Homework 2 Question 1

October 25, 2021

1 Question 1 - Items which have the highest Jaccard similarity compared to the first item (i.e., the item from the first review, '18471619')? Report both 10 items similarities and item IDs

[1]:
```
import gzip
from collections import defaultdict
import scipy
import scipy.optimize
import numpy
import random
```

[2]:
```
def parseDataFromFile(fname):
    for l in open(fname):
        yield eval(l)
```

[3]:
```
data=list(parseDataFromFile("goodreads_reviews_comics_graphic.json"))
print(data[0])
```

```
{'user_id': 'dc3763cdb9b2cae805882878eebb6a32', 'book_id': '18471619',
'review_id': '66b2ba840f9bd36d6d27f46136fe4772', 'rating': 3, 'review_text':
'Sherlock Holmes and the Vampires of London \n Release Date: April 2014 \n
Publisher: Darkhorse Comics \n Story by: Sylvain Cordurie \n Art by: Laci \n
Colors by: Axel Gonzabo \n Cover by: Jean Sebastien Rossbach \n ISDN:
9781616552664 \n MSRP: $17.99 Hardcover \n "Sherlock Holmes died fighting
Professor Moriarty in the Reichenbach Falls. \n At least, that\'s what the press
claims. \n However, Holmes is alive and well and taking advantage of his
presumed death to travel the globe. \n Unfortunately, Holmes\'s plans are
thwarted when a plague of vampirism haunts Britain. \n This book collects
Sherlock Holmes and the Vampires of London Volumes 1 and 2, originally created
by French publisher Soleil." - Darkhorse Comics \n 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! \n 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. \n 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. \n 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. \n 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. \n 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. \n That artwork is good, cover is
great, story is lacking so I am giving it 2.5 out of 5 stars.', 'date_added':
'Thu Dec 05 10:44:25 -0800 2013', 'date_updated': 'Thu Dec 05 10:45:15 -0800
2013', 'read_at': 'Tue Nov 05 00:00:00 -0800 2013', 'started_at': '', 'n_votes':
0, 'n_comments': 0}
```

[4]:
```
userPerItem=defaultdict(set)
itemPerUser=defaultdict(set)
```

[5]:
```
itemNames={}
```

[6]:
```
for d in data:
    user=d["user_id"]
    item=d["book_id"]
    userPerItem[item].add(user)
    itemPerUser[user].add(item)
```

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

[8]:
```
def mostSimilar(i,N=10):
    users=userPerItem[i]
    similarities=[]
    for j in userPerItem:
        if(i==j):
            continue
        sim=Jaccard(users,userPerItem[j])
        similarities.append((sim,j))
#     similarities.sort(reverse=True)
    s=sorted(similarities, key=lambda element:(-element[0],element[1]))
    return(s)
```

[9]:
```
query=data[0]['book_id']
print("Similarities and Item IDs")
mostSimilar(query)[:10]
```

```
Similarities and Item IDs
```

[9]:
```
[(0.16666666666666666, '25334626'),
 (0.14285714285714285, '25659811'),
 (0.13793103448275862, '18369278'),
 (0.13157894736842105, '18430205'),
 (0.12903225806451613, '20299669'),
 (0.125, '17995154'),
 (0.12121212121212122, '18853527'),
 (0.12121212121212122, '23093378'),
 (0.12121212121212122, '23241671'),
 (0.11764705882352941, '18734070')]
```

Homework 2 Question 2

October 25, 2021

1 Question 2 There are several ways similar-item recommendations could be used to make personalized recommendations for a particular user. For instance we could: User= "dc3763cdb9b2cae805882878eebb6a32"

1.1 (a) Choosing the N items most similar to the user's favorite (i.e., highest rated) item.

1.2 (b) Finding the N most similar users, and recommending each of their their favorite (highest rated) items.

[1]:
```
import gzip
from collections import defaultdict
import scipy
import scipy.optimize
import numpy
import random
```

[2]:
```
def parseDataFromFile(fname):
    for l in open(fname):
        yield eval(l)
```

