Metodi Numerici per l'Informatica
A.A. 2021-2022
Prof.ssa Angelamaria CARDONE

Marika Pia Salvato
Matr. 0522500897

Sistemi di raccomandazione basati sul metodo SGD

1 Modello dei fattori latenti

MODELLO DEI FATTORI LATENTI

Matrice dei Ratings

$$R_{nu,nv} = \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,nv} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,nv} \\ \vdots & \vdots & \ddots & \vdots \\ r_{nu,1} & r_{nu,2} & \cdots & r_{nu,nv} \end{pmatrix}$$

Matrici dei fattori latenti

$$U_{nu,d} = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_{nu} \end{pmatrix} \quad , \quad V_{d,nv} = (v_1 \quad v_2 \quad \cdots \quad v_{nv})$$

Matrice dei ratings approssimata

$$U_{nu,d} V_{d,nv} = R'_{nu,nd} \approx R_{nu,nv}$$

Obiettivo:

$$min\ SSE = min \sum_{(i,j) \in K} (r_{ij} - r'_{ij})^2 + \lambda(||u_i||^2 + ||v_j||^2)$$

Stochastic Gradient Descent

Ad ogni iterazione:

- Scelta randomica di un rating r_{ij} da K
- Aggiornamento della riga u_i e della colonna v_j tramite le formule (regolarizzate):

$$u'_{ik} = u_{ik} + \eta(e_{ij} v_{kj} - \lambda u_{ik}), \quad per\ k = 1, \ldots, d$$
$$v'_{kj} = v_{kj} + \eta(e_{ij} u_{ik} - \lambda v_{kj}), \quad per\ k = 1, \ldots, d$$

2 Algoritmo seriale

ALGORITMO SERIALE

Dataset di training **MovieLens** (ml-latest-small):

- Numero di Utenti = 610
- Numero di Film = 193'609
- ~ 20 ratings per utente

$R_{(610,193609)}$ è **molto sparsa**

ALGORITMO SERIALE

CARICAMENTO DEL DATASET

loadDataset() carica il dataset in R

generateR() genera una matrice di ratings nu x nv con sparsità dell'75% (75% di valori uguali a 0)

```
/* Inizializzazione di R*/
float *R;
int nu, nv;      // presi da input

if ((nu == 0)||(nv == 0))
{
    nu=610;
    nv=193609;
    R=(float *)malloc( nu * nv *(sizeof(float)));
    loadDataset(R,nu,nv,&n_miss,&sparsity);
}else
{
    R = (float *)malloc(nu*nv*sizeof(float));
    generateR(R,nu,nv,&n_miss,&sparsity);
}
```

CREAZIONE DEL TRAINING SET

R_train è una matrice n_R_train x 3

n_R_train = numero di ratings non nulli in R

```
/*
        Creazione del training set a partire da R
        id utente | id item | rating
*/

int n_R_train = ((nu*nv)-n_miss);
float R_train = (float *)malloc(n_R_train*3*sizeof(float));
int c = 0;

for ( i = 0; i < nu; i++)   // i == id utente
{
    for ( j = 0; j < nv; j++)   // j == id item
    {
        if (R[i*nv+j] != 0)
        {
            R_train[c*3+0] = i;
            R_train[c*3+1] = j;
            R_train[c*3+2] = R[i*nv+j];
            c++; // riga successiva
        }
    }
}
```

PREPARAZIONE DEI PARAMETRI

```
/* Parametri presi da input */
int d;      // numero di fattori latenti
float learning_rate, reg_param;
int n_iter; // numero di iterazioni

/* Inizializzazione di Rcap */
float Rcap = (float *)malloc(nu*nv*sizeof(float));

train(R_train, n_R_train, Rcap, nu, nv, d, learning_rate, reg_param, n_iter);
```

