HW3

November 13, 2019

[1]:
```
import gzip
from collections import defaultdict
import random
import numpy as np
```

[2]:
```
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(',')
```

[3]:
```
users = []
books = []

ub_all = defaultdict(set)
bu_train = defaultdict(set)
ub_train = defaultdict(set)
ub_validP = defaultdict(set)
popularity = defaultdict(int)

totalRead = 190000
count = 0

for user,book,r in readCSV("train_Interactions.csv.gz"):
    count += 1
    if not user in users:
        users.append(user)
    if not book in books:
        books.append(book)
    ub_all[user].add(book)
    if count <= 190000:
        ub_train[user].add(book)
        bu_train[book].add(user)
```

```
        popularity[book] += 1
    else:
        ub_validP[user].add(book)

sortedBook = sorted(popularity.items(), key=lambda x: x[1], reverse = True)

validN = defaultdict(set)
for user in ub_validP:
    for book in ub_validP[user]:
        newBook = books[random.randint(0,len(books)-1)]
        while (newBook in ub_all[user] or newBook in validN[user]):
            newBook = books[random.randint(0,len(books)-1)]
        validN[user].add(newBook)
```

[4]:
```
def mostPop(threshold):
    mostpop = []
    curRead = 0
    for book in sortedBook:
        curRead += book[1]
        mostpop.append(book[0])
        if (curRead > totalRead * threshold):
            break
    return mostpop
```

[5]:
```
def mostPopAcc(threshold):
    mostPopBook = mostPop(threshold)
    error = 0
    validNum = 0
    for user in ub_validP:
        for book in ub_validP[user]:
            validNum += 1
            if not book in mostPopBook:
                error += 1
    for user in validN:
        for book in validN[user]:
            validNum += 1
            if book in mostPopBook:
                error += 1
    accuracy = (validNum - error) / validNum
    return accuracy

print(mostPopAcc(0.5))
```

```
0.64355
```

1. The accuracy of baseline method shows the accuracy of 0.64355

[6]:
```
for i in np.arange(0.5, 0.7, 0.01):
    print("The accuracy of baseline method with threshold of %.2f is %.
 ↪5f"%(i,mostPopAcc(i)))
```

```
The accuracy of baseline method with threshold of 0.50 is 0.64355
The accuracy of baseline method with threshold of 0.51 is 0.64340
The accuracy of baseline method with threshold of 0.52 is 0.64325
The accuracy of baseline method with threshold of 0.53 is 0.64425
The accuracy of baseline method with threshold of 0.54 is 0.64590
The accuracy of baseline method with threshold of 0.55 is 0.64880
The accuracy of baseline method with threshold of 0.56 is 0.64935
The accuracy of baseline method with threshold of 0.57 is 0.65005
The accuracy of baseline method with threshold of 0.58 is 0.64900
The accuracy of baseline method with threshold of 0.59 is 0.64780
The accuracy of baseline method with threshold of 0.60 is 0.64700
The accuracy of baseline method with threshold of 0.61 is 0.64765
The accuracy of baseline method with threshold of 0.62 is 0.64675
The accuracy of baseline method with threshold of 0.63 is 0.64650
The accuracy of baseline method with threshold of 0.64 is 0.64540
The accuracy of baseline method with threshold of 0.65 is 0.64495
The accuracy of baseline method with threshold of 0.66 is 0.64420
The accuracy of baseline method with threshold of 0.67 is 0.64325
The accuracy of baseline method with threshold of 0.68 is 0.64160
The accuracy of baseline method with threshold of 0.69 is 0.64040
```

2. From above, we can know that we can find a better threshold at 0.57, with the accuracy of 0.650005

[7]:
```
def Jaccard(s1, s2):
    numer = len(s1.intersection(s2))
    denom = len(s1.union(s2))
    return numer / denom

def similarity(user, book):
    similarities = []
    for otherBook in ub_train[user]: # For all items
        if otherBook == book: continue # other than the query
        sim = Jaccard(bu_train[book], bu_train[otherBook])
        similarities.append((sim,otherBook))
    similarities.sort(reverse=True)
    return similarities[0][0]
```

