hw3_keliu

November 13, 2018

0.1 Problem 1 : purchase prediction of the baseline model

```
In [1]: import gzip
        from collections import defaultdict

In [2]: def readGz(f):
          for l in gzip.open(f):
            yield eval(l)

In [3]: businessCount = defaultdict(int)
        totalPurchases = 0
        count=0

        ###  all user ID and item ID in data
        user_set = []
        item_set = []

        ### already purchased data for validation ( should predicted positive )
        validation_set1 = defaultdict(list)
        user_items = defaultdict(list)
        for l in readGz("train.json.gz"):
          count+=1
          if not l['reviewerID'] in user_set :
                user_set.append(l['reviewerID'])
          if not l['itemID'] in item_set :
                item_set.append(l['itemID'])
          user_items[l['reviewerID']].append(l['itemID'])
          if count<=100000 :
             user,business = l['reviewerID'],l['itemID']
             businessCount[business] += 1
             totalPurchases += 1
          else :
             validation_set1[l['reviewerID']].append(l['itemID'])

In [4]: mostPopular = [(businessCount[x], x) for x in businessCount]
        mostPopular.sort()
        mostPopular.reverse()
```

```
In [5]: ### return "recommend-purchase" according to their popular (bought times)
        return1 = set()
        count = 0
        for ic, i in mostPopular:
          count += ic
          return1.add(i)
          if count > totalPurchases/2: break
```

```
In [6]: ### see the accuracy of baseline model on the text set
        count = 0
        predictions = open("predictions_purchase_base.txt", 'w')
        for l in open("pairs_Purchase.txt"):
          if l.startswith("reviewerID"):
            #header
            predictions.write(l)
            continue
          u,i = l.strip().split('-')
          if i in return1:
            predictions.write(u + '-' + i + ',' + str(1) + '\n')
            count += 1
          else :
            predictions.write(u + '-' + i + ',' + str(0) + '\n')

        predictions.close()
```

```
In [7]: import random
        ### generate randomly 100000 unpurchased user/item pairs

        def generate_random_unpurchased_pair(set1,users,items) :
            set2 = defaultdict(list)
            count = 0
            while count < 100000 :
              uID = users[random.randint(0,len(users)-1)]
              iID = items[random.randint(0,len(items)-1)]
              if (not uID in set1 or not iID in
                  set1[uID]) and (not uID in set2 or not iID in set2[uID]) :
                set2[uID].append(iID)
                count += 1
            return set2
```

```
In [8]: ### another half validation set that should predicted negative
        validation_set2 = generate_random_unpurchased_pair(user_items, user_set, item_set)
```

```
In [9]: ### the performance (accuracy) of the baseline model on the validation set
        err = 0
        for i in validation_set1 :
            for j in validation_set1[i] :
                if not j in return1 :
                    err += 1
        for i in validation_set2 :
            for j in validation_set2[i] :
                if j in return1 :
                    err += 1

        accuracy = (200000 - err)/200000
        print("the accuracy of baseline model is "+str(accuracy))
```

```
the accuracy of baseline model is 0.62919
```

0.2 Problem 2 - a better threshold

There is better a threshold. For example: $0.53 * totalPurchases$, with $acc = 0.629765 > 0.62919$

```
In [10]: parameter = 0.53   ## change parameter here and find the max accuracy
         return2 = set()
         count = 0
         for ic, i in mostPopular:
           count += ic
           return2.add(i)
           if count > totalPurchases*parameter: break


         ### the performance (accuracy) of the baseline model
         ### with a new threshold on the validation set
         err = 0
         for i in validation_set1 :
             for j in validation_set1[i] :
                 if not j in return2 :
                     err += 1
         for i in validation_set2 :
             for j in validation_set2[i] :
                 if j in return2 :
                     err += 1

         ### the 0.5 split:  0.628575
         accuracy1 = (200000 - err)/200000
         print("the accuracy of baseline model with threshold "
               + str(parameter)+" is "+str(accuracy1))
```

```
the accuracy of baseline model with threshold 0.53 is 0.629765
```

0.3 Problem 3 same-category baseline

```
In [32]: user_purchased_category = defaultdict(list)
         item_belong_to_category = defaultdict(list)
         count = 0
         for l in readGz("train.json.gz"):
           count+=1
           for i in l['categories'] :
              if not i in item_belong_to_category[l['itemID']] :
                    item_belong_to_category[l['itemID']].append(i)
              if not i in user_purchased_category[l['reviewerID']] :
                    user_purchased_category[l['reviewerID']].append(i)
           if count> 100000 :
                 break
```

```
In [33]: right = 0
         for i in validation_set1 :
             user_set = user_purchased_category[i]
             for j in validation_set1[i] :
                item_set = item_belong_to_category[j]
                for cat in item_set :
                     if cat in user_set :
                        right += 1
                        break
         error = 0
         for i in validation_set2 :
             user_set = user_purchased_category[i]
             for j in validation_set2[i] :
                item_set = item_belong_to_category[j]
                for cat in item_set :
                     if cat in user_set :
                        error += 1
                        break
         right_all = right + (100000 - error)
         print("accuracy of the same-category model is " + str(right_all/200000))
```

```
accuracy of the same-category model is 0.59234
```

0.4 Problem 4

```
In [34]: predictions = open("predictions_purchase1.txt", 'w')
         for l in open("pairs_Purchase.txt"):
           if l.startswith("reviewerID"):
             #header
             predictions.write(l)
             continue
           u,i = l.strip().split('-')
           user_set = user_purchased_category[u]
           item_set = item_belong_to_category[i]
           flag = False
```

```
            for category in item_set :
                  if category in user_set :
                      flag = True
                      predictions.write(u + '-' + i + ',' + str(1) + '\n')
                      break
            if not flag :
              predictions.write(u + '-' + i + ',' + str(0) + '\n')

        predictions.close()
```

My kaggle name is DaisyL . The result submitted to kaggle is 0.59371

1 Rating Predictions

1.1 Problem 5 - trival predictor

$\alpha = \frac{1}{N} \sum R_{ui}$

```
In [36]: sum_of_rating = 0
         u_i_r =[]

         count = 0
         for l in readGz("train.json.gz"):
             count += 1
             if count<=100000 :
                 sum_of_rating += l['rating']
             u_i_r.append([l['reviewerID'],l['itemID'],l['rating']])
```

```
In [37]: train = u_i_r[:len(u_i_r)//2]
         validation = u_i_r[len(u_i_r)//2:]

         alpha = sum_of_rating/100000
         alpha_global = alpha
         ### MSE

         err = 0
         for i in validation :
             err += (i[2] - alpha)**2
         mse = err/100000

         print("alpha is "+str(alpha))
         print("MSE of trival predictor is "+ str(mse))
```

```
alpha is 4.232
MSE of trival predictor is 1.222481119999121
```

