hw2

October 25, 2021

[109]:
```
import numpy
from urllib.request import urlopen
import scipy.optimize
import random
import gzip

def parseDataFromURL(fname):
  for l in urlopen(fname):

    yield eval(fname)

def parseData(fname):
  for l in open(fname):
    yield eval(l)

print("Reading data...")
```

```
Reading data...
```

[110]:
```
# Download from https://cseweb.ucsd.edu//classes/fa21/cse258-b/data/
 ↪goodreads_reviews_comics_graphic.json.gz"
data = list(parseData("./data/goodreads_reviews_comics_graphic.json"))
print(type(data)) # <class 'list'>
print(type(data[0])) # <class 'dictionary'>
print("done")

keys = data[0].keys()
print(keys)
print(len(data))
# dict_keys(['user_id', 'book_id', 'review_id', 'rating', 'review_text',␣
 ↪'date_added', 'date_updated', 'read_at', 'started_at', 'n_votes',␣
 ↪'n_comments'])

print(data[0]['user_id'])
print(data[0]['book_id'])
print(data[0]['rating'])
print(data[0]['review_text'])
```

```
<class 'list'>
```

<class 'dict'>
done
dict_keys(['user_id', 'book_id', 'review_id', 'rating', 'review_text',
'date_added', 'date_updated', 'read_at', 'started_at', 'n_votes', 'n_comments'])
542338
dc3763cdb9b2cae805882878eebb6a32
18471619
3
Sherlock Holmes and the Vampires of London
 Release Date: April 2014
 Publisher: Darkhorse Comics
 Story by: Sylvain Cordurie
 Art by: Laci
 Colors by: Axel Gonzabo
 Cover by: Jean Sebastien Rossbach
 ISDN: 9781616552664
 MSRP: $17.99 Hardcover
 "Sherlock Holmes died fighting Professor Moriarty in the Reichenbach Falls.
 At least, that's what the press claims.
 However, Holmes is alive and well and taking advantage of his presumed death to travel the globe.
 Unfortunately, Holmes's plans are thwarted when a plague of vampirism haunts Britain.
 This book collects Sherlock Holmes and the Vampires of London Volumes 1 and 2, originally created by French publisher Soleil." - Darkhorse Comics
 When I received this copy of "Sherlock Holmes and the Vampires of London" I was Ecstatic! The cover art was awesome and it was about two of my favorite things, Sherlock Holmes and Vampires. I couldn't wait to dive into this!
 Unfortunately, that is where my excitement ended. The story takes place a month after Sherlock Holmes supposed death in his battle with Professor Moriarty. Sherlock's plan to stay hidden and out of site are ruined when on a trip with his brother Mycroft, they stumble on the presence of vampires. That is about as much of Sherlock's character that comes through the book. I can't even tell you the story really because nothing and I mean nothing stuck with me after reading it. I never, ever got the sense of Sherlock Holmes anywhere in this graphic novel, nor any real sense of mystery or crime. It was just Sherlock somehow battling vampires that should have had absolutely no trouble snuffing him out in a fight, but somehow always surviving and holding his own against supernatural, super powerful, blazingly fast creatures.
 The cover art is awesome and it truly made me excited to read this but everything else feel completely flat for me. I tried telling myself that "it's a graphic novel, it would be hard to translate mystery, details, emotion" but then I remembered reading DC Comic's "Identity Crisis" and realized that was a load of crap. I know it's unfair to compare the two as "Identity Crisis" had popular mystery author Brad Meltzer writing it right? Yeah…no. The standard was set that day and there is more than enough talent out there to create a great story in a graphic novel.
 That being said, it wasn't a horrible story, it just didn't grip me for feel

anything like Sherlock Holmes to me. It was easy enough to follow but I felt no sense of tension, stakes or compassion for any of the characters.
As far as the vampires go, it's hard to know what to expect anymore as there are so many different versions these days. This was the more classic version which I personally prefer, but again I didn't find anything that portrayed their dominance, calm confidence or sexuality. There was definitely a presence of their physical prowess but somehow that was lost on me as easily as Sherlock was able to defend himself. I know it, wouldn't do to kill of the main character, but this would have a been a great opportunity to build around the experience and beguiling nature of a vampire that had lived so many years of experience. Another chance to showcase Sherlock's intellect in a battle of wits over strength in something more suitable for this sort of story as apposed to trying to make it feel like an action movie.
Maybe I expected to much and hoped to have at least a gripping premise or some sort of interesting plot or mystery but I didn't find it here. This may be a must have for serious Sherlock Holmes fans that have to collect everything about him, but if you are looking for a great story inside a graphic novel, I would have to say pass on this one.
That artwork is good, cover is great, story is lacking so I am giving it 2.5 out of 5 stars.

