Data Science [ITE4005]

# Programming Assignment #4

: Predict the ratings in test data using training data

2015004120
Software Engineering
Park Seonha

## 0. Environment

- OS : Windows
- Language : Python3 (version : 3.4.4)
- Executable file : recommender.py
  (../Programming_Assignment_4/project_recommender/recommender.py)

## 1. Summary of Algorithm"

This programming assignment's goal is "Predict the ratings of movies in test data by using the given traning data contaning movie ratings of users" with any algorithm which can use to predict such as content-based algorithm and collaborative filtering algorithms. I used user based nearest neighbor collaborative filtering algorithm to predict the ratings.

Before predict the rating, I calculated all necessary data like similarity and neighbor set because of its hard computation amount.

* precomputed data
- mean of all each users
- list of neighbor of each user
- similarity between all users.

To calculate similarity I used below formula.

$$sim(x,y) = \frac{\sum_{s \in Sxy} (r_{xs} - \bar{r}_s)(r_{ys} - \bar{r}_y)}{\sqrt{\sum_{s \in Sxy} (r_{xs} - \bar{r}_s)^2} \sqrt{\sum (r_{ys} - \bar{r}_y)^2}}$$

After calculate all necessary data, calculate the predict value with user_id and item_id. find the neighbors who have the rating data of certain item_id, and get the similarity between the user_id and each neighbor. To predict the user a's rating of item p, I used this formula.

$$\boldsymbol{pred(a,p) = \overline{r_a} + \frac{\sum_{b \in N} sim(a,b) * (r_{b,p} - \overline{r_b})}{\sum_{b \in N} sim(a,b)}}$$

And check the boundary of rating to fit in result value.

## 2. Description of codes

```
def main():
    # get command line argument
    train_file = sys.argv[1]
    test_file = sys.argv[2]
```

get training file name and test file name from command line argument.

```
    with open(train_file) as f:
        train_data = f.readlines()
    train_data = [d.strip() for d in train_data]
```

```
    # make record_list
    record_list = {}
    user_list = []
    item_list = []
    user_list = set(user_list)
    item_list = set(item_list)
    for line in train_data:
        each_line = line.split("\t")
        temp_key = (int(each_line[0]), int(each_line[1]))
        temp_value = int(each_line[2])
        record_list[temp_key] = temp_value
        user_list.add(int(each_line[0]))
        item_list.add(int(each_line[1]))
```

parse training file into record_list(dictionary whose key is tuple of user and item and whose value is the rating), user_list (set of all user_ids exist in training data), item_list(set of all item_id exist in trainng data)

```
    # make user_item
    user_item = {}
    user_mean = {}
    for user in user_list:
        mean = 0.0
        for item in item_list:
            if(user,item) in record_list:
                if user in user_item.keys():
                    user_item[user].append(item)
                    mean += record_list[(user,item)]
                else:
                    user_item[user] = []
                    user_item[user].append(item)
        mean = float(mean) / float(len(user_item[user]))
        user_mean[user] = mean
```

pre-compute all the mean of each users and make the dictionary whose key is user_id and value is the list of its item at the same time.

```
    # make get_intersect and get common average
    nume = 0
    deno = 0
    get_intersect = {}
    for k1, v1 in user_item.items():
        for k2, v2 in user_item.items():
            if k1 != k2:
                intersected = set(user_item[k1]).intersection((user_item[k2]))
                get_intersect[(k1,k2)] = intersected
                nume += len(intersected)
                deno += 1

    average_common = int(float(nume)/float(deno))
```

```
    # make sim dict
    sim_dict = {}
    for k,v in get_intersect.items():
        if len(v) > average_common:
            sim_dict[k] = get_sim(k[0],k[1],v,record_list)
```

make dictionary whose key is the tuple of two user and value is the set which is

intersection of two user's item. And pre-compute the average of it at same time.

pre-compute all similarities in get_intersect dictionary and store them in sim_dict with same key.

```
    # make neighbor dict
    neighbors = {}
    for user1 in user_list:
        for user2 in user_list:
            if user1 != user2 :
                if len(get_intersect[(user1,user2)]) > average_common:
                    if user1 in neighbors.keys():
                        neighbors[user1].append(user2)
                    else:
                        neighbors[user1] = []
                        neighbors[user1].append(user2)
        if user1 not in neighbors.keys():
            neighbors[user1] = []
```

