Three

November 13, 2019

Homework 3

Question 1

[954]:
```
import numpy as np
import pandas as pd
import urllib
import scipy.optimize
import random
import gzip
import ast
import csv
import gzip
from collections import defaultdict
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
from sklearn.metrics import confusion_matrix
```

[955]:
```
X = []
y = []
df = pd.read_csv('/home/jovyan/Desktop/CSE 258/assignment1/train_interactions.
↪csv.gz')
for i in range(len(df)):
    X.extend([[df['userID'][i], df['bookID'][i], df['rating'][i]]])
    y.append(1)
```

[956]:
```
xtrain = X[0:190000]
ytrain = y[0:190000]
xval = X[190000:200000]
yval = y[190000:200000]
```

[957]:
```
usersPerItem = defaultdict(set)
itemsPerUser = defaultdict(set)
all_books = set([x[1] for x in X])

for x in xtrain:
    u, i = x[0], x[1]
```

```
    usersPerItem[i].add(u)
    itemsPerUser[u].add(i)
```

[958]:
```
xvalid=[]

for userbook in xval:
    ub = userbook[0]
    rating=-1
    the_diff=list(all_books.difference(itemsPerUser[ub]))
    rindex = random.randrange(len(the_diff))
    new = the_diff[rindex]
    xvalid.append([ub, new,rating])
    yval.append(0)
```

[959]:
```
xval.extend(xvalid)
```

[960]:
```
bookCount = defaultdict(int)
totalRead = 0

for user,book,rating in xtrain:
    bookCount[book] += 1
    totalRead += 1

mostPopular = [(bookCount[x], x) for x in bookCount]
mostPopular.sort()
mostPopular.reverse()

amount = set()
count = 0
for ic, i in mostPopular:
    count += ic
    amount.add(i)
    if count > totalRead/2: break
```

[961]:
```
valid_predictions=[]

for l in range(len(xval)):
    user = xval[l][0]
    book = xval[l][1]
    if book in amount:
        valid_predictions.append(1)
    else:
        valid_predictions.append(0)
```

[962]:
```
valid_values = []

for l in range(len(xval)):
```

```
    user = xval[l][0]
    book = xval[l][1]
    rate = xval[l][2]

    if rate == -1:
        valid_values.append(0)
    else:
        valid_values.append(1)
```

[963]:
```
from sklearn.metrics import confusion_matrix
tn,fp,fn,tp = confusion_matrix(valid_values, valid_predictions).ravel()
accuracy=(tp+tn)/(tp+tn+fp+fn)
TNR=tn/(tn+fp)
TPR=tp/(tp+fn)
BER=1-0.5*(TPR+TNR)
recall=tp/(tp+fn)
precision=tp/(tp+fp)
F1=2*(precision*recall)/(precision+recall)
F10=101*(precision*recall)/(100*precision+recall)
```

[964]:
```
print("Accuracy:",accuracy)
```

```
Accuracy: 0.64465
```

[965]:
```
print("BER:",BER)
```

```
BER: 0.35535000000000005
```

[966]:
```
print("F1 score:",F1)
```

```
F1 score: 0.5768887301303804
```

[967]:
```
print("F10 score:",F10)
```

```
F10 score: 0.48604137676214765
```

Question 2

[968]:
```
thresholds = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]
accuracies = []
```

[969]:
```
for t in thresholds:
    print(t)
    bookcount= defaultdict(int)
    Read2 = 0

    for i in range(len(xtrain)):
        user2 = xtrain[i][0]
```

```
        book2 = xtrain[i][1]
        bookcount[book2] += 1
        Read2 += 1

    mostpopular = [(bookcount[x], x) for x in bookcount]
    mostpopular.sort()
    mostpopular.reverse()

    return2 = set()
    count2 = 0
    for ic, i in mostpopular:
        count2 += ic
        return2.add(i)
        if count2 > Read2*t: break

    valid_predictions2 = []

    for l in range(len(xval)):
    #continue
        user = xval[l][0]
        book = xval[l][1]
        if book in return2:
            valid_predictions2.append(1)
        else:
            valid_predictions2.append(0)

    tn2,fp2,fn2,tp2 = confusion_matrix(valid_values, valid_predictions2).ravel()
    secondaccuracy=(tp2+tn2)/(tp2+tn2+fp2+fn2)

    print(secondaccuracy)

    accuracies.append(secondaccuracy)
```

```
0.1
0.5404
0.2
0.5746
0.3
0.6072
0.4
0.6286
0.5
0.64465
0.6
0.6494
0.7
0.64225
```

```
0.8
0.62165
0.9
0.58015
```

The threshold 0.6 returns the highest accuracy of the thresholds used, at 0.65255.