[111]:
```
import pandas as pd
import numpy as np

def Jaccard(s1, s2):
    numer = len(s1.intersection(s2))
    denom = len(s1.union(s2))
    if denom == 0:
        return 0
    return numer / denom

book_ids = []
for d in data:
    book_ids.append(d['book_id'])
book_id_list = np.unique(book_ids)
```

[112]:
```
book_user_dict = {}
[book_user_dict.setdefault(b, []) for b in book_id_list]

for d in data:
    book_user_dict[d['book_id']].append(d['user_id'])
```

[113]:
```
# user list of
user_first_book_set = set(book_user_dict[data[0]['book_id']])

jaccard_sim_list = []
for book, users in book_user_dict.items():
```

```
    # jaccard similarity
    jaccard_sim_list.append(Jaccard(user_first_book_set, set(users)))
```

[114]:
```
jaccard_sim_sorted = jaccard_sim_list
jaccard_sim_sorted.sort()
jaccard_sim_sorted = list(reversed(jaccard_sim_sorted))

top_ten = jaccard_sim_sorted[1:11]
print(top_ten)
book_indices = [jaccard_sim_list.index(b) for b in top_ten]
books_most_similar = [list(book_user_dict.keys())[i] for i in book_indices]
print(books_most_similar)
```

```
[0.16666666666666666, 0.14285714285714285, 0.13793103448275862,
0.13157894736842105, 0.12903225806451613, 0.125, 0.12121212121212122,
0.12121212121212122, 0.12121212121212122, 0.11764705882352941]
['9998775', '9997912', '9997666', '9997584', '999589', '9995674', '9994190',
'9994190', '9994190', '9994164']
```

Answer for Problem 1

a. smilarities of top ten most similar book according to Jaccard Similarity

[0.16666666666666666, 0.14285714285714285, 0.13793103448275862, 0.13157894736842105, 0.12903225806451613, 0.125, 0.12121212121212122, 0.12121212121212122, 0.12121212121212122, 0.11764705882352941]

b. books id of top ten most similar book according to Jaccard Similarity

[‘9998775’, ‘9997912’, ‘9997666’, ‘9997584’, ‘999589’, ‘9995674’, ‘9994190’, ‘9994190’, ‘9994190’, ‘9994164’]

[115]:
```
# find the highest rating book for user 'dc3763cdb9b2cae805882878eebb6a32'
user_ids = []
for d in data:
    user_ids.append(d['user_id'])
user_id_list = np.unique(user_ids)
user_book_dict = {}
[user_book_dict.setdefault(u, []) for u in user_id_list]

for d in data:
    user_book_dict[d['user_id']].append(d['book_id'])

# user list of
user_book_set = set(user_book_dict['dc3763cdb9b2cae805882878eebb6a32'])
print(user_book_set)
```

```
{'18471619'}
```

[168]:
```
# choose the N items most similar to the user's favorite (i.e., highest rated)␣
↪item.
user_set = set(['dc3763cdb9b2cae805882878eebb6a32'])
jaccard_sim_list = []
for book, users in book_user_dict.items():
    # jaccard similarity
    jaccard_sim_list.append(Jaccard(user_set, set(users)))

jaccard_sim_sorted = jaccard_sim_list
jaccard_sim_sorted.sort()
jaccard_sim_sorted = list(reversed(jaccard_sim_sorted))

top_ten = jaccard_sim_sorted[1:11]
book_indices = [jaccard_sim_list.index(b) for b in top_ten]
books_most_similar = [list(book_user_dict.keys())[i] for i in book_indices]
print(books_most_similar)
print(sorted(book_id_list)[0:10])
```

```
['1000059', '1000059', '1000059', '1000059', '1000059', '1000059', '1000059',
'1000059', '1000059', '1000059']
['1000059', '10000736', '1000096', '10001021', '10001176', '10001434',
'1000163', '1000246', '10002633', '10002666']
```