1.2 Problem 6

$$f(u,i) = \alpha + \beta_u + \beta_i$$

```
In [38]: user_bought = defaultdict(list)
         item_customers = defaultdict(list)
         count = 0
         for l in readGz("train.json.gz"):
             count += 1
             if count>100000: break
             user_bought[l['reviewerID']].append([l['itemID'],l['rating']])
             item_customers[l['itemID']].append([l['reviewerID'],l['rating']])
```

```
In [39]: alpha = alpha_global
         sum_ui_all = sum_of_rating
         beta_u = defaultdict(int)
         beta_i = defaultdict(int)
         lamda = 1
         ##  judge from the result, the alpha and beta are all converged
         for k in range(150) :
             b_u_all = 0
             b_i_all = 0
             for ll in train :
                 b_u_all += beta_u[ll[0]]
                 b_i_all += beta_i[ll[1]]

             alpha = (sum_ui_all - (b_u_all + b_i_all))/100000

             for u in user_bought :
                 sum_ui_item = 0
                 alpha_item = 0
                 beta_i_item = 0
                 num_of_item = 0
                 for i in user_bought[u] :
                    sum_ui_item += i[1]
                    alpha_item += alpha
                    beta_i_item += beta_i[i[0]]
                    num_of_item += 1
                 beta_u[u] = (sum_ui_item - alpha_item - beta_i_item)/(lamda + num_of_item)

             for i in item_customers :
                 sum_ui_user = 0
                 alpha_user = 0
                 beta_u_user = 0
                 num_of_user = 0
                 for u in item_customers[i] :
                     sum_ui_user += u[1]
                     alpha_user += alpha
                     beta_u_user += beta_u[u[0]]
```

```
                num_of_user += 1
            beta_i[i] = (sum_ui_user - alpha_user - beta_u_user)/(lamda + num_of_user)
        print("alpha is " +str(alpha))
```

```
alpha is 4.232
alpha is 4.231707482679268
alpha is 4.231366081386166
alpha is 4.230995707891108
alpha is 4.230602266649138
alpha is 4.2301962707913034
alpha is 4.229789921472362
alpha is 4.2293941185337305
alpha is 4.229017191502577
alpha is 4.22866474134191
alpha is 4.228339989547173
alpha is 4.228044285522
alpha is 4.227777606034925
alpha is 4.227538981642016
alpha is 4.227326834313099
alpha is 4.227139231939572
alpha is 4.226974072996426
alpha is 4.226829215858752
alpha is 4.226702565881446
alpha is 4.226592131124168
alpha is 4.226496055334906
alpha is 4.2264126348059925
alpha is 4.226340324072468
alpha is 4.226277734124046
alpha is 4.226223625801517
alpha is 4.226176900291437
alpha is 4.2261365880676935
alpha is 4.226101837211386
alpha is 4.226071901735577
alpha is 4.226046130321279
alpha is 4.226023955713705
alpha is 4.226004884916985
alpha is 4.2259884902489455
alpha is 4.2259744012657015
alpha is 4.225962297531892
alpha is 4.2259519021913325
alpha is 4.22594297628075
alpha is 4.225935313723497
alpha is 4.22592873693856
alpha is 4.225923093001496
alpha is 4.225918250296928
alpha is 4.22591409560633
alpha is 4.22591053157947
alpha is 4.225907474542627
alpha is 4.225904852601477
alpha is 4.225902604001093
alpha is 4.225900675709713
alpha is 4.225899022196913
alpha is 4.225897604380348
alpha is 4.225896388718448
alpha is 4.22589534642932
alpha is 4.225894452818677
alpha is 4.2258936867018315
alpha is 4.225893029906804
alpha is 4.2258924668473155
alpha is 4.2258919841559415
alpha is 4.2258915703690585
alpha is 4.225891215656313
alpha is 4.2258909115883885
alpha is 4.225890650937675
alpha is 4.225890427507218
alpha is 4.225890235983943
alpha is 4.225890071812739
alpha is 4.225889931088439
alpha is 4.22588981046317
alpha is 4.225889707066888
alpha is 4.2258896184392265
alpha is 4.225889542471055
alpha is 4.225889477354367
alpha is 4.225889421539309
alpha is 4.225889373697333
alpha is 4.225889332689612
alpha is 4.225889297539954
alpha is 4.225889267411583
alpha is 4.225889241587234
alpha is 4.225889219452089
alpha is 4.225889200479143
alpha is 4.225889184216672
alpha is 4.22588917027747
alpha is 4.225889158329647
alpha is 4.2258891480887195
alpha is 4.225889139310843
alpha is 4.225889131787007
alpha is 4.225889125338057
alpha is 4.225889119810435
alpha is 4.225889115072516
alpha is 4.225889111011483
alpha is 4.225889107530633
alpha is 4.225889104547077
alpha is 4.225889101989772
alpha is 4.2258890997978185
alpha is 4.225889097919024
alpha is 4.225889096308646
alpha is 4.225889094928338
alpha is 4.225889093745229
alpha is 4.2258890927311485
alpha is 4.225889091861947
alpha is 4.225889091116926
alpha is 4.225889090478344
alpha is 4.2258890899309955
alpha is 4.2258890894618455
alpha is 4.225889089059722
alpha is 4.225889088715047
alpha is 4.225889088419616
alpha is 4.225889088166393
alpha is 4.225889087949347
alpha is 4.225889087763309
alpha is 4.225889087603851
alpha is 4.225889087467174
alpha is 4.2258890873500246
alpha is 4.225889087249612
alpha is 4.225889087163544
alpha is 4.225889087089774
alpha is 4.225889087026542
alpha is 4.225889086972344
alpha is 4.22588908692589
alpha is 4.225889086886072
alpha is 4.225889086851942
alpha is 4.225889086822689
alpha is 4.225889086797615
alpha is 4.225889086776124
alpha is 4.225889086757703
alpha is 4.225889086741914
alpha is 4.2258890867283805
alpha is 4.225889086716781
alpha is 4.225889086706838
alpha is 4.225889086698316
alpha is 4.22588908669101
alpha is 4.22588908668475
alpha is 4.225889086679383
alpha is 4.225889086674783
alpha is 4.225889086670841
alpha is 4.225889086667462
alpha is 4.225889086664565
alpha is 4.2258890866620815
alpha is 4.225889086659953
alpha is 4.225889086658128
alpha is 4.225889086656565
alpha is 4.225889086655226
alpha is 4.225889086654078
alpha is 4.225889086653094
alpha is 4.225889086652251
alpha is 4.225889086651527
alpha is 4.225889086650908
alpha is 4.225889086650376
alpha is 4.225889086649921
alpha is 4.22588908664953
alpha is 4.225889086649195
alpha is 4.225889086648909
alpha is 4.225889086648663
```

```
In [40]: ### MSE
         error = 0
         for i in validation :
             predict_rating = alpha + beta_u[i[0]] + beta_i[i[1]]
             error += (predict_rating - i[2])**2
         mse1 = error/len(validation)
         print("the MSE of the validation set is " + str(mse1))
```

```
the MSE of the validation set is 1.281118785404059
```

