CSE 258, Homework 3 ¶

By Amogha Sekhar, A53301791

Tasks (Read prediction)

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

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

def readCSV(path):
    f = gzip.open(path, 'rt')
    f.readline()
    for l in f:
        yield l.strip().split(',')

### Rating baseline: compute averages for each user, or return the global a

allRatings = []
userRatings = defaultdict(list)

for user,book,r in readCSV("train_Interactions.csv.gz"):
    r = int(r)
    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_Rating.txt", 'w')
for l in open("pairs_Rating.txt"):
    if l.startswith("userID"):
    #header
        predictions.write(l)
        continue
    u,b = l.strip().split('-')
    if u in userAverage:
        predictions.write(u + '-' + b + ',' + str(userAverage[u]) + '\n')
    else:
        predictions.write(u + '-' + b + ',' + str(globalAverage) + '\n')

predictions.close()

### Would-read baseline: just rank which books are popular and which are no

bookCount = defaultdict(int)
totalRead = 0

for user,book,_ in readCSV("train_Interactions.csv.gz"):
    bookCount[book] += 1
    totalRead += 1

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

return1 = set()
```

```
count = 0
for ic, i in mostPopular:
    count += ic
    return1.add(i)
    if count > totalRead/2: break

predictions = open("predictions_Read.txt", 'w')
for l in open("pairs_Read.txt"):
    if l.startswith("userID"):
    #header
        predictions.write(l)
        continue
    u,b = l.strip().split('-')
    if b in return1:
        predictions.write(u + '-' + b + ",1\n")
    else:
        predictions.write(u + '-' + b + ",0\n")

predictions.close()
```

Question 1: Although we have built a validation set, it only consists of positive samples. For this task we also need examples of user/item pairs that weren't read. For each entry (user,book) in the validation set, sample a negative entry by randomly choosing a book that user hasn't read.1 Evaluate the performance (accuracy) of the baseline model on the validation set you have built (1 mark).

In [2]:

```
allusers= []
allbooks= []
allratings= []
for u, b, r in readCSV("train_Interactions.csv.gz"):
    allusers.append(u)
    allbooks.append(b)
    allratings.append(r)

#print(len(user))
```

In [3]:

```
X= []
y= []
for i in range(len(allusers)):
    X.append([allusers[i], allbooks[i]])
    y.append(1)
```

In [4]:
```
X_train= X[:190000]
X_valid= X[190000:]
y_train= y[:190000]
y_valid= y[190000:]
```

In [5]:
```
userBooks = defaultdict(list)
for user,book,r in readCSV("train_Interactions.csv.gz"):
    userBooks[user].append(book)
print(len(userBooks))
```

```
11357
```

In [6]:
```
#Generating negative instances for the validation set
import random

def Diff(li1, li2):
    return (list(set(li1) - set(li2)))

count= 0

for u, b in X_valid:
    #print(count)

    user_books= userBooks[u]
    temp= Diff(allbooks,user_books)

    book= random.choice(temp)
    X_valid.append([user, book])
    count+=1

    if count==10000:
        break

i=0
while i<10000:
    y_valid.append(0)
    i+=1

print(len(y_valid))
print(len(X_valid))
```

```
20000
20000
```

In [7]:
```
#Baseline function
pred= []
def predict_read(X):
    for i in range(len(X)):
        u = X[i][0]
        b = X[i][1]
        if b in return1:
            pred.append(1)
        else:
            pred.append(0)

predict_read(X_valid)

#accuracy

c= 0
for i in range(len(y_valid)):
    if(y_valid[i] == pred[i]):
        c+= 1

print(c/len(y_valid))
```

```
0.65045
```

Answer for Question 1:

The accuracy on the validation set is 65.045%.

Question 2: The existing ‘read prediction’ baseline just returns True if the item in question is ‘popular,’ using a threshold of the 50th percentile of popularity (totalRead/2). Assuming that the ‘non-read’ test examples are a random sample of user-book pairs, this threshold may not be the best one. See if you can find a better threshold and report its performance on your validatin set (1 mark)

In [8]:
```
pred= []
bookCount = defaultdict(int)
totalRead = 0

for user,book,_ in readCSV("train_Interactions.csv.gz"):
    bookCount[book] += 1
    totalRead += 1

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

return1 = set()
count = 0
for ic, i in mostPopular:
    count += ic
    return1.add(i)
    if count > 2*totalRead/3: break


predict_read(X_valid)

