In [39]:
```
import gzip
import math
import random
from collections import defaultdict
import json
```

In [40]:
```
def parseDataFromFile(fname):
    for l in open(fname):
        yield eval(l)
```

In [41]:
```
# read json file data
data = list(parseDataFromFile('goodreads_reviews_comics_graphic.json'))
```

In [42]:
```
usersPerItem = defaultdict(set) # Maps an item to the users who rated it
itemsPerUser = defaultdict(set) # Maps a user to the items that they rated
ratingDict = {} # To retrieve a rating for a specific user/item pair

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

In [43]:
```
usersPerItem['18471619']
```

Out[43]:
```
{'033cf640dfa6f85eb146c39787289628',
 '071222e19ae29dc9fdbe225d983449be',
 '0fafb6f0843124383f4e2c5a2090fb09',
 '17f73ea38e97307935c0d3b6ca987b53',
 '26c41515b2144cf6a1545e831f8d2cd3',
 '41b1c110d428bbc49481036e896c0a6f',
 '42519f961f79b61701bda60787b031cf',
 '4674a9c5dc3fde5506d43d6a737fa059',
 '4ae069d704b11bdf12c25fe640f75ff0',
 '5510684ab6c18f2dd493787e66b2722c',
 '6470c7f5e3468ba34e9fe628960fbbf1',
 '6497ca91df3c182006874c96a8530b37',
 '65a7975989734fc6e18b7d2bd2bcb49f',
 '68dff5594b77c47aae96cbe97aba5206',
```

```
'714ed8e9b1814bf45dd9abd88431dbb8',
'7f63e4d65e873703970e71afabbc3b54',
'8d06514d97530ddb22a05b84dfe4daad',
'9d4feff5432a5a5243bf277e0d258042',
'9f6f9da3a71ded406f15764f8fbf5f51',
'a39b4249d201ef5ce5ea553bdd013e66',
'd286122fed6ded84ff53993335bfd59c',
'd7310760f68365d3ca747fa8b9310518',
'da7a0c5ee0c89973224d8853445be68e',
'dc3763cdb9b2cae805882878eebb6a32',
'dd669721e136c1be47d739b14fa23d20',
'eaa54d876d841293059657fb80a9bba6'}
```

In [44]:
```
# define Jaccard similarity function
def Jaccard(s1, s2):
    numer = len(s1.intersection(s2))
    denom = len(s1.union(s2))
    return numer / denom
```

In [45]:
```
# define function to return similarities and book_id
def mostSimilar_items(i):
    similarities = []
    users = usersPerItem[i]    # return users who reviewed the book with book_id
    for i2 in usersPerItem:    # iterate item
        if i2 == i: continue   # skip the same book_id
        # calculate the users intersation proportion of other books
        sim = Jaccard(users, usersPerItem[i2])
        similarities.append((sim,i2))
    similarities.sort(reverse=True)
    return similarities[:10]
```

In [46]:
```
query = '18471619'
```

In [47]:
```
mostSimilar_items(query)
```

Out[47]:
```
[(0.1666666666666666, '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 [ ]:

Question 2

In [48]:

```
# define function to return 10 items most similar to the user's favorite
def similar_favorite(i):
    items = list(itemsPerUser[i])
    favorite_item = items[0]
    for item in items:
        if ratingDict[(i,item)] > ratingDict[(i,favorite_item)]:
            favorite_item = item
        elif ratingDict[(i,item)] == ratingDict[(i,favorite_item)] and item < favorite_item:
            favorite_item = item
    similarities = []
    # return users who buy the book with the book_id
    users = usersPerItem[favorite_item]
    for i2 in usersPerItem:    # iterate item
        if i2 == i: continue   # skip the same book_id
        if i2 in items: continue
        # calculate the users intersation proportion of other books
        sim = Jaccard(users, usersPerItem[i2])
        similarities.append((sim,i2))
    similarities.sort(reverse=True)
    return similarities[:10]
```

