CSE 258 - HOMEWORK 2

Preparation

In [1]:
```
# import essential package
import gzip
import math
import random
from collections import defaultdict
import json
```

In [2]:
```
# read data and put into dataset[]
dataset = []
path = "goodreads_reviews_comics_graphic.json.gz"
with gzip.open(path, 'rt', encoding="utf8") as f:
    for line in f:
        dataset.append(json.loads(line))
```

Task1 - Similarity Functions

Question1

In [3]:
```
usersPerItem = defaultdict(set) # Maps an item to the users who rat
itemsPerUser = defaultdict(set) # Maps a user to the items that the
ratingDict = {} # To retrieve a rating for a specific user/item pai

for d in dataset:
    user,item = d['user_id'], d['book_id']
    usersPerItem[item].add(user)
    itemsPerUser[user].add(item)
    ratingDict[(user,item)] = d['rating']
```

In [4]:
```
# Similarity metrics of Jaccard similarity
def Jaccard(s1, s2):
    numer = len(s1.intersection(s2))
    denom = len(s1.union(s2))
    if denom == 0:
        return 0
    return numer / denom
```

In [5]:
```
def mostSimilar(i, N):
    similarities = []
    users = usersPerItem[i]
    for i2 in usersPerItem:
        if i2 == i: continue
        sim = Jaccard(users, usersPerItem[i2])
        similarities.append((sim,i2))
    similarities.sort(reverse=True)
    return similarities[:N]
```

In [6]:
```
query = dataset[0]['book_id']
```

In [7]:
```
ms = mostSimilar(query, 10)
print("\nQuestion 1 Answer: ")
ms
```

```
Question 1 Answer:
```

Out[7]:
```
[(0.16666666666666666, '25334626'),
 (0.14285714285714285, '25659811'),
 (0.13793103448275862, '18369278'),
 (0.13157894736842105, '18430205'),
 (0.12903225806451613, '20299669'),
 (0.125, '17995154'),
 (0.12121212121212122, '23241671'),
 (0.12121212121212122, '23093378'),
 (0.12121212121212122, '18853527'),
 (0.11764705882352941, '26778333')]
```

QUESTION 2

In [8]:
```
userID = 'dc3763cdb9b2cae805882878eebb6a32'
```

In [9]:
```
def favorite(userID):
    user = itemsPerUser[userID]
    itemRate = [x for x in zip(user, [ratingDict[(userID, rate)] fo
    itemRate = sorted(itemRate, key = lambda x: x[0])
    itemRate = sorted(itemRate, key = lambda x: x[1], reverse = Tru
    return itemRate[0][0]
```

In [10]:
```
favoriteItem = favorite(userID)
similarItem = mostSimilar(favoriteItem, 10)
print("\nQuestion 2(a) Answer: ")
similarItem
```

```
Question 2(a) Answer:

[(0.16666666666666666, '25334626'),
 (0.14285714285714285, '25659811'),
 (0.13793103448275862, '18369278'),
 (0.13157894736842105, '18430205'),
 (0.12903225806451613, '20299669'),
 (0.125, '17995154'),
 (0.12121212121212122, '23241671'),
 (0.12121212121212122, '23093378'),
 (0.12121212121212122, '18853527'),
 (0.11764705882352941, '26778333')]
```

In [11]:
```
query = dataset[0]['book_id']
```

In [12]:
```
def mostSimilarUser(i, N):
    similarities = []
    items = itemsPerUser[i]
    for i2 in itemsPerUser:
        if i2 == i: continue
        sim = Jaccard(items, itemsPerUser[i2])
        #sim = Pearson(i, i2) # Could use alternate similarity metr
        similarities.append((sim,i2))
    similarities.sort(reverse=True)
    return similarities[:N]
```