1.3 Problem 7 - the ID with the largest bias

```
In [43]: high_beta_u,low_beta_u,high_beta_i,low_beta_i = 0,0,0,0
         hi,lo = -10,10

         for i in beta_u :
             if beta_u[i] > hi :
                 high_beta_u = i
                 hi = beta_u[i]
             if beta_u[i] < lo :
                 low_beta_u = i
                 lo = beta_u[i]
         hi1,lo1 = -10,10
         for i in beta_i :
             if beta_i[i] > hi1 :
                 high_beta_i = i
                 hi1 = beta_i[i]
             if beta_i[i] < lo1 :
                 low_beta_i = i
                 lo1 = beta_i[i]

         print("max User ID    "+str(high_beta_u)+' with beta equals ' +str(hi))
         print("min User ID    "+str(low_beta_u)+' with beta equals ' +str(lo))
         print("max item ID    "+str(high_beta_i)+' with beta equals ' +str(hi1))
         print("min item ID    "+str(low_beta_i)+' with beta equals ' +str(lo1))
```

```
max User ID    U495776285 with beta equals 1.4867494280299913
min User ID    U204516481 with beta equals -2.551703536909886
max item ID    I809804570 with beta equals 1.270014825286867
min item ID    I511389419 with beta equals -2.575711953421698
```