Question 3

[970]:
```
xtrain_dataframe=pd.DataFrame(xtrain, columns = ['userID' , 'bookID', 'rating'])
xval_dataframe=pd.DataFrame(xval, columns = ['userID' , 'bookID', 'rating'])
```

[971]:
```
valid_predictions3 = []
similar = []
threshold = 0.00001
```

[972]:
```
def Jaccard(s1, s2):
    numer = len(s1.intersection(s2))
    denom = len(s1.union(s2))
    return numer / denom
```

[973]:
```
allBooks_train = xtrain_dataframe['bookID'].unique().tolist()
book_user_train = xtrain_dataframe.groupby('userID')
user_per_book_train = xtrain_dataframe.groupby('bookID')
```

[974]:
```
len(user_per_book_train)
```

[974]: 7169

[975]:
```
for i in range(len(xval_dataframe)):
    user = xval_dataframe.iloc[i][0]
    book4 = xval_dataframe.iloc[i][1]

    sim = []
    if book4 in allBooks_train:

        book4_users = set(user_per_book_train.get_group(book4)['userID'])
        Book_read = book_user_train.get_group(user)['bookID']

        if len(book4_users) == 1:
            users_book_read = {}
            sim.append(Jaccard(book4_users,users_book_read))
            if max(sim) > threshold:
                valid_predictions3.append(1)
            else:
                valid_predictions3.append(0)

        else:
```

```
            for i in Book_read:
                users_book_read = set(user_per_book_train.
↪get_group(i)['userID'])
                sim.append(Jaccard(book4_users,users_book_read))

            if max(sim) > threshold:
                valid_predictions3.append(1)
            else:
                valid_predictions3.append(0)
    else:
        sim = 0
        valid_predictions3.append(0)

    similar.append(sim)
```

[976]:
```
tn3,fp3,fn3,tp3= confusion_matrix(valid_values,valid_predictions3).ravel()
thirdaccuracy=(tp3+tn3)/(tp3+tn3+fp3+fn3)
thirdaccuracy
```

[976]: 0.5915

Performance on Validation Set: 0.5915

Question 4

[977]:
```
xtrain_dataframe4 = pd.DataFrame(xtrain, columns = ['userID' , 'bookID',␣
↪'rating'])
xval_dataframe4 = pd.DataFrame(xval, columns = ['userID' , 'bookID', 'rating'])
allBooks_train = xtrain_dataframe['bookID'].unique().tolist()
book_user_train = xtrain_dataframe.groupby('userID')
user_per_book_train = xtrain_dataframe.groupby('bookID')
```

[978]:
```
thresh = [0.6]
```

[979]:
```
for tr in thresh:
    print(tr)
    bookCount4= defaultdict(int)
    totalRead4 = 0

    for i in range(len(xtrain)):
            user4=xtrain[i][0]
            book4=xtrain[i][1]
            bookCount4[book4] += 1
            totalRead4 += 1

    mostPopular4 = [(bookCount4[x], x) for x in bookCount4]
    mostPopular4.sort()
```

```
    mostPopular4.reverse()

    return4 = set()
    count4 = 0

    for ic, i in mostPopular:
        count4 += ic
        return4.add(i)
        if count4 > totalRead4*tr: break
```