In [13]:
```
similarities = mostSimilarUser(userID, 20)
```

In [14]:
```
def findOtherFavor(similarities, N):
    favoriteOtherItems = []
    for a in similarities:
        bookID = favorite(a[1])
        if bookID == favoriteItem : continue
        favoriteOtherItems.append((a[0], a[1], bookID))
    return favoriteOtherItems[:N]
```

In [15]:
```
print("\nQuestion 2(b) Answer: Associated Scores, User ID, Recommen
findOtherFavor(similarities, 10)
```

```
Question 2(b) Answer: Associated Scores, User ID, Recommend Book I
D
```

Out[15]:
```
[(0.3333333333333333, '6470c7f5e3468ba34e9fe628960fbbf1', '1076746
6'),
 (0.25, '6497ca91df3c182006874c96a8530b37', '17570797'),
 (0.2, '033cf640dfa6f85eb146c39787289628', '15704307'),
 (0.14285714285714285, '5510684ab6c18f2dd493787e66b2722c', '101386
07'),
 (0.05555555555555555, '17f73ea38e97307935c0d3b6ca987b53', '124347
47'),
 (0.030303030303030304, 'a39b4249d201ef5ce5ea553bdd013e66', '17995
248'),
 (0.023809523809523808, '42519f961f79b61701bda60787b031cf', '10105
459'),
 (0.02040816326530612, '65a7975989734fc6e18b7d2bd2bcb49f', '109976
45'),
 (0.014925373134328358, '0fafb6f0843124383f4e2c5a2090fb09', '10361
139'),
 (0.0136986301369863, '071222e19ae29dc9fdbe225d983449be', '1026432
8')]
```

QUESTION 3

In [16]:
```
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)
```

In [17]:
```
def Pearson(i1, i2):
    # Between two items
    iBar1 = itemAverages[i1]
    iBar2 = itemAverages[i2]
    inter = usersPerItem[i1].intersection(usersPerItem[i2])
    numer = 0
    denom1 = 0
    denom2 = 0
    for u in inter:
        numer += (ratingDict[(u,i1)] - iBar1)*(ratingDict[(u,i2)] -
    for u in inter: #usersPerItem[i1]:
        denom1 += (ratingDict[(u,i1)] - iBar1)**2
    #for u in usersPerItem[i2]:
        denom2 += (ratingDict[(u,i2)] - iBar2)**2
    denom = math.sqrt(denom1) * math.sqrt(denom2)
    if denom == 0: return 0
    return numer / denom
```

In [18]:
```
def mostSimilar(i, N):
    similarities = []
    users = usersPerItem[i]
    for i2 in usersPerItem:
        if i2 == i: continue
        sim = Pearson(i, i2) # Could use alternate similarity metri
        similarities.append((sim,i2))
    similarities.sort(reverse=True)
    return similarities[:10]
```

In [19]:
```
query = dataset[0]['book_id']
ms = mostSimilar(query, 10)
print("\nQuestion 3.1 Answer: ")
ms
```

```
Question 3.1 Answer:
```

Out[19]:
```
[(1.0000000000000002, '993861'),
 (1.0000000000000002, '7986827'),
 (1.0000000000000002, '7342071'),
 (1.0000000000000002, '62953'),
 (1.0000000000000002, '33585240'),
 (1.0000000000000002, '3328828'),
 (1.0000000000000002, '31855855'),
 (1.0000000000000002, '31224404'),
 (1.0000000000000002, '30272308'),
 (1.0000000000000002, '29840108')]
```

In [20]:
```
def Pearson(i1, i2):
    # Between two items
    iBar1 = itemAverages[i1]
    iBar2 = itemAverages[i2]
    inter = usersPerItem[i1].intersection(usersPerItem[i2])
    numer = 0
    denom1 = 0
    denom2 = 0
    for u in inter:
        numer += (ratingDict[(u,i1)] - iBar1)*(ratingDict[(u,i2)] -
    for u in usersPerItem[i1]:
        denom1 += (ratingDict[(u,i1)] - iBar1)**2
    for u in usersPerItem[i2]:
        denom2 += (ratingDict[(u,i2)] - iBar2)**2
    denom = math.sqrt(denom1) * math.sqrt(denom2)
    if denom == 0: return 0
    return numer / denom
```