In [49]:

```
similar_favorite('dc3763cdb9b2cae805882878eebb6a32')
```

Out[49]:

```
[(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 [ ]:

In [50]:
```
# define function to get 10 most similar users and recommend their favorite
def mostSimilar_users(i):
    similarities = []
    items = itemsPerUser[i]    # return users who buy the book with the book_id
    for i2 in itemsPerUser:    # iterate item
        if i2 == i: continue   # skip the same book_id
        # calculate the users intersation proportion of other books
        sim = Jaccard(items, itemsPerUser[i2])
        if sim == 1: continue         # skip users who have the same books
        similarities.append((sim,i2))
    similarities.sort(reverse=True)
    favorite_books = []
    for k in similarities[:10]:
        m = list(k)[1]
        items = list(itemsPerUser[m])
        clean_items = []
        # avoid recommending items the user has already interacted
        for item in items:
            if item in itemsPerUser[i]: continue
            else:
                clean_items.append(item)
        favorite_item = clean_items[0]
        for item in clean_items:
            if ratingDict[(m,item)] > ratingDict[(m,favorite_item)]:
                favorite_item = item
            elif ratingDict[(m,item)] == ratingDict[(m,favorite_item)] and item < favorite_item:
                favorite_item = item  # order by alphabet with the same rate
        favorite_books.append((k[0],favorite_item))
    return favorite_books
```

In [51]:
```
mostSimilar_users('dc3763cdb9b2cae805882878eebb6a32')
```

Out[51]:
```
[(0.3333333333333333, '10767466'),
 (0.25, '17570797'),
 (0.2, '15704307'),
 (0.14285714285714285, '10138607'),
 (0.05555555555555555, '12434747'),
 (0.030303030303030304, '17995248'),
 (0.023809523809523808, '10105459'),
 (0.02040816326530612, '10997645'),
 (0.014925373134328358, '10361139'),
 (0.0136986301369863, '10264328')]
```

In [ ]:

In [ ]:

Question 3

In [54]:
```
userAverages = {} # get average rate for a user
itemAverages = {} # get average rate for a book

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 [55]:
```
# Pearson similarity be implemented only in terms of shared items
def Pearson_share(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)] - iBar2)
    for u in inter:
        denom1 += (ratingDict[(u,i1)] - iBar1)**2
    for u in inter:
        denom2 += (ratingDict[(u,i2)] - iBar2)**2
    denom =  math.sqrt(denom1) * math.sqrt(denom2)
    if denom == 0: return 0
    return numer / denom
```

In [56]:
```
# apply function above to get similarites
def Person_shared_similarity(i):
    similarities = []
    for item in usersPerItem:
        similarity = Pearson_share(i,item)
```

```
        similarities.append((similarity,item))
    similarities.sort(reverse=True)
    return similarities[:10]
```

In [57]:
```
Person_shared_similarity('18471619')
```

Out[57]:
```
[(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 [ ]:
```

```

In [58]:
```
# Pearson similarity be implemented in terms of all items each user consumed
def Pearson_either(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)] - iBar2)
    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 [59]:
```
# apply function above to get similarites
def Person_union_similarity(i):
    similarities = []
```

```
    for item in usersPerItem:
        similarity = Pearson_either(i,item)
        similarities.append((similarity,item))
    similarities.sort(reverse=True)
    return similarities[1:11]
```

In [60]:
```
Person_union_similarity('18471619')
```

Out[60]:
```
[(0.3189854900787419, '20300526'),
 (0.1878586543136926, '13280885'),
 (0.17896391275176454, '18208501'),
 (0.1626903669564168, '25430791'),
 (0.1626903669564168, '21521612'),
 (0.15550755955944487, '1341758'),
 (0.15263515662987517, '6314737'),
 (0.15204888048160348, '4009034'),
 (0.14944064441601537, '988744'),
 (0.14632419481281994, '18430205')]
```

In [ ]:
```

```

In [ ]:
```

```

Question 4

In [61]:
```
reviewsPerUser = defaultdict(list)
reviewsPerItem = defaultdict(list)
```

In [62]:
```
for d in data:
    user,item = d['user_id'], d['book_id']
    reviewsPerUser[user].append(d)
    reviewsPerItem[item].append(d)
```

In [74]:
```
# check the content of reviewsPerItem
reviewsPerItem['7966445'][:2]
```

Out[74]:
```
[{'user_id': '37b3e60b4e4152c580fd798d405150ff',
  'book_id': '7966445',
  'review_id': 'ff8fdb4a7617cf3fa089c3061509822c',
```

```
  'rating': 5,
  'review_text': 'Another excellent volume. The background on the Hydra leaders is terrific, as is the fact tha
t the main focus of the volume is character drama.',
  'date_added': 'Fri Jul 05 13:59:18 -0700 2013',
  'date_updated': 'Fri Jul 05 14:00:04 -0700 2013',
  'read_at': 'Sat Jun 16 00:00:00 -0700 2012',
  'started_at': '',
  'n_votes': 0,
  'n_comments': 0},
 {'user_id': '6943393b4bc61422a4a468cfee8b190e',
  'book_id': '7966445',
  'review_id': '0f52fb6f29b974116f1c7cb38315ea3f',
  'rating': 4,
  'review_text': 'The series is getting better and better.',
  'date_added': 'Tue Sep 13 03:14:57 -0700 2011',
  'date_updated': 'Fri Dec 16 03:30:50 -0800 2011',
  'read_at': 'Sun Dec 11 00:00:00 -0800 2011',
  'started_at': '',
  'n_votes': 1,
  'n_comments': 0}]
```

