A53271697
Bosi Cheng

1.

```
businessCount_train,businessCount_valid = defaultdict(int),defaultdict(int)

negative_valid=defaultdict(list)

train,positive_valid=defaultdict(list),defaultdict(list)

user_set,item_set=set(),set()

totalPurchases = 0



for l in readGz("train.json.gz"):

  if totalPurchases<100000:

    user,business = l['reviewerID'],l['itemID']

    user_set.add(user)

    item_set.add(business)

    train[user].append(business)

    businessCount_train[business] += 1

    totalPurchases += 1

  else:

    user,business = l['reviewerID'],l['itemID']

    user_set.add(user)

    item_set.add(business)
```

```
        positive_valid[user].append(business)
        businessCount_valid[business] += 1
        totalPurchases += 1


valid_if_purchase=0
while valid_if_purchase<100000:
    random_user=user_set.pop()
    user_set.add(random_user)
    random_item=item_set.pop()
    item_set.add(random_item)
    if random_item not in train[random_user] :
        if random_item not in negative_valid[random_user]:
            negative_valid[random_user].append(random_item)
            valid_if_purchase+=1


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


return1 = set()
```

```
count = 0

for ic, i in mostPopular:

    count += ic

    return1.add(i)

    if count > totalPurchases/2: break



correction=0

for user in positive_valid.keys():

    for item in positive_valid[user]:

        if item in return1:

            correction+=1




for user in negative_valid.keys():

    for item in negative_valid[user]:

        if item not in return1:

            correction+=1



print(len(positive_valid))

print(len(negative_valid))

print(correction)
```

```
print(float(correction/200000))
```

```
36402
39239
100472
0.50236
```

2.

No, it is not the best choice. This model has nothing to do with the training set. We should include the non-purchase pair only in the validation sets.

```
businessCount_train,businessCount_valid = defaultdict(int),defaultdict(int)

negative_valid=defaultdict(list)

train,positive_valid=defaultdict(list),defaultdict(list)

user_set,item_set=set(),set()

train_user_set=set()

totalPurchases = 0

valid_user_set,valid_item_set=set(),set()



for l in readGz("train.json.gz"):

    if totalPurchases<100000:

        user,business = l['reviewerID'],l['itemID']

        if user not in user_set:

            user_set.add(user)

        if business not in item_set:

            item_set.add(business)
```

```
        train[user].append(business)
        businessCount_train[business] += 1
        totalPurchases += 1
    else:
        user,business = l['reviewerID'],l['itemID']
        if user not in valid_user_set:
            valid_user_set.add(user)
        if business not in valid_item_set:
            valid_item_set.add(business)
        positive_valid[user].append(business)
        businessCount_valid[business] += 1
        totalPurchases += 1

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

return1 = set()
count = 0
for ic, i in mostPopular:
```

```
    count += ic
    return1.add(i)
    if count > totalPurchases/2: break


valid_if_purchase=0
while valid_if_purchase<100000:
    random_user=valid_user_set.pop()
    valid_user_set.add(random_user)
    random_item=valid_item_set.pop()
    valid_item_set.add(random_item)
    if random_item not in train[random_user] :
        if random_item not in negative_valid[random_user]:
            negative_valid[random_user].append(random_item)
            valid_if_purchase+=1




correction=0
for user in positive_valid.keys():
    for item in positive_valid[user]:
        if item in return1:
```

```
        correction+=1
print(correction)



for user in negative_valid.keys():
   for item in negative_valid[user]:
      if item not in return1:
         correction+=1


print(len(positive_valid))
print(len(negative_valid))
print(correction)
print(float(correction/200000))
```

```
115673
0.578365
```

3.

