_pred, y_valid)])
ACC = (TP + TN) / (TP + FP + TN + FN)
print(ACC)
```

```
0
20000
40000
60000
80000
100000
120000
140000
160000
180000
0.69627
```

Ans for Problem1 :

a. accuracy of the baseline model on validation set = 0.696805

[14]:
```
Image(url= "problem2.png")
```

[14]:
```
<IPython.core.display.Image object>
```

[15]:
```
def predict_by_threshold(percentile):
    return1 = set()
    count = 0
    for ic, i in mostPopular:
      count += ic
      return1.add(i)
      if count > totalCooked*percentile/100: break
    y_pred = []
    for u,i in valid :
        if i in return1:
            y_pred.append(1)
        else :
            y_pred.append(0)
    TP = sum([(p and l) for (p,l) in zip(y_pred, y_valid)])
    FP = sum([(p and not l) for (p,l) in zip(y_pred, y_valid)])
    TN = sum([(not p and not l) for (p,l) in zip(y_pred, y_valid)])
    FN = sum([(not p and l) for (p,l) in zip(y_pred, y_valid)])
    ACC = (TP + TN) / (TP + FP + TN + FN)
    print("percentile : ",percentile,", ACC = ",ACC)

for percentile in range(10,100,10):
    predict_by_threshold(percentile)
```

```
percentile :  10 , ACC =  0.548565
percentile :  20 , ACC =  0.59391
percentile :  30 , ACC =  0.634985
percentile :  40 , ACC =  0.66807
percentile :  50 , ACC =  0.69627
percentile :  60 , ACC =  0.723925
percentile :  70 , ACC =  0.754945
percentile :  80 , ACC =  0.783615
percentile :  90 , ACC =  0.8065
```

Ans for Problem2 :

a. accuracy of the best threshold model on validation set = 0.80843 with threshold = 90

[16]:
```
Image(url= "problem3.png")
```

[16]:
```
<IPython.core.display.Image object>
```

[17]:
```
# Jaccard similarity interchanging users and items
def Jaccard(s1, s2):
    numer = len(s1.intersection(s2))
    denom = len(s1.union(s2))
    if denom == 0:
        return 0
    return numer / denom

# prediction based on Jaccard Similarity
def predict_by_jaccard(threshold):
    y_pred = []
    y_valid = [1]*100000+[0]*100000
    for u,i in valid :
        items = recipe_per_user[u]
        s1 = user_per_recipe[i]
        jaccard_sim = [0]
        for g in items-{i} :
            s2 = user_per_recipe[g]
            jaccard_sim.append(Jaccard(s1, s2))
        #print(jaccard_sim)
        if max(jaccard_sim)> threshold/100:
            y_pred.append(1)
        else:
            y_pred.append(0)

    TP = sum([(p and l) for (p,l) in zip(y_pred, y_valid)])
    FP = sum([(p and not l) for (p,l) in zip(y_pred, y_valid)])
    TN = sum([(not p and not l) for (p,l) in zip(y_pred, y_valid)])
    FN = sum([(not p and l) for (p,l) in zip(y_pred, y_valid)])
    ACC = (TP + TN) / (TP + FP + TN + FN)
    print("threshold : ",threshold,", ACC = ",ACC)

for threshold in range(10,100,10):
    predict_by_jaccard(threshold)
```

```
threshold :  10 , ACC =  0.8077
threshold :  20 , ACC =  0.75093
threshold :  30 , ACC =  0.70707
threshold :  40 , ACC =  0.655665
threshold :  50 , ACC =  0.595455
threshold :  60 , ACC =  0.59426
threshold :  70 , ACC =  0.58028
threshold :  80 , ACC =  0.57944
threshold :  90 , ACC =  0.57944
```

Ans for Problem3 :

a. accuracy of the best jaccard model on validation set = 0.71121 with threshold = 0.1

[18]:
```
Image(url= "problem4.png")
```

[18]:
```
<IPython.core.display.Image object>
```

[19]:
```
def predict_by_combine(percentile, threshold):
    return1 = set()
    count = 0
    for ic, i in mostPopular:
      count += ic
      return1.add(i)
      if count > totalCooked*percentile/100: break
    y_pred = []
    for u,i in valid :
        if i in return1:
            y_pred.append(1)
        else :
            y_pred.append(0)

    TP = sum([(p and l) for (p,l) in zip(y_pred, y_valid)])
    FP = sum([(p and not l) for (p,l) in zip(y_pred, y_valid)])
    TN = sum([(not p and not l) for (p,l) in zip(y_pred, y_valid)])
    FN = sum([(not p and l) for (p,l) in zip(y_pred, y_valid)])
    ACC = (TP + TN) / (TP + FP + TN + FN)
    print("percentile : ",percentile,", ACC = ",ACC)
```

Ans for Problem4 :

a.

[20]:
```
Image(url= "problem5.png")
```

[20]:
```
<IPython.core.display.Image object>
```

[21]:
```
# train the model on train + validation and predict
def write_prediction_most_popular(threshold):
    predictions = open("predictions_Made_popular_%d.txt" % threshold,'w')
    return1 = set()
    count = 0
    for ic, i in mostPopular:
        count += ic
        return1.add(i)
        if count > totalCooked*threshold/100: break

    for l in open("stub_Made.txt"):
      if l.startswith("user_id"):
        #header
        predictions.write(l)
        continue
      u,i = l.strip().split('-')
      if i in return1:
        predictions.write(u + '-' + i + ",1\n")
      else:
        predictions.write(u + '-' + i + ",0\n")

    predictions.close()

for threshold in [50,60,70,80,90]:
    write_prediction_most_popular(threshold)
```

[22]:
```
# train the model on train + validation and predict
def write_prediction_jaccard(threshold):
    predictions = open("predictions_Made_jaccard_%d.txt"%threshold, 'w')

    for l in open("stub_Made.txt"):
      if l.startswith("user_id"):
        #header
        predictions.write(l)
        continue
      u,i = l.strip().split('-')
      items = recipe_per_user[u]
      s1 = user_per_recipe[i]
      jaccard_sim = [0]
      for g in items-{i} :
            s2 = user_per_recipe[g]
            jaccard_sim.append(Jaccard(s1, s2))
      #print(jaccard_sim)
      if max(jaccard_sim)> threshold/100:
            predictions.write(u + '-' + i + ",1\n")
      else:
            predictions.write(u + '-' + i + ",0\n")

