CSE 258 - HW 2

Jin Dai / A92408103

Tasks - Similarity Functions

1
```
import gzip
import io
import math
from collections import defaultdict
from urllib.request import urlopen
```

2
```
url = 'https://cseweb.ucsd.edu//classes/fa21/cse258-b/data/goodreads_reviews_comics_graphic.json
```

3
```
def download_and_decompose(url):
    print('Downloading data...')
    handle = urlopen(url)
    f = gzip.GzipFile(fileobj=io.BytesIO(handle.read()))
    print('Downloaded. Decomposing...')
    for line in f:
        yield eval(line)
    print('Decomposed.')
```

4
```
review_data = list(download_and_decompose(url))
```

```
Downloading data...
Downloaded. Decomposing...
Decomposed.
```

5
```
review_data[0]
```

5

```
{'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 Publis
 '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}
```


6
```
users_per_book = defaultdict(set)
for r in review_data:
    user, book = r['user_id'], r['book_id']
    users_per_book[book].add(user)
```

Question. 1

7
```
users_per_book[review_data[0]['book_id']]
```

7
```
{'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'}
```

8
```
def Jaccard(s1, s2):
    numer = len(s1.intersection(s2))
    denom = len(s1.union(s2))
    if denom == 0:
        return 0
    return numer / denom
```

9
```
def most_similar(i, N, users_per_item, sim_func):
    similarities = []
    users = users_per_item[i]
    for i2 in users_per_item:
        if i2 == i: continue
        sim = sim_func(users, users_per_item[i2])
        similarities.append((sim,i2))
    similarities.sort(reverse=True)
    return similarities[:N]
```

10
```
review_data[0]['book_id']
```

10
```
'18471619'
```

11
```
t10_jaccard_sim = most_similar(review_data[0]['book_id'], 10, users_per_book, Jaccard)
```

12
```
print('The top 10 items with the highest Jaccard similarity compared to the first item are:')
t10_jaccard_sim
```

The top 10 items with the highest Jaccard similarity compared to the first item are:

12
```
[(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

(a)

13
```
def most_similar_books(i, N, users_per_book, user_interactions, sim_func):
    similarities = []
    users = users_per_book[i]
    for i2 in users_per_book:
        if i2 == i: continue
        if i2 in user_interactions:
            print('The target user has already read book %s. Hence skipping...' % i2)
            continue
        sim = sim_func(users, users_per_book[i2])
        similarities.append((sim,i2))
    similarities.sort(reverse=True)
    return similarities[:N]
```

First we get the user's highest rated book.

14
```
user_id = 'dc3763cdb9b2cae805882878eebb6a32'
books_per_user = defaultdict(set)
rating_dict = {}

for r in review_data:
    user, book = r['user_id'], r['book_id']
    books_per_user[user].add(book)
    rating_dict[(user, book)] = r['rating']
```

15
```
[(b, rating_dict[(user_id, b)]) for b in books_per_user[user_id]]
```

15
```
[('18471619', 3)]
```

16
```
def top_rated_item_by(user, items_per_user, rating_dict):
    items = items_per_user[user]
    return max(items, key=lambda item : rating_dict[(user, item)])
```

17
```
top_rated = top_rated_item_by(user_id, books_per_user, rating_dict)
```

18
```
print('The highest rated book by user %s is %s' % (user_id, top_rated))
```

The highest rated book by user dc3763cdb9b2cae805882878eebb6a32 is 18471619

Then we recommend the 10 most similar books to book: '18471619' to the user.

19
```
books_per_user[user_id]
```

19
```
{'18471619'}
```

20
```
print('Our recommendations:')
most_similar_books(top_rated, 10, users_per_book, books_per_user[user_id], Jaccard)
```

```
Our recommendations:
```

20
```
[(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')]
```

(b)

21
```
def most_similar_users(u, N, items_per_user, sim_func):
    similarities = []
    items = items_per_user[u]
    for u2 in items_per_user:
        if u2 == u: continue
        sim = sim_func(items, items_per_user[u2])
        similarities.append((sim,u2))
    similarities.sort(reverse=True)
    return similarities[:N]
```

First, find the most similar users.

22
```
most_sim_users = most_similar_users(user_id, 20, books_per_user, Jaccard)
print('The most similar users to user %s are:' % user_id)
most_sim_users
```