[3]:
```
data=list(parseDataFromFile("goodreads_reviews_comics_graphic.json"))
print(data[0])
```

```
{'user_id': 'dc3763cdb9b2cae805882878eebb6a32', 'book_id': '18471619',
'review_id': '66b2ba840f9bd36d6d27f46136fe4772', 'rating': 3, 'review_text':
'Sherlock Holmes and the Vampires of London \n Release Date: April 2014 \n
Publisher: Darkhorse Comics \n Story by: Sylvain Cordurie \n Art by: Laci \n
Colors by: Axel Gonzabo \n Cover by: Jean Sebastien Rossbach \n ISDN:
9781616552664 \n MSRP: $17.99 Hardcover \n "Sherlock Holmes died fighting
Professor Moriarty in the Reichenbach Falls. \n At least, that\'s what the press
claims. \n However, Holmes is alive and well and taking advantage of his
presumed death to travel the globe. \n Unfortunately, Holmes\'s plans are
thwarted when a plague of vampirism haunts Britain. \n This book collects
Sherlock Holmes and the Vampires of London Volumes 1 and 2, originally created
by French publisher Soleil." - Darkhorse Comics \n 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! \n 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. \n 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. \n 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. \n 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. \n 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. \n That artwork is good, cover is great, story is lacking so I am giving it 2.5 out of 5 stars.', 'date_added': 'Thu Dec 05 10:44:25 -0800 2013', 'date_updated': 'Thu Dec 05 10:45:15 -0800 2013', 'read_at': 'Tue Nov 05 00:00:00 -0800 2013', 'started_at': '', 'n_votes': 0, 'n_comments': 0}

[4]:
```
userPerItem=defaultdict(set)
itemPerUser=defaultdict(set)
```

[5]:
```
itemNames={}
```

[6]:
```
for d in data:
    user=d["user_id"]
    item=d["book_id"]
    userPerItem[item].add(user)
    itemPerUser[user].add(item)
```

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

[8]:
```
def mostSimilar(i,N=10):
    users=userPerItem[i]
    similarities=[]
    for j in userPerItem:
        if(i==j):
            continue
        sim=Jaccard(users,userPerItem[j])
        similarities.append((sim,j))
    s=sorted(similarities, key=lambda element:(-element[0],element[1]))
    #similarities.sort(reverse=True)
    return(s[:10])
```

[9]:
```
w=[]
for d in data:
    if(d["user_id"]=="dc3763cdb9b2cae805882878eebb6a32"):
        o=[]
        o.append(d["book_id"])
        o.append(d["rating"])
        w.append(o)

rating_u=sorted(w, key=lambda element:(element[1]),reverse=True)
query=rating_u[0][0]
print(query)
```

```
18471619
```

1.3 (a) Part

[10]:
```
# Part A
mostSimilar(query)
```

[10]:
```
[(0.16666666666666666, '25334626'),
 (0.14285714285714285, '25659811'),
 (0.13793103448275862, '18369278'),
 (0.13157894736842105, '18430205'),
 (0.12903225806451613, '20299669'),
```

```
 (0.125, '17995154'),
 (0.12121212121212122, '18853527'),
 (0.12121212121212122, '23093378'),
 (0.12121212121212122, '23241671'),
 (0.11764705882352941, '18734070')]
```

[11]:
```
from operator import itemgetter
lis_item=[]
for d in data:
    if(d["user_id"]=="dc3763cdb9b2cae805882878eebb6a32" and d["rating"]):
        item=[]
        item.append(d["rating"])
        item.append(d["book_id"])
        lis_item.append(item)

w=(sorted(lis_item, key=itemgetter(0),reverse=True))
print(w)
mostSimilar(w[0][1])
```

```
[[3, '18471619']]
```

[11]:
```
[(0.16666666666666666, '25334626'),
 (0.14285714285714285, '25659811'),
 (0.13793103448275862, '18369278'),
 (0.13157894736842105, '18430205'),
 (0.12903225806451613, '20299669'),
 (0.125, '17995154'),
 (0.12121212121212122, '18853527'),
 (0.12121212121212122, '23093378'),
 (0.12121212121212122, '23241671'),
 (0.11764705882352941, '18734070')]
```

1.4 (b) Part

[12]:
```
e1=[]
def mostSimilarUser(i,N=10):
    item1=itemPerUser[i]

    for j in itemPerUser:
        similarities=[]
        if(i==j):
            continue
        sim=Jaccard(item1,itemPerUser[j])
        if(sim==1):
            continue
        similarities.append(sim)
        similarities.append(j)
        e1.append(similarities)
```

```
    return(e1)

# def mostSimilarUser(i,N=10):
#     item1=itemPerUser[i]
#     similarities=[]
#     for j in itemPerUser:
#         if(i==j):
#             continue
#         sim=Jaccard(item1,itemPerUser[j])
#         similarities.append((sim,j))
#     similarities.sort(reverse=True)
#     return(similarities)
```

[13]:
```
kab=[]
kab=mostSimilarUser("dc3763cdb9b2cae805882878eebb6a32")
kab.sort(key=lambda x:x[0],reverse=True)
print("Users similar to given ID:  ")
for i in range(10):
    print(kab[i])
```