precompute all user's neighbor(the intersection of both user's item is larger than average common items) and store it into dictionary whose key is user_id and value is the list of neighbor users.

```
def get_sim(user1, user2, intersected, record_list):
    mean_user1 = 0.0
    for item in intersected:
        mean_user1 += record_list[(user1,item)]
    mean_user1 = float(mean_user1)/float(len(intersected))

    mean_user2 = 0.0
    for item in intersected:
        mean_user2 += record_list[(user2,item)]
    mean_user2 = float(mean_user2)/float(len(intersected))

    dist_user1 = 0.0
    dist_list1 = []
    for item in intersected:
        temp = float(record_list[(user1,item)])-mean_user1
        dist_list1.append(temp)
        dist_user1 += math.pow(temp,2.0)
    dist_user1 = math.sqrt(dist_user1)

    dist_user2 = 0.0
    dist_list2 = []
    for item in intersected:
        temp = float(record_list[(user2,item)])-mean_user2
        dist_list2.append(temp)
        dist_user2 += math.pow(temp,2.0)
    dist_user2 = math.sqrt(dist_user2)

    sim_nume = 0.0
    for i,j in zip(dist_list1,dist_list2):
        sim_nume += i*j

    sim_deno = dist_user1 * dist_user2
    if abs(sim_deno) < 0.00001:
        return 1.0


    return sim_nume/sim_deno
```

compute the formula $sim(x,y) = \frac{\sum_{s \in Sxy} (r_{xs} - \bar{r}_s)(r_{ys} - \bar{r}_y)}{\sqrt{\sum_{s \in Sxy} (r_{xs} - \bar{r}_s)^2} \sqrt{\sum (r_{ys} - \bar{r}_y)^2}}$ between two users.

```
#start test
with open(test_file) as f:
    test_data = f.readlines()
test_data = [d.strip() for d in test_data]

test_list = []
for line in test_data:
    each_line = line.split("\t")
    temp_tuple = (int(each_line[0]), int(each_line[1]))
    test_list.append(temp_tuple)
```

open test data file and make the list of tuple (test_user_id, test_item) which to predict.

```
output_file = train_file+"_prediction.txt"
out_f = open(output_file,"w")
for test in test_list:
    out_f.write(str(test[0])+"\t"+str(test[1])+"\t")
    out_f.write(str(predict(test[0],test[1],record_list,user_item,get_intersect,average_common,user_mean,sim_dict,neighbors
    out_f.write("\n")
out_f.close()
```

predict the rating of each test and write in ouput file.

```
def predict(user, item, record_list, user_item, get_intersect,average_common, user_mean,sim_dict,neighbor_dict):
    mean_user = user_mean[user]

    neighbor = []
    for i in neighbor_dict[user]:
        if item in user_item[i]:
            neighbor.append(i)

    sim_list = []
    sim_l_nume = []

    for neigh in neighbor:
        temp = sim_dict[(user,neigh)]
        sim_list.append(temp)
        sim_l_nume.append(temp * (record_list[(neigh,item)] - user_mean[neigh]))

    if len(sim_list) > 0 and sum(sim_list) > 0 :
        result = mean_user + (sum(sim_l_nume)/sum(sim_list))
    else:
        result = mean_user

    if result < 1:
        result = 1
    elif result > 5:
        result = 5
    else:
        result = round(result)

    return result
```

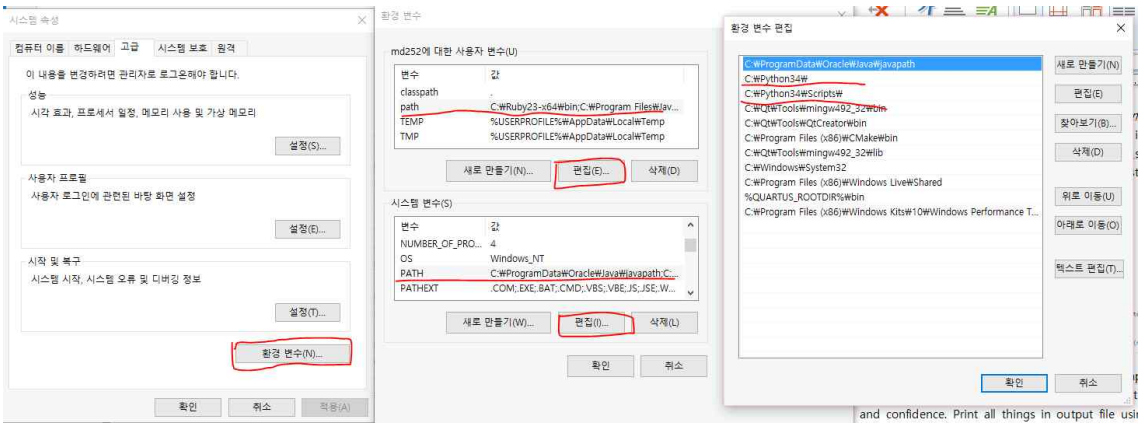
predict the rating of *item* which *user* rated with using pre computed datas and the formula $\boldsymbol{pred(a,p) = \overline{r_a} + \frac{\sum_{b \in N} sim(a,b) * (r_{b,p} - \overline{r_b})}{\sum_{b \in N} sim(a,b)}}$. If predict value is larger than 5, return 5. And if predict value is smaller than 1, return 1. Otherwise, return the round value of predicted result.