```
The most similar users to user dc3763cdb9b2cae805882878eebb6a32 are:
```

22
```
[(1.0, '4ae069d704b11bdf12c25fe640f75ff0'),
 (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'),
 (0.013157894736842105, '7f63e4d65e873703970e71afabbc3b54'),
 (0.007751937984496124, 'd7310760f68365d3ca747fa8b9310518'),
 (0.006622516556291391, '68dff5594b77c47aae96cbe97aba5206'),
 (0.006097560975609756, 'eaa54d876d841293059657fb80a9bba6'),
 (0.005780346820809248, '8d06514d97530ddb22a05b84dfe4daad'),
```

```
 (0.0051813471502590676, '9d4feff5432a5a5243bf277e0d258042'),
 (0.004694835680751174, 'da7a0c5ee0c89973224d8853445be68e'),
 (0.004484304932735426, '714ed8e9b1814bf45dd9abd88431dbb8'),
 (0.004098360655737705, '9f6f9da3a71ded406f15764f8fbf5f51')]
```

Then, we recommend the favorite books from each of the most similar users to the target user.

23
```
recommendations = []
for _, user in most_sim_users:
    top_pick = top_rated_item_by(user, books_per_user, rating_dict)
    print('Favorite book by user %s is %s' % (user, top_pick))
    if top_pick in books_per_user[user_id]:
        print('The target user has already read book %s. Hence skipping...' % top_pick)
        continue
    recommendations.append(top_pick)
```

```
Favorite book by user 4ae069d704b11bdf12c25fe640f75ff0 is 18471619
The target user has already read book 18471619. Hence skipping...
Favorite book by user 6470c7f5e3468ba34e9fe628960fbbf1 is 10767466
Favorite book by user 6497ca91df3c182006874c96a8530b37 is 23531233
Favorite book by user 033cf640dfa6f85eb146c39787289628 is 15931937
Favorite book by user 5510684ab6c18f2dd493787e66b2722c is 7736086
Favorite book by user 17f73ea38e97307935c0d3b6ca987b53 is 22454333
Favorite book by user a39b4249d201ef5ce5ea553bdd013e66 is 18720887
Favorite book by user 42519f961f79b61701bda60787b031cf is 6393176
Favorite book by user 65a7975989734fc6e18b7d2bd2bcb49f is 18260395
Favorite book by user 0fafb6f0843124383f4e2c5a2090fb09 is 24375349
Favorite book by user 071222e19ae29dc9fdbe225d983449be is 10638896
Favorite book by user 7f63e4d65e873703970e71afabbc3b54 is 917459
Favorite book by user d7310760f68365d3ca747fa8b9310518 is 32894544
Favorite book by user 68dff5594b77c47aae96cbe97aba5206 is 31220490
Favorite book by user eaa54d876d841293059657fb80a9bba6 is 13495085
Favorite book by user 8d06514d97530ddb22a05b84dfe4daad is 15953593
Favorite book by user 9d4feff5432a5a5243bf277e0d258042 is 7919401
Favorite book by user da7a0c5ee0c89973224d8853445be68e is 43747
Favorite book by user 714ed8e9b1814bf45dd9abd88431dbb8 is 105880
Favorite book by user 9f6f9da3a71ded406f15764f8fbf5f51 is 21535713
```

24
```
print('Our recommendations:')
recommendations[:10]
```

```
Our recommendations:
```

24
```
['10767466',
 '23531233',
 '15931937',
 '7736086',
 '22454333',
 '18720887',
 '6393176',
 '18260395',
 '24375349',
 '10638896']
```

Question. 3

25
```
item_avgs = {}

for i in users_per_book:
```

```
    rs = [rating_dict[(u,i)] for u in users_per_book[i]]
    item_avgs[i] = sum(rs) / len(rs)
```