```
Users similar to given ID:
[0.3333333333333333, '6470c7f5e3468ba34e9fe628960fbbf1']
[0.25, '6497ca91df3c182006874c96a8530b37']
[0.2, '033cf640dfa6f85eb146c39787289628']
[0.14285714285714285, '5510684ab6c18f2dd493787e66b2722c']
[0.05555555555555555, '17f73ea38e97307935c0d3b6ca987b53']
[0.030303030303030304, 'a39b4249d201ef5ce5ea553bdd013e66']
[0.023809523809523808, '42519f961f79b61701bda60787b031cf']
[0.02040816326530612, '65a7975989734fc6e18b7d2bd2bcb49f']
[0.014925373134328358, '0fafb6f0843124383f4e2c5a2090fb09']
[0.0136986301369863, '071222e19ae29dc9fdbe225d983449be']
```

[14]:
```
top_user=[]
for i in range(10):

    top_user.append(kab[i][1])
```

[15]:
```
# Part B
# N most similar users, and recommending each of their their favorite (highest␣
 ↪rated) items.
print("Favourite Items recommended by similar User ID")
for i in top_user:
    q=itemPerUser[i]
    q=list(q)
    w1=[]
    op=0
```

```
    for d in data:
        c1=[]
        if(d["user_id"]==i and d["book_id"] in q):
            c1.append(d["user_id"])
            c1.append(d["book_id"])
            c1.append(d["rating"])
            op=max(op,d["rating"])
        if(len(c1)>0):
            w1.append(c1)
    abc=(sorted(w1, key=itemgetter(2,1),reverse=False))

    for ab in abc:
        if(ab[2]==op):
            print(ab)
            break
    #print(abc)

#     abc.sort(key=lambda x:x[2],reverse=True)
#     print(abc)
```

```
Favourite Items recommended by similar User ID
['6470c7f5e3468ba34e9fe628960fbbf1', '10767466', 4]
['6497ca91df3c182006874c96a8530b37', '17570797', 5]
['033cf640dfa6f85eb146c39787289628', '15704307', 5]
['5510684ab6c18f2dd493787e66b2722c', '10138607', 5]
['17f73ea38e97307935c0d3b6ca987b53', '12434747', 5]
['a39b4249d201ef5ce5ea553bdd013e66', '17995248', 5]
['42519f961f79b61701bda60787b031cf', '10105459', 5]
['65a7975989734fc6e18b7d2bd2bcb49f', '10997645', 5]
['0fafb6f0843124383f4e2c5a2090fb09', '10361139', 5]
['071222e19ae29dc9fdbe225d983449be', '10264328', 5]
```

Homework 2 Question 3

October 25, 2021

1 Question 3 In class we briefly discussed whether the Pearson similarity should be implemented (a) only in terms of shared items (i.e., Ui Uj) in the denominator; or (b) in terms of all items each user consumed (i.e., Ui or Uj for each term in the denominator). (See last slide on Pearson similarity). Implement versions of the Pearson similarity based on both definitions, and report the 10 most similar items to the same query item from Question 1

[1]:
```
import gzip
from collections import defaultdict
import scipy
import scipy.optimize
import numpy
import random
```

[2]:
```
def parseDataFromFile(fname):
    for l in open(fname):
        yield eval(l)
```

[3]:
```
data=list(parseDataFromFile("goodreads_reviews_comics_graphic.json"))
```

[4]:
```
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']
```

[5]:
```
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)
```

1.1 (a) only in terms of shared items

[6]:
```
def sharedPearson(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
```

[7]:
```
def mostSimilar(i, N):
    similarities = []
    users = usersPerItem[i]
    for i2 in usersPerItem:
        if i2 == i: continue
        sim = sharedPearson(i, i2) # Could use alternate similarity metrics␣
 ↪straightforwardly
        similarities.append((sim,i2))
    #similarities.sort(reverse=True)
    s=sorted(similarities, key=lambda element:(-element[0],element[1]))
    return s[:10]
```

[8]:
```
query = data[0]['book_id']
```

[9]:
```
import math
ms = mostSimilar(query, 10)
```

[10]:
```
ms[0:10]
```

[10]:
```
[(1.0000000000000002, '1103951'),
 (1.0000000000000002, '11986350'),
 (1.0000000000000002, '16007365'),
 (1.0000000000000002, '17132269'),
 (1.0000000000000002, '17571519'),
 (1.0000000000000002, '18468852'),
 (1.0000000000000002, '18527488'),
 (1.0000000000000002, '18594657'),
 (1.0000000000000002, '18624024'),
 (1.0000000000000002, '1882498')]
```

1.2 (b) in terms of all items each user consumed

[11]:
```
def unionPearson(i1, i2):
    # Between two items
    iBar1 = itemAverages[i1]
    iBar2 = itemAverages[i2]
    inter1 = usersPerItem[i1].intersection(usersPerItem[i2])
    numer = 0
    denom1 = 0
    denom2 = 0
    for u in inter1:
        numer += (ratingDict[(u,i1)] - iBar1)*(ratingDict[(u,i2)] - iBar2)
    for u in usersPerItem[i1]: #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
```

[12]:
```
def mostSimilarUnion(i, N):
    similarities = []
    users = usersPerItem[i]
    for i2 in usersPerItem:
        if i2 == i: continue
        sim = unionPearson(i, i2) # Could use alternate similarity metrics␣
 ↪straightforwardly
        similarities.append((sim,i2))
    #similarities.sort(reverse=True)
    s=sorted(similarities, key=lambda element:(-element[0],element[1]))
    return s
```

[13]:
```
query = data[0]['book_id']
print(query)
```

