Data Science

Assignment4 : Recommender

2016025532 컴퓨터전공 심수정

1. Summary of Algorithm

이 과제는 *.base 에 정의된 user, item, rank, time stamp 등을 활용하여 *.test file 에서 기존의 정의되지 않은 user-item pair 에 대해 rank 를 예측하는 것을 목표로 합니다.

그것을 위해 저의 code 는 CF algorithm 을 사용합니다. CF algorithm 은 나와 가장 유사한 특성을 가지는 다른 user(neighbor user)들을 찾아, 이들이 평가한 내용을 바탕으로 평가하는 algorithm 입니다. 이 경우에는 '같은 item 에 대해 얼마나 비슷한 평가를 매겼는가'가 기준이 되며, 이를 위해 similarity function 을 사용하게 됩니다. 저의 algorithm 에서 사용한 similarity function 은 아래와 같습니다.

$$\mathrm{sim}(x,y) = \frac{\sum(r_{ax}-\overline{r_a})(r_{by}-\overline{r_b})}{\sqrt{\sum(r_{ax}-\overline{r_a})}\sqrt{\sum(r_{by}-\overline{r_b})}}$$

이 함수를 통해 similarity 가 0 이상인 user 중 가장 similarity 가 높은 user 대략 15 명 정도를 neighbor 로 가정합니다.

또한 "neighbor user 들을 통해 어떻게 평가할까"를 예측하기 위해 아래와 같은 함수를 사용합니다.

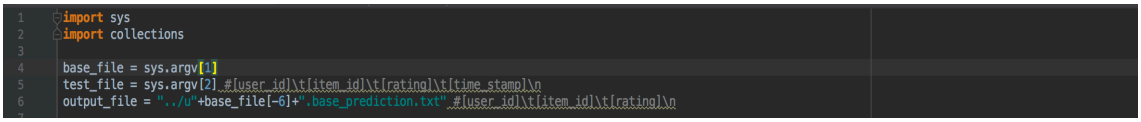
$$rate = \overline{r_a} + \frac{\sum(r_{bx}-\overline{r_b}) * weight}{\sum weight}$$

이 때 제가 이 project 에서 사용한 weight 값은 아래와 같습니다.

$$\mathrm{weight} = \mathrm{user\ similarity} * \mathrm{timestamp}$$

이를 통해서 예측을 하고, 평가를 하여 *.base_prediction.txt 파일에 예측된 값과 user, item 을 함께 출력하게 됩니다.

2. Detailed Description of Code



```
import sys
import collections

base_file = sys.argv[1]
test_file = sys.argv[2] #[user_id]\t[item_id]\t[rating]\t[time_stamp]\n
output_file = "../u"+base_file[-6]+".base_prediction.txt" #[user_id]\t[item_id]\t[rating]\n
```

code 를 실행할 때 base file, test file 을 받아옵니다. 그리고 이를 통하여 결과

를 쓸 output file 의 이름을 정합니다.

```
neighbor_number = 15
```

neighbor 의 수를 15 로 지정합니다.

```
with open(base_file) as base_:
    data_set = base.read().split()
    data_set = list(map(int, data_set))
    data_set = [data_set[i:i+4] for i in range(0, len(data_set), 4)]
```

base file 에서 user, item, rank, time stamp 를 읽어 이를 [[user1, item1, rank1, time stamp1], [user1, item2, rank2, time stamp2]] 의 형태로 data set 에 넣어 줍니다.

```
data_set_per_user = {}
for i in data_set_:
    if i[0] in data_set_per_user.keys()_:
        data_set_per_user[i[0]].append(([i[1], i[2], i[3]]))
    else_:
        data_set_per_user[i[0]] = [([i[1], i[2], i[3]])]
```