[8]:
```
def JaccardAcc(threshold):
    error = 0
    validNum = 0
    for user in ub_validP:
        for book in ub_validP[user]:
            validNum += 1
```

```
            if similarity(user, book) < threshold:
                error += 1
    for user in validN:
        for book in validN[user]:
            validNum += 1
            if similarity(user, book) >= threshold:
                error += 1
    accuracy = (validNum - error) / validNum
    return accuracy
```

[9]:
```
for i in np.arange(0, 0.1, 0.01):
    print("The accuracy of jaccard-based method with threshold of %.2f is %.
↪5f"%(i,JaccardAcc(i)))
```

```
The accuracy of jaccard-based method with threshold of 0.00 is 0.50000
The accuracy of jaccard-based method with threshold of 0.01 is 0.62265
The accuracy of jaccard-based method with threshold of 0.02 is 0.57715
The accuracy of jaccard-based method with threshold of 0.03 is 0.52935
The accuracy of jaccard-based method with threshold of 0.04 is 0.51250
The accuracy of jaccard-based method with threshold of 0.05 is 0.50555
The accuracy of jaccard-based method with threshold of 0.06 is 0.50210
The accuracy of jaccard-based method with threshold of 0.07 is 0.50100
The accuracy of jaccard-based method with threshold of 0.08 is 0.50025
The accuracy of jaccard-based method with threshold of 0.09 is 0.50010
```

3. We can show the performance of Jaccard-based method of different threshold as shown above. The better threshold is at 0.01, with the accuracy of 0.62265.

[10]:
```
def mostPopAndJacAcc(th_mp, th_jc):
    error = 0
    validNum = 0
    mostPopBook = mostPop(th_mp)
    for user in ub_validP:
        for book in ub_validP[user]:
            validNum += 1
            if similarity(user, book) < th_jc and not book in mostPopBook:
                error += 1
    for user in validN:
        for book in validN[user]:
            validNum += 1
            if similarity(user, book) >= th_jc or book in mostPopBook:
                error += 1
    accuracy = (validNum - error) / validNum
    return accuracy
```

[162]:
```
print(mostPopAndJacAcc(0.57, 0.03))
```

```
0.66275
```

4. By combine the baseline method with jaccard-based as shown above, we can get a imporved accuracy of validation set as 0.66275.

[12]:
```
predictions = open("predictions_Read.txt", 'w')
mostPopBook = mostPop(0.55)
for l in open("pairs_Read.txt"):
    if l.startswith("userID"):
        #header
        predictions.write(l)
        continue
    u,b = l.strip().split('-')
    if similarity(u, b) >= 0.03 or book in mostPopBook:
        predictions.write(u + '-' + b + ",1\n")
    else:
        predictions.write(u + '-' + b + ",0\n")

predictions.close()
```

5. The solution is gotten as above. My Kaggle user name is bbylzyw0524

[168]:
```
allRatings = []
userRatings = defaultdict(list)
bookRatings = defaultdict(list)
user_book = defaultdict(list)
book_user = defaultdict(list)
valid = []

count = 0

for user,book,r in readCSV("train_Interactions.csv.gz"):
    count += 1
    r = int(r)
    if count <= 190000:
        allRatings.append(r)
        userRatings[user].append(r)
        bookRatings[book].append(r)
        user_book[user].append(book)
        book_user[book].append(user)
    else:
        l = []
        l.append(user)
        l.append(book)
        l.append(r)
        valid.append(l)

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

```
    userAverage[u] = sum(userRatings[u]) / len(userRatings[u])
userSum = {}
for u in userRatings:
    userSum[u] = sum(userRatings[u])
bookSum = {}
for b in bookRatings:
    bookSum[b] = sum(bookRatings[b])

alpha = globalAverage
beta_u = defaultdict(int)
beta_b = defaultdict(int)
lambda_ = 1
```

[169]:
```
for u in userRatings:
    beta_u[u] = 0
for b in bookRatings:
    beta_b[b] = 0

for i in range(100):
    sum_beta_u = 0
    sum_beta_b = 0
#     beta_u_new = defaultdict(int)
#     beta_b_new = defaultdict(int)
#     for u in userRatings:
#         beta_u_new[u] = 0
#     for b in bookRatings:
#         beta_b_new[b] = 0
    for user in user_book:
        for book in user_book[user]:
            sum_beta_u += beta_u[user]
            sum_beta_b += beta_b[book]
    alpha = (sum(allRatings) - (sum_beta_b + sum_beta_u)) / 190000
    for user in user_book:
    #         print(userRatings[user])
        sum_ratings_u = userSum[user]
    #         print(sum_ratings_u)
        numOfBook = len(userRatings[user])
#         print(numOfBook)
        sum_beta_b = 0
        for book in user_book[user]:
            sum_beta_b += beta_b[book]
        beta_u[user] = (sum_ratings_u - alpha * numOfBook - sum_beta_b) /␣
↪(lambda_ + numOfBook)
    #         print(beta_u_new[user])