0.6

[980]:
```
Y_val_predictions4=[]
similar4 = []

for i in range(len(xval_dataframe4)):

    user = xval_dataframe4.iloc[i][0]
    book = xval_dataframe4.iloc[i][1]

    if book in return4:

        sim = []

        if BK in allBooks_train:


            book_users = set(user_per_book_train.get_group(book)['userID'])


            Book_read = book_user_train.get_group(book)['bookID']

        if len(book_users) == 1:
            users_book_read = {}
            sim.append(Jaccard(book_users,users_book_read))
            if max(sim) > thresh:
                Y_val_predictions4.append(1)
            else:
                Y_val_predictions4.append(0)

        else:

            for i in Book_read:

                users_book_read = set(user_per_book_train.
↪get_group(i)['userID'])
                sim.append(Jaccard(BK_users,users_book_read))
```

```
            if max(sim) > threshold4:
                Y_val_predictions4.append(1)
            else:
                Y_val_predictions4.append(0)
    else:
            sim = 0
            Y_val_predictions4.append(0)
```

[981]:
```
type(Y_val_predictions4)
```

[981]:
```
list
```

[982]:
```
type(valid_values)
```

[982]:
```
list
```

[983]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(valid_values, Y_val_predictions4)
print(confusion_matrix)
```

```
[[7424 2576]
 [4317 5683]]
```

[984]:
```
tp = 7424
fp = 2576
tn = 5683
fn = 4317
```

[985]:
```
fourthaccuracy = (tp + tn) / (tp + tn + fp + fn)
fourthaccuracy
```

[985]:
```
0.65535
```

Performance on validation set: 0.65535

Question 5

[122]:
```
thresh = [0.6]
```

[123]:
```
for tr in thresh:
    print(tr)
    bookCount4= defaultdict(int)
    totalRead4 = 0

    for i in range(len(xtrain)):
            user4=xtrain[i][0]
            book4=xtrain[i][1]
```

```
            bookCount4[book4] += 1
            totalRead4 += 1

    mostPopular4 = [(bookCount4[x], x) for x in bookCount4]
    mostPopular4.sort()
    mostPopular4.reverse()

    return4 = set()
    count4 = 0

    for ic, i in mostPopular:
        count4 += ic
        return4.add(i)
        if count4 > totalRead4*tr: break
```

0.6

[161]:
```
book5 = []
users5 = []
X5 = []
y5 = []
df5 = pd.read_csv('/home/jovyan/Desktop/CSE 258/assignment1/pairs_Read.txt',␣
↪sep ="-" )
df5.rename(columns = {'bookID,prediction':'bookID'}, inplace = True)
```

[163]:
```
df5.tail()
```

[163]:

	userID	bookID
19995	u14735314	b54613892
19996	u00713548	b10036624
19997	u53512754	b57070326
19998	u40683372	b06314002
19999	u11383735	b57480183

[165]:
```
df5.iloc[0][0]
```

[165]: 'u65407115'

[201]:
```
predictions = open("/home/jovyan/Desktop/CSE 258/assignment1/predictions_Read.
↪txt", 'w')
for i in range(len(df5)):
    #u,b = df5.iloc[1][0].strip().split("-")
    #book5.append(b)
   # users5.append(u)
    user = df5.iloc[i][0]
    book5 = df5.iloc[i][1]
    #print(user)
```

```
    if book5 in return4:
        #
        sim = []
        if book5 in allBooks_train:


            book5_users = set(user_per_book_train.get_group(book5)['userID'])
            Book_read = book_user_train.get_group(user)['bookID']
        if len(book5_users) == 1:
            users_book_read = {}
            sim.append(Jaccard(book5_users,users_book_read))
            if max(sim) > threshold:
                predictions.write(user + '-' + book5 + ",1\n")
            else:
                predictions.write(user + '-' + book5 + ",0\n")
        else:

            for i in Book_read:

                users_book_read = set(user_per_book_train.
 ↪get_group(i)['userID'])
                sim.append(Jaccard(book5_users,users_book_read))

            if max(sim) > threshold4:
                predictions.write(user + '-' + book5 + ",1\n")
            else:
                predictions.write(user + '-' + book5 + ",0\n")
    else:
            sim = 0
            predictions.write(user + '-' + book5 + ",0\n")

