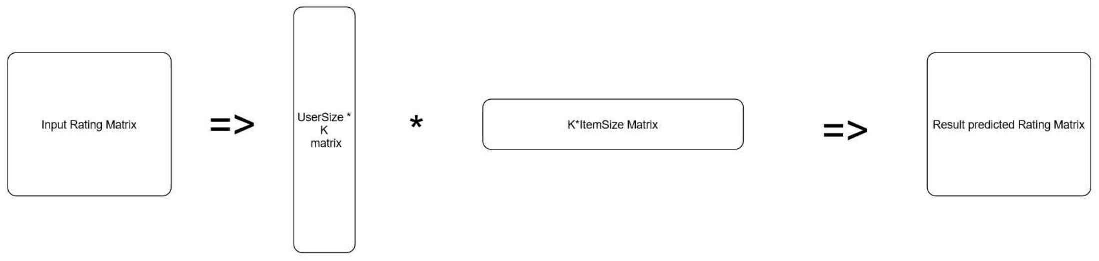
Assignment_Recommender System

2014004066 서왕규

<development enviroment>
Window 10
Visual studio19
C++
<dataset workload>
Dataset's id is not always sequential and id of item is same, too.
Rating is in range(1~5)

1. Algorithm

(1-1) Matrix factorization



① Make two matrix U, V^T
R(original matrix) = U * V^T
In This phase, how we can get matrix, U, V^T?
There is many method, This program chooses gradien descent.
② After make two matrix, Make result matrix by multipling them

(1-2) Gradient descent
① Loss function
Like Newton approximation, that is find a appropriate variable that make result minimum.
First, we make loss function that present error between input rating matrix and predicted rating matrix by matrix factorization.

$$Lossfunction = \sum_{i,j}^{size}(Original(i,j) - \mathrm{P}redict(i,j))^2 + b/2(\sum_{1}^{k}P(i,k)^2 + Q(,kj)^2)$$

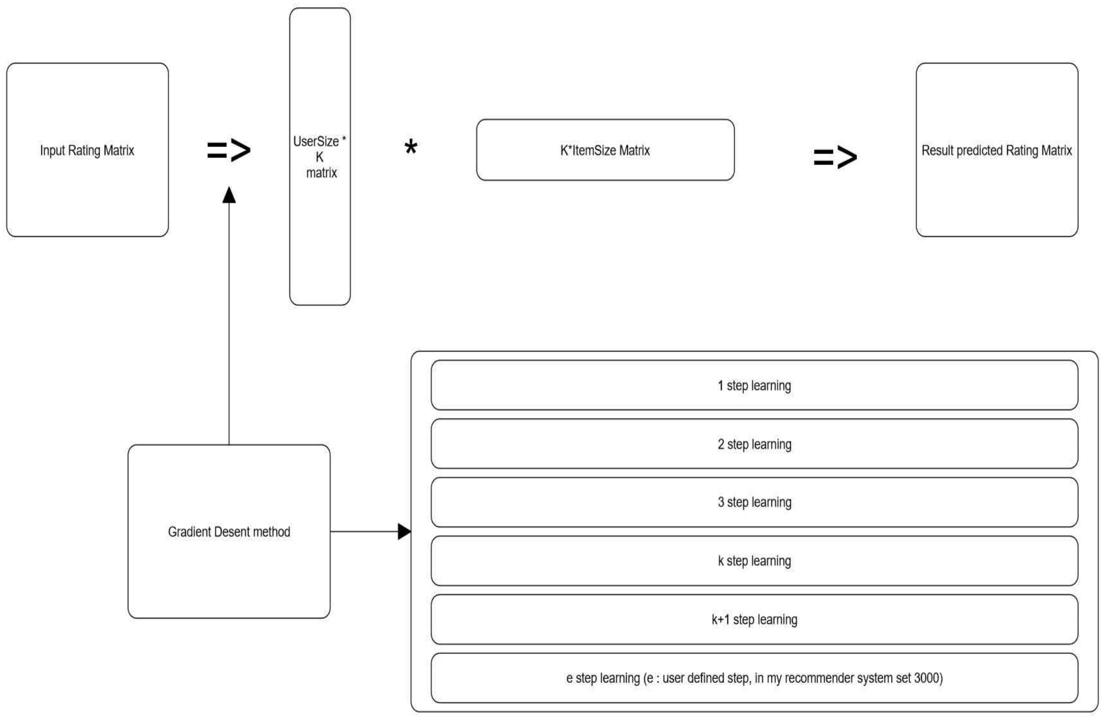
k : user defined parameter that means dimension of factorized matrix.
In my project, k is 3.
b : user defined parameter that composed to step size
P,Q : fatorized matrix
From this loss function, we get error between original matrix, and result of matrix factorization. If loss function is minimum, It means there is a minimum point.
② Find minimum point by using loss function