    for book in book_user:
        sum_ratings_b = bookSum[book]
```

```
        numOfUser = len(bookRatings[book])
#         print(numOfUser)
        sum_beta_u = 0
        for user in book_user[book]:
            sum_beta_u += beta_u[user]
        beta_b[book] = (sum_ratings_b - alpha * numOfUser - sum_beta_u) /␣
 ↪(lambda_ + numOfUser)

#     alpha = alpha_new
#     beta_u = beta_u_new
#     beta_b = beta_b_new
```

[171]:
```
sum_err = 0
for i in valid:
    predict_rating = alpha + beta_u[i[0]] + beta_b[i[1]]
#     print(predict_rating)
    sum_err += (predict_rating - i[2]) ** 2
MSE = sum_err / len(valid)
print("mse = ", MSE)
```

```
mse =  1.1159080829077774
```

9. The MSE on the validation set is 1.1159080829077774

[172]:
```
beta_u_num = [beta_u[user] for user in beta_u]
beta_b_num = [beta_b[book] for book in beta_b]
```

[173]:
```
print(min(beta_u_num), max(beta_u_num))
print(min(beta_b_num), max(beta_b_num))
```

```
-3.7467379856073335 1.3233584115575237
-1.7574168637067396 1.4265543793743276
```

10. The largest and smallest value for beta_u is 1.3233584115575237 and -3.7467379856073335. And the largest and smallest value for beta_b is 1.4265543793743276 and -1.7574168637067396

[174]:
```
alpha = globalAverage
beta_u = defaultdict(int)
beta_b = defaultdict(int)
lambda_ = 3

for u in userRatings:
    beta_u[u] = 0
for b in bookRatings:
    beta_b[b] = 0

for i in range(100):
    sum_beta_u = 0
```

```
    sum_beta_b = 0
    for user in user_book:
        for book in user_book[user]:
            sum_beta_u += beta_u[user]
            sum_beta_b += beta_b[book]
    alpha = (sum(allRatings) - (sum_beta_b + sum_beta_u)) / 190000
    for user in user_book:
        sum_ratings_u = userSum[user]
        numOfBook = len(userRatings[user])
        sum_beta_b = 0
        for book in user_book[user]:
            sum_beta_b += beta_b[book]
        beta_u[user] = (sum_ratings_u - alpha * numOfBook - sum_beta_b) /␣
↪(lambda_ + numOfBook)

    for book in book_user:
        sum_ratings_b = bookSum[book]
        numOfUser = len(bookRatings[book])
        sum_beta_u = 0
        for user in book_user[book]:
            sum_beta_u += beta_u[user]
        beta_b[book] = (sum_ratings_b - alpha * numOfUser - sum_beta_u) /␣
↪(lambda_ + numOfUser)
```

[175]:
```
sum_err = 0
for i in valid:
    predict_rating = alpha + beta_u[i[0]] + beta_b[i[1]]
#     print(predict_rating)
    sum_err += (predict_rating - i[2]) ** 2
MSE = sum_err / len(valid)
print("mse = ", MSE)
```

```
mse =  1.1080652410692764
```

11. I choose the $\lambda = 3$. And the MSE on the validation set is 1.1080652410692764. We get the ratings of test data as below

[176]:
```
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 userRatings and b in bookRatings:
        predict_rating = alpha + beta_u[u] + beta_b[b]
        predictions.write(u + '-' + b + ',' + str(predict_rating) + '\n')
    elif u in userRatings:
```

```
        predict_rating = alpha + beta_u[u]
        predictions.write(u + '-' + b + ',' + str(predict_rating) + '\n')
    elif b in bookRatings:
        predict_rating = alpha + beta_b[b]
        predictions.write(u + '-' + b + ',' + str(predict_rating) + '\n')
    else:
        predict_rating = alpha
        predictions.write(u + '-' + b + ',' + str(predict_rating) + '\n')
```