위에서 얻은 data set 을 user 당 data 로 표현합니다. 이는 {user : [[item1, rank1, time stamp1], [item2, rank2, time stamp2]]}의 형태가 됩니다.

```
rate_average = {}
for i in data_set_per_user.keys()_:
    rate_average[i] = sum([x[1] for x in data_set_per_user[i]]) / len(data_set_per_user[i])
```

user 가 평가한 item 들의 평균이 몇점인지 표시합니다. 이는 {user : rank}의 형태가 됩니다.

```
def get_neighbor (data_per_user)_:
    intersection_per_user = {}_# common # : user pair

    users = list(data_per_user.keys())

    count = 0
    for i in users_:
        for j in users_:
            if users.index(i) >= users.index(j)_:
                continue

            """
            common = get_common_item(data_per_user[i], data_per_user[j])

            if len(common) in list(intersection_per_user.keys()) :
                intersection_per_user[len(common)].append((i, j))
            else :
                intersection_per_user[len(common)] = [(i, j)]
            """
            similarity = get_similarity(data_per_user[i], data_per_user[j])
            if count % 10000 == 0_:
                print(i, j, similarity)
            count = count + 1
            if (similarity) in list(intersection_per_user.keys()) and similarity > 0.009:
                intersection_per_user[similarity].append((i, j))
            else_:
                intersection_per_user[similarity] = [(i, j)]

    intersection_per_user = collections.OrderedDict(sorted(intersection_per_user.items(), reverse=True))
    neighbor_dict = {}
    neighbor_list = []

    for i in users_:
        user_neighbor = []
        for j in intersection_per_user.keys()_:
            if j > 0_:
                for k in intersection_per_user[j]_:
                    if i in k_:
                        user_neighbor.append([j, list(set(k)-set([i]))[0]])

                if len(user_neighbor) > neighbor_number_:
                    break

        neighbor_list.append(user_neighbor)
        neighbor_dict[i] = neighbor_list

    return neighbor_dict
```

간단하게 말하면 다른 user 들과의 similarity 를 확인하고, 이중 similarity 가 높은 항목들을 이웃으로 넣어 반환하는 함수입니다. 우선적으로 user 들의 pair 를 simillarity 를 key 로 저장하는 dictionary 를 만듭니다. 이 dictionary 는 {similarity : [user pair1, user pair2]}의 형태를 가집니다.

후에는 이를 similarity 순으로 정렬하고, user 별 neighbor 를 neighbor_dict 에

{user : [[similarity1, neighbor1], [similarity2, neighbor2]]}의 형태로 저장합니다. 만약 같은 similarity 에서 이 작업을 한 후 이웃의 수가 neighbor number 로 지정된 수를 넘는다면 이 user 에 대해서 종료합니다.

최종적으로는 이렇게 형성된 neighbor dict 를 return 하게 됩니다.

```
def get_similarity (item_x, item_y) :
    item_x_rank = list(map(lambda x: x[1], item_x))
    item_y_rank = list(map(lambda y: y[1], item_y))
    avg_x = sum(item_x_rank) / len(item_x_rank)
    avg_y = sum(item_y_rank) / len(item_y_rank)
    x_square_root = sum(map(lambda x_: (x - avg_x)**2, item_x_rank))**0.5
    y_square_root = sum(map(lambda y_: (y - avg_y)**2, item_y_rank))**0.5
    similarity = sum([(ix[1]-avg_x)*(iy[1]-avg_y) for ix, iy in zip(item_x, item_y) if ix[0] == iy[0]])
    return ((similarity) / (x_square_root * y_square_root))
```

간단하게 similarity 를 구해서 return 하는 함수입니다. 위에서 언급한 식이였던 " $\mathrm{sim}(x,y) = \frac{\sum(r_x-\overline{r_x})(r_y-\overline{r_y})}{\sqrt{\sum(r_x-\overline{r_x})}\sqrt{\sum(r_y-\overline{r_y})}}$ "를 활용해 계산하여 return 합니다.

```
def predict(neighbor, data_per_user, rate_avg, test_set):
    neighbor_rank = []

    for i in neighbor[test_set[0]]:
        for j in i:
            for k in data_per_user[j[1]]:
                if k[0] == test_set[1]:
                    new_val = (k[1]-rate_avg[j[1]])
                    neighbor_rank.append([new_val, j[0], k[2]])

    if len(neighbor_rank) > 0:
        new_val = round(rate_avg[test_set[0]] + sum(map(lambda x: x[0] * (x[1] * x[2]), neighbor_rank)) / sum(map(lambda x: (x[1] * x[2]), neighbor_rank)))
        if new_val < 1:
            return 1
        elif new_val > 5:
            return 5
        else:
            return new_val
    else:
        rank_list = [x[1] for x in data_per_user[test_set[0]]]
        #rank_list = list(map(lambda x: x[1], data_per_user[test_set[0]]))
        return max(set(rank_list), key=rank_list.count)
```

rank 를 예측하는 함수입니다. neighbor 를 확인하고, 이들 중 평가할 아이템에 대해 rank 를 가지고 있는 항목들을 이 neighbor 의 평균에 상대적인 값으로 저장합니다.

그리고 나의 평균과 time stamp, similarity 를 weight 로 더한 평균을 반올림하여 새로 저장합니다. 만약 1 보다 작다면 1 로, 5 보다 크다면 5 로 저장합니다.

하지만 이웃 중 내가 평가하려는 item 의 정보를 가진 이웃이 없다면 내가 가장 많이 준 rank 를 부여합니다.

```
neighbor = get_neighbor(data_set_per_user)
```

이웃 dictionary 를 가져옵니다.

```
# test result print

with open(test_file) as test_:
    test_set = test.read().split()
    test_set = list(map(int, test_set))
    test_set = [test_set[i:i+4] for i in range(0, len(test_set), 4)]
test_set = list(map(lambda x_: [x[0], x[1]], test_set))
```

test file 을 열여 test set 에 data set 과 같은 형태로 넣어줍니다.

```
count = 0
with open(output_file, "w") as prediction_:
    for i in test_set_:
        if count % 100 == 0_:
            print(count)
        count = count+1
        prediction.write("{}\t{}\t{}\n".format(i[0], i[1], str(predict(neighbor, data_set_per_user, rate_average, i))))
```

output file 을 열어 예측한 값을 user, item 와 함께 적어줍니다. 이의 형태는

"{user}₩t{item}₩t{predicted rank}₩n" 입니다.

3. Instruction for Compiling Source Code

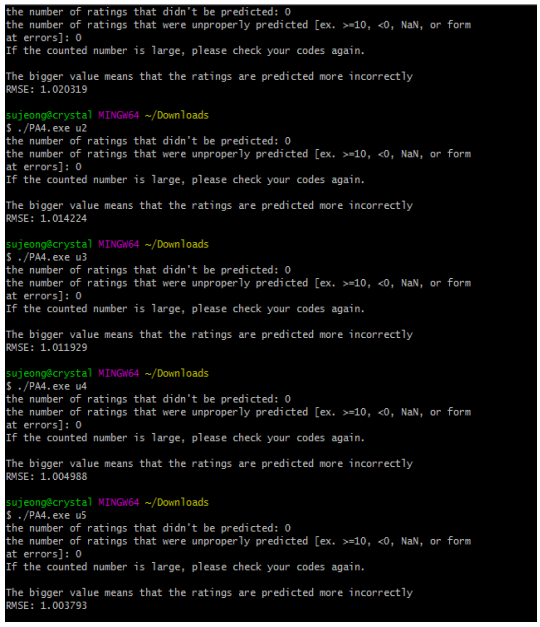
이 코드는 python3 을 기반으로 작성되었습니다. 따라서 python3 가 설치되어 있어야 합니다.

"**python3 recommender.py [base file] [test file]**"의 형태로 complie, 실행하면 됩니다. output file 은 실행한 현재 폴더의 상위 폴더에 생성됩니다.

4. Other Sepcification of Implementation and Testing

우선적으로, 실행 시간이 매우 깁니다. 하지만 실행을 하는 동안은 현재 진행중인 user-user similarity, test set 이 몇 번째 등인지를 계속 일정 간격으로 출력합니다.

아래 그림은 주어진 test file, base file 과 주어진 test program 을 바탕으로 성능을 측정한 결과입니다.



대부분 1 에 근접하는 RMSE 값을 가지는 것을 확인할 수 있습니다.