ALGORITMO SERIALE

FASE DI TRAINING #1

U di dimensioni nu x d

V di dimensioni d x nv

Inizializzazione con valori 0 e 1.

```
void train(float *R_train, int n_R_train, float *Rcap,
           int nu, int nv, int d, float l_r, float reg_param, int n_iter)
{
    float *U, *V;
    int i,j;

    /* Inizializzazione delle matrici U e V */
    U = (float *)malloc(nu*d*sizeof(float));
    V = (float *)malloc(nv*d*sizeof(float));

    srand((int)time(0));
    for ( i = 0; i < nu; i++)
    {
        for ( j = 0; j < d; j++)
        {
            U[i*d + j] = rand() % 2;   // genera numeri da 0 a 1
        }

    }

    for ( i = 0; i < d; i++)
    {
        for ( j = 0; j < nv; j++)
        {
            V[i*nv + j] = rand() % 2;   // genera numeri da 0 a 1
        }
    }
```

FASE DI TRAINING #2

```
void train(float *R_train, int n_R_train, float *Rcap,
           int nu, int nv, int d, float l_r, float reg_param, int n_iter)
{
    ...
    float, e=0, r_ui, r_ui_pred=0;
    int r, utente, item;

    /* Inizia la fase di training */
    for (int iter = 0; iter < n_iter; iter++)
    {
        // scelta random di 1 rating dal training set
        r = rand() % n_R_train;   // genera numeri da 0 a # dei ratings

        utente = (int) R_train[r*3+0]; // id utente
        item = (int) R_train[r*3+1]; // id item
        r_ui = R_train[r*3+2]; // rating

        // calcolo della predizione del rating
        r_ui_pred = calcPreds(U,V,nu,nv,d,utente,item);
```

Ad ogni iterazione, si sceglie un rating e si calcola la predizione con U e V

```
float calcPreds(float *U, float *V, int nu, int nv,
                int d, int utente, int item)
{
    int i;

    // calcolo della predizione del rating
    float r_ui_pred = 0;
    for ( i = 0; i < d; i++)
    {
        r_ui_pred += U[utente*d+i]*V[i*nv+item];
    }

    return r_ui_pred;
}
```

FASE DI TRAINING #3

Update di U e V tramite le formule derivate dal gradiente

Rcap contiene i rating calcolati con U e V alla fine delle n_iter iterazioni

```
void train(float *R_train, int n_R_train, float *Rcap,
           int nu, int nv, int d, float l_r, float reg_param, int n_iter)
{
        ...
        // calcolo dell'errore
        e = r_ui - r_ui_pred;

        // update dei fattori latenti
        for ( i = 0; i < d; i++)
        {
            U[utente*d+i] += l_r * (e*V[i*nv+item] - reg_param*U[utente*d+i]);
            V[i*nv+item] += l_r * (e*U[utente*d+i] - reg_param*V[i*nv+item]);
        }
    } // end for n_iter

    /* Generazione di Rcap a partire da U e V */
    for ( i = 0; i < nu; i++)
    {
        for ( j = 0; j < nv; j++)
        {
            Rcap[i*nv+j] = calcPreds(U,V,nu,nv,d,i,j);
        }
    }
} // end dichiarazione train
```

ALGORITMO SERIALE

VALUTAZIONE DEL MODELLO

```
... // nel main
/* Calcolo di RMSD */
rmsd = calcRMSD(R_train, n_R_train, Rcap, nv);

// end main
```

```
float calcRMSD(float *R_train, int n_R_train,
               float *Rcap, int nv)
{
    int i, utente, item;
    float r_ui_pred, r_ui, rmsd;

    for ( i = 0; i < n_R_train; i++)
    {
        utente = (int) R_train[i*3];
        item = (int) R_train[i*3+1];
        r_ui = R_train[i*3+2];

        r_ui_pred = Rcap[utente*nv+item];

        rmsd += pow(r_ui_pred-r_ui,2);
    }
    rmsd=sqrt(rmsd/n_R_train);

    return rmsd;
}
```

$$e_{ij} = r_{ij} - \hat{r}_{ij} \quad \text{for } (i,j) \in K \ (\subset R) \ \ (K \text{ insieme dei rating non mancanti})$$
$$e'_{ij} = \frac{1}{2} e_{ij}^2$$

$$SSE = \sum_{(i,j)\in K} e_{i,j}^2 \ , \qquad SSE' = \frac{1}{2} SEE = \sum_{(i,j)\in K} e'_{ij}$$
$$RMSE = \sqrt{SSE/|K|}$$