$X_{k+1} = X_k - \lambda f'(X_k)$

lambda is user defined parameter that means step size

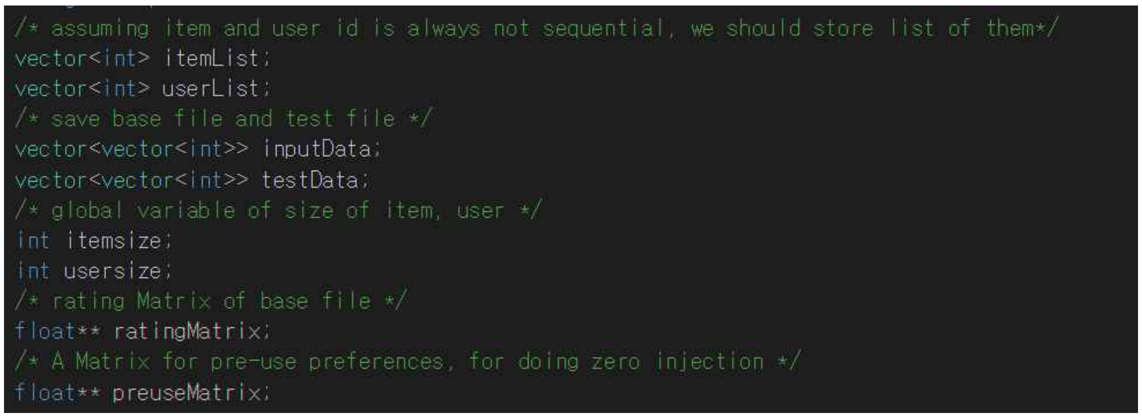
In k+1 step, if tangent in k step point is negative, we choose moving + direction, if tangent in k step point is positive, we choose moving – direction for finding minimum point of loss function.

Repeat, this step, we can find a appropriate approximation fatorized matrixs. And multiply that two matrix, we can get a predicted matrix.



2. detailed description of my codes

① global variable

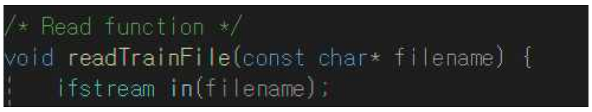


```
/* assuming item and user id is always not sequential, we should store list of them*/
vector<int> itemList;
vector<int> userList;
/* save base file and test file */
vector<vector<int>> inputData;
vector<vector<int>> testData;
/* global variable of size of item, user */
int itemsize;
int usersize;
/* rating Matrix of base file */
float** ratingMatrix;
/* A Matrix for pre-use preferences, for doing zero injection */
float** preuseMatrix;
```

- itemList and userList : save information of dataset.
- inputData, testData : save base file and test file

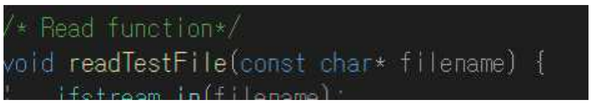
- ratingMatrix : save rating to matrix from input data

② read and write file



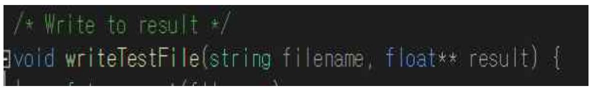
```
/* Read function */
void readTrainFile(const char* filename) {
    ifstream in(filename);
```

- read base file to program. filename is in parameter



```
/* Read function*/
void readTestFile(const char* filename) {
    ifstream in(filename);
```

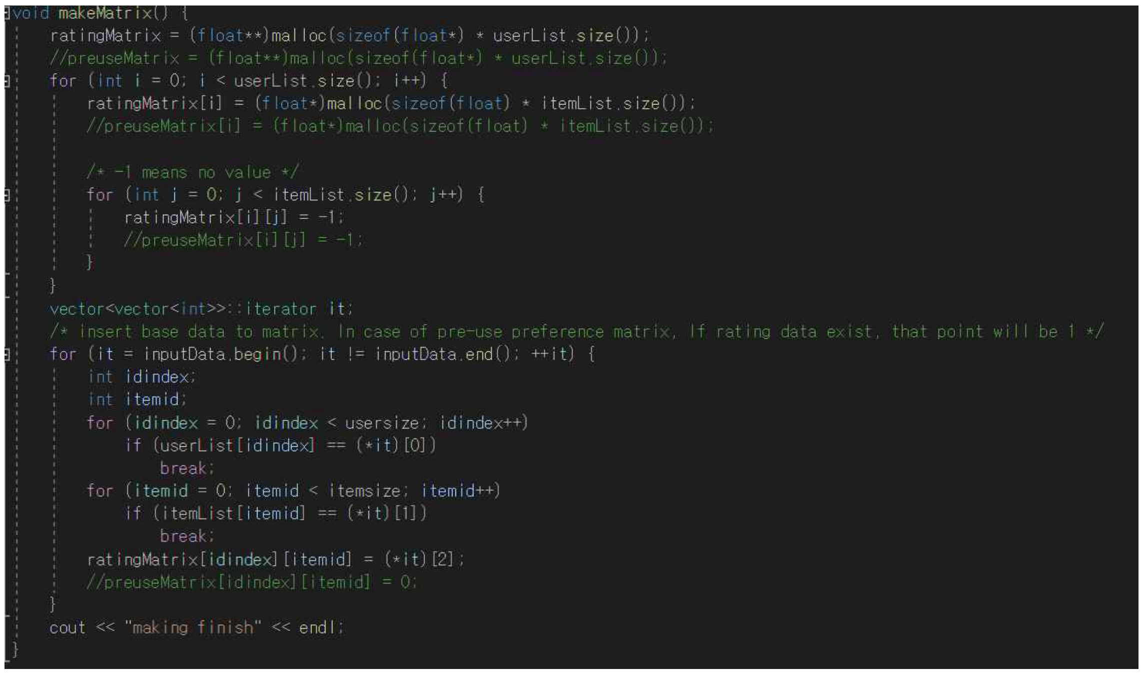
- read information that what is target of predict from test file



```
/* Write to result */
void writeTestFile(string filename, float** result) {
```

