Recommender systems

Xiaoyang Zhang

B00708854

Overview

A recommender system is a subclass of information **filtering system** that seeks to **predict** the "rating" or "preference" a user would give to an item.

A : Retail Trade

B : Information Distribution System

Java + Python (Jupyter notebook)

01 Input file analysis

02 Non-negative matrix factorization (NMF)

03 Test & Error Analysis

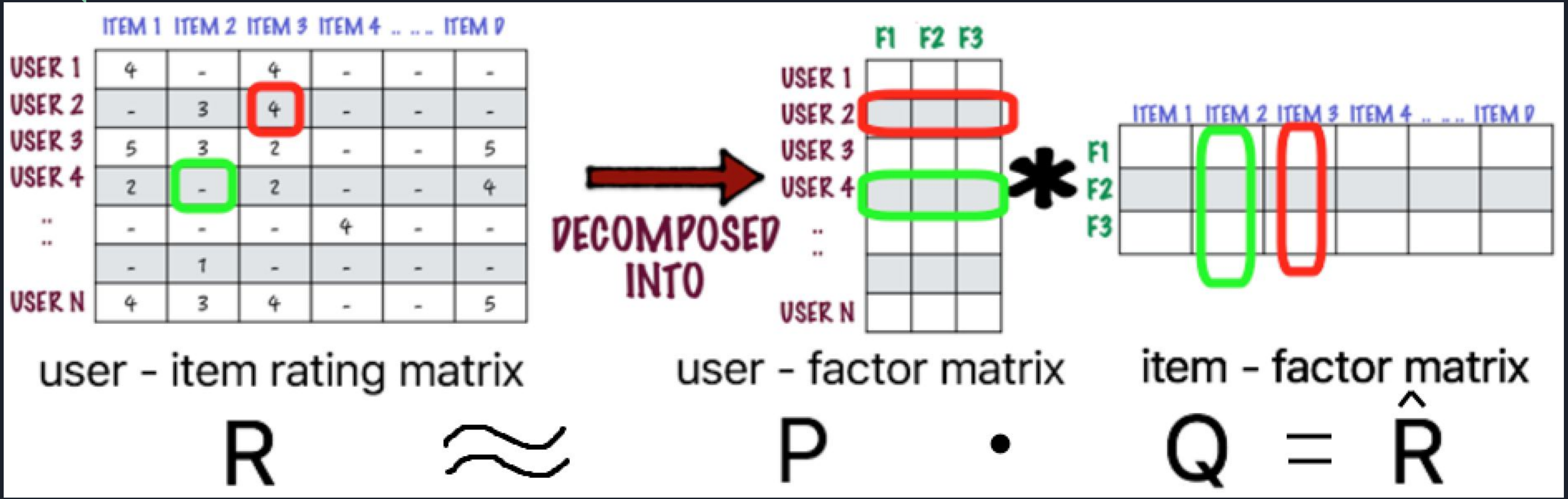
01 Input file analysis

A: 943 users / 1650 items (requirement: 943/1682)

B: Sparse matrix: valid data 5.14%

userID	1	2	3	4	5	6	7	8	9	10	...	934	935	936	937	938	939	940	941	942	943
itemID																					
1	5.0	4.0	NaN	NaN	4.0	NaN	NaN	NaN	NaN	4.0	...	2.0	3.0	4.0	NaN	4.0	NaN	NaN	5.0	NaN	NaN
2	3.0	NaN	NaN	NaN	3.0	NaN	NaN	NaN	NaN	NaN	...	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	5.0
3	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	3.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4.0	...	5.0	NaN	NaN	NaN	NaN	NaN	2.0	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

02 Non-negative matrix factorization (NMF)



A matrix R is factorized into (usually) two matrices P and Q, with the property that these matrices have **no negative** elements.

02 Non-negative matrix factorization (NMF)

A: $\mathbf{R} \approx \mathbf{P} \times \mathbf{Q} \quad = \hat{\mathbf{R}}$ $\qquad \hat{r}_{ij} = p_i \, q_j = \sum_{k=1}^{k} p_{ik} q_{kj}$

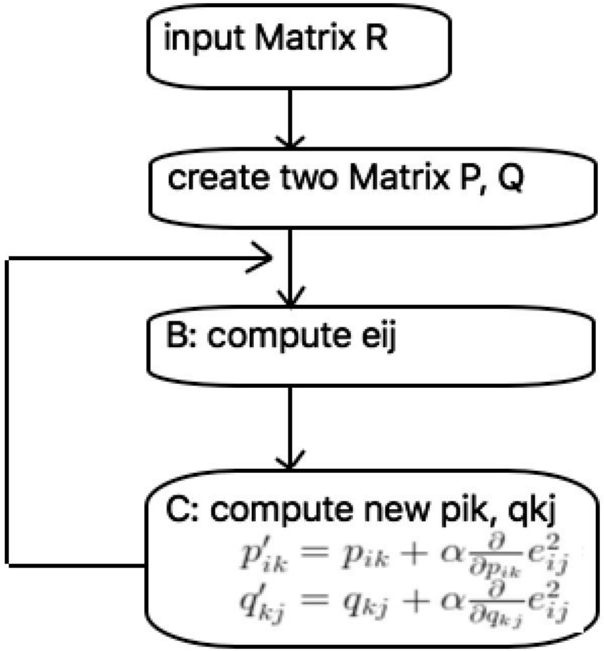
B: $e_{ij}^2 = (r_{ij} - \sum_{k=1}^{K} p_{ik} q_{kj})^2 + \frac{\beta}{2} \sum_{k=1}^{K} (||P||^2 + ||Q||^2)$

C:

$$p'_{ik} = p_{ik} + \alpha \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + \alpha(2e_{ij} q_{kj} - \beta p_{ik})$$

$$q'_{kj} = q_{kj} + \alpha \frac{\partial}{\partial q_{kj}} e_{ij}^2 = q_{kj} + \alpha(2e_{ij} p_{ik} - \beta q_{kj})$$

02 Non-negative matrix factorization (NMF)



R[user][item]

P[user][k], Q[k][item]
init P/Q, put random value in them

training, keep improving the values in P/Q matrix

```
// Run 5000 steps, Alpha 0.002, Beta 0.04
NNmf nmf = new NNmf(trainRM, PM, QM, Kc, Uc, Oc, 5000, 0.002, 0.04);
nmf.run();
```

Result & Error Analysis

A: training data: 80%

test data: 20%

B: 5-fold Cross-validation

C: RMSE: 0.96

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} (X_i - X_{is})^2}{n}}$$

RMSE has the nice property that it amplifies the contributions of egregious errors, both false positives and false negatives.

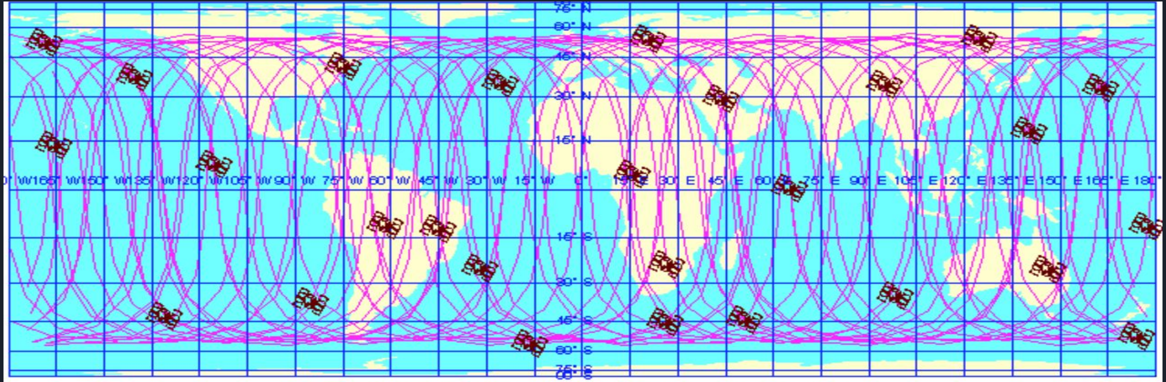
```
//RMSE part
double errorSum = 0;
double RMSE = 0;
for (int i = 0; i < Uc; i++) {
    for (int j = 0; j < Oc; j++) {
        if(testRM[i][j] != 0){
            errorSum = errorSum + Math.pow(testRM[i][j] - trainRM[i][j], 2);
        }
    }
}
RMSE = Math.sqrt(errorSum/testData);
```

--Netflix Prize (Recommender systems)

real-world application problems with a solution of matrix completion

Global positioning

The global positioning problem emerges naturally in sensor networks. The problem is to recover the global positioning of points in space from a local or partial set of pairwise distances. Thus it is a matrix completion problem with rank two if the sensors are located in a 2-D plane and three if they are in a 3-D space.

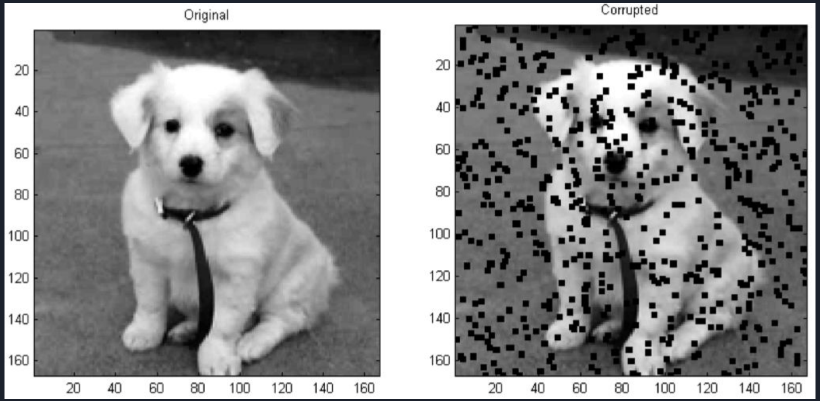


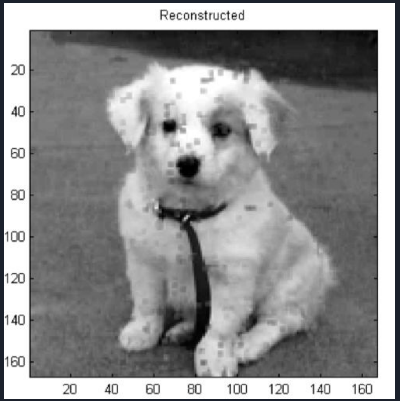
Global Positioning System Satellites and Orbits

real-world application problems with a solution of matrix completion

Image Processing

Matrix completion problem deals with the reconstruction of a data matrix from a small subset of its entries. It has been shown that, under certain conditions, the missing entries can be recovered, when the data matrix has a low rank.





Thank you !