```
26 def Pearson_1(i1, i2, users_per_item, item_avg):
    # Between two items
    i_bar1 = item_avg[i1]
    i_bar2 = item_avg[i2]
    inter = users_per_item[i1].intersection(users_per_item[i2])
    numer = 0
    denom1 = 0
    denom2 = 0
    for u in inter:
        numer += (rating_dict[(u,i1)] - i_bar1)*(rating_dict[(u,i2)] - i_bar2)
    for u in inter: #usersPerItem[i1]:
        denom1 += (rating_dict[(u,i1)] - i_bar1)**2
    #for u in usersPerItem[i2]:
        denom2 += (rating_dict[(u,i2)] - i_bar2)**2
    denom = math.sqrt(denom1) * math.sqrt(denom2)
    if denom == 0: return 0
    return numer / denom
```

```
27 def Pearson_2(i1, i2, users_per_item, item_avg):
    # Between two items
    i_bar1 = item_avg[i1]
    i_bar2 = item_avg[i2]
    inter = users_per_item[i1].intersection(users_per_item[i2])
    numer = 0
    denom1 = 0
    denom2 = 0
    for u in inter:
        numer += (rating_dict[(u,i1)] - i_bar1)*(rating_dict[(u,i2)] - i_bar2)
    for u in users_per_item[i1]:
        denom1 += (rating_dict[(u,i1)] - i_bar1)**2
    for u in users_per_item[i2]:
        denom2 += (rating_dict[(u,i2)] - i_bar2)**2
    denom = math.sqrt(denom1) * math.sqrt(denom2)
    if denom == 0: return 0
    return numer / denom
```

```
28 def most_similar_pearson(i, N, users_per_item, item_avg, pearson):
    similarities = []
    for i2 in users_per_item:
        if i2 == i: continue
        sim = pearson(i, i2, users_per_item, item_avg)
        similarities.append((sim,i2))
    similarities.sort(reverse=True)
    return similarities[:N]
```

```
29 t10_pearson_1_sim = most_similar_pearson(review_data[0]['book_id'], 10, users_per_book, item_avg
```

```
30 print('The top 10 items with the highest similarity based on Pearson with implementation 1 compa
t10_pearson_1_sim
```

```
The top 10 items with the highest similarity based on Pearson with implementation 1 compared to
```

30
```
[(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')]
```

31
```
t10_pearson_2_sim = most_similar_pearson(review_data[0]['book_id'], 10, users_per_book, item_avg
```

32
```
print('The top 10 items with the highest similarity based on Pearson with implementation 2 compa
t10_pearson_2_sim
```

```
The top 10 items with the highest similarity based on Pearson with implementation 2 compared to 
```

32
```
[(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'),
 (0.14632419481281997, '18430205')]
```

32

CSE 258 - HW 2

Jin Dai / A92408103

Tasks - Rating Prediction

1
```
import gzip
import io
from collections import defaultdict
from urllib.request import urlopen
```

2
```
url = 'https://cseweb.ucsd.edu//classes/fa21/cse258-b/data/goodreads_reviews_comics_graphic.json
```

3
```
def download_and_decompose(url):
    print('Downloading data...')
    handle = urlopen(url)
    f = gzip.GzipFile(fileobj=io.BytesIO(handle.read()))
    print('Downloaded. Decomposing...')
    for line in f:
        yield eval(line)
    print('Decomposed.')
```

4
```
review_data = list(download_and_decompose(url))
```

```
Downloading data...
Downloaded. Decomposing...
Decomposed.
```

5
```
def Jaccard(s1, s2):
    numer = len(s1.intersection(s2))
    denom = len(s1.union(s2))
    if denom == 0:
        return 0
    return numer / denom
```

Question. 4

6
```
users_per_item = defaultdict(set)
items_per_user = defaultdict(set)
rating_dict = {}
reviews_per_user = defaultdict(list)
reviews_per_item = defaultdict(list)
for r in review_data:
    user,item = r['user_id'], r['book_id']
    users_per_item[item].add(user)
    items_per_user[user].add(item)
    rating_dict[(user, item)] = r['rating']
```

```
    reviews_per_user[user].append(r)
    reviews_per_item[item].append(r)
```

7