    predictions.close()

for threshold in [10,20]:
    write_prediction_jaccard(threshold)
```

Ans for Problem5 :

a. accuracy of the most popular 50 model on validation set = 0.68660 accuracy of the most popular 60 model on validation set = 0.69930 accuracy of the most popular 70 model on validation set = 0.68480 accuracy of the most popular 80 model on validation set = 0.70230 accuracy of the most popular 90 model on validation set = 0.63180 accuracy of the jaccard threshold 10 model on validation set = 0.52500 accuracy of the jaccard threshold 20 model on validation set = 0.50849

[23]:
```
Image(url= "problem9.png")
```

[23]:
```
<IPython.core.display.Image object>
```

[8]:
```
rating_data = []
user_per_recipe = defaultdict(set)
recipe_per_user = defaultdict(set)

for user,recipe,d in readCSV("trainInteractions.csv.gz"):
    rating_data.append([user,recipe,int(d['rating'])])
    recipe_per_user[user].add(recipe)
    user_per_recipe[recipe].add(user)
```

```python

rating_train = rating_data[:400000]
rating_valid = rating_data[400000:]

N = len(rating_data)
nUsers = len(list(recipe_per_user.keys()))
nItems = len(list(user_per_recipe.keys()))
users = list(recipe_per_user.keys())
items = list(user_per_recipe.keys())
#print(nUsers, nItems)
#print(len(users),len(items))
#print(len(users)+len(items))
```

[9]:

```python
import numpy as np
alpha = np.mean(np.array([d[2] for d in rating_data]))
print(alpha)
```

```
4.580794
```

[10]:

```python
userBiases = defaultdict(float)
itemBiases = defaultdict(float)
```

[11]:

```python
def prediction(user, item):
    return alpha + userBiases[user] + itemBiases[item]
```

[12]:

```python
def unpack(theta):
    #print("unpack, len(theta) = ", len(theta))
    global alpha
    global userBiases
    global itemBiases
    alpha = theta[0]
    #print(users)
    #print(items)
    userBiases = dict(zip(users, theta[1:nUsers+1]))
    itemBiases = dict(zip(items, theta[1+nUsers:]))
```

[13]:

```python
def MSE(predictions, labels):
    differences = [(x-y)**2 for x,y in zip(predictions,labels)]
    return sum(differences) / len(differences)

def cost(theta, labels, lamb):
    unpack(theta)
    #print("cost, len(theta) = ", len(theta))
    predictions = [prediction(d[0], d[1]) for d in rating_data]
    cost = MSE(predictions, labels)
    print("MSE = " + str(cost))
    for u in userBiases:
        cost += lamb*userBiases[u]**2
    for i in itemBiases:
        cost += lamb*itemBiases[i]**2
    return cost
```

[14]:
```
def derivative(theta, labels, lamb):
    unpack(theta)
    N = len(rating_data)
    dalpha = 0
    dUserBiases = defaultdict(float)
    dItemBiases = defaultdict(float)
    for d in rating_data:
        u,i = d[0], d[1]
        pred = prediction(u, i)
        diff = pred - d[2]
        dalpha += 2/N*diff
        dUserBiases[u] += 2/N*diff
        dItemBiases[i] += 2/N*diff
    for u in userBiases:
        dUserBiases[u] += 2*lamb*userBiases[u]
    for i in itemBiases:
        dItemBiases[i] += 2*lamb*itemBiases[i]
    dtheta = [dalpha] + [dUserBiases[u] for u in users] + [dItemBiases[i] for i␣
 ↪in items]
    #print("len [dUserBiases[u] for u in users] = ",len([dUserBiases[u] for u␣
 ↪in users])) #13533
    #print("len [dItemBiases[i] for i in items] = ",len([dItemBiases[i] for i␣
 ↪in items])) #163899 ???
    #print("len(items) = ",len(items))
    #print("derivative, len(dtheta) = ", len(dtheta))
    return np.hstack((dalpha,[dUserBiases[u] for u in users],[dItemBiases[i]␣
 ↪for i in items]))
```

[15]:
```
import scipy
import numpy
theta = np.hstack(([alpha],[0]*(nUsers+nItems)))
#print(len(theta)) 164996
labels = [d[2] for d in rating_data]
lamb = 1
scipy.optimize.fmin_l_bfgs_b(cost, theta, derivative, args = (labels, lamb))
```

```
MSE = 0.9008923295603781
MSE = 0.8880111212499086
MSE = 0.90075870005668
MSE = 0.9007586515538737
```

[15]:
```
(array([ 4.58067116e+00, -4.78253046e-05, -6.45546452e-06, …,
         8.38808750e-07,  8.38537481e-07,  8.38151494e-07]),
 0.9008253818180506,
 {'grad': array([ 5.34144372e-07, -1.97833492e-07, -7.32105272e-09, …,
         -1.18395761e-10, -1.14568445e-10, -1.09296871e-10]),
  'task': 'CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL',
  'funcalls': 4,
  'nit': 2,
  'warnflag': 0})
```

Answer for Problem9:

a. MSE on the validation set :

[34]:
```
Image(url= "problem10.png")
```

[34]:
```
<IPython.core.display.Image object>
```

[41]:
```
userBiases_max = max(userBiases, key= lambda x: userBiases[x])
print("maximum value of userBiases:",userBiases_max, userBiases[userBiases_max])
userBiases_min = min(userBiases, key= lambda x: userBiases[x])
print("minimum value of userBiases:",userBiases_min, userBiases[userBiases_min])
itemBiases_max = max(itemBiases, key= lambda x: itemBiases[x])
print("maximum value of itemBiases:",itemBiases_max, itemBiases[itemBiases_max])
itemBiases_min = min(itemBiases, key= lambda x: itemBiases[x])
print("minimum value of itemBiases:",itemBiases_min, itemBiases[itemBiases_min])
```