predictions.close()
```

[185]:
```
test = pd.read_csv('/home/jovyan/Desktop/CSE 258/assignment1/predictions_Read.
 ↪txt',
                   header = None, names = ['userID-bookID', 'prediction'])
test.head()
```

[185]:

	userID-bookID	prediction
0	u65407115-b69897799	0
1	u53740605-b39436893	0
2	u88031275-b83889575	0
3	u99759913-b39270822	1
4	u20090895-b47380623	1

[187]:
```
export_csv = test.to_csv (r'/home/jovyan/Desktop/CSE 258/assignment1/
 ↪predictions_Read.csv', index = None, header=True)
```

Kaggle Username: Ellen Walsh

Question 9

[1026]:
```
#read the data in
reviewsPerUser = defaultdict(list)
reviewsPerItem = defaultdict(list)
path = '/home/jovyan/Desktop/CSE 258/assignment1/train_interactions.csv.gz'
f = gzip.open(path, 'rt', encoding="utf8")
header = f.readline()
header = header.strip().split(',')
header
```

[1026]:
```
['userID', 'bookID', 'rating']
```

[1027]:
```
dataset1 = []
for line in f:
    fields = line.strip().split(',')
    d = dict(zip(header, fields))
    d['rating'] = int(d['rating'])
    dataset1.append(d)
```

[1028]:
```
dataset = dataset1[0:190000]
```

[1029]:
```
#allusers = []
#allitems = []
for d in dataset:
    user,item = d['userID'], d['bookID']
    allusers.append(user)
    allitems.append(item)
    reviewsPerUser[user].append(user)
    reviewsPerItem[item].append(item)
```

[1030]:
```
ratingMean = sum([d['rating'] for d in dataset]) / len(dataset)
alpha = ratingMean
```

[1031]:
```
userBiases = defaultdict(float)
itemBiases = defaultdict(float)
```

[1032]:
```
N = len(dataset1) #length of our dataset
nUsers = len(reviewsPerUser)
nItems = len(reviewsPerItem)
users = list(reviewsPerUser.keys())
items = list(reviewsPerItem.keys())
```

[1033]:
```
userGamma = {}
itemGamma = {}
K = 2
```

[1034]:
```
for u in reviewsPerUser:
    userGamma[u] = [random.random() * 0.1 - 0.05 for k in range(K)]
```

[1035]:
```
for i in reviewsPerItem:
    itemGamma[i] = [random.random() * 0.1 - 0.05 for k in range(K)]
```

[1036]:
```
def unpack(theta):
    global alpha
    global userBiases
    global itemBiases
    global userGamma
    global itemGamma
    index = 0
    alpha = theta[index]
    index += 1
    userBiases = dict(zip(users, theta[index:index+nUsers]))
    index += nUsers
    itemBiases = dict(zip(items, theta[index:index+nItems]))
    index += nItems
    for u in users:
        userGamma[u] = theta[index:index+K]
        index += K
    for i in items:
        itemGamma[i] = theta[index:index+K]
        index += K
```

[1037]:
```
def prediction1(user, item):
    return alpha + userBiases[user] + itemBiases[item] + inner(userGamma[user],␣
 ↪itemGamma[item])
```

[1038]:
```
betau = 0
betai = 0
def prediction(user, item, betau, betai):
    if user in userBiases:
        betau = userBiases[user]
    else:
        betau = 0
        if item in itemBiases:
            betai = itemBiases[item]
        else:
            betai = 0
    return alpha + betau + betai
```

[1039]:
```
def cost(theta, labels, lamb):
    unpack(theta)
    predictions = [prediction1(d['userID'], d['bookID']) for d in dataset]
    cost = MSE(predictions, labels)
```

```
    print("MSE = " + str(cost))
    for u in users:
        cost += lamb*userBiases[u]**2
        for k in range(K):
            cost += lamb*userGamma[u][k]**2
    for i in items:
        cost += lamb*itemBiases[i]**2
        for k in range(K):
            cost += lamb*itemGamma[i][k]**2
    return cost
```

[1040]:
```
def derivative(theta, labels, lamb):
    unpack(theta)
    N = len(dataset)
    dalpha = 0
    dUserBiases = defaultdict(float)
    dItemBiases = defaultdict(float)
    dUserGamma = {}
    dItemGamma = {}
    for u in reviewsPerUser:
        dUserGamma[u] = [0.0 for k in range(K)]
    for i in reviewsPerItem:
        dItemGamma[i] = [0.0 for k in range(K)]
    for d in dataset:
        u,i = d['userID'], d['bookID']
        pred = prediction1(u, i)
        diff = pred - d['rating']
        dalpha += 2/N*diff
        dUserBiases[u] += 2/N*diff
        dItemBiases[i] += 2/N*diff
        for k in range(K):
            dUserGamma[u][k] += 2/N*itemGamma[i][k]*diff
            dItemGamma[i][k] += 2/N*userGamma[u][k]*diff
    for u in userBiases:
        dUserBiases[u] += 2*lamb*userBiases[u]
        for k in range(K):
            dUserGamma[u][k] += 2*lamb*userGamma[u][k]
    for i in itemBiases:
        dItemBiases[i] += 2*lamb*itemBiases[i]
        for k in range(K):
            dItemGamma[i][k] += 2*lamb*itemGamma[i][k]
    dtheta = [dalpha] + [dUserBiases[u] for u in users] + [dItemBiases[i] for i␣
↪in items]
    for u in users:
        dtheta += dUserGamma[u]
    for i in items:
        dtheta += dItemGamma[i]
```

```
    return numpy.array(dtheta)
```

[1041]:
```
scipy.optimize.fmin_l_bfgs_b(cost, [alpha] + # Initialize alpha
                                   [0.0]*(nUsers+nItems) + # Initialize beta
                                   [random.random() * 0.1 - 0.05 for k in␣
↪range(K*(nUsers+nItems))], # Gamma
                              derivative, args = (labels, 0.001))
```

```
MSE = 1.4772597921688049
MSE = 1.4624297066919072
MSE = 9.209821541522478
MSE = 1.4607033633568773
MSE = 1.3624823904517256
MSE = 1.3622399822338649
MSE = 1.3613537241943185
MSE = 1.3602510998673492
MSE = 1.3607498387397916
MSE = 1.3610118025805495
MSE = 1.361012638122509
MSE = 1.3610119473413964
```

[1041]:
```
(array([ 3.87908988e+00, -8.03060680e-03,  4.32681312e-02, …,
        -9.94879563e-06,  9.92707913e-06, -4.49551173e-06]),
 1.4137264183136011,
 {'grad': array([ 3.18211991e-07, -5.48976406e-09, -3.05460324e-08, …,
         -1.98879006e-08,  1.88117753e-08, -9.09601085e-09]),
  'task': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH',
  'funcalls': 12,
  'nit': 8,
  'warnflag': 0})
```

[1042]:
```
alpha = ratingMean
pred = []
for x in dataset:
    user = x['userID']
    book = x['bookID']
    p = prediction1(user, book)
    pred.append(p)
```

[1044]:
```
len(pred)
```

[1044]:
```
190000
```

[1045]:
```
labels = [d['rating'] for d in dataset]
```

[1046]:
```
def MSE(predictions, labels):
    differences = [(x-y)**2 for x,y in zip(predictions,labels)]
```

```
    return sum(differences) / len(differences)
```

[1047]:
```
MSE(pred, labels)
```

[1047]: 1.3576186439771218

[1048]:
```
valid = dataset1[190000:]
```

[1049]:
```
len(dataset1)
```

[1049]: 200000

[1050]:
```
len(valid)
```

[1050]: 10000

[1051]:
```
validlabels = [d['rating'] for d in valid]
validpredict = []
```

[1052]:
```
alpha = ratingMean
validpred = []
for x in valid:
    user = x['userID']
    book = x['bookID']
    p = prediction(user, book, 0, 0)
    validpred.append(p)
```

[1053]:
```
for user,book,rating in valid:
    validpredict.append(prediction(user,book,0,0))
print(MSE(validpred, validlabels))
```

1.4153923875358738

The MSE for validation set is 1.3918320892193197

Question 10

[1054]:
```
import operator
maxuser = max(userBiases.items(),key=operator.itemgetter(1))[0]
maxitem = max(itemBiases.items(),key=operator.itemgetter(1))[0]
minuser = min(userBiases.items(),key=operator.itemgetter(1))[0]
minitem = min(itemBiases.items(),key=operator.itemgetter(1))[0]
```

[1055]:
```
print("The user with the highest beta: ", maxuser)
print("The user with the lowest beta: ", minuser)
print("The item with the highest beta: ", maxitem)
print("The item with the lowest beta: ", minitem)
```