## 3. Instructions for compiling this code

This code is written in Python3 and tested in Python 3.4.4. Since this code has **sqrt** and **pow**(to calculate similarity between two user), and it exists in Python3 math module, to run this code, we need python3 interpreter.

We can get python 3.4.4 in (https://www.python.org/downloads/release/python-344/)

If python2 is already installed in PC, change path Python2x to Python34 in advanced system settings – environment variables.



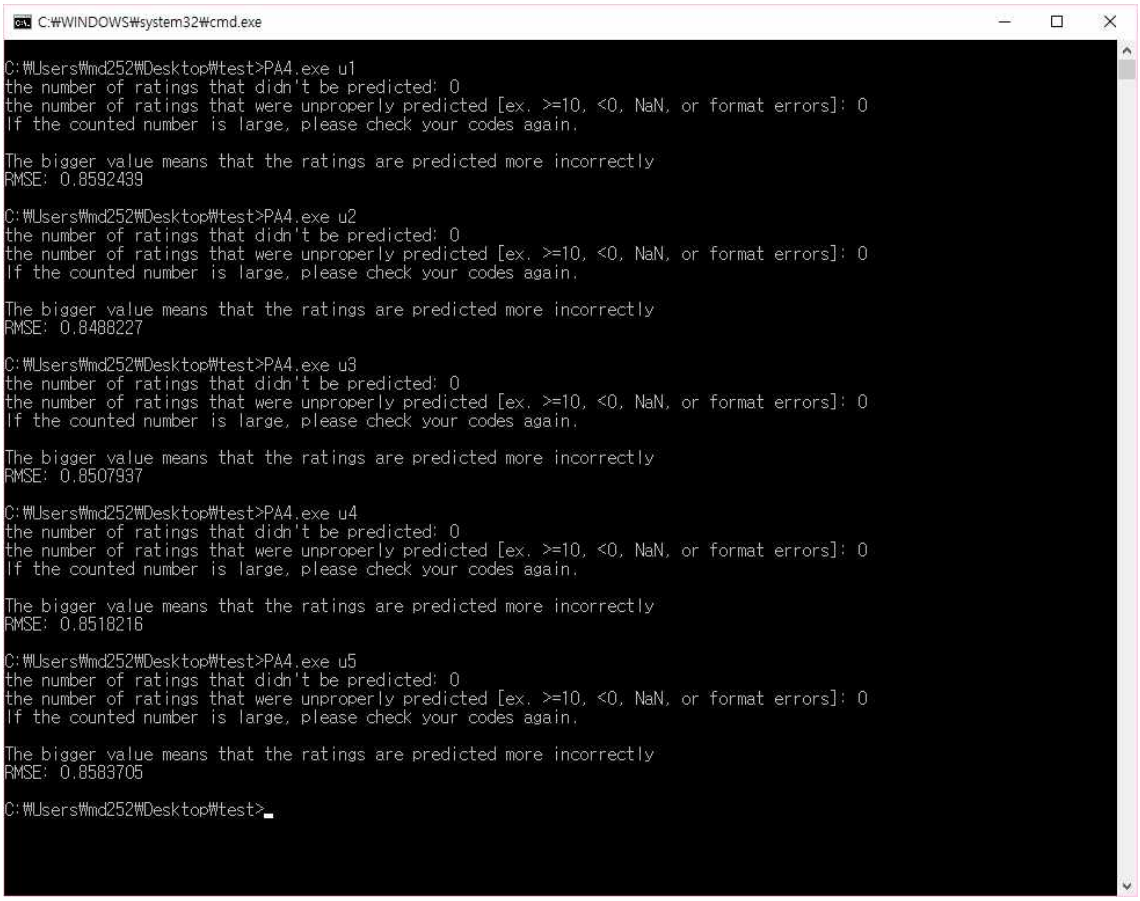


After environment variables setting, we can get python3 default in cmd.

In code directory(python file is in /project_recommender), we can run this program on cmd "python recommender.py [training file name] [test file name]".

The output file will be located in same directory with training file.

## 4. Other Specifications




In testing, some computer can take quite long time to pre-compute similarity befor predict. I only calculate the similarity between each neighbor just one time, it is necessary step and it can't make infinite loop, wait until it will finish.