Homework 3

November 8, 2021

[58]:
```
import gzip
from collections import defaultdict
from sklearn import linear_model
import csv
```

[59]:
```
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)
    for l in c:
        d = dict(zip(header,l))
        yield d['user_id'],d['recipe_id'],d
```

[60]:
```
allRatings = []
userRatings = defaultdict(list)
user_list=[]


for user,recipe,d in readCSV("trainInteractions.csv.gz"):
    user_list.append([user,recipe,int(d["rating"])])
    r = int(d['rating'])
    allRatings.append(r)
    userRatings[user].append(r)

globalAverage = sum(allRatings) / len(allRatings)
userAverage = {}
for u in userRatings:
  userAverage[u] = sum(userRatings[u]) / len(userRatings[u])
```

[61]:
```
train_set=user_list[:400000]
valid_set=user_list[400000:500000]
```

1 Question 1

[62]:
```
setRecipe=set()
setUser=set()
dic_rating=defaultdict(set)
userPerRecipe=defaultdict(set)
recipePerUser=defaultdict(set)
for i in user_list:
    setRecipe.add(i[1])
    setUser.add(i[0])
    userPerRecipe[i[1]].add(i[0])
    recipePerUser[i[0]].add(i[1])
    dic_rating[(i[0],i[1])]=i[2]
listRecipe=list(setRecipe)
listUser=list(setUser)
```

[63]:
```
question_1=[]
for i in valid_set:
    question_1.append([i[0],i[1],1,0])
```

[64]:
```
import random
random_index = random.randint(0,len(listRecipe)-1)
cookedNot=[]
for i in valid_set:
    user=i[0]
    while(True):
        random_index = random.randint(0,len(listRecipe)-1)
        rec=listRecipe[random_index]
        if(rec not in recipePerUser[user]):
            temp=[]
            temp.append(user)
            temp.append(rec)
            cookedNot.append(temp)
            break
```

[65]:
```
for i in cookedNot:
    question_1.append([i[0],i[1],0,0])
```

[66]:
```
def popularityFunction(threshold):
    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()

    return1 = set()
    count = 0
    for ic, i in mostPopular:
        count += ic
        return1.add(i)
        if count > (totalCooked*threshold): break
    return return1
```

[67]:
```
def predictor_accuracy(threshold):
    popularItems=popularityFunction(threshold)
#     popularItems=list(popularItems)
    for i in question_1:
        if(i[1] in popularItems):
            i[3]=1
        else:
            i[3]=0
    tptn=0
    fpfn=0
    for i in question_1:
        if(i[2]==i[3]):
            tptn=tptn+1
        else:
            fpfn=fpfn+1
    return(tptn/(tptn+fpfn))
```

[68]:
```
print("Performance on Validation Set: ",predictor_accuracy(0.5))
```

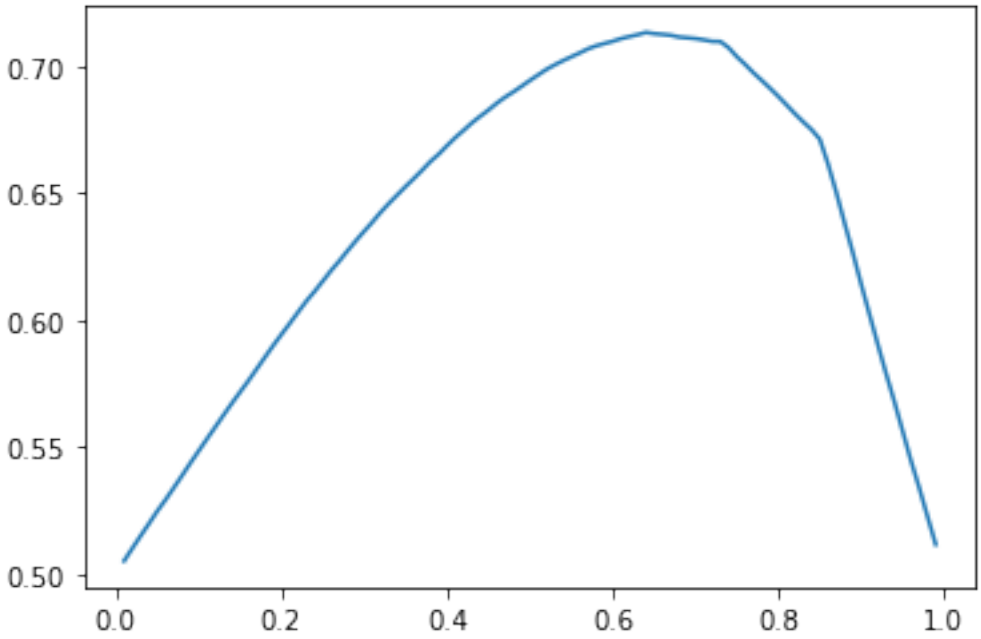
```
Performance on Validation Set:  0.694335
```