- write to base_prediction file

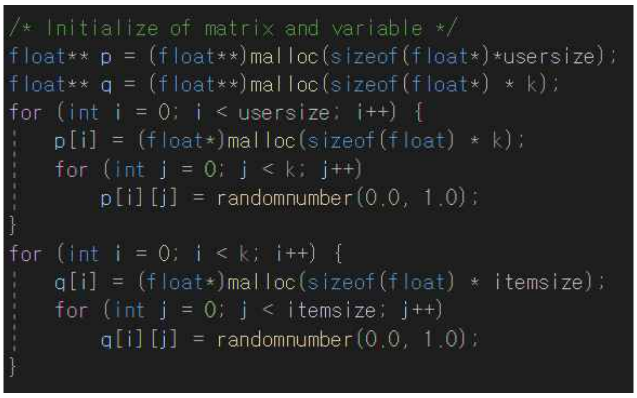
③ makeMatrix



```
void makeMatrix() {
    ratingMatrix = (float**)malloc(sizeof(float*) * userList.size());
    //preuseMatrix = (float**)malloc(sizeof(float*) * userList.size());
    for (int i = 0; i < userList.size(); i++) {
        ratingMatrix[i] = (float*)malloc(sizeof(float) * itemList.size());
        //preuseMatrix[i] = (float*)malloc(sizeof(float) * itemList.size());

        /* -1 means no value */
        for (int j = 0; j < itemList.size(); j++) {
            ratingMatrix[i][j] = -1;
            //preuseMatrix[i][j] = -1;
        }
    }
    vector<vector<int>>::iterator it;
    /* insert base data to matrix. In case of pre-use preference matrix, If rating data exist, that point will be 1 */
    for (it = inputData.begin(); it != inputData.end(); ++it) {
        int idindex;
        int itemid;
        for (idindex = 0; idindex < usersize; idindex++)
            if (userList[idindex] == (*it)[0])
                break;
        for (itemid = 0; itemid < itemsize; itemid++)
            if (itemList[itemid] == (*it)[1])
                break;
        ratingMatrix[idindex][itemid] = (*it)[2];
        //preuseMatrix[idindex][itemid] = 0;
    }
    cout << "making finish" << endl;
}
```

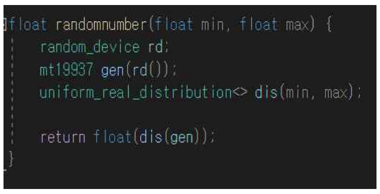
- make rating matrix. If there is no rating, set -1(means no rating)
- There is preuseMatrix initialize code that mark comment. That code of zero injection is just for testing which algorithm is better, but it is not efficient in matrix factorization by gradient descent, I remove it in my program.

④ matrixFactorization



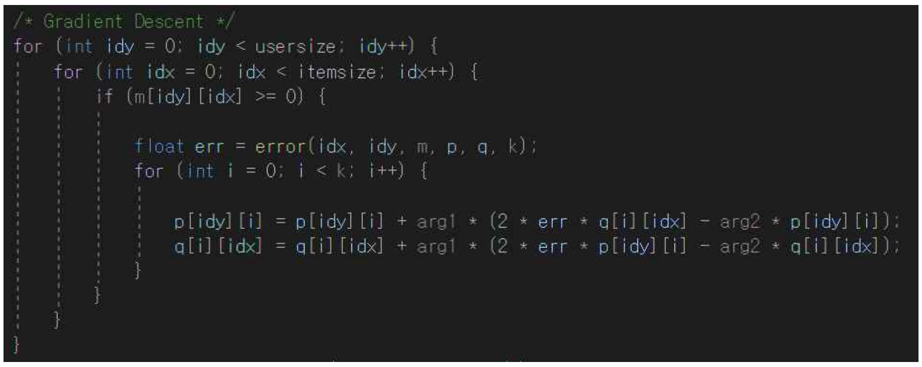
```
/* Initialize of matrix and variable */
float** p = (float**)malloc(sizeof(float*)*usersize);
float** q = (float**)malloc(sizeof(float*) * k);
for (int i = 0; i < usersize; i++) {
    p[i] = (float*)malloc(sizeof(float) * k);
    for (int j = 0; j < k; j++)
        p[i][j] = randomnumber(0.0, 1.0);
}
for (int i = 0; i < k; i++) {
    q[i] = (float*)malloc(sizeof(float) * itemsize);
    for (int j = 0; j < itemsize; j++)
        q[i][j] = randomnumber(0.0, 1.0);
}
```

- In matrixFactorization function, initialize fatorized matrix in first using random uniform distribution



```
float randomnumber(float min, float max) {
    random_device rd;
    mt19937 gen(rd());
    uniform_real_distribution<> dis(min, max);

    return float(dis(gen));
}
```

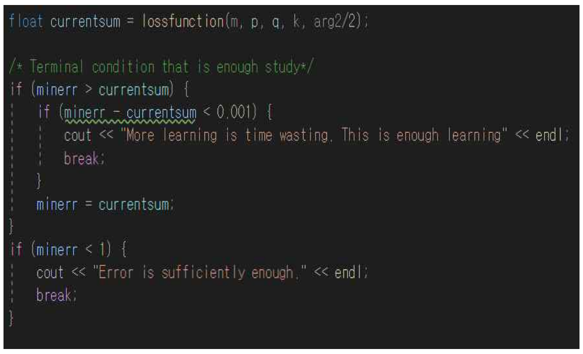
- It is make a random number function



```
/* Gradient Descent */
for (int idy = 0; idy < usersize; idy++) {
    for (int idx = 0; idx < itemsize; idx++) {
        if (m[idy][idx] >= 0) {

            float err = error(idx, idy, m, p, q, k);
            for (int i = 0; i < k; i++) {

                p[idy][i] = p[idy][i] + arg1 * (2 * err * q[i][idx] - arg2 * p[idy][i]);
                q[i][idx] = q[i][idx] + arg1 * (2 * err * p[idy][i] - arg2 * q[i][idx]);
            }
        }
    }
}
```

- error function returns m[i][j] - p[i][k]*q[k][j]
- using this formula, moving to minimum point of loss function

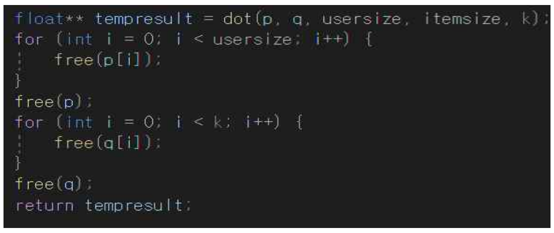
$$X_{k+1} = X_k - \lambda f^{'}(X_k)$$



```
float currentsum = lossfunction(m, p, q, k, arg2/2);

/* Terminal condition that is enough study*/
if (minerr > currentsum) {
    if (minerr - currentsum < 0.001) {
        cout << "More learning is time wasting. This is enough learning" << endl;
        break;
    }
    minerr = currentsum;
}
if (minerr < 1) {
    cout << "Error is sufficiently enough." << endl;
    break;
}
```

- calculate loss function by calling lossfunction() and using it, check terminal condition.

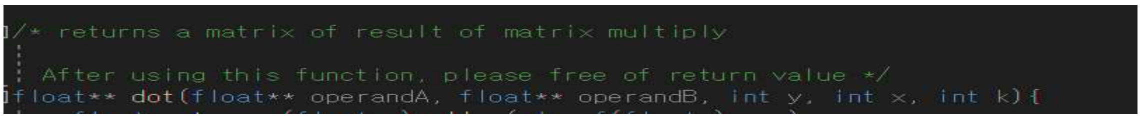
- terminal condition is lossfunction returns too small change between last learning and current learning or lossfunction is so small that already enough learning



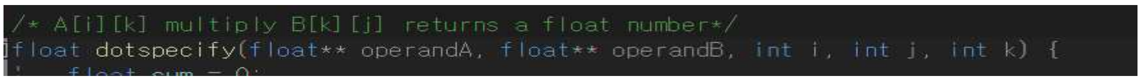
```
float** tempresult = dot(p, q, usersize, itemsize, k);
for (int i = 0; i < usersize; i++) {
    free(p[i]);
}
free(p);
for (int i = 0; i < k; i++) {
    free(q[i]);
}
free(q);
return tempresult;
```

- After learning, calculate predict matrix by multiply p,q
- And then, free p, q matrix

⑤ dot, dotspecify



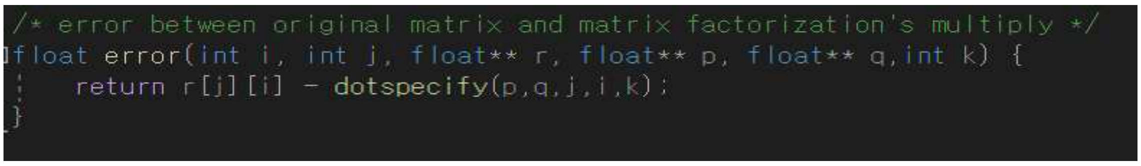
```
/* returns a matrix of result of matrix multiply

 After using this function, please free of return value */
float** dot(float** operandA, float** operandB, int y, int x, int k){
```



```
/* A[i][k] multiply B[k][j] returns a float number*/
float dotspecify(float** operandA, float** operandB, int i, int j, int k) {
```

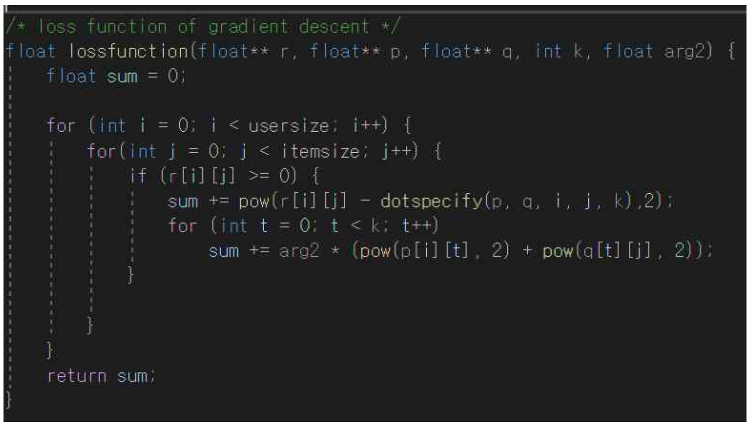
- It is functions just for calculating matrix.
- dotspecify is multiply specific column and row, and then return a float value
- dot is multiply two matrix and then return a result matrix

⑥ loss function, error



```
/* error between original matrix and matrix factorization's multiply */
float error(int i, int j, float** r, float** p, float** q,int k) {
    return r[j][i] - dotspecify(p,q,j,i,k);
}
```

- a original rating matrix's point subtract result matrix's point at specific learning section
- It returns a difference of two matrix.
- This function is for gain $f'(x)$



```
/* loss function of gradient descent */
float lossfunction(float** r, float** p, float** q, int k, float arg2) {
    float sum = 0;

    for (int i = 0; i < usersize; i++) {
        for(int j = 0; j < itemsize; j++) {
            if (r[i][j] >= 0) {
                sum += pow(r[i][j] - dotspecify(p, q, i, j, k),2);
                for (int t = 0; t < k; t++)
                    sum += arg2 * (pow(p[i][t], 2) + pow(q[t][j], 2));
            }


        }
    }
    return sum;
}
```

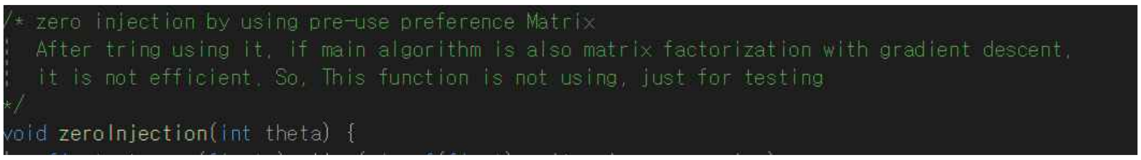
- This is Loss Function returns

$$Lossfunction = \sum_{i,j}^{size}(Original(i,j) - \mathrm{P}redict(i,j))^2 + b/2(\sum_{1}^{k}P(i,k)^2 + Q(,kj)^2)$$

k : user defined parameter that means dimension of factorized matrix.
 In my project, k is 3.
b : user defined parameter that composed to step size
P,Q : fatorized matrix

- In gradient descent, we try to minimize this loss function's value

⑦ using zero injection(for testing which algorithm is better)

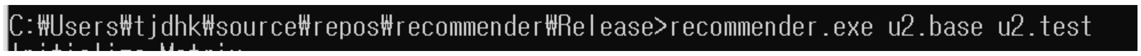


```
/* zero injection by using pre-use preference Matrix
   After tring using it, if main algorithm is also matrix factorization with gradient descent,
   it is not efficient. So, This function is not using, just for testing
*/
void zeroInjection(int theta) {
```

- This is a function for zero injection by using pre-use preference matrix.
- According my document section 4, I compare precision MF by gradient descent with MF with zeroinjection by gradient descent, It make poor results. So In my program, I don't choose zero injection to MF. Therefore, it is just function for testing.

3. Instruction for compiling and result

Executable file recommender.exe using by

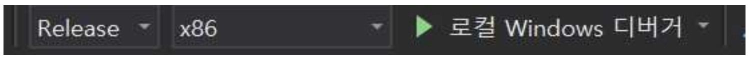


```
C:\Users\tjdhk\source\repos\recommender\Release>recommender.exe u2.base u2.test
```

this syntax.

        recommender.exe <base file name> <test file name>

Make exe by using visual studio, you import my source file(recommender.cpp) and change mode to release and build it.



<result>

Initial factorized matrix has random value of uniform real distribution, result slightly difference each execution. So, I take average value of 3 times execution of each base file.
parameter setting is alpha = 0.001, beta = 0.2, e = 3000

u1	RSME
1st Test	RMSE : 0.9336742
2nd Test	RMSE : 0.9348789
3rd Test	RMSE : 0.9326875

average	0.9337468

u2	RSME
1st Test	RMSE : 0.9258804
2nd Test	RMSE : 0.9268222
3rd Test	RMSE : 0.9260392
average	0.9262472

u3	RSME
1st Test	RMSE : 0.923207
2nd Test	RMSE : 0.9227527
3rd Test	RMSE : 0.9218636
average	0.9226077

u4	RSME
1st Test	RMSE : 0.9240443
2nd Test	RMSE : 0.9245628
3rd Test	RMSE : 0.9245047
average	0.9243706

u5	RSME
1st Test	RMSE : 0.9229636
2nd Test	RMSE : 0.9235064
3rd Test	RMSE : 0.9235466
average	0.9233388

4. Statistic about Parameter

e : number of learning step

theta : % of zero injection, alpha = 0.0001, beta = 0.02

e / method	Only MF	MF + Zero injection(theta = 10%)
500	0.9651146	1.79
1000	0.9513451	1.65
1500	0.9403142	1.56
2000	0.9381456	1.52
3000	0.9365149	1.52

Initial fatorized matrix set randomly, result RSME different slightly. So, average of 5 test of 6 datasets.(5 * 6 = 30 tests)

Test result show in gradient descent matrix factorization, non zero injection is better performance. Because in gradient descent, loss function distort zero value.

So in my project, I just implementation function of zero injection, but don't use it.

When number of iteration is 3000, change parameter of step size

test is average of 5 test about u5.base and u4.base.

alpha	beta	Result
0.0001	0.02	0.93316
0.0002	0.04	0.93305
0.0004	0.08	0.93156
0.001	0.2	0.92367
0.002	0.4	0.94430

In this result and from formula, in sufficient number of learning step, there is no huge difference about step size that is appropriate small.
But, we can know small step size needs many learning step.
at e = 3000 -> e = 10000 steps :
alpha = 0.0001 and beta = 0.02 improve there precision. However, alpha = 0.001 and beta = 0.2 don't improve there precision with e-value changing.
In my program, I choose alpha 0.01, beta 0.2, because considering execution time of program and precision.