ALGORITMO SERIALE

Esecuzione su MovieLens

```
$ ./mf_sgd

***  Fattorizzazione di Matrice - SGD   ***

Dimensioni della matrice di rating R
    nu = 0
    nv = 0

estratto 10x10 di R =
4.00    0.00    4.00    0.00    0.00    4.00    0.00    0.00    0.00    0.00
0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
0.00    0.00    0.00    0.00    0.00    0.00    5.00    0.00    0.00    5.00
0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    3.00
0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00

R dim = (610,193609),   sparsità = 99.91%
```

#1

```
Numero di fattori latenti d = 10
Learning rate = 0.01
Parametro di regolarizzazione = 0.03
Numero di iterazioni = 100000

RMSD = 1.043248

Tempo di esecuzione = 4.644269 secondi
```

#2

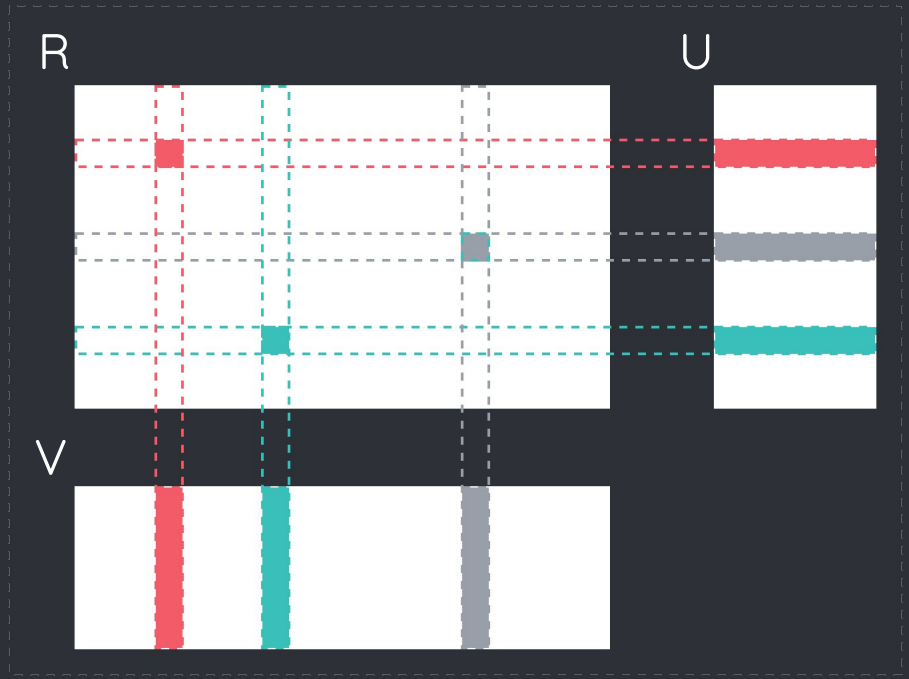
```
Numero di fattori latenti d = 10
Learning rate = 0.01
Parametro di regolarizzazione = 0.03
Numero di iterazioni = 1000000

RMSD = 0.803702

Tempo di esecuzione = 5.377245 secondi
```

3 Algoritmi paralleli

HogWild!