2 Question 2

[69]:
```
import numpy as np
xab=[]
yab=[]
for thresh in np.arange(0.01, 1, 0.01):
    xab.append(thresh)
    yab.append(predictor_accuracy(thresh))
```

[70]:
```
import matplotlib.pyplot as plt
def plot_function():
    plt.plot(xab, yab)
```

[71]:
```
plot_function()
```



[72]:
```
for i in range(len(xab)):
    if(yab[i]>yab[i+1]):
        print("Threshold",xab[i],"Performance on Validation Set: ",yab[i])
        break
```

```
Threshold 0.64 Performance on Validation Set:  0.71323
```

3 Question 3

[73]:
```
setRecipeT=set()
setUserT=set()
dic_ratingT=defaultdict(set)
userPerRecipeT=defaultdict(set)
recipePerUserT=defaultdict(set)
for i in train_set:
    setRecipeT.add(i[1])
    setUserT.add(i[0])
    userPerRecipeT[i[1]].add(i[0])
    recipePerUserT[i[0]].add(i[1])
    dic_ratingT[(i[0],i[1])]=i[2]
listRecipeT=list(setRecipeT)
listUserT=list(setUserT)
```

[74]:
```
valid_data=random.choice(valid_set)
```

[75]:
```
print(valid_data)
```

```
['51048569', '00413511', 5]
```

[76]:
```
def Jaccard(s1,s2):
    numer=len(s1.intersection(s2))
    deno=len(s1.union(s2))
    return(numer/deno)
```

[77]:
```
def mostSimilar(user,recipe):
    users=userPerRecipeT[recipe]
    t=[]
    for j in recipePerUserT[user]:
        if(i==j):
            continue
        similarities=[]
        sim=Jaccard(users,userPerRecipeT[j])
        similarities.append(sim)
        similarities.append(j)
#     s=sorted(similarities, key=lambda element:(-element[0]))
        t.append(similarities)
    t.sort(reverse=True)
    return(t)
```

[78]:
```
def accuracy(question_1):
    tptn=0
    fpfn=0
    for i in question_1:
        if(i[2]==i[3]):
            tptn=tptn+1
        else:
            fpfn=fpfn+1
    return(tptn/(tptn+fpfn))
```

[79]:
```
threshold=0.5
highestJaccard=[]
for i in question_1:
    t=mostSimilar(i[0],i[1])
    if(len(t)==0):
        highestJaccard.append(0)
        continue
    highestJaccard.append(t[0][0])
```

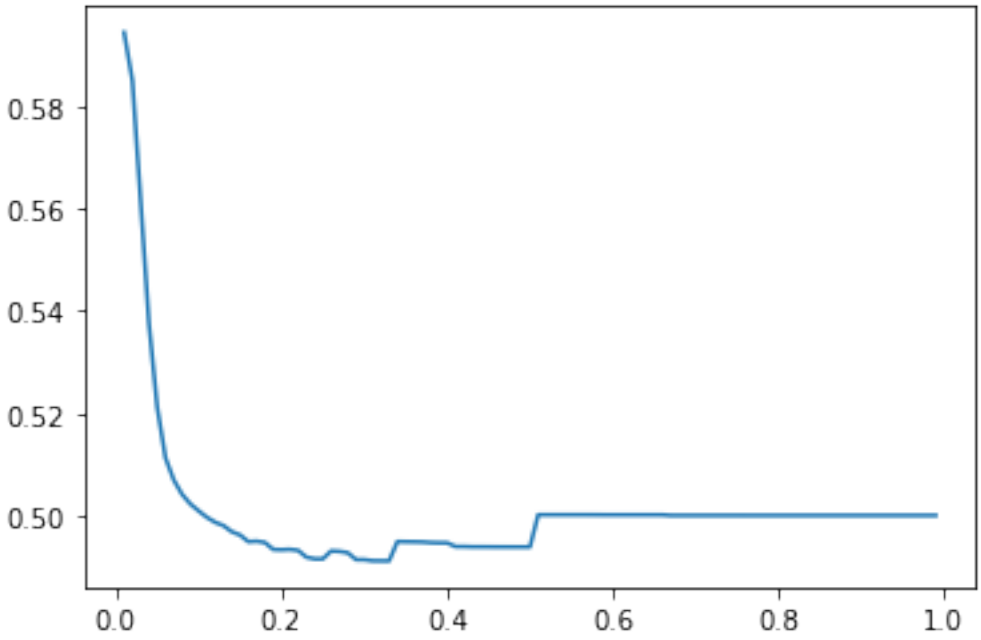
[80]:
```
import numpy as np
xab=[]
yab=[]
for threshold in np.arange(0.01, 1, 0.01):
    for i in range(len(highestJaccard)):
```

```
        if(highestJaccard[i]>=threshold):
            question_1[i][3]=1
        else:
            question_1[i][3]=0

    xab.append(threshold)
    yab.append(accuracy(question_1))
```

[81]:
```
import matplotlib.pyplot as plt
def plot_function():
    plt.plot(xab, yab)
```

[82]:
```
plot_function()
```



[83]:
```
for i in range(len(xab)):
    if(yab[i]>yab[i+1]):
        print("Threshold: ",xab[i],"Performance on Validation Set: ",yab[i])
        break
```

```
Threshold:  0.01 Performance on Validation Set:  0.594545
```

4 Question 4

[27]:
```
import numpy as np
values=[]
for threshold1 in np.arange(0.01, 1, 0.01):
    for threshold2 in np.arange(0.01, 1, 0.01):
        yab=[]
        o=popularityFunction(threshold2)
        for i in range(len(highestJaccard)):
            if(highestJaccard[i]>=threshold1 or question_1[i][1] in o):
                question_1[i][3]=1
            else:
                question_1[i][3]=0
        yab.append(accuracy(question_1))
        yab.append(threshold1)
        yab.append(threshold2)
        values.append(yab)
```

[56]:
```
maximum_acc=0
for i in range(len(values)):
    maximum_acc=max(maximum_acc,values[i][0])
for i in range(len(values)):
    if(values[i][0]==maximum_acc):
        print("Accuracy on Training Set :  ",maximum_acc)
        print("Threshold 1: ",values[i][1],"Threshold 2: ",values[i][2])
        threshold1=values[i][1]
        threshold2=values[i][2]
        break
```

```
Accuracy on Training Set :   0.715245
Threshold 1:  0.51 Threshold 2:  0.64
```

[52]:
```
o=popularityFunction(threshold2)
```

[57]:
```
for i1 in range(len(question_1)):
    u,i = question_1[i1][0],question_1[i1][1]
    modelJaccard=mostSimilar(u,i)
    if(len(modelJaccard)==0):
        if(i in o):
            question_1[i1][3]=1
            continue
        else:
            question_1[i1][3]=0
            continue
    if(modelJaccard[0][0]>threshold1 or i in o) :
        question_1[i1][3]=1
    else:
```

```
        question_1[i1][3]=0
print("Accuracy on Validation Set: ",accuracy(question_1))
```

```
Accuracy on Validation Set:  0.71298
```

5 Question 5 Kaggle User Name: Kakshak Porwal

[30]:
```
predictions = open("predictions_Made.txt", 'w')
for l in open("stub_Made.txt"):
    if l.startswith("user_id"):
        predictions.write(l)
        continue
    u,i = l.strip().split('-')
    modelJaccard=mostSimilar(u,i)
    if(len(modelJaccard)==0):
        if(i in o):
            predictions.write(u + '-' + i + ",1\n")
            continue
        else:
            predictions.write(u + '-' + i + ",0\n")
            continue

    if(modelJaccard[0][0]>threshold1 or i in o) :
        predictions.write(u + '-' + i + ",1\n")
    else:
        predictions.write(u + '-' + i + ",0\n")

predictions.close()
```