[117]:
```
# find the N most similar users, and recommending each of their their favorite␣
↪(highest rated) items
jaccard_sim_list = []
for user, books in user_book_dict.items():
    jaccard_sim_list.append(Jaccard(user_book_set, set(books)))
jaccard_sim_sorted = jaccard_sim_list
jaccard_sim_sorted.sort()
jaccard_sim_sorted = list(reversed(jaccard_sim_sorted))

top_ten = jaccard_sim_sorted[1:11]
print(top_ten)
user_indices = [jaccard_sim_list.index(u) for u in top_ten]
users_most_similar = [list(user_book_dict.keys())[i] for i in user_indices]
print(users_most_similar)

for u in users_most_similar:
    print(user_book_dict[u])
```

```
[1.0, 0.3333333333333333, 0.25, 0.2, 0.14285714285714285, 0.05555555555555555,
0.030303030303030304, 0.023809523809523808, 0.02040816326530612,
0.014925373134328358]
['ffffbb062a8b208c9c1031b529c08f7a', 'ffff7cafdaf5196383cb2efca08fb6fe',
'ffff601c0ffa34bd5ffbbf2caee30644', 'fffe3fca0160bd78ae5828b44fbeb72d',
'fffdbe24990b7e9e78653f97fc8cecd1', 'fffc34d137f5c5c5e1ca1d6f325a4dcf',
'fffba5600c5d01693b75964e7fbe193f', 'fffa84546640e2fc35ec5e7c39a126cd',
```

```
'fff9e2b47952a14cbe37ab8a7bfbc63c', 'fff99682ca5ff8aab27863cb42f24e16']
['16131399', '11029842']
['16071892', '18630542', '23093367', '19358975', '17131869', '15704307',
'18659623', '20575446', '15799936']
['18474136', '13602241', '22424']
['24000357']
['19351043']
['34895950', '34934766', '34506918', '32479654', '32479653', '29066588',
'28862494', '25076719', '23503834', '22789111', '23982972', '23241668',
'1294247', '15985364', '6492676', '9640779']
['17571564']
['18586340']
['25652706']
['22416']
```

[118]:
```
rec = []
for d in data:
    if (d['user_id'] in users_most_similar) :
        rec.append([d['user_id'],d['book_id'],d['rating']])
print(rec)
for u in users_most_similar:
    ratings = []
    for b in user_book_dict[u]:
        for r in rec :
            if (r[0]==u) & (r[1]==b):
                ratings.append(r[2])
    print(ratings)
```

```
[['ffff601c0ffa34bd5ffbbf2caee30644', '18474136', 5],
['ffff601c0ffa34bd5ffbbf2caee30644', '13602241', 5],
['ffff601c0ffa34bd5ffbbf2caee30644', '22424', 5],
['ffff7cafdaf5196383cb2efca08fb6fe', '16071892', 2],
['ffff7cafdaf5196383cb2efca08fb6fe', '18630542', 4],
['ffff7cafdaf5196383cb2efca08fb6fe', '23093367', 4],
['ffff7cafdaf5196383cb2efca08fb6fe', '19358975', 4],
['ffff7cafdaf5196383cb2efca08fb6fe', '17131869', 3],
['ffff7cafdaf5196383cb2efca08fb6fe', '15704307', 4],
['ffff7cafdaf5196383cb2efca08fb6fe', '18659623', 4],
['ffff7cafdaf5196383cb2efca08fb6fe', '20575446', 4],
['ffff7cafdaf5196383cb2efca08fb6fe', '15799936', 4],
['fffdbe24990b7e9e78653f97fc8cecd1', '19351043', 5],
['fff9e2b47952a14cbe37ab8a7bfbc63c', '25652706', 5],
['fffc34d137f5c5c5e1ca1d6f325a4dcf', '34895950', 0],
['fffc34d137f5c5c5e1ca1d6f325a4dcf', '34934766', 2],
['fffc34d137f5c5c5e1ca1d6f325a4dcf', '34506918', 3],
['fffc34d137f5c5c5e1ca1d6f325a4dcf', '32479654', 4],
['fffc34d137f5c5c5e1ca1d6f325a4dcf', '32479653', 4],
['fffc34d137f5c5c5e1ca1d6f325a4dcf', '29066588', 3],
```

```
['fffc34d137f5c5c5e1ca1d6f325a4dcf', '28862494', 3],
['fffc34d137f5c5c5e1ca1d6f325a4dcf', '25076719', 3],
['fffc34d137f5c5c5e1ca1d6f325a4dcf', '23503834', 4],
['fffc34d137f5c5c5e1ca1d6f325a4dcf', '22789111', 4],
['fffc34d137f5c5c5e1ca1d6f325a4dcf', '23982972', 3],
['fffc34d137f5c5c5e1ca1d6f325a4dcf', '23241668', 3],
['fffc34d137f5c5c5e1ca1d6f325a4dcf', '1294247', 3],
['fffc34d137f5c5c5e1ca1d6f325a4dcf', '15985364', 4],
['fffc34d137f5c5c5e1ca1d6f325a4dcf', '6492676', 3],
['fffc34d137f5c5c5e1ca1d6f325a4dcf', '9640779', 4],
['ffffbb062a8b208c9c1031b529c08f7a', '16131399', 5],
['ffffbb062a8b208c9c1031b529c08f7a', '11029842', 1],
['fffe3fca0160bd78ae5828b44fbeb72d', '24000357', 5],
['fffa84546640e2fc35ec5e7c39a126cd', '18586340', 5],
['fff99682ca5ff8aab27863cb42f24e16', '22416', 4],
['fffba5600c5d01693b75964e7fbe193f', '17571564', 4]]
[5, 1]
[2, 4, 4, 4, 3, 4, 4, 4, 4]
[5, 5, 5]
[5]
[5]
[0, 2, 3, 4, 4, 3, 3, 3, 4, 4, 3, 3, 3, 4, 3, 4]
[4]
[5]
[5]
[4]
```