```
maximum value of userBiases: 32445558 3.7003036684773426e-06
minimum value of userBiases: 70705426 -1.2678107941701826e-06
maximum value of itemBiases: 98124873 1.8437616886670768e-07
minimum value of itemBiases: 29147042 -2.62689645997895e-07
```

Answer for Problem10:

a. user ID that have the largest value of beta : 32445558, value : 3.7003036684773426e-06

b. user ID that have the smallest value of beta : 70705426, value : -1.2678107941701826e-06

c. recipe ID that have the largest value of beta : 98124873, value : 1.8437616886670768e-07

d. recipe ID that have the smallest value of beta : 29147042, value : -2.62689645997895e-07

[35]:
```
Image(url= "problem11.png")
```

[35]:
```
<IPython.core.display.Image object>
```

[16]:
```
# MSE: 0.4547707140445709 -> MSE: 0.45477062463760376

for lamb in [0.00001, 0.0001] :
    print("lamb = ",lamb)
    scipy.optimize.fmin_l_bfgs_b(cost, theta, derivative, args = (labels, lamb))

    predictions = open("predictions_Rated_bias_%f.txt" % lamb,'w')
    for l in open("stub_Rated.txt"):
        if l.startswith("user_id"):
            #header
            predictions.write(l)
            continue
        u,i = l.strip().split('-')
        pred = alpha
        if u in userBiases.keys():
            pred+=userBiases[u]
        if i in itemBiases.keys():
            pred+=itemBiases[i]
        predictions.write(u + '-' + i + ",%f\n"%pred)
    predictions.close()
```

```
lamb =  1e-05
MSE = 0.9008923295603781
MSE = 0.8880111212499086
MSE = 1.069611774925789
MSE = 0.8854287133812021
MSE = 0.8800495326243006
MSE = 0.8792288815381499
MSE = 0.8761198198963718
MSE = 0.8552503387512107
MSE = 0.844030763897731
MSE = 0.8287385168931087
MSE = 0.8180375474910421
MSE = 0.7979963501373877
MSE = 0.7840140130811376
MSE = 0.7711449292816135
MSE = 0.7613025945278198
MSE = 0.7507671736011382
MSE = 5.345930082893958
MSE = 0.7506959492153554
MSE = 0.7467219604102234
MSE = 0.7387969604882767
MSE = 0.7338251748249827
MSE = 0.7261427552356622
MSE = 0.7185612868667165
MSE = 0.7098738719307657
MSE = 0.7009736748541021
MSE = 0.6902352307829346
MSE = 0.6820939765389208
MSE = 0.6809004970478701
MSE = 0.6806377856494903
MSE = 0.6800865674487122
MSE = 0.678011146489186
MSE = 0.6759285824177541
MSE = 0.6724558791953841
MSE = 0.6714563598671184
MSE = 0.6674767145274385
MSE = 0.6684374809302834
MSE = 0.6645768392321414
MSE = 0.6613946514295945
MSE = 0.6588469783561169
MSE = 0.6552680178801122
MSE = 0.6539755911054936
MSE = 0.6543714316487619
MSE = 0.6531478504559842
MSE = 0.6531738264247221
MSE = 0.6529455323919525
MSE = 0.6535219415400308
MSE = 0.6530414744047724
MSE = 0.6527522885467963
MSE = 0.6526004924888641
MSE = 0.6514953316059208
MSE = 0.651810231679115
MSE = 0.6520867071200405
MSE = 0.6521098608774898
MSE = 0.6521126974666346
MSE = 0.6531642611458578
MSE = 0.6518599106408396
MSE = 0.6516069778686175
MSE = 0.6512596293787687
MSE = 0.6511829470869952
MSE = 0.6508753821953974
MSE = 0.6505275168614008
MSE = 0.6501753646144617
MSE = 0.6498228235911943
MSE = 0.649807975088368
MSE = 0.6652478963758147
MSE = 0.6497554666090166
MSE = 0.6497079432578234
MSE = 0.6496194653288437
MSE = 0.6493059306629612
MSE = 0.649118283358505
MSE = 0.6490430164481296
MSE = 0.6489093634430095
MSE = 0.6488839070733912
MSE = 0.6488500876007357
MSE = 0.6475682814562694
MSE = 0.6478068105953899
MSE = 0.647731819273153
MSE = 0.6475946766583445
MSE = 0.6474794480709908
MSE = 0.6462640955460146
MSE = 0.6470189726939326
MSE = 0.6572492238531034
MSE = 0.6468564325407815
MSE = 0.6467023375260005
MSE = 0.6566834270930518
MSE = 0.6468183801573265
MSE = 0.6466180436784982
MSE = 0.6463515641940021
MSE = 0.6462976800216745
MSE = 0.646234396488296
MSE = 0.6461257112967079
MSE = 0.6458761278634012
MSE = 0.6455151938665773
MSE = 0.6437553881100531
MSE = 0.6448948171018407
MSE = 0.6444063441935024
MSE = 0.644624743793296
MSE = 0.6446383576573183
MSE = 0.6446563232787742
MSE = 0.644557695665424
MSE = 0.6441749125287669
MSE = 0.6441362702641132
MSE = 0.6440147217351855
MSE = 0.6437541447091862
MSE = 0.6435599628333424
MSE = 0.6434353415035318
MSE = 0.643357605861816
MSE = 0.6432587830953387
MSE = 0.6431719549755747
MSE = 0.6428553117495017
MSE = 0.6464380147726588
MSE = 0.6428882362182476
MSE = 0.6427652684251897
MSE = 0.6426761407346012
MSE = 0.6426075575934442
MSE = 0.6424777431850217
MSE = 0.6422846023501654
MSE = 0.6424138868690303
MSE = 0.6420828351389939
MSE = 0.6417127169935639
MSE = 0.6415730033940412
MSE = 0.6414769715651176
MSE = 0.641450693395469
MSE = 0.6414390079183784
MSE = 0.6415337276850459
MSE = 0.6415654290388586
MSE = 0.6415836794577897
MSE = 0.6412634316419313
MSE = 0.6415921834562589
MSE = 0.6413793687111206
MSE = 0.6411670124770998
MSE = 0.6408587484932727
MSE = 0.6407474739664187
MSE = 0.6407974204303061
MSE = 0.6403841670870936
MSE = 0.6406170258398457
MSE = 0.6406020960990854
MSE = 0.6405919568344804
MSE = 0.6404699671325975
MSE = 0.6401402914929535
MSE = 0.641437301097975
MSE = 0.6401668302571034
MSE = 0.6398702035651449
MSE = 0.6453945389107121
MSE = 0.6398309683157737
MSE = 0.6395892447232924
MSE = 0.6389924063628618
MSE = 0.6392824221818221
MSE = 0.6389800210128352
MSE = 0.6381670807331046
MSE = 0.6384894046474477
MSE = 0.6384781967006361
MSE = 0.6385447100234887
MSE = 0.6381952040760621
MSE = 0.6374711404205651
MSE = 0.6390956106507142
MSE = 0.6374457463115207
MSE = 0.6369639311051181
MSE = 0.6366839822392462
MSE = 0.6369218345271305
MSE = 0.636835706225587
MSE = 0.6368147408372649
MSE = 0.6368299360427108
MSE = 0.636667140090628
MSE = 0.6365166921374855
MSE = 0.6363187847697235
MSE = 0.6361851003850275
MSE = 0.6361166061586757
MSE = 0.636096575553414
MSE = 0.6360908060416661
MSE = 0.6368643067730128
MSE = 0.6361281494428047
MSE = 0.6361399781915908
MSE = 0.6361921296837435
MSE = 0.6362455980949782
MSE = 0.6362558474694692
MSE = 0.6360645912765704
MSE = 0.6361688148498138
MSE = 0.6360872803115023
MSE = 0.635865509040343
MSE = 0.6357759390949179
MSE = 0.635938821633157
MSE = 0.6357849968580424
MSE = 0.6356697170487369
MSE = 0.6357827838093234
MSE = 0.6356902936432969
MSE = 0.6356612016334707
MSE = 0.635689548513651
MSE = 0.6357132319490375
MSE = 0.6356993501468186
MSE = 0.6357120142191314
MSE = 0.6356984366591759
MSE = 0.6355253187283547
MSE = 0.6355155602284275
MSE = 0.6354121587066774
MSE = 0.6353062759344451
MSE = 0.6350512524944544
MSE = 0.6351815887501837
MSE = 0.6348770526228078
MSE = 0.635148604590614
MSE = 0.6350996952868735
MSE = 0.6350883909708442
MSE = 0.6351158323268064
MSE = 0.6350848273329036
MSE = 0.6350937480965854
MSE = 0.6351165988534044
MSE = 0.6351210247110727
MSE = 0.6351279570205869
MSE = 0.6344114400696008
MSE = 0.6350105291776783
MSE = 0.63499359886741
MSE = 0.6349407010138794
MSE = 0.634895949657667
MSE = 0.6347321648650878
MSE = 0.6348243888255913
MSE = 0.6348167483492404
MSE = 0.6348096345106009
MSE = 0.6348086637135137
MSE = 0.6348206143790429
MSE = 0.6348100699449816
MSE = 0.6348095136028468
MSE = 0.6348039278835477
MSE = 0.6348095330605363
MSE = 0.6348058545986139
MSE = 0.6347524836436221
MSE = 0.6352762753776684
MSE = 0.6347978479363615
MSE = 0.6347780739758608
MSE = 0.6347516485712363
MSE = 0.6347667346207378
MSE = 0.6347326792972932
MSE = 0.6347163177930022
MSE = 0.6346002956881384
MSE = 0.634675748370534
MSE = 0.6346938329560146
MSE = 0.6347265616049144
MSE = 0.6347255729761029
MSE = 0.6347115577660264
MSE = 0.6348671460047283
MSE = 0.6347708569151191
MSE = 0.6346148406915597
MSE = 0.6346746083198328
MSE = 0.6346892742224873
MSE = 0.634702807893328
MSE = 0.6342540772423353
MSE = 0.634676624070215
MSE = 0.6347037256792801
MSE = 0.6346729338686231
MSE = 0.6346481929112168
MSE = 0.6346259367457087
MSE = 0.6346171282646317
MSE = 0.6346105822515712
MSE = 0.6346047398113261
MSE = 0.6345848473256432
MSE = 0.6345534557006808
MSE = 0.6338886915560242
MSE = 0.6345117545536385
MSE = 0.6343841970459558
MSE = 0.6344653665413593
MSE = 0.6344629956910439
MSE = 0.6344431420836587
MSE = 0.6344371539215351
MSE = 0.6344284278942932
MSE = 0.6344158840617314
MSE = 0.6344427829622108
MSE = 0.634418119340311
MSE = 0.634407061274982
MSE = 0.6344012977194741
MSE = 0.6344003035124637
MSE = 0.6343986803760061
MSE = 0.6343909249757739
MSE = 0.634372447742189
MSE = 0.6343265598938645
MSE = 0.6343662262078718
MSE = 0.6343280804301162
MSE = 0.6342934667617972
MSE = 0.6342744840271014
MSE = 0.6342679235131122
MSE = 0.6342488452212572
MSE = 0.6342622067223109
MSE = 0.6342507404416353
MSE = 0.6342312448754917
MSE = 0.6342140040996228
MSE = 0.6342116412359292
MSE = 0.6341291575505147
MSE = 0.6341851607405737
MSE = 0.634202345013962
MSE = 0.6341880632397595
MSE = 0.6341913193545488
MSE = 0.6341947955073256
MSE = 0.6340009684145611
MSE = 0.6341715750487976
MSE = 0.6341628376078816
MSE = 0.6341514918490772
MSE = 0.6341294078022928
MSE = 0.6341183840468194
MSE = 0.6341386696833684
MSE = 0.6341157730039024
MSE = 0.6340993258003779
MSE = 0.6340823961292988
MSE = 0.6340841848833918
MSE = 0.6338827757059475
MSE = 0.6340772928867582
MSE = 0.634081380401378
MSE = 0.6340657003597402
MSE = 0.6340710466695122
MSE = 0.6340779066565423
MSE = 0.6340891966040787
MSE = 0.6340802247293351
MSE = 0.6340763386987056
MSE = 0.6340714184032632
MSE = 0.6340750815206039
MSE = 0.6340669554401532
MSE = 0.6340630783795247
MSE = 0.6340597354086867
MSE = 0.6340702099180981
MSE = 0.6342934036366215
MSE = 0.6340801018729273
MSE = 0.6341045422585108
MSE = 0.6340869138720852
MSE = 0.6341007662138642
MSE = 0.6341134327733907
MSE = 0.6341235802878193
MSE = 0.634136497156714
MSE = 0.6341883370776465
MSE = 0.6341383460311428
MSE = 0.6341457890983845
MSE = 0.6341553033299521
MSE = 0.6341525961754956
MSE = 0.634145990024387
MSE = 0.6341634033962597
MSE = 0.6341469348284917
MSE = 0.6341452700570915
MSE = 0.6341473471083766
MSE = 0.6341167133131254
MSE = 0.634141858898768
MSE = 0.6341444278273037
MSE = 0.6341526355067247
MSE = 0.634154020626509
MSE = 0.634150240106463
MSE = 0.6341477487661004
MSE = 0.634145570355057
MSE = 0.6341447120348698
MSE = 0.6341463156870584
MSE = 0.6341380505559471
MSE = 0.6341449388211792
MSE = 0.6341481788010801
MSE = 0.634154497656838
MSE = 0.6341558386685454
MSE = 0.6341637617328665
MSE = 0.634150696594303
MSE = 0.6341612331188466
MSE = 0.6341645257273949
MSE = 0.6341671913991425
MSE = 0.6341687176764904
MSE = 0.6341680289471773
MSE = 0.6341742650069225
MSE = 0.634177277246967
MSE = 0.6341978263757083
MSE = 0.6341859919882981
MSE = 0.6341923920606669
MSE = 0.6341900242523433
MSE = 0.6341886618744861
MSE = 0.6341885776394552
MSE = 0.6341829802146746
MSE = 0.6341906479203796
MSE = 0.6341951836312085
MSE = 0.6342176687400954
MSE = 0.6341981711512272
MSE = 0.6342059525417488
MSE = 0.6342085135199366
MSE = 0.634216546397375
MSE = 0.6342175410616744
MSE = 0.6342178465119325
MSE = 0.6342187558052469
MSE = 0.6342205535718703
MSE = 0.6342228663976861
MSE = 0.6342332172050339
MSE = 0.6342303027493539
MSE = 0.6342296344512754
MSE = 0.6342323064787018
MSE = 0.6342343922378433
MSE = 0.6342451866298886
MSE = 0.6342452347935369
MSE = 0.6342450632205238
MSE = 0.634245037715461
MSE = 0.6342513479234594
MSE = 0.6342382024011222
MSE = 0.6342482162347715
MSE = 0.6342512387196795
MSE = 0.6342579474901072
MSE = 0.634268733921549
MSE = 0.6342598115125848
MSE = 0.6342623472025214
MSE = 0.6342688243081908
MSE = 0.6342656867380435
MSE = 0.6342669840684998
MSE = 0.6342659539636893
MSE = 0.6342878454516789
MSE = 0.6342663899331522
MSE = 0.6342660707701869
MSE = 0.6342688102643227
MSE = 0.6342746137585848
MSE = 0.6342857179476071
MSE = 0.6342995876914765
MSE = 0.6343253154136154
MSE = 0.6343510523696653
MSE = 0.6343333543712552
MSE = 0.6343295687227958
MSE = 0.6343230711858318
MSE = 0.6343270033693744
MSE = 0.6343308205994943
lamb =  0.0001
MSE = 0.9008923295603781
MSE = 0.8880111212499086
MSE = 1.06862279077
MSE = 0.8855270897361742
MSE = 0.8800733442598857
MSE = 0.8792401321678299
MSE = 0.8760801200129987
MSE = 0.85543794918346
MSE = 0.8447671461635389
MSE = 0.8297331772329899
MSE = 0.8204837554157354
MSE = 0.8065773473043875
MSE = 0.7976501064097553
MSE = 0.7942577438023959
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
/var/folders/x9/xv9fr3td34x85pl4rz2hy7kh0000gn/T/ipykernel_23443/3993606945.py␣
 ↪in <module>
      3 for lamb in [0.00001, 0.0001, 0.001, 0.01, 0.1, 10,] :
      4     print("lamb = ",lamb)
----> 5     scipy.optimize.fmin_l_bfgs_b(cost, theta, derivative, args =␣
 ↪(labels, lamb))
      6
      7     predictions = open("predictions_Rated_bias_%f.txt" % lamb,'w')

/usr/local/lib/python3.9/site-packages/scipy/optimize/lbfgsb.py in␣
 ↪fmin_l_bfgs_b(func, x0, fprime, args, approx_grad, bounds, m, factr, pgtol,␣
 ↪epsilon, iprint, maxfun, maxiter, disp, callback, maxls)
    195                 'maxls': maxls}
    196
--> 197     res = _minimize_lbfgsb(fun, x0, args=args, jac=jac, bounds=bounds,

    198                            **opts)
    199     d = {'grad': res['jac'],

/usr/local/lib/python3.9/site-packages/scipy/optimize/lbfgsb.py in␣
 ↪_minimize_lbfgsb(fun, x0, args, jac, bounds, disp, maxcor, ftol, gtol, eps,␣
 ↪maxfun, maxiter, iprint, callback, maxls, finite_diff_rel_step,␣
 ↪**unknown_options)
    358                 # until the completion of the current minimization iteratio
    359                 # Overwrite f and g:
--> 360                 f, g = func_and_grad(x)
    361             elif task_str.startswith(b'NEW_X'):
    362                 # new iteration

/usr/local/lib/python3.9/site-packages/scipy/optimize/_differentiable_functions
 ↪py in fun_and_grad(self, x)
    266             self._update_x_impl(x)
    267         self._update_fun()
--> 268         self._update_grad()
    269         return self.f, self.g
    270
```

```
/usr/local/lib/python3.9/site-packages/scipy/optimize/_differentiable_functions
 ↪py in _update_grad(self)
    236     def _update_grad(self):
    237         if not self.g_updated:
--> 238             self._update_grad_impl()
    239             self.g_updated = True
    240

/usr/local/lib/python3.9/site-packages/scipy/optimize/_differentiable_functions
 ↪py in update_grad()
    147
    148             def update_grad():
--> 149                 self.g = grad_wrapped(self.x)
    150
    151         elif grad in FD_METHODS:

/usr/local/lib/python3.9/site-packages/scipy/optimize/_differentiable_functions
 ↪py in grad_wrapped(x)
    144             def grad_wrapped(x):
    145                 self.ngev += 1
--> 146                 return np.atleast_1d(grad(np.copy(x), *args))
    147
    148             def update_grad():

/var/folders/x9/xv9fr3td34x85pl4rz2hy7kh0000gn/T/ipykernel_23443/860518651.py i
 ↪derivative(theta, labels, lamb)
     10         diff = pred - d[2]
     11         dalpha += 2/N*diff
---> 12         dUserBiases[u] += 2/N*diff
     13         dItemBiases[i] += 2/N*diff
     14     for u in userBiases:

KeyboardInterrupt:
```

[17]:

```
for lamb in [0.00001, 0.0001, 0.001] :
    print("lamb = ",lamb)
    scipy.optimize.fmin_l_bfgs_b(cost, theta, derivative, args = (labels, lamb))

    predictions = open("predictions_Rated_bias_%f_20000.txt" % lamb,'w')
    for l in open("stub_Made.txt"):
        if l.startswith("user_id"):
            #header
            predictions.write(l)
            continue
        u,i = l.strip().split('-')
        pred = alpha
        if u in userBiases.keys():
            pred+=userBiases[u]
        if i in itemBiases.keys():
            pred+=itemBiases[i]
        predictions.write(u + '-' + i + ",%f\n"%pred)
    predictions.close()
```

```
lamb =  1e-05
MSE = 0.9008923295603781
MSE = 0.8880111212499086
MSE = 1.069611774925789
MSE = 0.8854287133812021
MSE = 0.8800495326243006
MSE = 0.8792288815381499
MSE = 0.8761198198963718
MSE = 0.8552503387512107
MSE = 0.844030763897731
MSE = 0.8287385168931087
MSE = 0.8180375474910421
MSE = 0.7979963501373877
MSE = 0.7840140130811376
MSE = 0.7711449292816135
MSE = 0.7613025945278198
MSE = 0.7507671736011382
MSE = 5.345930082893958
MSE = 0.7506959492153554
MSE = 0.7467219604102234
MSE = 0.7387969604882767
MSE = 0.7338251748249827
MSE = 0.7261427552356622
MSE = 0.7185612868667165
MSE = 0.7098738719307657
MSE = 0.7009736748541021
MSE = 0.6902352307829346
MSE = 0.6820939765389208
MSE = 0.6809004970478701
MSE = 0.6806377856494903
MSE = 0.6800865674487122
MSE = 0.678011146489186
MSE = 0.6759285824177541
MSE = 0.6724558791953841
MSE = 0.6714563598671184
MSE = 0.6674767145274385
MSE = 0.6684374809302834
MSE = 0.6645768392321414
MSE = 0.6613946514295945
MSE = 0.6588469783561169
MSE = 0.6552680178801122
MSE = 0.6539755911054936
MSE = 0.6543714316487619
MSE = 0.6531478504559842
MSE = 0.6531738264247221
MSE = 0.6529455323919525
MSE = 0.6535219415400308
MSE = 0.6530414744047724
MSE = 0.6527522885467963
MSE = 0.6526004924888641
MSE = 0.6514953316059208
MSE = 0.651810231679115
MSE = 0.6520867071200405
MSE = 0.6521098608774898
MSE = 0.6521126974666346
MSE = 0.6531642611458578
MSE = 0.6518599106408396
MSE = 0.6516069778686175
MSE = 0.6512596293787687
MSE = 0.6511829470869952
MSE = 0.6508753821953974
MSE = 0.6505275168614008
MSE = 0.6501753646144617
MSE = 0.6498228235911943
MSE = 0.649807975088368
MSE = 0.6652478963758147
MSE = 0.6497554666090166
MSE = 0.6497079432578234
MSE = 0.6496194653288437
MSE = 0.6493059306629612
MSE = 0.649118283358505
MSE = 0.6490430164481296
MSE = 0.6489093634430095
MSE = 0.6488839070733912
MSE = 0.6488500876007357
MSE = 0.6475682814562694
MSE = 0.6478068105953899
MSE = 0.647731819273153
MSE = 0.6475946766583445
MSE = 0.6474794480709908
MSE = 0.6462640955460146
MSE = 0.6470189726939326
MSE = 0.6572492238531034
MSE = 0.6468564325407815
MSE = 0.6467023375260005
MSE = 0.6566834270930518
MSE = 0.6468183801573265
MSE = 0.6466180436784982
MSE = 0.6463515641940021
MSE = 0.6462976800216745
MSE = 0.646234396488296
MSE = 0.6461257112967079
MSE = 0.6458761278634012
MSE = 0.6455151938665773
MSE = 0.6437553881100531
MSE = 0.6448948171018407
MSE = 0.6444063441935024
MSE = 0.644624743793296
MSE = 0.6446383576573183
MSE = 0.6446563232787742
MSE = 0.644557695665424
MSE = 0.6441749125287669
MSE = 0.6441362702641132
MSE = 0.6440147217351855
MSE = 0.6437541447091862
MSE = 0.6435599628333424
MSE = 0.6434353415035318
MSE = 0.643357605861816
MSE = 0.6432587830953387
MSE = 0.6431719549755747
MSE = 0.6428553117495017
MSE = 0.6464380147726588
MSE = 0.6428882362182476
MSE = 0.6427652684251897
MSE = 0.6426761407346012
MSE = 0.6426075575934442
MSE = 0.6424777431850217
MSE = 0.6422846023501654
MSE = 0.6424138868690303
MSE = 0.6420828351389939
MSE = 0.6417127169935639
MSE = 0.6415730033940412
MSE = 0.6414769715651176
MSE = 0.641450693395469
MSE = 0.6414390079183784
MSE = 0.6415337276850459
MSE = 0.6415654290388586
MSE = 0.6415836794577897
MSE = 0.6412634316419313
MSE = 0.6415921834562589
MSE = 0.6413793687111206
MSE = 0.6411670124770998
MSE = 0.6408587484932727
MSE = 0.6407474739664187
MSE = 0.6407974204303061
MSE = 0.6403841670870936
MSE = 0.6406170258398457
MSE = 0.6406020960990854
MSE = 0.6405919568344804
MSE = 0.6404699671325975
MSE = 0.6401402914929535
MSE = 0.641437301097975
MSE = 0.6401668302571034
MSE = 0.6398702035651449
MSE = 0.6453945389107121
MSE = 0.6398309683157737
MSE = 0.6395892447232924
MSE = 0.6389924063628618
MSE = 0.6392824221818221
MSE = 0.6389800210128352
MSE = 0.6381670807331046
MSE = 0.6384894046474477
MSE = 0.6384781967006361
MSE = 0.6385447100234887
MSE = 0.6381952040760621
MSE = 0.6374711404205651
MSE = 0.6390956106507142
MSE = 0.6374457463115207
MSE = 0.6369639311051181
MSE = 0.6366839822392462
MSE = 0.6369218345271305
MSE = 0.636835706225587
MSE = 0.6368147408372649
MSE = 0.6368299360427108
MSE = 0.636667140090628
MSE = 0.6365166921374855
MSE = 0.6363187847697235
MSE = 0.6361851003850275
MSE = 0.6361166061586757
MSE = 0.636096575553414
MSE = 0.6360908060416661
MSE = 0.6368643067730128
MSE = 0.6361281494428047
MSE = 0.6361399781915908
MSE = 0.6361921296837435
MSE = 0.6362455980949782
MSE = 0.6362558474694692
MSE = 0.6360645912765704
MSE = 0.6361688148498138
MSE = 0.6360872803115023
MSE = 0.635865509040343
MSE = 0.6357759390949179
MSE = 0.635938821633157
MSE = 0.6357849968580424
MSE = 0.6356697170487369
MSE = 0.6357827838093234
MSE = 0.6356902936432969
MSE = 0.6356612016334707
MSE = 0.635689548513651
MSE = 0.6357132319490375
MSE = 0.6356993501468186
MSE = 0.6357120142191314
MSE = 0.6356984366591759
MSE = 0.6355253187283547
MSE = 0.6355155602284275
MSE = 0.6354121587066774
MSE = 0.6353062759344451
MSE = 0.6350512524944544
MSE = 0.6351815887501837
MSE = 0.6348770526228078
MSE = 0.635148604590614
MSE = 0.6350996952868735
MSE = 0.6350883909708442
MSE = 0.6351158323268064
MSE = 0.6350848273329036
MSE = 0.6350937480965854
MSE = 0.6351165988534044
MSE = 0.6351210247110727
MSE = 0.6351279570205869
MSE = 0.6344114400696008
MSE = 0.6350105291776783
MSE = 0.63499359886741
MSE = 0.6349407010138794
MSE = 0.634895949657667
MSE = 0.6347321648650878
MSE = 0.6348243888255913
MSE = 0.6348167483492404
MSE = 0.6348096345106009
MSE = 0.6348086637135137
MSE = 0.6348206143790429
MSE = 0.6348100699449816
MSE = 0.6348095136028468
MSE = 0.6348039278835477
MSE = 0.6348095330605363
MSE = 0.6348058545986139
MSE = 0.6347524836436221
MSE = 0.6352762753776684
MSE = 0.6347978479363615
MSE = 0.6347780739758608