Homework 3

November 8, 2021

1 Question 9

[1]:
```
import gzip
from collections import defaultdict
from sklearn import linear_model
import csv
import numpy
```

[2]:
```
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)
    for l in c:
        d = dict(zip(header,l))
        yield d['user_id'],d['recipe_id'],d
```

[3]:
```
allRatings = []
userRatings = defaultdict(list)
user_list=[]


for user,recipe,d in readCSV("trainInteractions.csv.gz"):
    user_list.append([user,recipe,int(d["rating"])])
    r = int(d['rating'])
    allRatings.append(r)
    userRatings[user].append(r)

globalAverage = sum(allRatings) / len(allRatings)
userAverage = {}
for u in userRatings:
    userAverage[u] = sum(userRatings[u]) / len(userRatings[u])
train_set=user_list[:400000]
valid_set=user_list[400000:500000]
```

[4]:
```
reviewsPerUser = defaultdict(list)
reviewsPerItem = defaultdict(list)
```

[5]:
```
for d in train_set:
    user,item = d[0],d[1]
    reviewsPerUser[user].append(d)
    reviewsPerItem[item].append(d)
```

[6]:
```
usersPerItem = defaultdict(set) # Maps an item to the users who rated it
itemsPerUser = defaultdict(set) # Maps a user to the items that they rated
itemNames = {}
ratingDict = {} # To retrieve a rating for a specific user/item pair

for d in train_set:
    user,item = d[0], d[1]
    usersPerItem[item].add(user)
    itemsPerUser[user].add(item)
    ratingDict[(user,item)] = d[2]
```

[7]:
```
userAverages=defaultdict()
itemAverages=defaultdict()
# userAverages = {}
# itemAverages = {}

for u in itemsPerUser:
    rs = [ratingDict[(u,i)] for i in itemsPerUser[u]]
    userAverages[u] = sum(rs) / len(rs)

for i in usersPerItem:
    rs = [ratingDict[(u,i)] for u in usersPerItem[i]]
    itemAverages[i] = sum(rs) / len(rs)
```

[8]:
```
averageRating=0
for d in train_set:
    averageRating=averageRating+d[2]

ratingMean = averageRating/len(train_set)
```

[9]:
```
reviewPerRecipe=defaultdict(list)
reviewPerUser= defaultdict(list)
for i in train_set:
    reviewPerRecipe[i[1]].append(i[2])
    reviewPerUser[i[0]].append(i[2])
```

[10]:
```
N = len(train_set)
nUsers = len(reviewsPerUser)
nItems = len(reviewsPerItem)
```

```
users = list(reviewsPerUser.keys())
items = list(reviewsPerItem.keys())
```

[11]:
```
alpha = ratingMean
```

[12]:
```
userBiases = defaultdict(float)
itemBiases = defaultdict(float)
```

[ ]:
```
```

[13]:
```
def prediction(user, item):
    if(user not in userBiases.keys() and item not in itemBiases.keys()):
        return(alpha)
    if(user not in userBiases.keys() and item in itemBiases.keys()):
        return(alpha + itemBiases[item])
    if(user in userBiases.keys() and item not in itemBiases.keys()):
        return(alpha + userBiases[user])
    return(alpha + userBiases[user] + itemBiases[item])
```

[14]:
```
def unpack(theta):
    global alpha
    global userBiases
    global itemBiases
    alpha = theta[0]
    userBiases = dict(zip(users, theta[1:nUsers+1]))
    itemBiases = dict(zip(items, theta[1+nUsers:]))
```

[47]:
```
def cost(theta, labels, lamb):
    unpack(theta)
    predictions = [prediction(d[0], d[1]) for d in train_set]
    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
```

[16]:
```
def derivative(theta, labels, lamb):
    unpack(theta)
    N = len(train_set)
    dalpha = 0
    dUserBiases = defaultdict(float)
    dItemBiases = defaultdict(float)
    for d in train_set:
        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]
    return numpy.array(dtheta)
```

[17]:
```
def MSE(predictions, labels):
    differences = [(x-y)**2 for x,y in zip(predictions,labels)]
    return sum(differences) / len(differences)
```