Asincrono

- Approccio lock-free
- Thread indipendenti

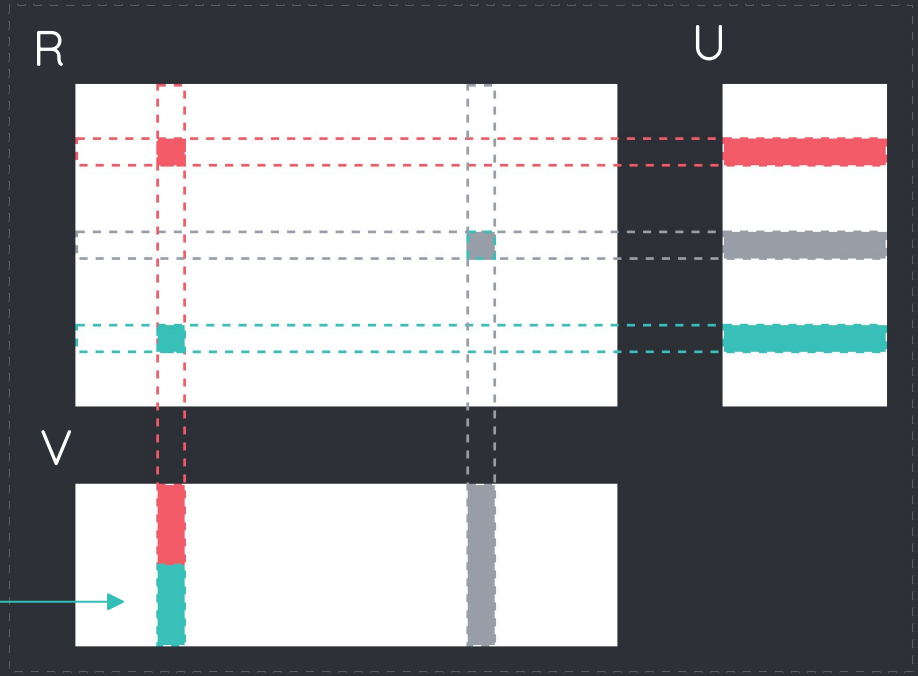


HogWild!

Asincrono

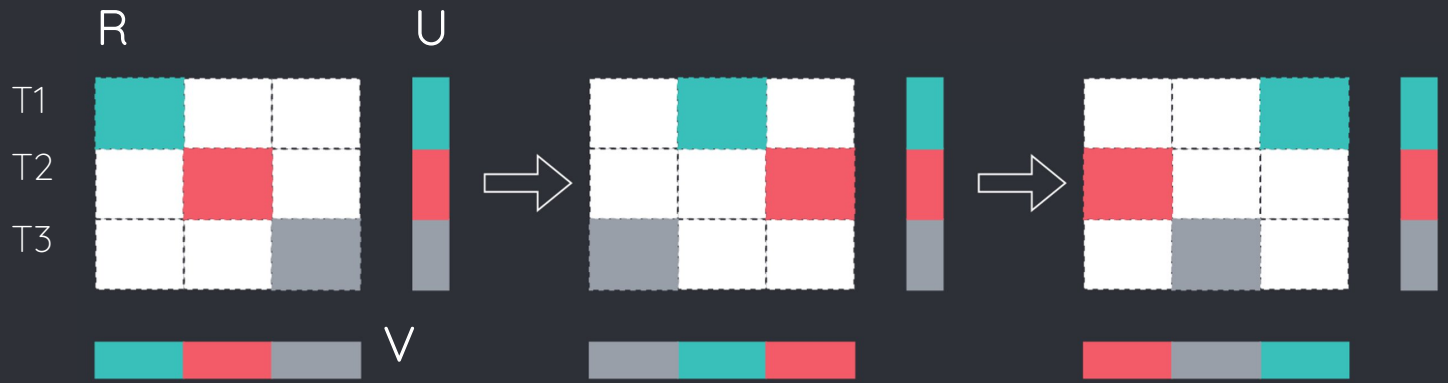
- Garantisce la convergenza

Overwrite



Stratified (o sincrono)

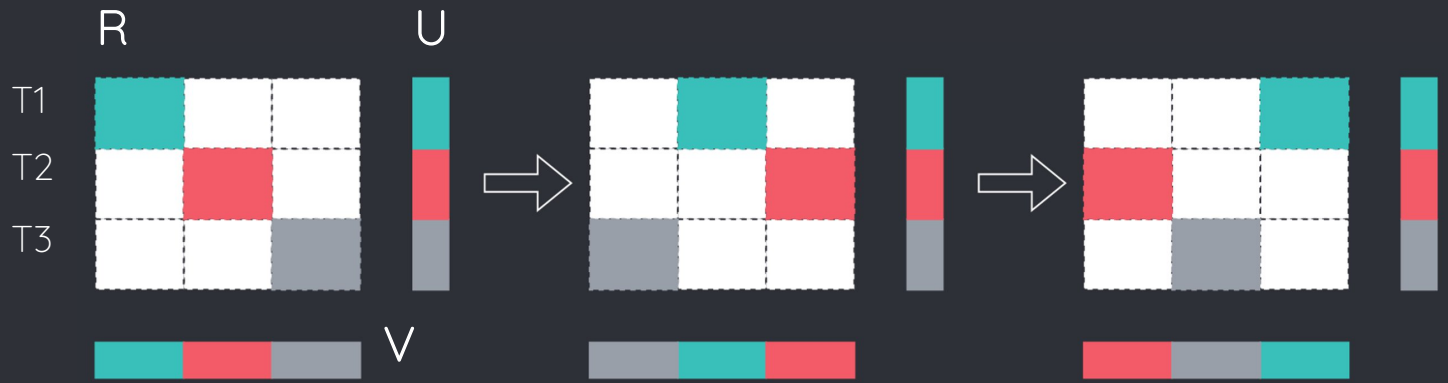
- Numero di thread t
- R in blocchi txt
- Numero di iterazioni n_iter



DSGD

Stratified (o sincrono)

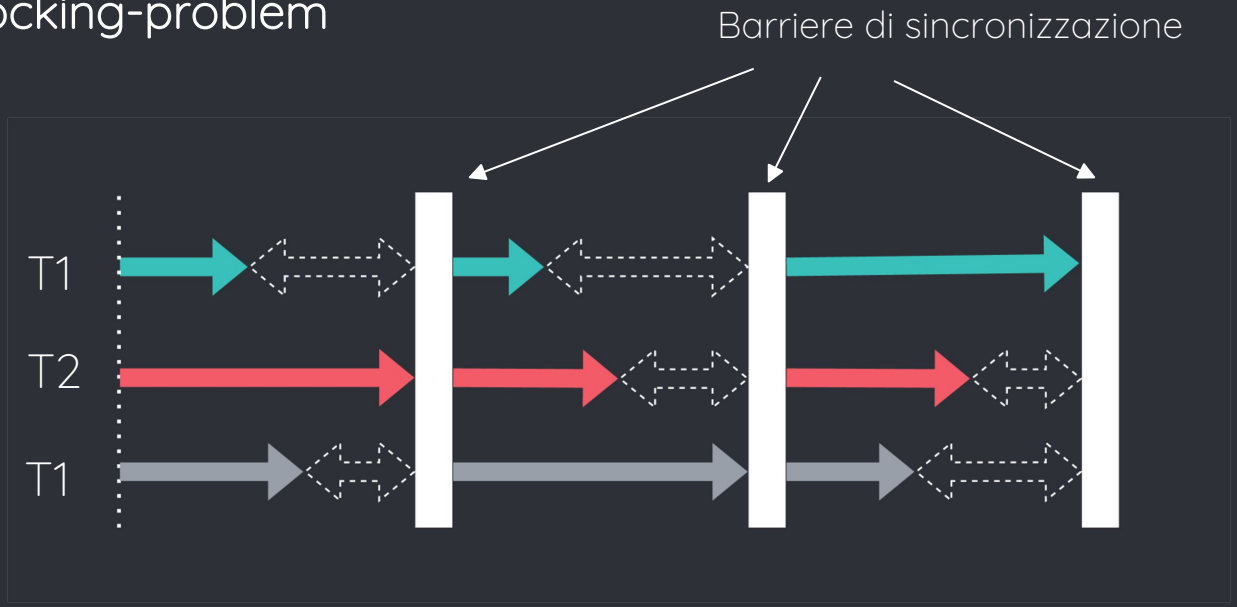
1. Per ogni iterazione (n_iter):
2. Assegna un blocco finché tutta R non è stata processata (per un totale di t sub-iterazioni).
3. In modo parallelo: ogni thread sceglie un rating dal blocco e aggiorna U e V.



DSGD

Stratified (o sincrono)

- Locking-problem



RIFERIMENTI

al materiale teorico

- Matrix factorization Techniques for recommender systems, Koren et al. (2009)
- Investigation of Various Matrix Factorization Methods for Large Recommender Systems, Gabor et al. (2008)

Ad altri algoritmi

- A Fast Parallel SGD for Matrix Factorization in Shared Memory Systems, Zhuang et al. (2013)
- Fast and Robust Parallel SGD Matrix Factorization, Jinoh Oh et al. (2015)
- Scalable Task-Parallel SGD on Matrix Factorization in Multicore Architectures, Nishioka et al. (2015)
- CuMF_SGD: Parallelized Stochastic Gradient Descent for Matrix Factorization on GPUs, Xie et al. (2020)

Grazie per l'attenzione!