In [21]:
```
query = dataset[0]['book_id']
ms = mostSimilar(query, 10)
print("\nQuestion 3.2 Answer: ")
ms
```

```
Question 3.2 Answer:
```

Out[21]:
```
[(0.31898549007874194, '20300526'),
 (0.18785865431369264, '13280885'),
 (0.17896391275176457, '18208501'),
 (0.16269036695641687, '25430791'),
 (0.16269036695641687, '21521612'),
 (0.1555075595594449, '1341758'),
 (0.1526351566298752, '6314737'),
 (0.15204888048160353, '4009034'),
 (0.1494406444160154, '988744'),
 (0.1463241948128199, '18430205')]
```

Task 2 - Rating Prediction

Question 4

In [22]:
```
reviewsPerUser = defaultdict(list)
reviewsPerItem = defaultdict(list)
```

In [23]:
```
for d in dataset:
    user,item = d['user_id'], d['book_id']
    reviewsPerUser[user].append(d)
    reviewsPerItem[item].append(d)
```

In [24]:
```
ratingMean = sum([d['rating'] for d in dataset]) / len(dataset)
```

In [25]:
```
def predictRating(user,item):
    ratings = []
    similarities = []
    for d in reviewsPerUser[user]:
        i2 = d['book_id']
        if i2 == item: continue
        ratings.append(d['rating'] - itemAverages[i2])
        similarities.append(Jaccard(usersPerItem[item],usersPerItem
    if (sum(similarities) > 0):
        weightedRatings = [(x*y) for x,y in zip(ratings,similaritie
        return itemAverages[item] + sum(weightedRatings) / sum(simi
    else:
        # User hasn't rated any similar items
        return itemAverages[item]
```

In [26]:
```
def MSE(predictions, labels):
    differences = [(x-y)**2 for x,y in zip(predictions,labels)]
    return sum(differences) / len(differences)
```

In [27]:
```
alwaysPredictMean = [ratingMean for d in dataset]
```

In [28]:
```
dataset = dataset[:10000]
```

In [29]:
```
simPredictions = [predictRating(d['user_id'], d['book_id']) for d i
```

In [30]:
```
labels = [d['rating'] for d in dataset]
```

In [31]:
```
print("\nQuestion 4 Answer: ")
print("MSE: ", MSE(simPredictions, labels))
```

```
Question 4 Answer:
MSE:  0.7017041185560355
```

Question 6

In [32]:
```
import time
import dateutil.parser

def predictRating(user, item, timestamp):
    ratings = []
    similarities = []
    for d in reviewsPerUser[user]:
        timestamp2 = dateutil.parser.parse(d['date_added']).timesta
        diff = abs(timestamp2 - timestamp) / 86400
        parameter = 0.0022
        ft = math.exp(-diff * parameter)
        i2 = d['book_id']
        if i2 == item: continue
        ratings.append(d['rating'] - itemAverages[i2])
        similarities.append(Jaccard(usersPerItem[item], usersPerIte
    if (sum(similarities) > 0):
        weightedRatings = [(x*y) for x,y in zip(ratings,similaritie
        return itemAverages[item] + sum(weightedRatings) / sum(simi
    else:
        # User hasn't rated any similar items
        return itemAverages[item]

time = dateutil.parser.parse(dataset[0]['date_added'])
dateParsed = [dateutil.parser.parse(d['date_added']).timestamp() fo
labels = [d['rating'] for d in dataset]
simPredictions = [predictRating(d['user_id'], d['book_id'], dateuti

print("\nQuestion 6 Answer: ")
print("MSE: ", MSE(simPredictions, labels))
```

```
Question 6 Answer:
MSE:  0.6975936563299326
```