Answer for Problem 2

a. recommendation by strategy a : all equal to zero so sort by alphabetical order

['1000059', '10000736', '1000096', '10001021', '10001176', '10001434', '1000163', '1000246', '10002633', '10002666']

b. recommendation by strategy b :

['16131399', '15704307', '22424', '24000357', '19351043', '9640779', '17571564', '18586340', '25652706', '22416']

[119]:
```python
import gzip
import math
import random
from collections import defaultdict

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

for d in data:
```

```
    user,item = d['user_id'], d['book_id']
    usersPerItem[item].add(user)
    itemsPerUser[user].add(item)
    ratingDict[(user,item)] = d['rating']
    #itemNames[item] = d['product_title']
```

[120]:
```
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)
```

[121]:
```
def Pearson_inter(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: #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

def Pearson_union(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
```

[122]:
```
# item = '18471619'
item = '18471619'
sim_pearson_inter = []
sim_pearson_union = []
for b in book_id_list:
    sim_pearson_inter.append(Pearson_inter(item,str(b)))
    sim_pearson_union.append(Pearson_union(item,str(b)))

sim_pearson_inter_sorted = sorted(sim_pearson_inter)
sim_pearson_inter_sorted.reverse()
sim_pearson_union_sorted = sorted(sim_pearson_union)
sim_pearson_union_sorted.reverse()

pearson_inter_top_ten_books = [book_id_list[sim_pearson_inter.index(s)] for s␣
↪in sim_pearson_inter_sorted[0:10]]
pearson_union_top_ten_books = [book_id_list[sim_pearson_union.index(s)] for s␣
↪in sim_pearson_union_sorted[0:10]]
print(pearson_inter_top_ten_books)
print(pearson_union_top_ten_books)
```

```
['1103951', '1103951', '1103951', '1103951', '1103951', '1103951', '1103951',
'1103951', '1103951', '1103951']
['18471619', '20300526', '13280885', '18208501', '21521612', '21521612',
'1341758', '6314737', '4009034', '988744']
```

Answer for Problem 3

a. top 10 items recommended based on Pearson Similarity (denominator with intersection)

['1103951', '1103951', '1103951', '1103951', '1103951', '1103951', '1103951', '1103951', '1103951', '1103951']

b. top 10 items recommended based on Pearson Similarity (denominator with union)

['18471619', '20300526', '13280885', '18208501', '21521612', '21521612', '1341758', '6314737', '4009034', '988744']

[129]:
```
import numpy as np

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 ratingMean

reviewsPerUser = defaultdict(list)
reviewsPerItem = defaultdict(list)
for d in data:
    user,item = d['user_id'], d['book_id']
    reviewsPerUser[user].append(d)
    reviewsPerItem[item].append(d)
ratingMean = sum([d['rating'] for d in data]) / len(data)

ratings_data = []
ratings_pred = []
for d in data:
    ratings_data.append(d['rating'])
    ratings_pred.append(predictRating(d['user_id'],d['book_id']))

mse = np.square(np.subtract(np.array(ratings_data),np.array(ratings_pred))).
 ↪mean()
```

