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.