< Data Science (ITE4005) >

Term Project – recommender system

소프트웨어 전공

2013010926 강은석

1. Introduction

: This is an assignment using Collaborative filtering of matrix factorization.

Collaborative filtering method is explained that find similar other object and use object's information. And Matrix Factorization is explained that we have a matrix that have missing data. so by using gradient descent, fill missing data.

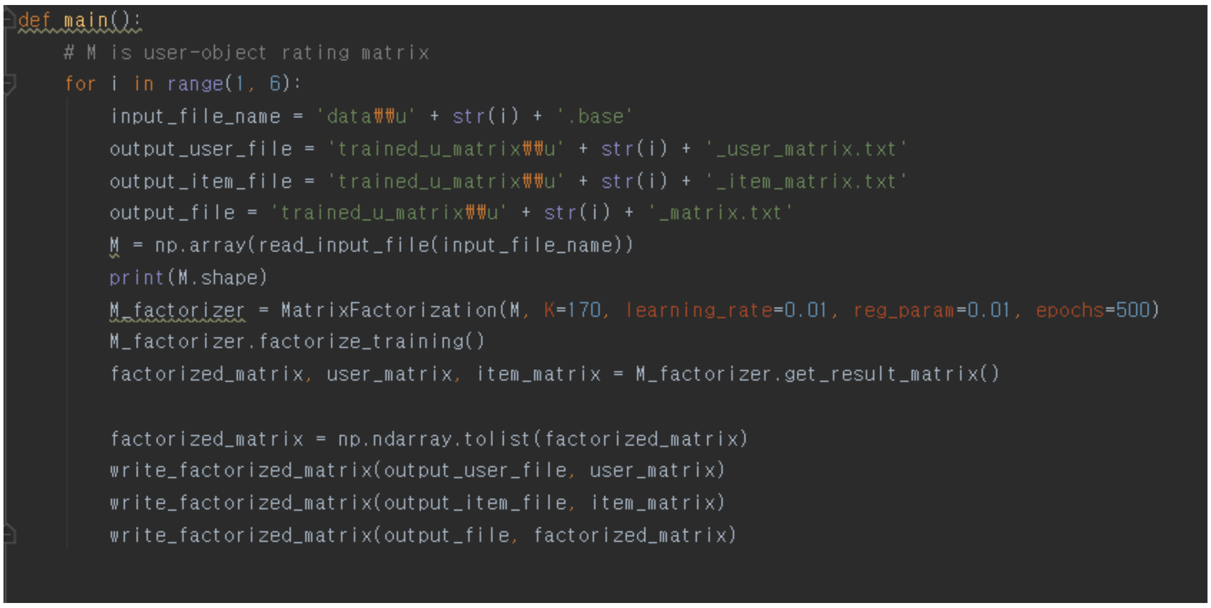
2. Summary of My algorithm

: In this case, I tried collaborative filtering, matrix factorization and use both.

When I use collaborative filtering to get similarity, I tried 'pearson correlation', 'cosine similarity', 'euclidean distance'. The result of each algorithm that calculated by PA4.exe is almost 1.12-1.14. so I tried matrix factorization. Matrix factorization is an algorithm that find NxK and KxM matrix that are well explain NxM matrix by gradient descent. The result of MF algorithm is almost 1.06. Finally, I tried both algorithm. First make matrix by matrix factorization and calculate similarity. But I can't get better result. So I conclude to using matrix factorization method.

3-1. Description of codes (matrix_fatorization.py)

: fill missing data by using matrix factorization code. I using python ‘Numpy’ library.

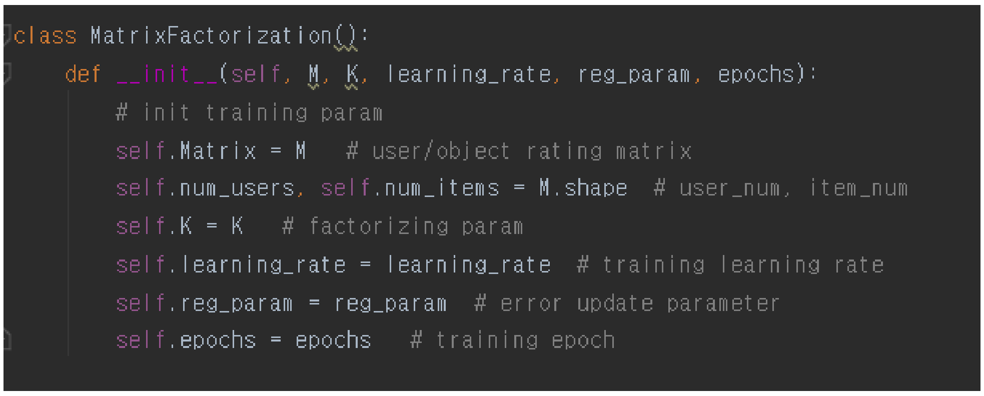


```
def main():
    # M is user-object rating matrix
    for i in range(1, 6):
        input_file_name = 'data₩₩u' + str(i) + '.base'
        output_user_file = 'trained_u_matrix₩₩u' + str(i) + '_user_matrix.txt'
        output_item_file = 'trained_u_matrix₩₩u' + str(i) + '_item_matrix.txt'
        output_file = 'trained_u_matrix₩₩u' + str(i) + '_matrix.txt'
        M = np.array(read_input_file(input_file_name))
        print(M.shape)
        M_factorizer = MatrixFactorization(M, K=170, learning_rate=0.01, reg_param=0.01, epochs=500)
        M_factorizer.factorize_training()
        factorized_matrix, user_matrix, item_matrix = M_factorizer.get_result_matrix()

        factorized_matrix = np.ndarray.tolist(factorized_matrix)
        write_factorized_matrix(output_user_file, user_matrix)
        write_factorized_matrix(output_item_file, item_matrix)
        write_factorized_matrix(output_file, factorized_matrix)
```

그림 1 - main process

: I decide hyper parameter K = 170, learing rate = 0.01, reg_param = 0.01, epochs = 500. K is dememsion value that split NxM matrix to NxK and KxM matrix. Learning rate is value that how much reflect the gradient value. Reg_param is value that regularize predicted missing data.

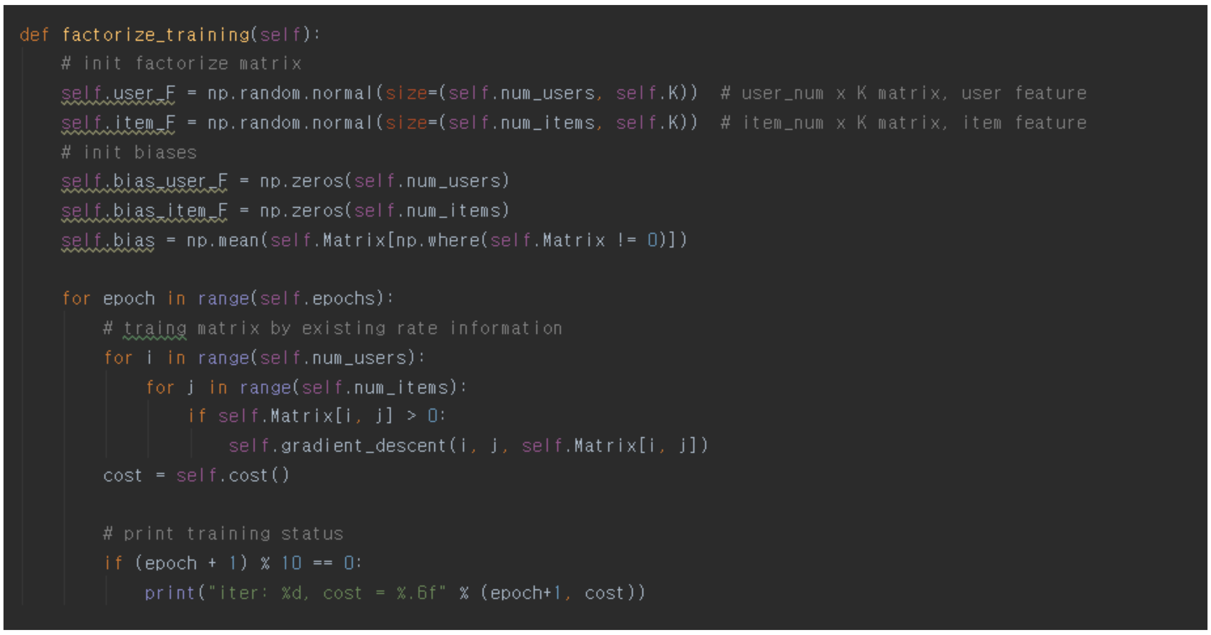
Output file is saved ‘traind_u_matrix’ directory. File name is ‘u#_matrix.txt’



```
class MatrixFactorization():
    def __init__(self, M, K, learning_rate, reg_param, epochs):
        # init training param
        self.Matrix = M   # user/object rating matrix
        self.num_users, self.num_items = M.shape  # user_num, item_num
        self.K = K   # factorizing param
        self.learning_rate = learning_rate  # training learning rate
        self.reg_param = reg_param  # error update parameter
        self.epochs = epochs   # training epoch
```

그림 2 - class initialize parameter

: initialize class parameter by values that I decided. I explained 그림-1 that Hyper parameter values.

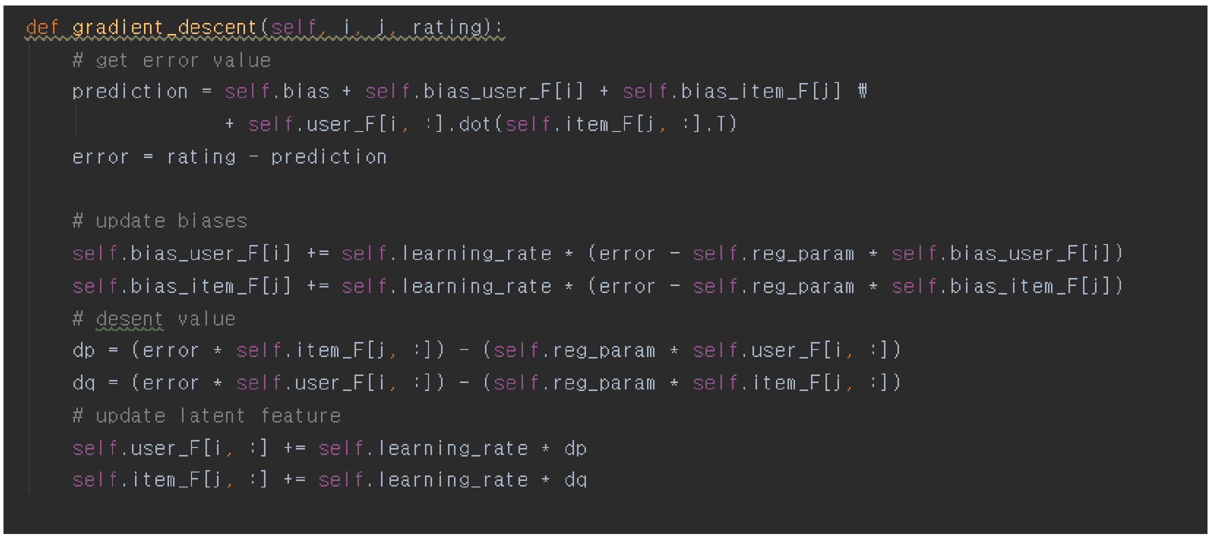


```python
def factorize_training(self):
    # init factorize matrix
    self.user_F = np.random.normal(size=(self.num_users, self.K))  # user_num x K matrix, user feature
    self.item_F = np.random.normal(size=(self.num_items, self.K))  # item_num x K matrix, item feature
    # init biases
    self.bias_user_F = np.zeros(self.num_users)
    self.bias_item_F = np.zeros(self.num_items)
    self.bias = np.mean(self.Matrix[np.where(self.Matrix != 0)])

    for epoch in range(self.epochs):
        # traing matrix by existing rate information
        for i in range(self.num_users):
            for j in range(self.num_items):
                if self.Matrix[i, j] > 0:
                    self.gradient_descent(i, j, self.Matrix[i, j])
        cost = self.cost()

        # print training status
        if (epoch + 1) % 10 == 0:
            print("iter: %d, cost = %.6f" % (epoch+1, cost))
```

그림 3 - training function

: initialize user feature(N x K), item feature(K x M), and bias matrix.

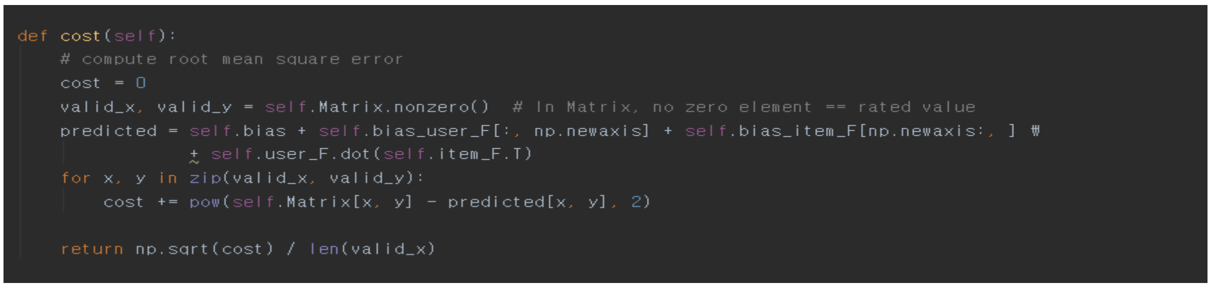
Each epoch, for existed data, calculate gradient decent value and update Matrix value.



```python
def gradient_descent(self, i, j, rating):
    # get error value
    prediction = self.bias + self.bias_user_F[i] + self.bias_item_F[j] ₩
                 + self.user_F[i, :].dot(self.item_F[j, :].T)
    error = rating - prediction

    # update biases
    self.bias_user_F[i] += self.learning_rate * (error - self.reg_param * self.bias_user_F[i])
    self.bias_item_F[j] += self.learning_rate * (error - self.reg_param * self.bias_item_F[j])
    # desent value
    dp = (error * self.item_F[j, :]) - (self.reg_param * self.user_F[i, :])
    dq = (error * self.user_F[i, :]) - (self.reg_param * self.item_F[j, :])
    # update latent feature
    self.user_F[i, :] += self.learning_rate * dp
    self.item_F[j, :] += self.learning_rate * dq
```

그림 4 - gradient decent function

: calculate gradient decent value and update user feature matrix, and item feature matrix.

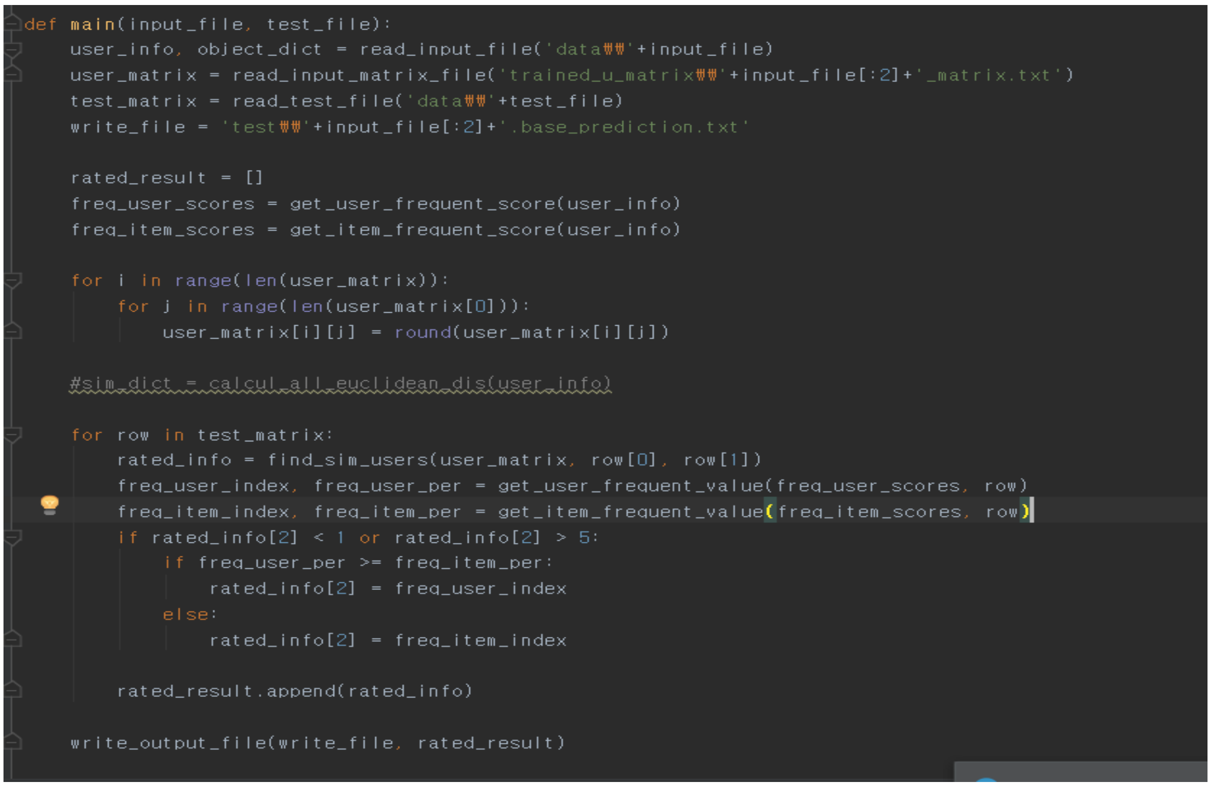


```
def cost(self):
    # compute root mean square error
    cost = 0
    valid_x, valid_y = self.Matrix.nonzero()  # In Matrix, no zero element == rated value
    predicted = self.bias + self.bias_user_F[:, np.newaxis] + self.bias_item_F[np.newaxis:, ] ₩
                + self.user_F.dot(self.item_F.T)
    for x, y in zip(valid_x, valid_y):
        cost += pow(self.Matrix[x, y] - predicted[x, y], 2)

    return np.sqrt(cost) / len(valid_x)
```

그림 5 - cost function

: calculate cost value that explain how much difference to original matrix and predicted matrix.

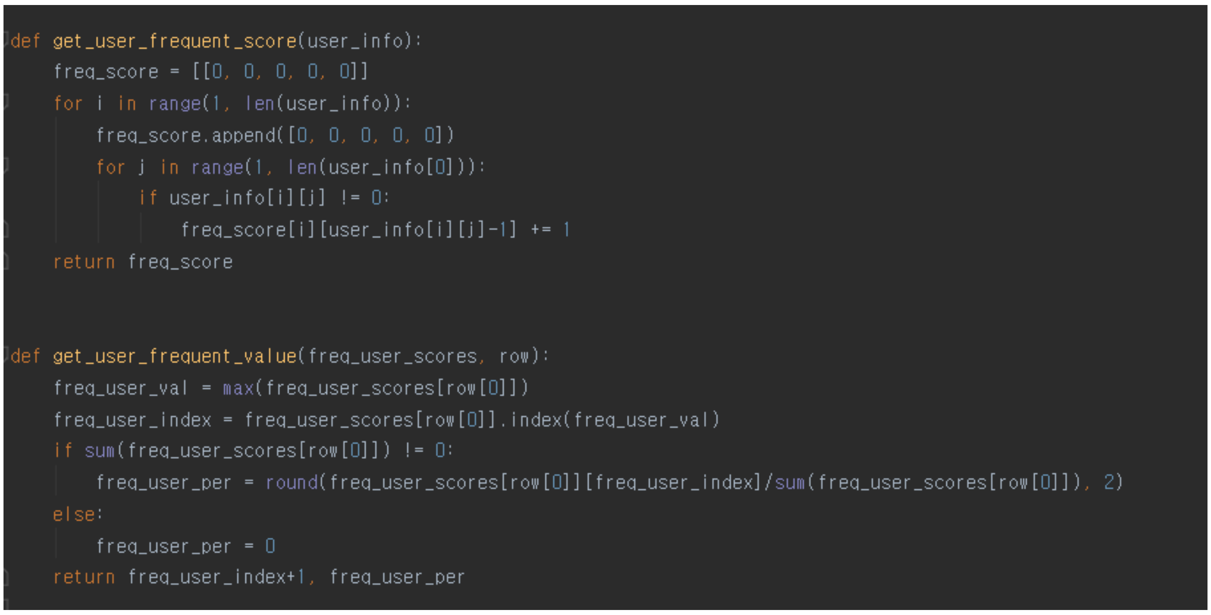
3-2. Description of codes (recommender.py)



```
def main(input_file, test_file):
    user_info, object_dict = read_input_file('data₩₩'+input_file)
    user_matrix = read_input_matrix_file('trained_u_matrix₩₩'+input_file[:2]+'_matrix.txt')
    test_matrix = read_test_file('data₩₩'+test_file)
    write_file = 'test₩₩'+input_file[:2]+'.base_prediction.txt'

    rated_result = []
    freq_user_scores = get_user_frequent_score(user_info)
    freq_item_scores = get_item_frequent_score(user_info)

    for i in range(len(user_matrix)):
        for j in range(len(user_matrix[0])):
            user_matrix[i][j] = round(user_matrix[i][j])

    #sim_dict = calcul_all_euclidean_dis(user_info)

    for row in test_matrix:
        rated_info = find_sim_users(user_matrix, row[0], row[1])
        freq_user_index, freq_user_per = get_user_frequent_value(freq_user_scores, row)
        freq_item_index, freq_item_per = get_item_frequent_value(freq_item_scores, row)
        if rated_info[2] < 1 or rated_info[2] > 5:
            if freq_user_per >= freq_item_per:
                rated_info[2] = freq_user_index
            else:
                rated_info[2] = freq_item_index

        rated_result.append(rated_info)

    write_output_file(write_file, rated_result)
```

그림 6 - main function

: first read file 'u#.base', 'u#.test', 'trained_u_matrix₩₩u#_matrix.txt'(matrix factorized matrix that made from matrix_factorization.py). calculate user base frequent scores, item base frequent scores. For each test file row, get rate from factorized matrix. if rate score is out of range, chose higher percentage score between user base frequent score, item base frequent score.



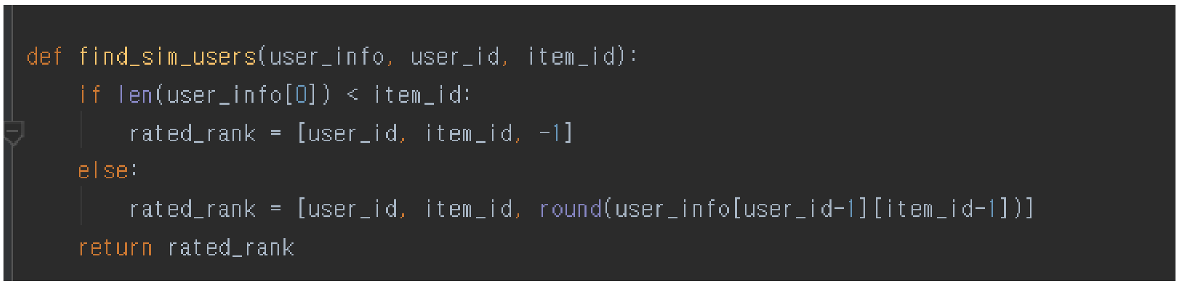
```python
def get_user_frequent_score(user_info):
    freq_score = [[0, 0, 0, 0, 0]]
    for i in range(1, len(user_info)):
        freq_score.append([0, 0, 0, 0, 0])
        for j in range(1, len(user_info[0])):
            if user_info[i][j] != 0:
                freq_score[i][user_info[i][j]-1] += 1
    return freq_score


def get_user_frequent_value(freq_user_scores, row):
    freq_user_val = max(freq_user_scores[row[0]])
    freq_user_index = freq_user_scores[row[0]].index(freq_user_val)
    if sum(freq_user_scores[row[0]]) != 0:
        freq_user_per = round(freq_user_scores[row[0]][freq_user_index]/sum(freq_user_scores[row[0]]), 2)
    else:
        freq_user_per = 0
    return freq_user_index+1, freq_user_per
```

그림 7 user base frequent score (same as item base frequent score)

: calculate user base frequent scores. Get the most frequent rate value User#

And calculate the most frequent rate value's ratio.

For example User#'s the most frequent score is 4 and calculate rate 4's ratio



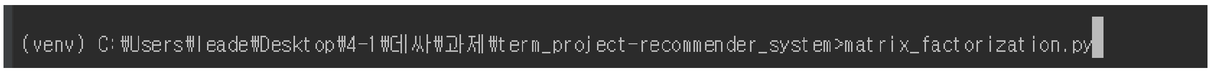
```python
def find_sim_users(user_info, user_id, item_id):
    if len(user_info[0]) < item_id:
        rated_rank = [user_id, item_id, -1]
    else:
        rated_rank = [user_id, item_id, round(user_info[user_id-1][item_id-1])]
    return rated_rank
```

그림 8 - get rate value

: get rate value from factorized matrix.

because factorized matrix element is not integer, use round method.

4-1. How to compile code (matrix_factorization.py)



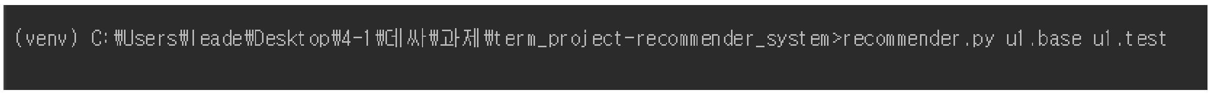
```
(venv) C:₩Users₩leade₩Desktop₩4-1₩데사₩과제₩term_project-recommender_system>matrix_factorization.py
```

그림 9 - run matrix_factorization.py

: [python matrix_factorization.py]

Before run matrix_factorization.py, you must install 'Numpy' library.

By [pip install numpy]



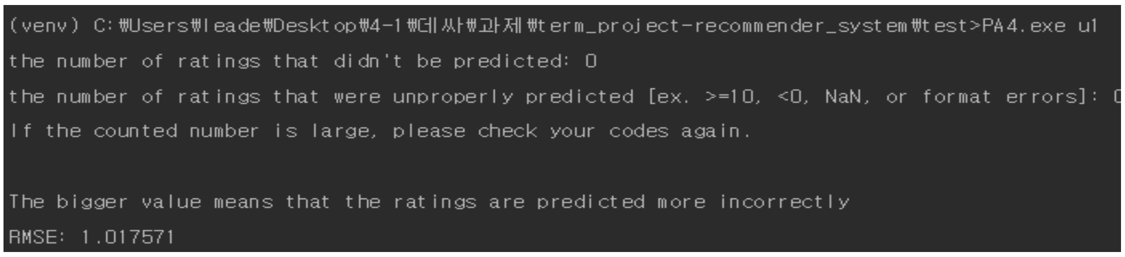
```
(venv) C:₩Users₩leade₩Desktop₩4-1₩데사₩과제₩term_project-recommender_system>recommender.py u1.base u1.test
```

그림 10 - run recommender.py

[python recommender.py u#.base u#.test]

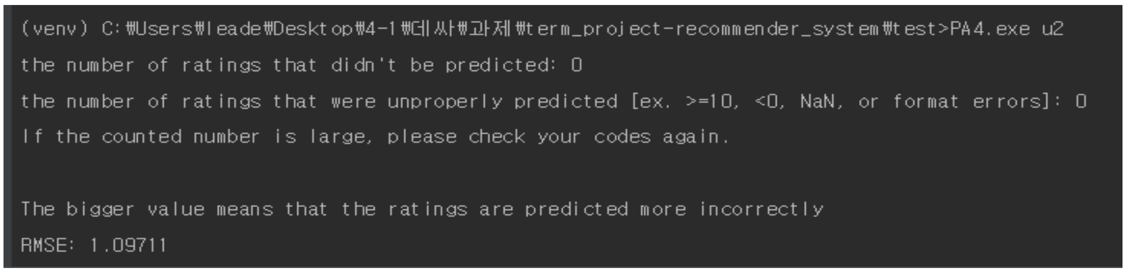
u#.base and u# test file is in data directory. and output file (u#.base_prediction.txt) is saved in test directory.

5. Result of test



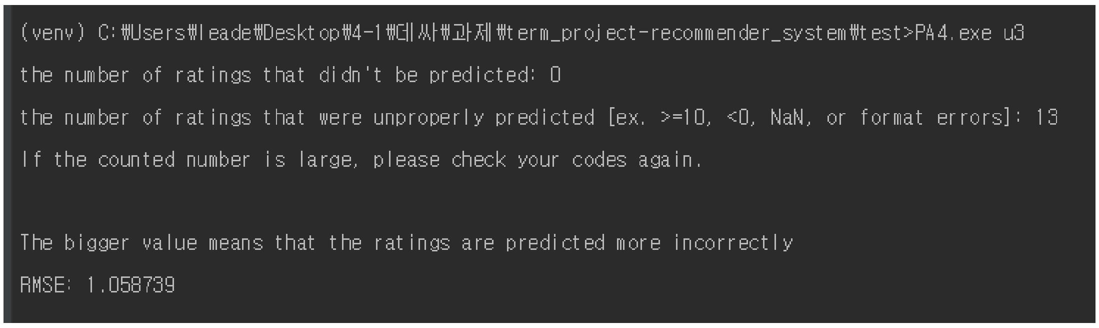
```
(venv) C:₩Users₩leade₩Desktop₩4-1₩데사₩과제₩term_project-recommender_system₩test>PA4.exe u1
the number of ratings that didn't be predicted: 0
the number of ratings that were unproperly predicted [ex. >=10, <0, NaN, or format errors]:
If the counted number is large, please check your codes again.

The bigger value means that the ratings are predicted more incorrectly
RMSE: 1.017571
```

그림 11 RMSE result u1



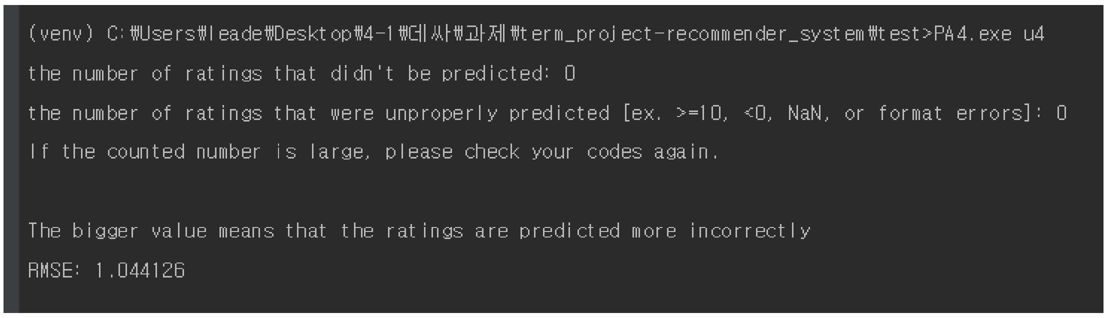
```
(venv) C:₩Users₩leade₩Desktop₩4-1₩데사₩과제₩term_project-recommender_system₩test>PA4.exe u2
the number of ratings that didn't be predicted: 0
the number of ratings that were unproperly predicted [ex. >=10, <0, NaN, or format errors]: 0
If the counted number is large, please check your codes again.

The bigger value means that the ratings are predicted more incorrectly
RMSE: 1.09711
```

그림 12 RMSE result u2



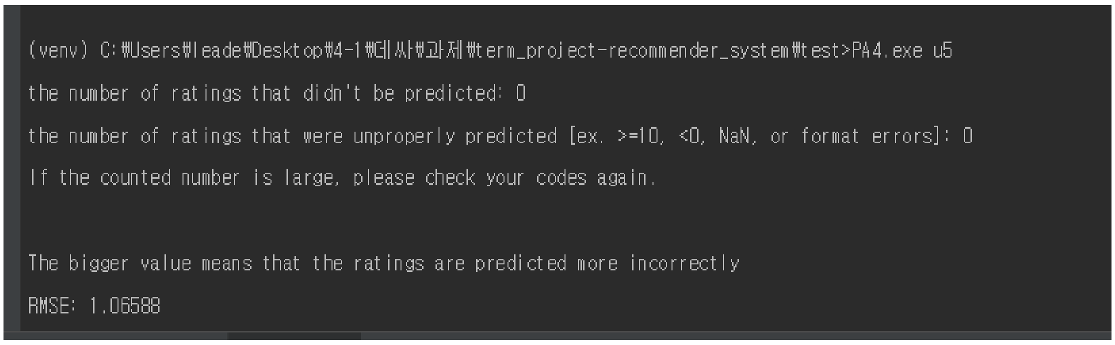
```
(venv) C:₩Users₩leade₩Desktop₩4-1₩데사₩과제₩term_project-recommender_system₩test>PA4.exe u3
the number of ratings that didn't be predicted: 0
the number of ratings that were unproperly predicted [ex. >=10, <0, NaN, or format errors]: 13
If the counted number is large, please check your codes again.

The bigger value means that the ratings are predicted more incorrectly
RMSE: 1.058739
```

그림 13 RMSE result u3



```
(venv) C:₩Users₩leade₩Desktop₩4-1₩데사₩과제₩term_project-recommender_system₩test>PA4.exe u4
the number of ratings that didn't be predicted: 0
the number of ratings that were unproperly predicted [ex. >=10, <0, NaN, or format 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: 1.044126
```

그림 14 RMSE result u4



```
(venv) C:₩Users₩leade₩Desktop₩4-1₩데사₩과제₩term_project-recommender_system₩test>PA4.exe u5
the number of ratings that didn't be predicted: 0
the number of ratings that were unproperly predicted [ex. >=10, <0, NaN, or format 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: 1.06588
```

그림 15 RMSE result u5