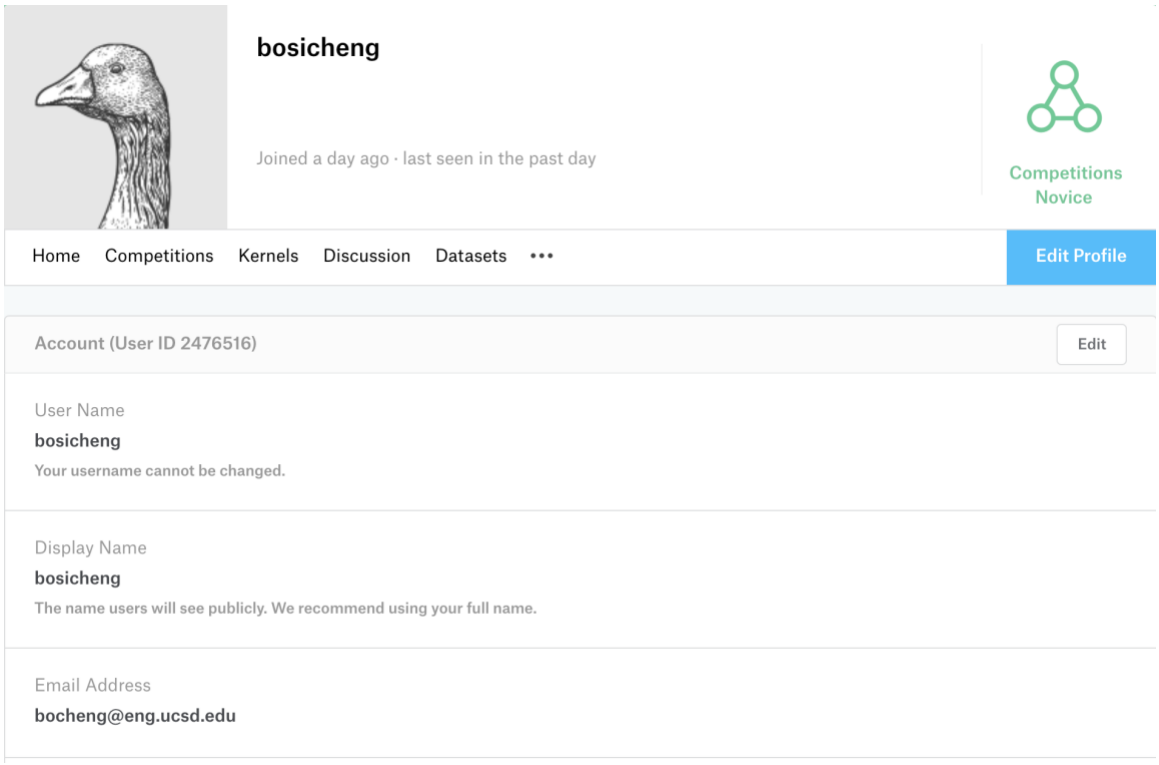
while loading the json, create a dict that matches every item and its categories item_categories[item].

```
repeat_purchase=defaultdict()
for user in user_set:
   temp=set()
   for item in purchase[user]:
      for category in item_categories[item]:
```

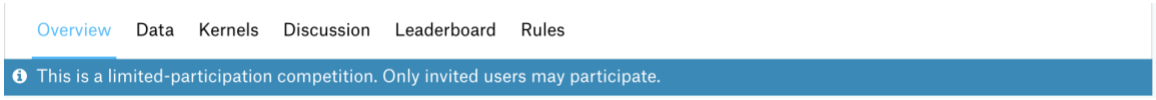
```
        if category in temp:
            repeat_purchase[user]=True
        temp.add(category)
    if user not in repeat_purchase.keys():
        repeat_purchase[user]=False
```

4.

My username is bosicheng. But I can't participate because I'm not invited. (why?) I swear it's my UCSD email.



Here's the issue:



5.

```
allRatings_train = []

userRatings = defaultdict(list)

totalPurchase=0

sum_mse=0

for l in readGz("train.json.gz"):

    user,business = l['reviewerID'],l['itemID']

    if totalPurchase<100000:

        allRatings_train.append(l['rating'])

        userRatings[user].append(l['rating'])

        totalPurchase+=1

    else:

        if totalPurchase==100000:

            globalAverage = sum(allRatings_train) / len(allRatings_train)

        sum_mse+=abs(l['rating']-globalAverage)*abs(l['rating']-globalAverage)

        totalPurchase+=1



globalAverage = sum(allRatings_train) / len(allRatings_train)

userAverage = {}

for u in userRatings:

    userAverage[u] = sum(userRatings[u]) / len(userRatings[u])
```

```
print(globalAverage)

print(sum_mse/100000)
```

```
4.232
1.222481119999121
```