1.4 Problem 8 - find better lambda and MSE

```
In [44]: def findLamda(lamda,alpha,user_bought,item_customers ) :
             alpha = alpha
             beta_u = defaultdict(int)
             beta_i = defaultdict(int)
           #  lamda = 1
             for k in range(200) :
                sum_ui_all = sum_of_rating
                b_u_all = 0
                b_i_all = 0
                for ll in train :
                    b_u_all += beta_u[ll[0]]
                    b_i_all += beta_i[ll[1]]

                alpha = (sum_ui_all - (b_u_all + b_i_all))/100000

                for u in user_bought :
                   sum_ui_item = 0
                   alpha_item = 0
                   beta_i_item = 0
                   num_of_item = 0
                   for i in user_bought[u] :
                      sum_ui_item += i[1]
                      alpha_item += alpha
                      beta_i_item += beta_i[i[0]]
                      num_of_item += 1
                   beta_u[u] = (sum_ui_item - alpha_item - beta_i_item)/(lamda + num_of_item)

                for i in item_customers :
                   sum_ui_user = 0
                   alpha_user = 0
                   beta_u_user = 0
                   num_of_user = 0
                   for u in item_customers[i] :
                      sum_ui_user += u[1]
                      alpha_user += alpha
                      beta_u_user += beta_u[u[0]]
                      num_of_user += 1
                   beta_i[i] = (sum_ui_user - alpha_user - beta_u_user)/(lamda + num_of_user)

             error = 0
```

```
        for i in validation :
            predict_rating = alpha + beta_u[i[0]] + beta_i[i[1]]
            error += (predict_rating - i[2])**2
        mse1 = error/len(validation)
        print("the MSE of the validation set with lamda " + str(lamda) +" is " + str(mse1
```

```
In [67]: findLamda(0.1,alpha_global,user_bought,item_customers)
         findLamda(1,alpha_global,user_bought,item_customers)
         findLamda(10,alpha_global,user_bought,item_customers)
         findLamda(100,alpha_global,user_bought,item_customers)
```

```
the MSE of the validation set with lamda 0.1 is 1.7488629120006902
the MSE of the validation set with lamda 1 is 1.281118785404067
the MSE of the validation set with lamda 10 is 1.1416030979547038
the MSE of the validation set with lamda 100 is 1.1998248086659447
```

```
In [45]: findLamda(3,alpha_global,user_bought,item_customers)
         findLamda(5,alpha_global,user_bought,item_customers)
         findLamda(7,alpha_global,user_bought,item_customers)
```

```
the MSE of the validation set with lamda 3 is 1.1583185109618461
the MSE of the validation set with lamda 5 is 1.1398956065799264
the MSE of the validation set with lamda 7 is 1.1377680309690605
```

```
In [46]: findLamda(6,alpha_global,user_bought,item_customers)
         findLamda(8,alpha_global,user_bought,item_customers)
```

```
the MSE of the validation set with lamda 6 is 1.1379239574804347
the MSE of the validation set with lamda 8 is 1.1385919338283539
```

Through comparing, we found when lamda is 7, the MSE is lower. So choose lambda = 7 and the MSE is 1.13792.

```
In [48]: alpha = alpha_global
         beta_u = defaultdict(int)
         beta_i = defaultdict(int)
         lamda = 7
         for k in range(200) :
             sum_ui_all = sum_of_rating
             b_u_all = 0
             b_i_all = 0
             for ll in train :
                 b_u_all += beta_u[ll[0]]
                 b_i_all += beta_i[ll[1]]
```

```
            alpha = (sum_ui_all - (b_u_all + b_i_all))/100000

            for u in user_bought :
                sum_ui_item = 0
                alpha_item = 0
                beta_i_item = 0
                num_of_item = 0
                for i in user_bought[u] :
                   sum_ui_item += i[1]
                   alpha_item += alpha
                   beta_i_item += beta_i[i[0]]
                   num_of_item += 1
                beta_u[u] = (sum_ui_item - alpha_item - beta_i_item)/(lamda + num_of_item)

            for i in item_customers :
                sum_ui_user = 0
                alpha_user = 0
                beta_u_user = 0
                num_of_user = 0
                for u in item_customers[i] :
                    sum_ui_user += u[1]
                    alpha_user += alpha
                    beta_u_user += beta_u[u[0]]
                    num_of_user += 1
                beta_i[i] = (sum_ui_user - alpha_user - beta_u_user)/(lamda + num_of_user)
```

```
In [49]: predictions_rating = open("predictions_rating8.txt", 'w')
         for l in open("pairs_rating.txt"):
           if l.startswith("reviewerID"):
             #header
             predictions_rating.write(l)
             continue
           u,i = l.strip().split('-')
           rting = alpha + beta_u[u] +beta_i[i]
           predictions_rating.write(u + '-' + i + ',' + str(rting) + '\n')

         predictions_rating.close()
```

the submission resulit is 1.18462

```
In [ ]:
```