In [156…

```python
# define the function to calculate r(u, i)
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[i2]))
    if (sum(similarities) > 0):
        weightedRatings = [(x*y) for x,y in zip(ratings,similarities)]
        return itemAverages[item] + sum(weightedRatings) / sum(similarities)
    else:
        # User hasn't rated any similar items
        return itemAverages[item]
```

In [76]:

```python
# define function to calculate MSE
def MSE(predictions, labels):
    differences = [(x-y)**2 for x,y in zip(predictions,labels)]
    return sum(differences) / len(differences)
```

In [81]:

```python
# combine the data to prepare for later steps
user_book_data = [(d['user_id'],d['book_id'],d['rating']) for d in data]
```

In [82]:
```
user_book_data[:10]
```

Out[82]:
```
[('dc3763cdb9b2cae805882878eebb6a32', '18471619', 3),
 ('bafc2d50014200cda7cb2b6acd60cd73', '6315584', 4),
 ('bafc2d50014200cda7cb2b6acd60cd73', '29847729', 4),
 ('bafc2d50014200cda7cb2b6acd60cd73', '18454118', 5),
 ('bafc2d50014200cda7cb2b6acd60cd73', '2239435', 4),
 ('bafc2d50014200cda7cb2b6acd60cd73', '13094398', 3),
 ('bafc2d50014200cda7cb2b6acd60cd73', '13526176', 3),
 ('bafc2d50014200cda7cb2b6acd60cd73', '13638518', 5),
 ('bafc2d50014200cda7cb2b6acd60cd73', '13630859', 5),
 ('0ef32090550901ead25cb0ea21c4d36b', '15984307', 3)]
```

In [84]:
```
import pandas as pd  # import pandas to sample data because of laptop limit
```

In [85]:
```
user_book_data_pandas = pd.DataFrame(user_book_data)
```

In [88]:
```
# randomly sample 10000 rows data
user_book_data_pandas_sample = user_book_data_pandas.sample(n=10000,random_state=10)
```

In [106…
```
x = user_book_data_pandas_sample.iloc[:,:2]
```

In [107…
```
labels = user_book_data_pandas_sample.iloc[:,2]
```

In [111…
```
# transform DataFrame data to list
x = x.values.tolist()
```

In [113…
```
labels = labels.values.tolist()
```

In [157…
```
# predict the rating
simPredictions = [predictRating(d[0],d[1]) for d in x]
```

In [158…
```
# MSE for 10000 sampled data
```

```
MSE(simPredictions, labels)
```

Out[158… 0.7946198152367399

In [ ]:

Question 6

In [159…
```
# import library and define function to change string time to timestamp
import datetime
import dateutil.parser

def getDateTime(s):
    d = dateutil.parser.parse(s)
    return d
```

In [160…
```
# import math library for later calculation
import math
```

In [161…
```
# define a function to find date added for item we are interested
def find_item(user,item):
    for k in reviewsPerUser[user]:
        if k['book_id'] == item:
            return k['date_added']
        else: continue
```

Model explanation: in this model, we're gonna do temporal recommendation. I choose f(t) = e**(−λt) for decay and choose λ = 1. Then, use days' time intervals (|tu,i − tu,j|) of date_added as parameter. That means the influence of other items' rate on the item we are interested will decay on time basis. The days' time interval can be large such as hundreds or thousands days, therefore, I take log10 of the time interval to avoid time intervals affect the results excessively.

In [162…
```
# use f(t) = e-λt and make λ = 1; use days' time intervals(t) for decay function
def predictRatingdecay(user,item):
    ratings = []
    similarities = []
    time_period = []
    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[i2]))
        if abs((getDateTime(find_item(user,item)) - getDateTime(d['date_added'])).days) == 0:
# as take log10 later, avoid domain problem for two reviews added at the same day
            t = 0
        else:
            # take log10 to avoid time intervals(days) affect the results excessively
            t = math.log10(abs((getDateTime(find_item(user,item)) - getDateTime(d['date_added'])).days))
        time_period.append(math.e**(-t))
    if (sum(similarities) > 0):
        weightedRatings = [(x*y*z) for x,y,z in zip(ratings,similarities,time_period)]
        decay_similarities = [(y*z) for y,z in zip(similarities,time_period)]
        return itemAverages[item] + sum(weightedRatings) / sum(decay_similarities)
    else:
        # User hasn't rated any similar items
        return itemAverages[item]
```

In [163…

```
simPredictionsdecay = [predictRatingdecay(d[0],d[1]) for d in x]
```

In [164…

```
# MSE for the same 10000 sampled data
MSE(simPredictionsdecay, labels)
```

Out[164… 0.7818071355736506

In [165…

```
print('The refined model\'s improvement in MSE:',
      100 * (MSE(simPredictions, labels)-
             MSE(simPredictionsdecay, labels))/ MSE(simPredictions, labels),'%')
```

```
The refined model's improvement in MSE: 1.6124289147347826 %
```

In [ ]: