

# Data Science

## **Programming Assignment 4**

recommender :Predict the ratings of movies

## Design Document

**2015004011**

**major in Software**

**Dohyun Kim**

## 1. Summary of my algorithm

This assignment is to implement the 'Recommender system' that predict the rating of movies in test data by using the given training data containing movie rating of users.

I used concept of Collaborative Filtering .This concept follows the method below.

First, Finding a group of users(neighbors) whose preferences are similar to that of an active user  $c$ . I used Cosine similarity.

$$w(a, i) = \sum_j \frac{v_{a,j}}{\sqrt{\sum_{k \in I_a} v_{a,k}^2}} \frac{v_{i,j}}{\sqrt{\sum_{k \in I_i} v_{i,k}^2}}$$

Second, Estimating  $r(c,s)$ , the rating of item  $s$  for active user  $c$ , based on the ratings given to item  $s$  by  $c$ 's neighbors.I used this formula during different methods for aggregation.

$$(c) \ r_{c,s} = \bar{r}_c + k \sum_{c' \in C} sim(c, c') \times (r_{c',s} - \bar{r}_{c'}),$$

Finally,

Recommending a few items with the ratings estimated high

I will explain the implementation details through the code.

## 2. Detailed description of my codes

### 1. main

```
#main
if __name__ == '__main__':
    train_f_name = sys.argv[1]
    test_f_name = sys.argv[2]

    global train_dic
    global test_dic
    train_dic = pre_treatment_train(train_f_name)
    test_dic = pre_treatment_test(test_f_name)
    user_rec(test_dic, train_f_name)
```

The main function is as clean as possible. There are functions that is preprocessing and predicting score.

### 2. pre\_treatment\_train

```
#Training file preprocessing
def pre_treatment_train(input_f_name):
    train_dic = {}
    f = open(input_f_name)

    def make_clean(str):
        str = str.strip()
        return str

    train_line = [map(make_clean, line.split('\t')) for line in f]

    for i in range(len(train_line)):
        user_id = int(train_line[i][0])
        item_id = int(train_line[i][1])
        rating = int(train_line[i][2])
        train_dic.setdefault(user_id, {})
        train_dic[user_id][item_id] = rating

    f.close()
    return train_dic
```

This function is a function that takes a training data set and organizes it in dictionary form. The format is {user\_id : {item\_id:rating , item\_id:rating ...}}

### 3.pre\_treatment\_test

```
#Test_file_preprocessing
def pre_treatment_test(input_f_name):
    test_dic = {}
    error_list = []
    f = open(input_f_name)

    def make_clean(str):
        str = str.strip()
        return str

    test_line = [map(make_clean, line.split('\t')) for line in f]

    for i in range(len(test_line)):
        user_id = int(test_line[i][0])
        item_id = int(test_line[i][1])
        test_dic.setdefault(user_id, {})
        test_dic[user_id].setdefault(item_id, 0)

    f.close()
    return test_dic
```

This function is a function that takes a test data set and organizes it in dictionary form. The format is {user\_id : {item\_id:0 , item\_id:0...}} Test data sets do not receive ratings, so this function is made different from the training pre-processing functions.

### 4. user\_simil

```
#A function to obtain cosine-similarity for two users
def user_simil(user_id_1, user_id_2):

    # To get both rated items
    both_rated = {}
    for item in train_dic[user_id_1]:
        if item in train_dic[user_id_2]:
            both_rated[item] = 1

    number_of_ratings = len(both_rated)

    # Checking for ratings in common
    if number_of_ratings == 0:
        return 0

    # Sum up the squares of rated value of each user
    user_1_square_sum = sum([(train_dic[user_id_1][item])**2 for item in train_dic[user_id_1]])
    user_2_square_sum = sum([(train_dic[user_id_2][item])**2 for item in train_dic[user_id_2]])

    # Sum up the product value of both rated value for each item
    both_sum = sum([train_dic[user_id_1][item] * train_dic[user_id_2][item] for item in both_rated])

    # Calculate the cosine-similarity
    numerator_value = both_sum
    denominator_value = math.sqrt(user_1_square_sum) * math.sqrt(user_2_square_sum)

    if denominator_value == 0:
        return 0
    else:
        r = numerator_value / denominator_value
        return r
```

This function is neighborhood-based algorithm that calculates the similarity between two users. I used cosine-similarity value. The formula for cosine-similarity is attached above.

## 5. most\_similar\_users

```
#Finds 30 other users who are most similar to user
def most_similar_users(user_id, item ,number_of_users):

    # returns the number_of_users (similar persons) for a given specific person
    most_simil_list = [(user_simil(user_id, other_user), other_user) for other_user in train_dic \
                        if other_user != user_id and item in train_dic[other_user]]

    # Sort the similar persons so the highest scores person will appear at the first
    most_simil_list.sort()
    most_simil_list.reverse()
    return most_simil_list[0:number_of_users]
```

This function returns a certain number of people in highest value of similarity value. This list consists of a tuple containing the similarity value and the user ID of the other user like (similarity\_value(user\_1, user\_2), user\_2\_id). Sort the highest value to be forward and return the wanted number of names. This number is determined by calling in the 'user\_rec' function, and I set it to 30.

## 6. get\_average

```
#The average of all the scores the user has
def get_average(user):
    user_data = train_dic[user].items()
    length = len(user_data)
    sum = 0
    for item, rating in train_dic[user].items():
        sum += rating
    return float(sum) / float(length)
```

This function is a function that averages all the scores rated by the user as arguments. This one is used by calculating formula as an aggregation of some similar users' rating of the item.

## 7. user\_rec

```
#Expect a score.
def user_rec(test_dic, train_f_name):

    filename = train_f_name + "_prediction.txt"
    output_file = open(filename, 'w')

    for test_user in test_dic:
        test_user_average = get_average(test_user)
        for unknown_item in test_dic[test_user]:
            most_simil_list = most_similar_users(test_user, unknown_item, 30)
            sum_simil = 0
            sum_simil_rank = 0
            for other_user in most_simil_list:
                if unknown_item in train_dic[other_user[1]]:
                    sum_simil += abs(other_user[0])
                    sum_simil_rank += other_user[0]*\
                        (train_dic[other_user[1]][unknown_item] -\
                         get_average(other_user[1]))

            if sum_simil == 0:
                r = test_user_average
                if r<1:
                    r = 1
            else:
                r = test_user_average + (sum_simil_rank / sum_simil)
                if r<1:
                    r=1

            result_str = str(test_user)+"\t"+str(unknown_item)+"\t"+str(r)+"\n"
            output_file.write(result_str)

    output_file.close()
```

This function predicts the score as a core function of this program.

$$r_{u,i} = \bar{r}_u + k \sum_{u' \in U} \text{simil}(u, u') (r_{u',i} - \bar{r}_{u'})$$

where k is a normalizing factor defined as  $k = 1 / \sum_{u' \in U} |\text{simil}(u, u')|$ , and  $\bar{r}_u$  is the average rating of user u for all the items rated by u.

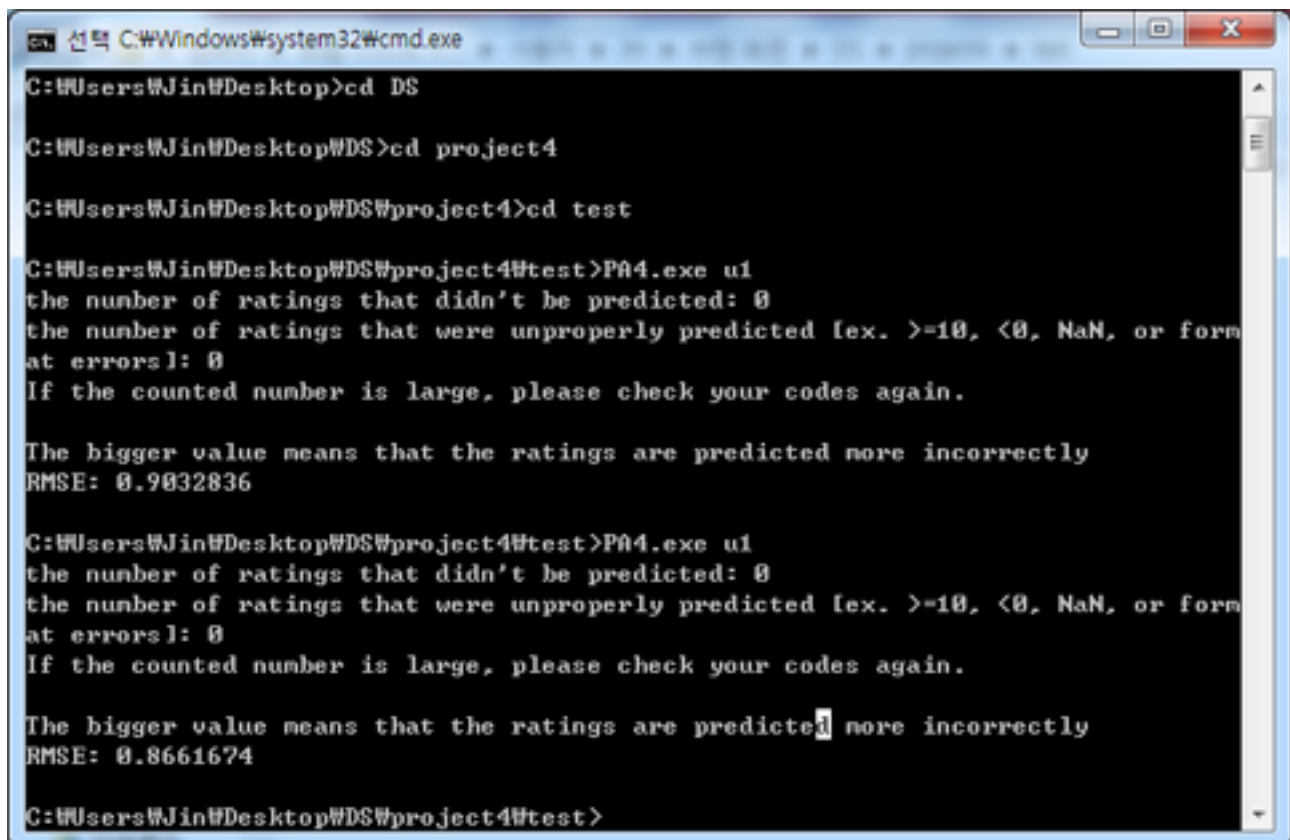
I used this formula, Because the error rate is the lowest. If the sum of the value of similarity is zero, it is arbitrarily determined as the average value of the scores already evaluated by the user. And if the predicted value is less than 1, there is no score value less than 1, so I changed it to 1.

### 3. Detailed description of my codes

My code was written in Python so when you run it you need to input first 'python' like this "python recommender.py u1.base u1.test"

Enter python first, followed by the training data and test data

### 4. result



```
선택 C:\Windows\system32\cmd.exe
C:\Users\Jin\Desktop>cd DS
C:\Users\Jin\Desktop\DS>cd project4
C:\Users\Jin\Desktop\DS\project4>cd test
C:\Users\Jin\Desktop\DS\project4\test>PA4.exe u1
the number of ratings that didn't be predicted: 0
the number of ratings that were improperly predicted [ex. >=10, <0, NaN, or form
at errors]: 0
If the counted number is large, please check your codes again.

The bigger value means that the ratings are predicted more incorrectly
RMSE: 0.9032836

C:\Users\Jin\Desktop\DS\project4\test>PA4.exe u1
the number of ratings that didn't be predicted: 0
the number of ratings that were improperly predicted [ex. >=10, <0, NaN, or form
at errors]: 0
If the counted number is large, please check your codes again.

The bigger value means that the ratings are predicted more incorrectly
RMSE: 0.8661674

C:\Users\Jin\Desktop\DS\project4\test>
```