6.

```
import gzip

import numpy as np



from collections import defaultdict



#get the data

def readGz(f):

    for l in gzip.open(f):

        yield eval(l)



allRatings = []

data=[]

userRatings = defaultdict(list)

for l in readGz("train.json.gz"):

    user,business = l['reviewerID'],l['itemID']
```

```
    allRatings.append(l['rating'])
    userRatings[user].append(l['rating'])
    data.append(l)



#shuffle the data
np.random.shuffle(data)



def getPrediction2(alpha,uB,iB,i,j,y_u,y_i,uMap,iMap):
        rating = alpha    + (uB[i] if i in uB else 0)    + (iB[j] if j in iB else 0)
        if i in uMap and j in iMap:
                rating +=np.inner(y_u[uMap[i]],y_i[iMap[j]])
        return rating



#Method to Train The Latent Factor Model. This method doesn't use any Machine Learning library.
def trainLFModel(lam,tData,vData,trials):
        uTrainDict = defaultdict(lambda: defaultdict(int))
        iTrainDict = defaultdict(lambda: defaultdict(int))
        uValidDict = defaultdict(lambda: defaultdict(int))
        # iValidDict = defaultdict(lambda: defaultdict(int))
        uB = defaultdict(float)
```

```
iB = defaultdict(float)

uMap = defaultdict(int)

uCount=0

iMap = defaultdict(int)

iCount=0

for i in tData:

    user, item, rating = i['reviewerID'], i['itemID'], i['rating']

    uTrainDict[user][item] = rating

    iTrainDict[item][user] = rating

    if user not in uMap:

        uMap[user]=uCount

        uCount+=1

    if item not in iMap:

        iMap[item]=iCount

        iCount+=1

for i in vData:

    user, item, rating = i['reviewerID'], i['itemID'], i['rating']

    uValidDict[user][item] = rating



y_u=np.random.normal(scale=1,size=(len(uTrainDict),1))
```

```
y_i=np.random.normal(scale=1,size=(len(iTrainDict),1))



alpha = 0

totalTrials=trials

for _ in range(totalTrials):

    alpha=0

    for i in uTrainDict:

        for j in uTrainDict[i]:

            alpha += uTrainDict[i][j] - uB[i] -iB[j] - np.inner(y_u[uMap[i]],y_i[iMap[j]])

    alpha /= len(tData)

    print(alpha)

    for i in uTrainDict:

        uB[i] = 0

        for j in uTrainDict[i]:

            uB[i] += uTrainDict[i][j]    - alpha - iB[j] - np.inner(y_u[uMap[i]],y_i[iMap[j]])

        uB[i] /= (lam + len(uTrainDict[i]))

    for j in iTrainDict:

        iB[j] = 0

        for i in iTrainDict[j]:

            iB[j] += iTrainDict[j][i]    -alpha - uB[i] - np.inner(y_u[uMap[i]],y_i[iMap[j]])
```

```
                iB[j] /= (lam + len(iTrainDict[j]))



        for i in uTrainDict:

                for lf in range(1):

                        y_u[uMap[i]][lf] = 0

                        for j in uTrainDict[i]:

                                y_u[uMap[i]][lf] += y_i[iMap[j]][lf]*(uTrainDict[i][j]    - alpha - iB[j]
+y_i[iMap[j]][lf]*y_i[iMap[j]][lf]-np.inner(y_u[uMap[i]],y_i[iMap[j]]) )

                                y_u[uMap[i]][lf]    /= (lam + y_i[iMap[j]][lf]*y_i[iMap[j]][lf])

        for j in iTrainDict:

                for lf in range(1):

                        y_i[iMap[j]][lf] = 0

                        for i in iTrainDict[j]:

                                y_i[iMap[j]][lf] += y_u[uMap[i]][lf]*(uTrainDict[i][j]    - alpha - uB[i] -
np.inner(y_u[uMap[i]],y_i[iMap[j]]) +y_u[uMap[i]][lf]*y_u[uMap[i]][lf] )

                                y_i[iMap[j]][lf] /= (lam + y_u[uMap[i]][lf]*y_u[uMap[i]][lf])

    vMSE = 0

    for i in uValidDict:

        for j in uValidDict[i]:

#                   vMSE += ((alpha    + (uB[i] if i in uB else 0)    + (iB[j] if j in iB else 0) - uValidDict[i][j]) **2)
```

```
                vMSE += ((getPrediction2(alpha,uB,iB,i,j,y_u,y_i,uMap,iMap) - uValidDict[i][j]) **2)
    vMSE /= len(vData)
    print (vMSE)
    return vMSE,alpha,uB,iB,uMap,iMap
print("done")



tData=data[:100000]
vData=data[100000:]
lamdas=[1,2,3,5]
trials=[2]
for i in lamdas:
    tempvMSE=1
    for t in trials:
        tempvMSE,alpha,uB,iB,uMap,iMap=trainLFModel(i,tData,vData,t)
        print ("MSE:"+str(tempvMSE))
```

```
MSE: 1.2608578988522912
```

```
7. biggest: 'U536579649', 'U910050838', 'I471768594'
```

```
smallest: 'I011994385', 'U926666668'
```

8.

when λ=5:

```
MSE: 1.1453497407928057
```