```
18471619
```

[14]:
```
import math
ms = mostSimilarUnion(query, 10)
```

[15]:
```
ms[:10]
```

[15]:
```
[(0.31898549007874194, '20300526'),
 (0.18785865431369264, '13280885'),
 (0.17896391275176457, '18208501'),
 (0.16269036695641687, '21521612'),
 (0.16269036695641687, '25430791'),
 (0.1555075595594449, '1341758'),
 (0.1526351566298752, '6314737'),
 (0.15204888048160353, '4009034'),
 (0.1494406444160154, '988744'),
 (0.14632419481281994, '18430205')]
```

Homework 2 Question 4

October 25, 2021

1 Question 4 Implement a rating prediction model based on the similarity function Report the MSE of this rating prediction function when Sim(i, j) = Jaccard(i, j)

[1]:
```
import gzip
from collections import defaultdict
import scipy
import scipy.optimize
import numpy
import random
```

[2]:
```
def parseDataFromFile(fname):
    for l in open(fname):
        yield eval(l)
```

[3]:
```
data=list(parseDataFromFile("goodreads_reviews_comics_graphic.json"))
print(data[1])
```

```
{'user_id': 'bafc2d50014200cda7cb2b6acd60cd73', 'book_id': '6315584',
'review_id': '72f1229aba5a88f9e72f0dcdc007dd22', 'rating': 4, 'review_text':
"I've never really liked Spider-Man. I am, however, a huge fan of the Dresden
Files. Jim Butcher is clever and sarcastic and probably the perfect choice to
pen a superhero novel. I really enjoyed this book!", 'date_added': 'Wed Aug 10
06:06:48 -0700 2016', 'date_updated': 'Fri Aug 12 08:49:54 -0700 2016',
'read_at': 'Fri Aug 12 08:49:54 -0700 2016', 'started_at': 'Wed Aug 10 00:00:00
-0700 2016', 'n_votes': 0, 'n_comments': 0}
```

[4]:
```
reviewsPerUser = defaultdict(list)
reviewsPerItem = defaultdict(list)
```

[5]:
```
for d in data:
    user,item = d['user_id'], d['book_id']
    reviewsPerUser[user].append(d)
    reviewsPerItem[item].append(d)
```

[6]:
```
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']
```

[7]:
```
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)
```

[8]:
```
ratingMean = sum([d['rating'] for d in data]) / len(data)
```

[9]:
```
def Jaccard(s1, s2):
    numer = len(s1.intersection(s2))
    denom = len(s1.union(s2))
    if denom == 0:
        return 0
    return numer / denom
```

[10]:
```
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]
```

1.1 For 10,000 Items

[11]:
```
predictions=[]
for d in data[:10000]:
    t=predictRating(d["user_id"],d["book_id"])
    predictions.append(t)
```

[12]:
```
def MSE(predictions, labels):
    differences = [(x-y)**2 for x,y in zip(predictions,labels)]
    return sum(differences) / len(differences)
```

[13]:
```
labels = [d['rating'] for d in data[:10000]]
```

[14]:
```
print("MSE for 10,000 items")
MSE(predictions,labels)
```

```
MSE for 10,000 items
```

[14]: 0.7017041185560355

1.2 For all Items

[15]:
```
predictions=[]
for d in data:
    t=predictRating(d["user_id"],d["book_id"])
    predictions.append(t)
```

[16]:
```
def MSE(predictions, labels):
    differences = [(x-y)**2 for x,y in zip(predictions,labels)]
    return sum(differences) / len(differences)
```

[17]:
```
labels = [d['rating'] for d in data]
```

[18]:
```
print("MSE for all items")
MSE(predictions,labels)
```

```
MSE for all items
```

[18]: 0.7908367015187353

Homework 2 Question 6

October 25, 2021

1 Question 6 Recommender Systems based on Temporal Dynamics.

[1]:
```
import gzip
from collections import defaultdict
import scipy
import scipy.optimize
import numpy
import random
import math
```

[2]:
```
def parseDataFromFile(fname):
    for l in open(fname):
        yield eval(l)
```

[3]:
```
data=list(parseDataFromFile("goodreads_reviews_comics_graphic.json"))
print(data[1])
```

```
{'user_id': 'bafc2d50014200cda7cb2b6acd60cd73', 'book_id': '6315584',
'review_id': '72f1229aba5a88f9e72f0dcdc007dd22', 'rating': 4, 'review_text':
"I've never really liked Spider-Man. I am, however, a huge fan of the Dresden
Files. Jim Butcher is clever and sarcastic and probably the perfect choice to
pen a superhero novel. I really enjoyed this book!", 'date_added': 'Wed Aug 10
06:06:48 -0700 2016', 'date_updated': 'Fri Aug 12 08:49:54 -0700 2016',
'read_at': 'Fri Aug 12 08:49:54 -0700 2016', 'started_at': 'Wed Aug 10 00:00:00
-0700 2016', 'n_votes': 0, 'n_comments': 0}
```

[4]:
```
reviewsPerUser = defaultdict(list)
reviewsPerItem = defaultdict(list)
```

[5]:
```
for d in data:
    user,item = d['user_id'], d['book_id']
    reviewsPerUser[user].append(d)
    reviewsPerItem[item].append(d)
```

[6]:
```
import dateutil.parser
import time
d1=data[0]
dt1 = dateutil.parser.parse(d1["date_added"])
```

```
q1=int(time.mktime(dt1.timetuple()))
mi=q1
ma=0
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
timeStamp = {}
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']
    dt = dateutil.parser.parse(d["date_added"])
    q=int(time.mktime(dt.timetuple()))
    mi=min(mi,q)
    ma=max(ma,q)
    timeStamp[(user,item)]=q
```

[7]:
```
for d in data:
    oab=timeStamp[(d["user_id"],d["book_id"])]
    oab=oab-mi
    ran=ma-mi
    timeStamp[(d["user_id"],d["book_id"])]=(oab/ran)
```

[8]:
```
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)
```

[9]:
```
ratingMean = sum([d['rating'] for d in data]) / len(data)
avgTimeList=[timeStamp[key] for key in timeStamp]
avgTime=sum(avgTimeList)/len(avgTimeList)
```

[10]:
```
def timeFunction(item,i2):
    eConstant=math.e
    co=timeStamp[(user,item)]
    lamdat=-5*(abs(co-avgTime)/avgTime)
    val=math.pow(eConstant,lamdat)
    return(val)
```

[11]:
```
def Jaccard(s1, s2):
    numer = len(s1.intersection(s2))
    denom = len(s1.union(s2))
    if denom == 0:
        return 0
    return numer / denom
```

[12]:
```
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])
        ab=Jaccard(usersPerItem[item],usersPerItem[i2])

        #timefunction
        t1=timeStamp[(user,item)]
        t2=timeStamp[(user,i2)]
        tabs=abs(t1-t2)
        eConstant=math.e
        lamdat=-5*tabs
        val=math.pow(eConstant,lamdat)

        ab=ab*val
        similarities.append(ab)
    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]
```

1.1 For 10,000 Items

[13]:
```
predictions=[]
for d in data[:10000]:
    t=predictRating(d["user_id"],d["book_id"])
    predictions.append(t)
```

[14]:
```
def MSE(predictions, labels):
    differences = [(x-y)**2 for x,y in zip(predictions,labels)]
    return sum(differences) / len(differences)
```

[15]:
```
labels = [d['rating'] for d in data[:10000]]
```

[16]:
```
print("MSE for 10,000 items")
MSE(predictions,labels)
```

```
MSE for 10,000 items
```

[16]:
```
0.6974678265068076
```

1.2 For all Items

[17]:
```
predictions=[]
for d in data:
    t=predictRating(d["user_id"],d["book_id"])
    predictions.append(t)
```

[18]:
```
def MSE(predictions, labels):
    differences = [(x-y)**2 for x,y in zip(predictions,labels)]
    return sum(differences) / len(differences)
```

[19]:
```
labels = [d['rating'] for d in data]
```

[20]:
```
print("MSE for all items")
MSE(predictions,labels)
```

```
MSE for all items
```

[20]:
```
0.7827220281567892
```