[130]:
```
print(mse)
```

```
0.8097344461671871
```

Answer for Problem 4

a. MSE = 0.8097344461671871

[146]:
```
# Cosine similarity
def Cosine(i1, i2):
    # Between two items
    inter = usersPerItem[i1].intersection(usersPerItem[i2])
    numer = 0
    denom1 = 0
    denom2 = 0
    for u in inter:
        numer += ratingDict[(u,i1)]*ratingDict[(u,i2)]
    for u in usersPerItem[i1]:
        denom1 += ratingDict[(u,i1)]**2
    for u in usersPerItem[i2]:
        denom2 += ratingDict[(u,i2)]**2
    denom = math.sqrt(denom1) * math.sqrt(denom2)
```

```
    if denom == 0: return 0
    return numer / denom

def predictRating_Cosine(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(Cosine(item,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 ratingMean

ratings_pred = []
for d in data[0:10000]:
    ratings_data.append(d['rating'])
    ratings_pred.append(predictRating_Cosine(d['user_id'],d['book_id']))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/var/folders/x9/xv9fr3td34x85pl4rz2hy7kh0000gn/T/ipykernel_51326/3858428067.py␣
 ↪in <module>
     36     ratings_pred.append(predictRating_Cosine(d['user_id'],d['book_id'])
     37
---> 38 mse = np.square(np.subtract(np.array(ratings_data),np.
 ↪array(ratings_pred))).mean()
     39 print(mse)

ValueError: operands could not be broadcast together with shapes (20000,)␣
 ↪(10000,)
```

[148]:
```
ratings_data = [d['rating'] for d in data[0:10000] ]
mse = np.square(np.subtract(np.array(ratings_data),np.array(ratings_pred))).
 ↪mean()
print(mse)
```

```
0.7107374206362238
```

[152]:
```
# Pearson similarity
def Pearson_inter(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: #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

def predictRating_Pearson_inter(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(Pearson_inter(item,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 ratingMean

ratings_data = []
ratings_pred = []
for d in data[0:10000]:
    ratings_data.append(d['rating'])
    ratings_pred.append(predictRating_Pearson_inter(d['user_id'],d['book_id']))

mse = np.square(np.subtract(np.array(ratings_data),np.array(ratings_pred))).
 ↪mean()
print(mse)
```

```
8595.26558013147
```

[160]:
```
m = np.square(np.subtract(np.array(ratings_data),np.array(ratings_pred)))
```

[165]:
```
m.max()
```

[165]: 25385820.982252207

[166]:
```
def Pearson_union(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

def predictRating_Pearson_union(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(Pearson_union(item,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 ratingMean

ratings_data = []
ratings_pred = []
for d in data[0:10000]:
    ratings_data.append(d['rating'])
    ratings_pred.append(predictRating_Pearson_union(d['user_id'],d['book_id']))

mse = np.square(np.subtract(np.array(ratings_data),np.array(ratings_pred))).
 ↪mean()
print(mse)
```

9602.525337537289

[167]:
```
# Jaccard similarity interchanging users and items
def Jaccard(s1, s2):
    numer = len(s1.intersection(s2))
    denom = len(s1.union(s2))
    if denom == 0:
        return 0
    return numer / denom

def predictRating_Jaccard(user,item):
    ratings = []
    similarities = []
    for d in reviewsPerItem[item]:
        u2 = d['book_id']
        if u2 == user: continue
        ratings.append(d['rating'] - usersAverages[u2])
        similarities.append(Jaccard(itemsPerUser[user],itemsPerUser[u2]))
    if (sum(similarities) > 0):
        weightedRatings = [(x*y) for x,y in zip(ratings,similarities)]
        return userAverages[user] + sum(weightedRatings) / sum(similarities)
    else:
        # User hasn't rated any similar items
        return ratingMean

ratings_data = []
ratings_pred = []
for d in data[0:10000]:
    ratings_data.append(d['rating'])
    ratings_pred.append(predictRating_Pearson_union(d['user_id'],d['book_id']))

mse = np.square(np.subtract(np.array(ratings_data),np.array(ratings_pred))).
 ↪mean()
print(mse)
```

```
9602.525337537289
```

Answer for Problem 5

a. MSE for Cosine Similarity

   mse = 0.7107374206362238

b. MSE for Pearson Similarity, Denominator Intersection

   mse = 8595.26558013147

   MSE for Pearson Similarity, Denominator Union

   mse = 9602.525337537289

c. MSE for Jaccard Similarity interchanging users and items

   mse = 9602.525337537289

[ ]:

Answer for Problem 6