[18]:
```
alwaysPredictMean = [ratingMean for d in train_set]
```

[19]:
```
labels = [d[2] for d in train_set]
```

[40]:
```
import scipy
q=scipy.optimize.fmin_l_bfgs_b(cost, [alpha] + [0.0]*(nUsers+nItems),␣
 ↪derivative, args = (labels, 1))
```

```
MSE = 0.8987313760374103
MSE = 0.8863220312891775
MSE = 0.8985953121629906
MSE = 0.8985952330504594
```

[41]:
```
labels_valid=[]
predictions_valid=[]
for i in range(len(valid_set)):
    labels_valid.append(valid_set[i][2])
    predictions_valid.append(prediction(valid_set[i][0],valid_set[i][1]))
```

[42]:
```
print("MSE on Validation Set: ",MSE(predictions_valid,labels_valid))
```

```
MSE on Validation Set:  0.9094108423819184
```

2 Question 10

[43]:
```
min(itemBiases.items(), key=lambda x: x[1])
```

[43]: ('29147042', -0.0002853173040833448)

[44]:
```
max(itemBiases.items(), key=lambda x: x[1])
```

[44]:
```
('98124873', 0.00020946448810490916)
```

[45]:
```
min(userBiases.items(), key=lambda x: x[1])
```

[45]:
```
('70705426', -0.0012946223969233764)
```

[46]:
```
max(userBiases.items(), key=lambda x: x[1])
```

[46]:
```
('32445558', 0.0036701233370137437)
```

3 Question 11

[31]:
```
def mse_valid():
    labels_valid=[]
    predictions_valid=[]
    for i in range(len(valid_set)):
        labels_valid.append(valid_set[i][2])
        predictions_valid.append(prediction(valid_set[i][0],valid_set[i][1]))
    return(MSE(predictions_valid,labels_valid))
```

[32]:
```
import numpy as np
mse_valid_l=[]
for lamda in np.arange(0.01, 1, 0.01):
    l=[]
    scipy.optimize.fmin_l_bfgs_b(cost, [alpha] + [0.0]*(nUsers+nItems),␣
 ↪derivative, args = (labels, lamda))
    l.append(mse_valid())
    l.append(lamda)
    mse_valid_l.append(l)
```

[36]:
```
o=[]
for i in range(len(mse_valid_l)):
    o.append(mse_valid_l[i][0])
t=min(o)
for i in range(len(mse_valid_l)):
    if(t==mse_valid_l[i][0]):
        print("MSE on validation set :",mse_valid_l[i][0],"Optimal Value of␣
 ↪Lambda : ",mse_valid_l[i][1])
        break
```

```
MSE on validation set : 0.9005787810061352 Optimal Value of Lambda :  0.01
```

[37]:
```
scipy.optimize.fmin_l_bfgs_b(cost, [alpha] + [0.0]*(nUsers+nItems), derivative,␣
 ↪args = (labels, 0.01))
```

[37]:
```
(array([ 4.57175696e+00, -7.42311770e-03, -7.82074978e-04, ...,
        -1.42300842e-04,  9.04312545e-05, -1.40592993e-04]),
```

```
 0.8932767891570182,
 {'grad': array([-4.03404592e-07, -1.51643898e-07,  2.21462608e-08, ...,
         4.78841165e-09, -2.92573350e-09,  2.88283312e-09]),
  'task': 'CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL',
  'funcalls': 8,
  'nit': 6,
  'warnflag': 0})
```

4 Kaggle User Name: Kakshak Porwal

[34]:
```
allRatings = []
userRatings = defaultdict(list)

for user,recipe,d in readCSV("trainInteractions.csv.gz"):
    r = int(d['rating'])
    allRatings.append(r)
    userRatings[user].append(r)

globalAverage = sum(allRatings) / len(allRatings)
userAverage = {}
for u in userRatings:
    userAverage[u] = sum(userRatings[u]) / len(userRatings[u])

predictions = open("predictions_Rated.txt", 'w')
for l in open("stub_Rated.txt"):
    if l.startswith("user_id"):
        #header
        predictions.write(l)
        continue
    u,i = l.strip().split('-')
    itPred=prediction(u,i)
#     itPred=itPred+predictRating(u,i)
#     itPred=itPred/2
    predictions.write(u + '-' + i + ',' + str(itPred) + '\n')


predictions.close()
```