homework_2

October 18, 2021

1 Homework 2

1.1 Tasks — Similarity Functions

1.1.1 Task 1

[2]:
```
import pandas as pd

URL = 'https://cseweb.ucsd.edu//classes/fa21/cse258-b/data/
↪goodreads_reviews_comics_graphic.json.gz'
reviews = pd.read_json(URL, lines=True)
```

[3]:
```
def Jaccard(s1, s2):
    numerator = len(s1.intersection(s2))
    denominator = len(s1.union(s2))
    if(denominator == 0):
        return 0
    return numerator/denominator

def mostSimilar(item, K):
    similarities = []
    users = usersPerItem[item]
    for i in usersPerItem:
        if (i==item): continue
        similarity = Jaccard(users, usersPerItem[i])
        similarities.append((similarity, i))
    similarities.sort(reverse=True)
    return similarities[:K]
```

[4]:
```
from collections import defaultdict

usersPerItem = defaultdict(set)
itemsPerUser = defaultdict(set)
ratingDict = {}
timeDict = {}

ratingsPerItem = defaultdict(list)
```

```
for index in reviews.index:
    user, item, rating, time = reviews['user_id'][index],␣
↪reviews['book_id'][index], reviews['rating'][index],␣
↪reviews['date_added'][index]
    usersPerItem[item].add(user)
    itemsPerUser[user].add(item)
    ratingDict[(user,item)] = rating
    timeDict[(user, item)] = time
    ratingsPerItem[item].append(rating)
```

[5]:
```
firstItem = reviews['book_id'][0]
tenHighest = mostSimilar(firstItem, 10)

print('\t Item ID \t Similarity')
for index, elem in enumerate(tenHighest):
    print(f"{index+1} \t {elem[1]} \t {elem[0]}")
```

```
         Item ID         Similarity
1        25334626        0.16666666666666666
2        25659811        0.14285714285714285
3        18369278        0.13793103448275862
4        18430205        0.13157894736842105
5        20299669        0.12903225806451613
6        17995154        0.125
7        23241671        0.12121212121212122
8        23093378        0.12121212121212122
9        18853527        0.12121212121212122
10       26778333        0.11764705882352941
```

1.1.2 Task 2

[6]:
```
userId = 'dc3763cdb9b2cae805882878eebb6a32'

userReviews = {}
for index in reviews.index:
    if (reviews['user_id'][index] == userId):
        userReviews[reviews['book_id'][index]] = reviews['rating'][index]

sortedUserReviews = sorted(userReviews.items(), reverse=True, key=lambda elem :␣
↪elem[1])
```

Task a)

[7]:
```
def mostSimilarItemsNotInteractedWith(item, K):
    similarities = []
    users = usersPerItem[item]
    for i in usersPerItem:
        if (i==item): continue
        if i in userReviews.keys(): continue
        similarity = Jaccard(users, usersPerItem[i])
        similarities.append((similarity, i))
    similarities.sort(reverse=True)
    return similarities[:K]

tenBestA = mostSimilarItemsNotInteractedWith(sortedUserReviews[0][0], 10)

print('\t Item ID \t Similarity')
for index, elem in enumerate(tenBestA):
    print(f"{index+1} \t {elem[1]} \t {elem[0]}")
```

```
         Item ID         Similarity
1        25334626        0.16666666666666666
2        25659811        0.14285714285714285
3        18369278        0.13793103448275862
4        18430205        0.13157894736842105
5        20299669        0.12903225806451613
6        17995154        0.125
7        23241671        0.12121212121212122
8        23093378        0.12121212121212122
9        18853527        0.12121212121212122
10       26778333        0.11764705882352941
```

Task b)