MSE = 0.6347516485712363
MSE = 0.6347667346207378
MSE = 0.6347326792972932
MSE = 0.6347163177930022
MSE = 0.6346002956881384
MSE = 0.634675748370534
MSE = 0.6346938329560146
MSE = 0.6347265616049144
MSE = 0.6347255729761029
MSE = 0.6347115577660264
MSE = 0.6348671460047283
MSE = 0.6347708569151191
MSE = 0.6346148406915597
MSE = 0.6346746083198328
MSE = 0.6346892742224873
MSE = 0.634702807893328
MSE = 0.6342540772423353
MSE = 0.634676624070215
MSE = 0.6347037256792801
MSE = 0.6346729338686231
MSE = 0.6346481929112168
MSE = 0.6346259367457087
MSE = 0.6346171282646317
MSE = 0.6346105822515712
MSE = 0.6346047398113261
MSE = 0.6345848473256432
MSE = 0.6345534557006808
MSE = 0.6338886915560242
MSE = 0.6345117545536385
MSE = 0.6343841970459558
MSE = 0.6344653665413593
MSE = 0.6344629956910439
MSE = 0.6344431420836587
MSE = 0.6344371539215351
MSE = 0.6344284278942932
MSE = 0.6344158840617314
MSE = 0.6344427829622108
MSE = 0.634418119340311
MSE = 0.634407061274982
MSE = 0.6344012977194741
MSE = 0.6344003035124637
MSE = 0.6343986803760061
MSE = 0.6343909249757739
MSE = 0.634372447742189
MSE = 0.6343265598938645
MSE = 0.6343662262078718
MSE = 0.6343280804301162
MSE = 0.6342934667617972
MSE = 0.6342744840271014
MSE = 0.6342679235131122
MSE = 0.6342488452212572
MSE = 0.6342622067223109
MSE = 0.6342507404416353
MSE = 0.6342312448754917
MSE = 0.6342140040996228
MSE = 0.6342116412359292
MSE = 0.6341291575505147
MSE = 0.6341851607405737
MSE = 0.634202345013962
MSE = 0.6341880632397595
MSE = 0.6341913193545488
MSE = 0.6341947955073256
MSE = 0.6340009684145611
MSE = 0.6341715750487976
MSE = 0.6341628376078816
MSE = 0.6341514918490772
MSE = 0.6341294078022928
MSE = 0.6341183840468194
MSE = 0.6341386696833684
MSE = 0.6341157730039024
MSE = 0.6340993258003779
MSE = 0.6340823961292988
MSE = 0.6340841848833918
MSE = 0.6338827757059475
MSE = 0.6340772928867582
MSE = 0.634081380401378
MSE = 0.6340657003597402
MSE = 0.6340710466695122
MSE = 0.6340779066565423
MSE = 0.6340891966040787
MSE = 0.6340802247293351
MSE = 0.6340763386987056
MSE = 0.6340714184032632
MSE = 0.6340750815206039
MSE = 0.6340669554401532
MSE = 0.6340630783795247
MSE = 0.6340597354086867
MSE = 0.6340702099180981
MSE = 0.6342934036366215
MSE = 0.6340801018729273
MSE = 0.6341045422585108
MSE = 0.6340869138720852
MSE = 0.6341007662138642
MSE = 0.6341134327733907
MSE = 0.6341235802878193
MSE = 0.634136497156714
MSE = 0.6341883370776465
MSE = 0.6341383460311428
MSE = 0.6341457890983845
MSE = 0.6341553033299521
MSE = 0.6341525961754956
MSE = 0.634145990024387
MSE = 0.6341634033962597
MSE = 0.6341469348284917
MSE = 0.6341452700570915
MSE = 0.6341473471083766
MSE = 0.6341167133131254
MSE = 0.634141858898768
MSE = 0.6341444278273037
MSE = 0.6341526355067247
MSE = 0.634154020626509
MSE = 0.634150240106463
MSE = 0.6341477487661004
MSE = 0.634145570355057
MSE = 0.6341447120348698
MSE = 0.6341463156870584
MSE = 0.6341380505559471
MSE = 0.6341449388211792
MSE = 0.6341481788010801
MSE = 0.634154497656838
MSE = 0.6341558386685454
MSE = 0.6341637617328665
MSE = 0.634150696594303
MSE = 0.6341612331188466
MSE = 0.6341645257273949
MSE = 0.6341671913991425
MSE = 0.6341687176764904
MSE = 0.6341680289471773
MSE = 0.6341742650069225
MSE = 0.634177277246967
MSE = 0.6341978263757083
MSE = 0.6341859919882981
MSE = 0.6341923920606669
MSE = 0.6341900242523433
MSE = 0.6341886618744861
MSE = 0.6341885776394552
MSE = 0.6341829802146746
MSE = 0.6341906479203796
MSE = 0.6341951836312085
MSE = 0.6342176687400954
MSE = 0.6341981711512272
MSE = 0.6342059525417488
MSE = 0.6342085135199366
MSE = 0.634216546397375
MSE = 0.6342175410616744
MSE = 0.6342178465119325
MSE = 0.6342187558052469
MSE = 0.6342205535718703
MSE = 0.6342228663976861
MSE = 0.6342332172050339
MSE = 0.6342303027493539
MSE = 0.6342296344512754
MSE = 0.6342323064787018
MSE = 0.6342343922378433
MSE = 0.6342451866298886
MSE = 0.6342452347935369
MSE = 0.6342450632205238
MSE = 0.634245037715461
MSE = 0.6342513479234594
MSE = 0.6342382024011222
MSE = 0.6342482162347715
MSE = 0.6342512387196795
MSE = 0.6342579474901072
MSE = 0.634268733921549
MSE = 0.6342598115125848
MSE = 0.6342623472025214
MSE = 0.6342688243081908
MSE = 0.6342656867380435
MSE = 0.6342669840684998
MSE = 0.6342659539636893
MSE = 0.6342878454516789
MSE = 0.6342663899331522
MSE = 0.6342660707701869
MSE = 0.6342688102643227
MSE = 0.6342746137585848
MSE = 0.6342857179476071
MSE = 0.6342995876914765
MSE = 0.6343253154136154
MSE = 0.6343510523696653
MSE = 0.6343333543712552
MSE = 0.6343295687227958
MSE = 0.6343230711858318
MSE = 0.6343270033693744
MSE = 0.6343308205994943
lamb =  0.0001
MSE = 0.9008923295603781
MSE = 0.8880111212499086
MSE = 1.06862279077
MSE = 0.8855270897361742
MSE = 0.8800733442598857
MSE = 0.8792401321678299
MSE = 0.8760801200129987
MSE = 0.85543794918346
MSE = 0.8447671461635389
MSE = 0.8297331772329899
MSE = 0.8204837554157354
MSE = 0.8065773473043875
MSE = 0.7976501064097553
MSE = 0.7942577438023959
MSE = 0.7927018130999864
MSE = 0.7905653925745705
MSE = 0.8512358769363845
MSE = 0.790515783166518
MSE = 0.7874443580766012
MSE = 0.7849211547156516
MSE = 0.7828499720498525
MSE = 0.7817458649577731
MSE = 0.7812932943269393
MSE = 0.7816922802022033
MSE = 0.7815845470920835
MSE = 0.7812976190180073
MSE = 0.7806994521071291
MSE = 0.7802859419157254
MSE = 0.780265547372516
MSE = 0.7802750756929726
MSE = 0.7803137870650587
MSE = 0.7798456511351345
MSE = 0.7800902021799689
MSE = 0.7801480192103055
MSE = 0.78010359241491
MSE = 0.7800419460191758
MSE = 0.7799744959900314
MSE = 0.7799250986224038
MSE = 0.7799175040116846
MSE = 0.7799189169771109
MSE = 0.7799399996321964
MSE = 0.7802509677831934
MSE = 0.7799410324002617
MSE = 0.779952286292096
MSE = 0.7801414106869669
MSE = 0.7799693706079754
MSE = 0.7799719895664099
MSE = 0.7799934607568919
MSE = 0.7800080167203193
MSE = 0.7800174629415336
lamb =  0.001
MSE = 0.9008923295603781
MSE = 0.8880111212499086
MSE = 1.0607060073644645
MSE = 0.8863010253223489
MSE = 0.881258376036975
MSE = 0.8803560986339667
MSE = 0.8769389926187163
MSE = 0.8643200138886875
MSE = 0.8606151882312933
MSE = 0.857579086062278
MSE = 0.8574139564458994
MSE = 0.857512232280082
MSE = 0.857484544764681
MSE = 0.8574310680388667
MSE = 0.8573958898569245
MSE = 0.8573993276748094
MSE = 0.8573996194370783
MSE = 0.8574021993238108
MSE = 0.8574001821737313
MSE = 0.8574004686509782
```

[ ]:

```
for lamb in [100, 1000] :
    print("lamb = ",lamb)
    scipy.optimize.fmin_l_bfgs_b(cost, theta, derivative, args = (labels, lamb))

    predictions = open("predictions_Rated_bias_%f.txt" % lamb,'w')
    for l in open("stub_Rated.txt"):
        if l.startswith("user_id"):
            #header
            predictions.write(l)
            continue
        u,i = l.strip().split('-')
        pred = alpha
        if u in userBiases.keys():
            pred+=userBiases[u]
        if i in itemBiases.keys():
            pred+=itemBiases[i]
        predictions.write(u + '-' + i + ",%f\n"%pred)
    predictions.close()
```

[58]:

```
import numpy as np
user_bias_mean = np.array(list(userBiases.values())).mean()
item_bias_mean = np.array(list(itemBiases.values())).mean()
print(user_bias_mean)
print(item_bias_mean)
```

```
-8.76262562739984e-25
-7.549711999005067e-26
```

Answer for Problem11:

a. value lamb wrt MSE :

lamb = 1e-05, MSE = 0.6343308205994943 lamb = 0.0001, MSE = 0.7800174629415336 lamb = 0.001, MSE = 0.8574004686509782 lamb = 0.01, MSE = 0.8911307388311662 lamb = 0.1, MSE = 0.8996097642865012 lamb = 10, MSE = 0.9008789032254846 lamb = 100, MSE = 0.9008909863002668 lamb = 1000, MSE = 0.900892195231244