#accuracy

c= 0
for i in range(len(y_valid)):
    if(y_valid[i] == pred[i]):
        c+= 1

print(c/len(y_valid))
```

```
0.65315
```

Answer for Question 2:

A better threshhold will be 66.67th percentile of popularity. (2* totalRead/3) The performance on the validation set goes up from 65.045% to 65.315%.

Question 3: A stronger baseline than the one provided might make use of the Jaccard similarity (or another similarity metric). Given a pair (u, b) in the validation set, consider all training items b 0 that user u has read. For each, compute the Jaccard similarity between b and b 0 , i.e., users (in the training set) who have read b and users who have read b 0. Predict as ‘read’ if the maximum of these Jaccard similarities exceeds a threshold (you may choose the threshold that works best). Report the performance on your validation set (1 mark).

In [9]:
```
bookUsers = defaultdict(list)
for user,book,r in readCSV("train_Interactions.csv.gz"):
    bookUsers[book].append(user)
print(len(bookUsers))
```

```
7170
```

In [10]:
```
pred_jaccard= []
def Jaccard(b1, b2):
    s1= bookUsers[b1] #set of users who have read b1
    s2= bookUsers[b2] #set of users who have read b2
    s1= set(s1)
    s2= set(s2)
    numer = len(s1.intersection(s2))
    denom = len(s1.union(s2))
    return numer / denom


def mostSimilar(X):
    for i in range(len(X)):
        u = X[i][0]
        b = X[i][1]
        user_books= userBooks[u] #all the books user u has read
        similarities = []
        for j in range(len(user_books)):
            if b == user_books[j]:
                continue
            sim= Jaccard(b, user_books[j])
            similarities.append(sim)
        if max(similarities)> 0.02:
            pred_jaccard.append(1)
        else:
            pred_jaccard.append(0)


mostSimilar(X_valid)
```

In [11]:
```
cj= 0

for i in range(len(y_valid)):
    if(y_valid[i] == pred_jaccard[i]):
        cj+= 1

print(cj/len(y_valid))
```

```
0.76405
```

Answer for Question 3:

The performance on the validation set was maximum at a threshhold of 0.02. The accuracy was 76.405%.

Question 4: Improve the above predictor by incorporating both a Jaccard-based threshold and a popularity based threshold. Report the performance on your validation set (1 mark).

In [12]:
```
#need to consider ensembling model combining popularity and jaccard

print(y_valid[:20])
print(pred[:20])
print(pred_jaccard[:20])
```

```
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
[1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1]
[1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1]
```

In [13]:
```
predv= []

return1 = set()
count = 0
for ic, i in mostPopular:
    count += ic
    return1.add(i)
    if count > 2*totalRead/3: break


for i in range(len(X_valid)):
    u = X_valid[i][0]
    b = X_valid[i][1]
    user_books= userBooks[u] #all the books user u has read
    similarities = []
    for j in range(len(user_books)):
        if b == user_books[j]:
                continue
        sim= Jaccard(b, user_books[j])
        similarities.append(sim)
    if max(similarities)> 0.01 and b in return1:
        predv.append(1)
    elif max(similarities)> 0.035 and b not in return1:
        predv.append(1)
    else:
        predv.append(0)


cv= 0

for i in range(len(y_valid)):
    if(y_valid[i] == predv[i]):
        cv+= 1

print(cv/len(y_valid))
```

```
0.7985
```

Answer for Question 4:

By combining popularity and Jaccard-similarity, the accuracy on the validation set is 79.85%, which is an improvement from 76.405%.

Question 5: To run our model on the test set, we'll have to use the files 'pairs Read.txt' to find the reviewerID/itemID pairs about which we have to make predictions. Using that data, run the above model and upload your solution to Kaggle. Tell us your Kaggle user name (1 mark). If you've already uploaded a better solution to Kaggle, that's fine too!

Answer for Question 5:

Username: amoghasekhar

Display name: Amogha Sekhar

Name on leaderboard: Amogha Sekhar

Email: amogha.sekhar@gmail.com (mailto:amogha.sekhar@gmail.com)

Submitted my solution on Kaggle.

Tasks (Rating prediction)

9. Fit a predictor of the form rating(user, item) ' α + βuser + βitem, by fitting the mean and the two bias terms as described in the lecture notes. Use a regularization parameter of λ = 1. Report the MSE on the validation set (1 mark).

In [14]:
```
#Getting the data in the form we need
import numpy

data= []
for user,book,r in readCSV("train_Interactions.csv.gz"):
    data.append([user, book, int(r)])

train= data[:190000]
valid= data[190000:]

#print(train[:10])
```

In [15]:
```
#function to get the features of the training set: namely alpha, beta_u, be
def getFeatures(data):
    alpha = 0*random.random()
    betau = {}
    betai = {}
    itemsWithUser = {}
    usersWithItem = {}
    ratings = defaultdict(dict)
    for i in range(len(data)):
        itemid = data[i][1]
        userid = data[i][0]
        rating = int(data[i][2])
        betau[userid] = 0*random.random();
        betai[itemid] = 0*random.random();

        if userid in itemsWithUser:
            itemsWithUser[userid].append(itemid)
        else:
            itemsWithUser[userid] = [itemid]

        if itemid in usersWithItem:
            usersWithItem[itemid].append(userid)
        else:
            usersWithItem[itemid] = [userid]