[8]:
```
def mostSimilarUsers(user):
    similarities = []
    for item in userReviews:
        for u in usersPerItem[item]:
            if user == u: continue
            similarity = Jaccard(itemsPerUser[user], itemsPerUser[u])
            similarities.append((similarity, u))
    similarities.sort(reverse=True)
    return similarities

sortedSimilarUsers = mostSimilarUsers(userId)

tenBestB = []
booksAdded = []
#spaghetti
for otherUser in sortedSimilarUsers:
    if(len(tenBestB) == 10): break
    otherUserReviews = {}
    for index in reviews.index:
        if (reviews['user_id'][index] == otherUser[1]):
            otherUserReviews[reviews['book_id'][index]] =␣
↪reviews['rating'][index]
    sortedOtherUserReviews = sorted(otherUserReviews.items(), reverse=True,␣
↪key=lambda elem : elem[1])
    for elem in sortedOtherUserReviews:
        if elem[0] in booksAdded: continue
        if elem[0] in userReviews.keys(): continue
        tenBestB.append((otherUser[0], sortedOtherUserReviews[0][0]))
        booksAdded.append(sortedOtherUserReviews[0][0])
        break

print('\t Item ID \t Similarity')
for index, elem in enumerate(tenBestB):
    print(f"{index+1}   \t {elem[1]}    \t {elem[0]}")
```

```
         Item ID         Similarity
1        10767466        0.3333333333333333
2        23531233        0.25
3        59715           0.2
4        26400739        0.14285714285714285
5        22454333        0.05555555555555555
6        21432474        0.030303030303030304
7        20696439        0.023809523809523808
8        17689253        0.02040816326530612
9        10361139        0.014925373134328358
10       6238080         0.0136986301369863
```

1.1.3 Task 3

[9]:
```
import math

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)
```

[10]:
```
def PearsonSharedItems(item1, item2):
    item1Bar, item2Bar = itemAverages[item1], itemAverages[item2]
    intersect = usersPerItem[item1].intersection(usersPerItem[item2])
    numerator, denominator1, denominator2 = 0, 0, 0
    for user in intersect:
        numerator += (ratingDict[(user, item1)] -␣
↪item1Bar)*(ratingDict[(user,item2)] - item2Bar)
    for user in intersect:
        denominator1 += (ratingDict[(user,item1)] - item1Bar)**2
        denominator2 += (ratingDict[(user,item2)] - item2Bar)**2
    denominator = math.sqrt(denominator1) * math.sqrt(denominator2)
    if denominator == 0: return 0
    return numerator/denominator

def mostSimilar(item, K):
    similarities = []
    users = usersPerItem[item]
    for i in usersPerItem:
        if (i==item): continue
        similarity = PearsonSharedItems(item, i)
        similarities.append((similarity, i))
    similarities.sort(reverse=True)
    return similarities[:K]

firstItem = reviews['book_id'][0]
tenHighest = mostSimilar(firstItem, 10)
print('Pearson similarity in terms of SHARED items in the denominator')
print('\t Item ID \t Similarity')
for index, elem in enumerate(tenHighest):
    print(f"{index+1} \t {elem[1]} \t {elem[0]}")
```

```
Pearson similarity in terms of SHARED items in the denominator
         Item ID         Similarity
1        33585240        1.0000000000000002
2        31855855        1.0000000000000002
3        31224404        1.0000000000000002
4        30272308        1.0000000000000002
5        29840108        1.0000000000000002
6        29431094        1.0000000000000002
7        28926893        1.0000000000000002
8        28084929        1.0000000000000002
9        26251358        1.0000000000000002
10       26013087        1.0000000000000002
```

[11]:
```
def PearsonAllItems(item1, item2):
    item1Bar, item2Bar = itemAverages[item1], itemAverages[item2]
    intersect = usersPerItem[item1].intersection(usersPerItem[item2])
    numerator, denominator1, denominator2 = 0, 0, 0
    for user in intersect:
        numerator += (ratingDict[(user, item1)] -␣
↪item1Bar)*(ratingDict[(user,item2)] - item2Bar)
    for user in usersPerItem[item1]:
        denominator1 += (ratingDict[(user,item1)] - item1Bar)**2
    for user in usersPerItem[item2]:
        denominator2 += (ratingDict[(user,item2)] - item2Bar)**2
    denominator = math.sqrt(denominator1) * math.sqrt(denominator2)
    if denominator == 0: return 0
    return numerator/denominator

def mostSimilar(item, K):
    similarities = []
    users = usersPerItem[item]
    for i in usersPerItem:
        if (i==item): continue
        similarity = PearsonAllItems(item, i)
        similarities.append((similarity, i))
    similarities.sort(reverse=True)
    return similarities[:K]

firstItem = reviews['book_id'][0]
tenHighest = mostSimilar(firstItem, 10)

print('Pearson similarity in terms of ALL items in the denominator')
print('\t Item ID \t Similarity')
for index, elem in enumerate(tenHighest):
    print(f"{index+1} \t {elem[1]} \t {elem[0]}")
```

```
Pearson similarity in terms of ALL items in the denominator
         Item ID         Similarity
1        20300526        0.31898549007874194
2        13280885        0.18785865431369264
3        18208501        0.17896391275176457
4        25430791        0.16269036695641687
5        21521612        0.16269036695641687
6        1341758         0.1555075595594449
7        6314737         0.1526351566298752
8        4009034         0.15204888048160353
9        988744          0.1494406444160154
10       18430205        0.14632419481281997
```

1.2 Tasks — Rating Prediction

1.2.1 Task 4

[12]:
```
def predictRating(user, item):
    avgRating = sum([rating for rating in ratingsPerItem[item]])/
↪len(ratingsPerItem[item])
    numerator, denominator = 0, 0
    for item2 in itemsPerUser[user]:
        if item == item2: continue
        similarity = Jaccard(usersPerItem[item], usersPerItem[item2])
        avgRating2 = sum([rating for rating in ratingsPerItem[item2]])/
 ↪len(ratingsPerItem[item2])
        numerator += (ratingDict[(user, item)] - avgRating2)*similarity
        denominator += similarity
    if(denominator!=0):
        return avgRating + numerator/denominator
    else:
        return avgRating

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

simPredictions = [predictRating(reviews['user_id'][index],␣
 ↪reviews['book_id'][index]) for index in range(0, 10000)]

labels = [reviews['rating'][index] for  index in range(0, 10000)]

mse = MSE(simPredictions, labels)

print(f"MSE: {mse}")
```

```
MSE: 0.36449548691864403
```

1.2.2 Task 6

[13]:
```
import dateutil.parser

def f(t_i, t_j):
    t_i = dateutil.parser.parse(t_i)
    t_j = dateutil.parser.parse(t_j)
    return math.e**-(abs((t_i-t_j).days)*1e-03)

def predictRating(user, item):
    avgRating = sum([rating for rating in ratingsPerItem[item]])/
 ↪len(ratingsPerItem[item])
    numerator, denominator = 0, 0
    for item2 in itemsPerUser[user]:
        if item == item2: continue
        t_item = timeDict[(user, item)]
        t_item2 = timeDict[(user, item2)]
        decay = f(t_item, t_item2)
        similarity = Jaccard(usersPerItem[item], usersPerItem[item2])
        avgRating2 = sum([rating for rating in ratingsPerItem[item2]])/
 ↪len(ratingsPerItem[item2])
        numerator += (ratingDict[(user, item)] - avgRating2)*similarity*decay
        denominator += similarity*decay
    if(denominator!=0):
        return avgRating + numerator/denominator
    else:
        return avgRating

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

simPredictions = [predictRating(reviews['user_id'][index],␣
 ↪reviews['book_id'][index]) for index in range(0, 10000)]

labels = [reviews['rating'][index] for index in range(0, 10000)]

mse = MSE(simPredictions, labels)

print(f"MSE: {mse}")
```

```
MSE: 0.36110988912894737
```