```
user_avg = {}
item_avg = {}

for u in items_per_user:
    rs = [rating_dict[(u,i)] for i in items_per_user[u]]
    user_avg[u] = sum(rs) / len(rs)

for i in users_per_item:
    rs = [rating_dict[(u,i)] for u in users_per_item[i]]
    item_avg[i] = sum(rs) / len(rs)
```

8

```
def predict_rating(user, item):
    weighted_ratings_sum = 0
    similarities_sum = 0
    for r in reviews_per_user[user]:
        i2 = r['book_id']
        if i2 == item: continue
        sim = Jaccard(users_per_item[item],users_per_item[i2])
        weighted_ratings_sum += (r['rating'] - item_avg[i2]) * sim
        similarities_sum += sim
    if similarities_sum > 0:
        return item_avg[item] + weighted_ratings_sum / similarities_sum
    else:
        # User hasn't rated any similar items
        return item_avg[item]
```

9

```
sim_predictions = [predict_rating(r['user_id'], r['book_id']) for r in review_data]
```

10

```
labels = [r['rating'] for r in review_data]
```

11

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

12

```
MSE(sim_predictions, labels)
```

12

```
0.7908367015187353
```

Question. 6

Design:

1. First, we want to base the decay function on top of the delta between timestamps when reviews are added. We use months as the delta's granularity since intuitively a reader's rating toward a book would not change much on a finer granularity like minutes, hours, or days.

2. We choose exponential decay function over the month-based delta since we believe ratings with the shortest delta tend to have the most strong indication of the next user's rating at present. The indication or effect would drop exponentially to a relatively static level if the delta becomes too large ($\lim_{\delta_t \to \infty} f(\delta_t) = 0$).
3. To choose the best λ for the exponential decay function, ideally we should use gradient decent to get the best answer. But for simplicity, we prepare a few candidates and compute the MSE using each. The candidate that gives the smallest MSE should be a good enough choice for our λ to outperform the trivial function.

13
```
import math
```

14
```
def time_weight_factor(time_diff, l):
    return math.exp(-l * time_diff)
```

15
```
review_data[0]
```

15
```
{'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 Publi
 '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}
```

16
```
from dateutil.parser import parse
```

17
```
parse(review_data[0]['date_added'])
```

17
```
datetime.datetime(2013, 12, 5, 10, 44, 25, tzinfo=tzoffset(None, -28800))
```

18
```
parse(review_data[0]['date_added']).year
```

18
```
2013
```

19
```
parse(review_data[0]['date_added']).month
```

19
```
12
```

20
```
def parse_to_months(t):
    parsed = parse(t)
    return parsed.year * 12 + parsed.month
```

21
```
timestamp_dict = {}
for r in review_data:
```

```
    user,item = r['user_id'], r['book_id']
    timestamp_dict[(user, item)] = parse_to_months(r['date_added'])
```

22

```
def time_weighted_predict_rating(user, item, l):
    weighted_ratings_sum = 0
    similarities_sum = 0
    for r in reviews_per_user[user]:
        i2 = r['book_id']
        if i2 == item: continue
        sim = Jaccard(users_per_item[item],users_per_item[i2])
        time_weight = time_weight_factor(abs(timestamp_dict[(user, item)] - timestamp_dict[(user
        time_weighted_sim = sim * time_weight
        weighted_ratings_sum += (r['rating'] - item_avg[i2]) * time_weighted_sim
        similarities_sum += time_weighted_sim
    if similarities_sum > 0:
        return item_avg[item] + weighted_ratings_sum / similarities_sum
    else:
        # User hasn't rated any similar items
        return item_avg[item]
```

*

```
for l in [1, 0.1, 0.01, 0.001, 0.0001]:
    time_weighted_sim_predictions = [time_weighted_predict_rating(r['user_id'], r['book_id'], l)
    mse = MSE(time_weighted_sim_predictions, labels)
    print('lambda: %f, mse: %f' % (l, mse))
```

```
lambda: 1.000000, mse: 0.872494
lambda: 0.100000, mse: 0.786729
lambda: 0.010000, mse: 0.786887
lambda: 0.001000, mse: 0.790366
```

Therefore, based on the design of our decay function, the best MSE we can get is 0.786729 at $\lambda = 0.1$. This result outperforms MSE=0.7908367 using the trivial decay function by ~0.0041.