2014000082 구희강

Data Science

TERM PROJECT – RECOMMENDER SYSTEMS

How to run program

$\gg$ python recommender.py [base file name] [*test file name*]

need numpy library to run

Code Description and Algorithm

On this program I used **Item and User based Collaborative filtering** to predict the ratings of movies in test data.

<main>

```
def main():
    base_file = sys.argv[1]
    test_file = sys.argv[2]
    if base_file.split('.')[0] != test_file.split('.')[0]:
        print("base test file do not match")
        exit()
    name = base_file.split('.')[0]
    base_data = readData(base_file)
    test_data = readData(test_file)
    preprocess(base_data,test_data)
    output_data = prediction(test_data)
    writeData("{}.base_prediction.txt".format(name),output_data)
```

This is the flow of entire code. It requires 2 arguments: base file name, test file name. First, make ‘base_data’ and ‘test_data’ from input file. Then do preprocess. And predict the ratings of movies in test data and write the result.

<readData>

```
def readData(file_name):
    fin = open(file_name,'r')
    data = {} #dict
    while True:
        tmp = fin.readline() #user item rating time
        if not tmp:
            break
        tmp = list(tmp.strip('\n').split('\t'))
        tmp.pop() # delete time
        for i in range(len(tmp)):
            tmp[i] = int(tmp[i])
        if not tmp[0] in data:
            data[tmp[0]] = [tmp[1:]]
        elif tmp[0] in data:
            data[tmp[0]].append(tmp[1:])
    fin.close()
    # user: [item,rating],[item,rating],....
    return data
```

This function reads input text file and makes dict type data whose element is a list of item and rating pairs.

<preprocess>

```
    # make table based on base_data
    base_table = np.zeros((user_size+1,item_size+1),dtype=float)
    for key in base_data.keys():
        for rating in base_data[key]:
            user = key
            item = rating[0]
            rate = rating[1]
            base_table[user][item] = rate
    # average rating of each user
    for i,user in enumerate(base_table):
        if i == 0:
            user_average.append(0)
        else:
            user_average.append(np.sum(user)/np.count_nonzero(user))
    # norm of each item
    for i,item in enumerate(base_table.T):
        if i == 0:
            item_norm.append(0)
        else:
            item_norm.append(np.sqrt(np.dot(item,item)))
    # user 의 평균을 빼서 보정해줌
    table_user = copy.deepcopy(base_table)
    for i in range(1,len(table_user)):
        for index in np.nonzero(table_user[i])[0]:
            table_user[i][index] -= user_average[i]
    # 속도 향상을 위해 cache table
    cache_item = np.full((item_size+1,item_size+1),-1,dtype='float64')
    cache_user = np.full((user_size+1,user_size+1),-1,dtype='float64')
```

base_table : (#user x #item) size of rate array table based on base_table. The zero element means there is no rating info of that (user, item) pair.

user_average : average rating of each user.

Item_norm : L2 norm of each item.

table_user : base_table adjusted by user's average rating.

cache_item, cache_user : cache table of similarity.

<prediction>

```
def prediction(test_data):
    global test_cnt
    cnt = 0
    output_data = []
    for key in test_data.keys():
        for rating in test_data[key]:
            user = key
            item = rating[0]
            predict = 0.5*userBasedPrediction(user,item) +
                          0.5*itemBasedPrediction(user,item)
            if predict == 0:
                predict = 1
            tmp = "{}\t{}\t{}".format(user,item,predict)
            output_data.append(tmp)
            cnt += 1
            if cnt%1000 == 0:
                print("{}/{}".format(cnt,test_cnt))
    return output_data
```

This function finds user based predicted rating value and item based predicted rating value. And the final predicted value is the average of them.

<UserBasedPrediction>

```
def userBasedPrediction(target_user,target_item):
    global base_table, cache_user, user_average
    similarity = {}
    for i in range(1,base_table.shape[0]): #user 전체에 대해서
        if base_table[i][target_item] == 0 or i == target_user:
            continue #본인이거나 rating이 안된 경우 안구함
        if cache_user[target_user][i] != -1:
            similarity[i] = cache_user[target_user][i]
        else:
            similarity[i] = simUser(target_user,i)
    top = 0; bottom = 0; tmp = {}
    for i in range(1,len(base_table)):
        if i == target_user: # 본인이면 패스
            continue
        if i in similarity: # similarity 구한 경우만
            if similarity[i] > 0:
                tmp[i] = similarity[i]
    tmp = sorted(tmp.items(),key=operator.itemgetter(1),reverse=True)
    tmp = tmp[:50]
    for i,s in tmp:
            bottom += s
            top += base_table[i][target_item]*s
    if top==0:
        return user_average[target_user]
    else:
        return top/bottom
```

For all of user in base_table except target user, if the user has rated target_item, get similarity of target user and that user. By similarity value, we can get the target user's neighbor users. The predicted value is weighted(by similarity) average of neighbor's rating value on target_item.

<simUser>

```
# Adjusted cosine similarity
def simUser(i,j):
    global table_user, cache_user
    user_i = table_user[i]
    user_j = table_user[j]
    ret
np.dot(user_i,user_j)/(np.sqrt(np.dot(user_i,user_i))*np.sqrt(np.dot(user_j,user_j
)))
    cache_user[i][j] = ret
    cache_user[j][i] = ret
    return ret
```

This function returns adjusted (by the average of user's rating value) similarity of user i and user j.

$$sim(i,j) = \frac{\sum_{u \in U}(R_{u,i} - \bar{R_u})(R_{u,j} - \bar{R_u})}{\sqrt{\sum_{u \in U}(R_{u,i} - \bar{R_u})^2}\sqrt{\sum_{u \in U}(R_{u,j} - \bar{R_u})^2}}.$$

<ItemBasedPrediction>

```
def itemBasedPrediction(target_user,target_item):
    global base_table, cache_item, user_average
    similarity = {} # target_item 과 의 similarity 전부다 구하기
    for i in range(1,base_table.shape[1]): # item 전체에 대해서
        if i == target_item:
            continue
        if cache_item[target_user][i] != -1:
            similarity[i] = cache_item[target_user][i]
        else:
            similarity[i] = simItem(target_user,i)
    top = 0; bottom = 0; tmp = {}
    for i,rate in enumerate(base_table[target_user]):
        if rate != 0 and similarity[i] > 0:
            tmp[i] = similarity[i]
    tmp = sorted(tmp.items(),key=operator.itemgetter(1),reverse=True)
    for i,s in tmp:
            bottom += s
            top += base_table[target_user][i]*s
    if top==0:
        return user_average[target_user]
    else:
        return top/bottom
```

For all items in base_table except target_item, get similarity of target item and that item. Among the items that the target_user has already rated, we can get the target_item's neighbor items by similarity value (larger than 0). The predicted rating value is weighted (by similarity) average of neighbor items' rating value by the target user.

<simItem>

```
# cosine-based similarity
def simItem(i,j):
    global base_table, item_norm, cache_item
    col_i = base_table[:,i]
    col_j = base_table[:,j]
    norm2_i = item_norm[i]
    norm2_j = item_norm[j]
    if norm2_i == 0  or norm2_j == 0:
        ret = 0
    else:
        ret = np.dot(col_i,col_j)/(norm2_i*norm2_j)
    cache_item[i][j] = ret
    cache_item[j][i] = ret
    return ret
```

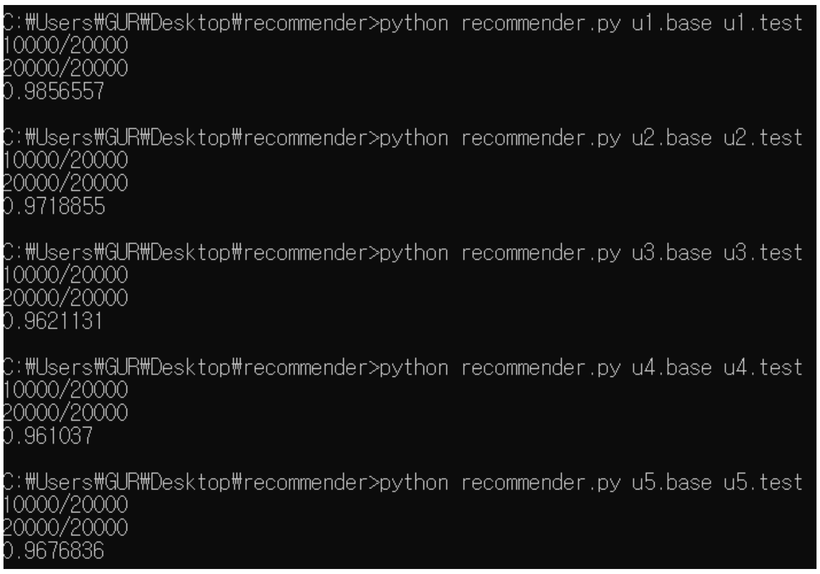
This function returns cosine-based similarity of item i and item j.

$$sim(i,j) = \cos(\vec{i},\vec{j}) = \frac{\vec{i}\cdot\vec{j}}{\|\vec{i}\|_2 * \|\vec{j}\|_2}$$

Results

Test environment - Windows 10

Test requirement – **Python 3** (with **numpy** library)



```
C:₩Users₩GUR₩Desktop₩recommender>python recommender.py u1.base u1.test
10000/20000
20000/20000
0.9856557

C:₩Users₩GUR₩Desktop₩recommender>python recommender.py u2.base u2.test
10000/20000
20000/20000
0.9718855

C:₩Users₩GUR₩Desktop₩recommender>python recommender.py u3.base u3.test
10000/20000
20000/20000
0.9621131

C:₩Users₩GUR₩Desktop₩recommender>python recommender.py u4.base u4.test
10000/20000
20000/20000
0.961037

C:₩Users₩GUR₩Desktop₩recommender>python recommender.py u5.base u5.test
10000/20000
20000/20000
0.9676836
```