```
The user with the highest beta:  u92864068
The user with the lowest beta:  u11591742
The item with the highest beta:  b84299066
The item with the lowest beta:  b57299824
```

Question 11

[1056]:
```
N = len(dataset1) #length of our dataset
nUsers = len(reviewsPerUser)
nItems = len(reviewsPerItem)
users = list(reviewsPerUser.keys())
items = list(reviewsPerItem.keys())
```

[1063]:
```
theta = scipy.optimize.fmin_l_bfgs_b(cost, [alpha] + # Initialize alpha
                                     [0.0]*(nUsers+nItems) + # Initialize beta
                                     [random.random() * 0.1 - 0.05 for k in␣
 ↪range(K*(nUsers+nItems))], # Gamma
                                derivative, args = (labels, 0.001))
```

```
MSE = 1.4735565803111743
MSE = 1.4587283881377358
MSE = 9.783893732312077
MSE = 1.457181042857805
MSE = 1.358786849701721
MSE = 1.3585201087294905
MSE = 1.357551359830003
MSE = 1.3565318805064703
MSE = 1.3570337413231155
MSE = 1.357294031329298
MSE = 1.357293703330912
```

[1065]:
```
lam = [0.01,0.1,1,10,100]
li = []
for l in lam:
    print(l)
    theta = scipy.optimize.fmin_l_bfgs_b(cost, [alpha] + # Initialize alpha
                                         [0.0]*(nUsers+nItems) + # Initialize beta
                                         [random.random() * 0.1 - 0.05 for k in␣
 ↪range(K*(nUsers+nItems))], # Gamma
                                    derivative, args = (labels, l))
    li.append(theta)
```

```
0.01
MSE = 1.4735503547236848
MSE = 1.470735526031969
MSE = 1.7271193749542109
MSE = 1.5169376022864252
```

```
MSE = 1.4618689825896254
MSE = 1.4582734481029622
MSE = 1.4582769664529067
MSE = 1.4582910413686998
MSE = 1.4584227022114604
0.1
MSE = 1.4735487519447041
MSE = 1.4732636204959095
MSE = 1.5180458636582757
MSE = 3.7090917362908127
MSE = 1.471966821024494
MSE = 1.4719781601315696
1
MSE = 1.4735431344428382
MSE = 1.4735158307424183
MSE = 1.4733899537539272
10
MSE = 1.4735513750500677
MSE = 1.4735469735002826
MSE = 1.4735317405169936
MSE = 1.4735317404042878
100
MSE = 1.4735403958208166
MSE = 1.4735421413137837
MSE = 1.4735459253702416
MSE = 1.4735459253713235
MSE = 1.4735459253763914
```

Value of lambda: 0.01

MSE = 1.4584227022114604

[1087]:
```
test11 = pd.read_csv('/home/jovyan/Desktop/CSE 258/assignment1/pairs_Rating.
↪txt')
```

[1088]:
```
test11.head()
```

[1088]:

	userID-bookID	prediction
0	u39027358-b98920686	NaN
1	u50800253-b93497672	NaN
2	u07295538-b19850286	NaN
3	u41427072-b39758017	NaN
4	u89648987-b25118404	NaN

[1089]:
```
test11['prediction'] = validpred
```

[1091]:
```
test11.tail()
```

[1091]:

	userID-bookID	prediction
9995	u78346778-b73812430	3.915340
9996	u71616100-b10876672	3.914581
9997	u64609829-b62615825	3.888772
9998	u28944466-b02268479	3.907948
9999	u07954759-b23242339	3.945518

[1092]:
```
export_csv = test11.to_csv (r'/home/jovyan/Desktop/CSE 258/assignment1/
↪predictions_Rating.csv', index = None, header=True)
```