        ratings[userid][itemid] = rating
    return alpha, betau, betai, itemsWithUser, usersWithItem, ratings
```

In [16]:
```
#function to train the parameters based on the update rule in lecture notes
def trainParam(alpha, betau, betai, itemsWithUser, usersWithItem, ratings,
    N = 0
    alpha = 0
    for user in ratings:
        for item in ratings[user]:
            alpha = alpha + (ratings[user][item] - betau[user] - betai[item
            N = N + 1
    alpha = alpha/N

    for user, items in itemsWithUser.items():
        betaUpdate = 0
        l = len(items)
        for item in items:
            betaUpdate = betaUpdate + (ratings[user][item] - alpha - betai[
        betau[user] = betaUpdate/(lmbda + l)

    for item, users in usersWithItem.items():
        betaUpdate = 0
        l = len(users)
        for user in users:
            betaUpdate = betaUpdate + (ratings[user][item] - alpha - betau[
        betai[item] = betaUpdate/(lmbda + l)

    return alpha
```

In [17]:
```
#function to form the test set
def getTestFeatures(data):
    features = []
    for i in range(len(data)):
        feature = []
        itemid = data[i][1]
        userid = data[i][0]
        rating = data[i][2]
        feature.append(userid)
        feature.append(itemid)
        feature.append(rating)
        features.append(feature)
    return features
```

In [18]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error

#function to find MSE on the validation set
def validation(features, alpha, betau, betai):
    predictions = []
    true = []
    for feature in features:
            prediction = alpha
            if feature[0] in betau:
                prediction = prediction + betau[feature[0]]
            if feature[1] in betai:
                prediction = prediction + betai[feature[1]]
            predictions.append(prediction)
            true.append(feature[-1])
    predictions = numpy.array(predictions)
    true = numpy.array(true)
    return mean_squared_error(true, predictions)
```

In [19]:
```
alpha, betau, betai, itemsWithUser, usersWithItem, ratings = getFeatures(tr
```

In [20]:
```
alpha= trainParam(alpha, betau, betai, itemsWithUser, usersWithItem, rating
```

In [21]:
```
features= getTestFeatures(valid)
```

In [22]:
```
mse= validation(features, alpha, betau, betai)
print(mse)
```

```
1.1203483020335931
```

Answer for Question 9:

The MSE on the validation set is 1.1203483020335931.

Question 10: Report the user and book IDs that have the largest and smallest values of β (1 mark).

In [23]:
```
maxbetauser = ''
temp = -1
for user in betau:
    if betau[user] > temp:
        maxbetauser = user
        temp = betau[user]

minbetauser = ''
temp = 1000
for user in betau:
    if betau[user] < temp:
        minbetauser = user
        temp = betau[user]

maxbetaitem = ''
temp = -1
for item in betai:
    if betai[item] > temp:
        maxbetaitem = item
        temp = betai[item]

minbetaitem = ''
temp = 1000
for item in betai:
    if betai[item] < temp:
        minbetaitem = item
        temp = betai[item]


print (maxbetauser + " : " + str(betau[maxbetauser]) + " " + minbetauser +
print (maxbetaitem + " : " + str(betai[maxbetaitem]) + " " + minbetaitem +
print (maxbetauser in ratings)
print (minbetauser in ratings)
```

```
u81539151 : 1.0835301939058184 u76571258 : -3.7672170175438575
b19925500 : 1.1720514785959866 b84091840 : -1.6952980452472723
True
True
```

Answer for Question 10:

The user ID which has the smallest beta value= u76571258

The user ID which has the largest beta value= u81539151

The book ID which has the smallest beta value= b84091840

The book ID which has the largest beta value= b19925500

Question 11: Find a better value of λ using your validation set. Report the value you chose, its MSE, and upload your solution to Kaggle by running it on the test data (1 mark).

In [24]:
```
for lmbda in range(1,10):
    alpha, betau, betai, itemsWithUser, usersWithItem, ratings = getFeature
    features = getTestFeatures(valid)
    prev = 100;
    while(True):
        mse = validation(features, alpha, betau, betai)
        if prev - mse < .000000001:
            break
        prev = mse
        alpha = trainParam(alpha, betau, betai, itemsWithUser, usersWithIte
    print (lmbda, mse)
```

```
1 1.1159069673742232
2 1.1099085617893674
3 1.1080652400772688
4 1.1087757564475635
5 1.111117718270156
6 1.1145221947938972
7 1.1186217976272135
8 1.1231713328901287
9 1.1280031181106889
```

Answer for Question 11:

As can been seen from above:

When lambda=2, MSE= 1.1099085617893674

When lambda= 3, MSE= 1.1080652400772688

When lambda= 4, MSE= 1.1087757564475635

When lambda= 5, MSE= 1.111117718270156

When lambda= 6, MSE= 1.1145221947938972

Basically, any of these values of lambda will work!

The best value is lambda=3, and I have uploaded my solution on kaggle.

In [ ]: