Recommender Systems

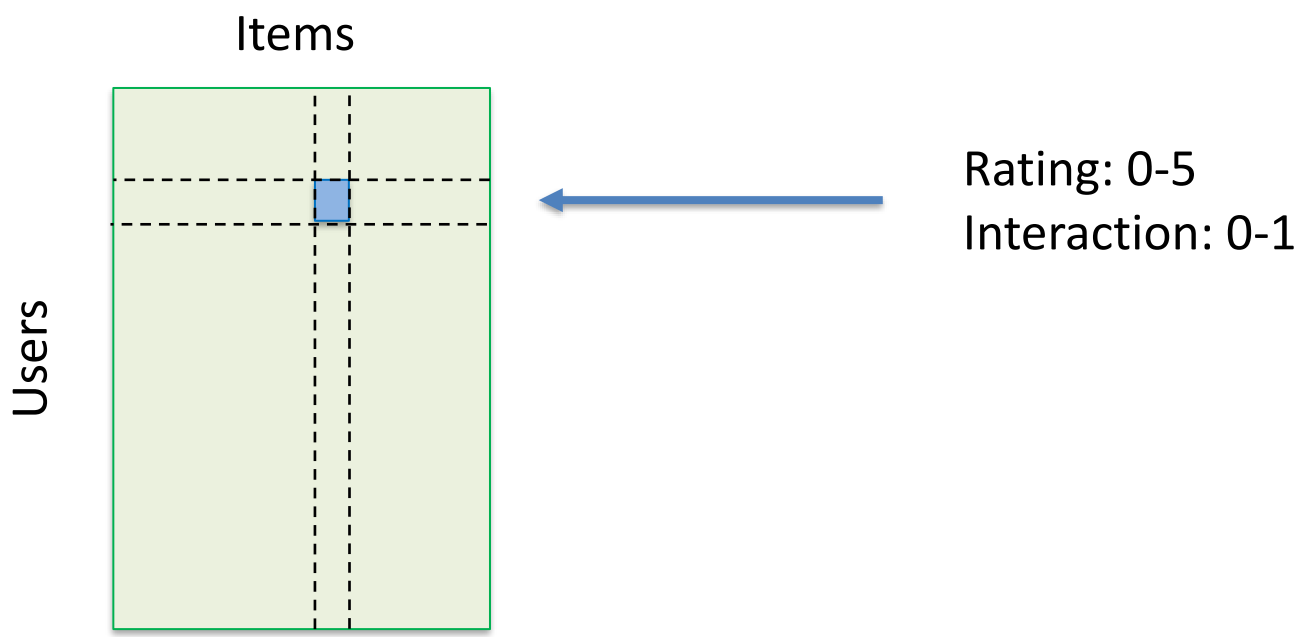
Sparse matrices with Scipy

Maurizio Ferrari Dacrema

A.Y. 2021/22

User Rating Matrix (URM)

In Recommender Systems we often rely on user-item interaction data, which tells us the items any user interacted with.



User Rating Matrix (URM)

The URM has a number of cells equal to $|items| \cdot |users|$

Consider a relatively small case:

- $100'000 = 10^5$ users and items
- 10^{10} cells or $10^{10} \cdot 16\ bit = 80\ GB$ to store integer interaction data
- Most of the data will be zero. The number of interactions that occurred is typically in the range of 10^{-2} / 10^{-5}

Need better solution!

Sparse matrix

Underlying idea:

- Assume all cells have a default value (zero)
- Store only the cells that have a different value

Challenges:

- We need fast access
- We need fast linear algebra operations, e.g., multiplication

COOrdinate Format

The COOrdinate format stores values as triples (row, col, data)

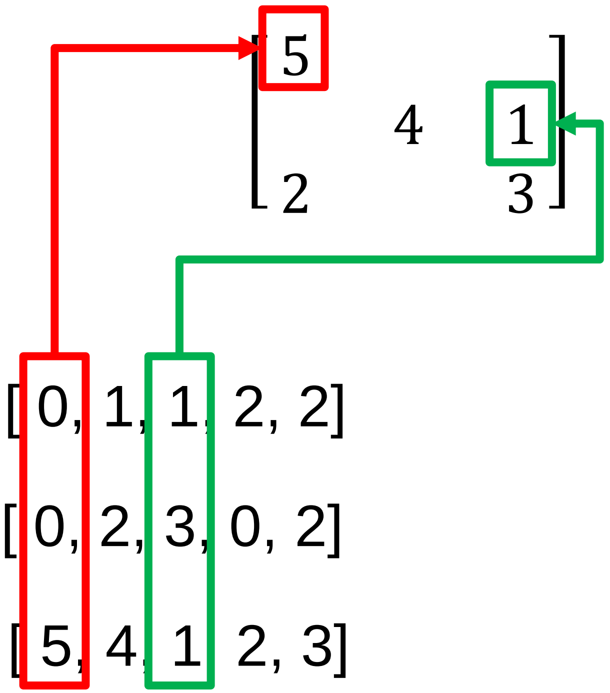
$$\begin{bmatrix} 5 & & & \\ & & 4 & 1 \\ 2 & & & 3 \end{bmatrix}$$

In scipy:

```
matrix.row  = [ 0, 1, 1, 2, 2]
matrix.col   = [ 0, 2, 3, 0, 2]
matrix.data = [ 5, 4, 1, 2, 3]
```

COOrdinate Format

The COOrdinate format stores values as triples (row, col, data)



$$\begin{bmatrix} \boxed{5} & & & \\ & & 4 & \boxed{1} \\ 2 & & 3 & \end{bmatrix}$$

In scipy:

```
matrix.row  = [0, 1, 1, 2, 2]
matrix.col   = [0, 2, 3, 0, 2]
matrix.data = [5, 4, 1, 2, 3]
```

COOrdinate Format

COO format is easy to read

$$\begin{array}{ll} row: & 0,0,0,0,0,0,0,0,1,1,1,1,1,1,2,2,2,3,3,3\ \ldots \\ col: & 1,5,0,2,5,7,0,3,2,0,8,6,0,1,1,2,0,6,5,3\ \ldots \end{array}$$

COOrdinate Format

COO format is easy to read but not usable for matrix operations

$$
\begin{array}{ll}
row: & 0,0,0,0,0,0,0,0,1,1,1,1,1,1,2,2,2,3,3,3\ \dots \\
col: & 1\boxed{,5,}0,2\boxed{,5,}7,0,3,2,0,8,6,0,1,1,2,0,6\boxed{,5,}3\ \dots
\end{array}
$$

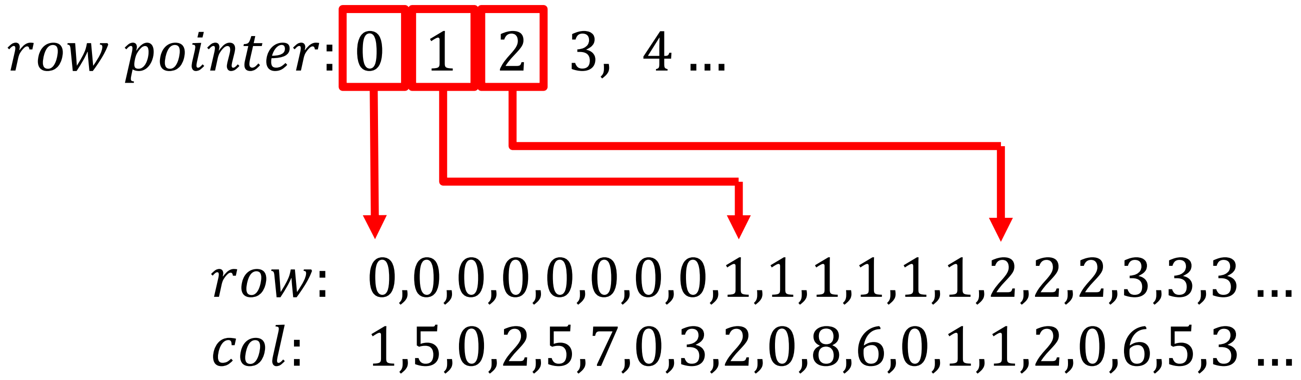
How to find the elements in column 5? Loop all the “col” array

How to find row 2569? Loop the “row” array until you find it

Impossible to do fast access and matrix operations

From COO to Compressed formats

What if we store separately a pointer to where a row begins?



$row\ pointer$: 0 1 2 3, 4 ...
row: 0,0,0,0,0,0,0,0,1,1,1,1,1,1,2,2,2,3,3,3 ...
col: 1,5,0,2,5,7,0,3,2,0,8,6,0,1,1,2,0,6,5,3 ...

Compressed Sparse Row (CSR) format

CSR format is developed to provide fast row access

The idea:

- Store only the (col, data) explicitly but not the row
- Use a “indptr” data structure to know where the row begins in the (col, data) structure

Data for row i will be between positions $indptr[i]\colon indptr[i+1]$

Compressed Sparse Row (CSR) format

$$\begin{bmatrix} 5 & & \\ & 4 & 1 \\ 2 & & 3 \end{bmatrix}$$

In scipy:

```
matrix.indptr   = [ 0, 1, 3, 5 ]

matrix.indices  = [ 0, 1, 2, 0, 2 ]

matrix.data     = [ 5, 4, 1, 2, 3 ]
```

Compressed Sparse Row (CSR) format

$$\begin{bmatrix} \boxed{5} & & \\ & 4 & 1 \\ 2 & & 3 \end{bmatrix}$$

In scipy:

matrix.indptr = [$\boxed{0, 1}$, 3, 5]

matrix.indices = [$\boxed{0}$, 1, 2, 0, 2]

matrix.data = [$\boxed{5}$, 4, 1, 2, 3]

Row 0 corresponds to the data from coordinate 0 to 1 excluded

Compressed Sparse Row (CSR) format

$$\begin{bmatrix} 5 & & \\ & \boxed{4 \quad\quad 1} & \\ 2 & & 3 \end{bmatrix}$$

In scipy:

```
matrix.indptr   = [ 0, 1, 3, 5 ]
matrix.indices  = [ 0, 1, 2, 0, 2 ]
matrix.data     = [ 5, 4, 1, 2, 3 ]
```

Row 1 corresponds to the data from coordinate 1 to 3 excluded

Compressed Sparse Row (CSR) format

$$\begin{bmatrix} 5 & & \\ & 4 & 1 \\ \boxed{2} & & \boxed{3} \end{bmatrix}$$

In scipy:

$$\begin{array}{ll} \text{matrix.indptr} & = [\ 0,\ 1,\ \boxed{3,\ 5}\ ] \\ \text{matrix.indices} & = [\ 0,\ 1,\ 2,\ \boxed{0,\ 2}\ ] \\ \text{matrix.data} & = [\ 5,\ 4,\ 1,\ \boxed{2,\ 3}\ ] \end{array}$$

Row 2 corresponds to the data from coordinate 3 to 5 excluded

Compressed Sparse Row (CSR) format

$$\begin{bmatrix} 5 & & \\ & 4 & 1 \\ 2 & & 3 \end{bmatrix}$$

In scipy:

matrix.indptr = [0, 1, 3, $\boxed{5}$] $\xleftarrow{\quad}$ Row 3 (coordinate 5) does not exist in the matrix. It is added because it allows to loop indptr without a special case for the last row

matrix.indices = [0, 1, 2, 0, 2]

matrix.data = [5, 4, 1, 2, 3]

Compressed Sparse Row (CSR) format

Pros:

- Very fast row access (just two array accesses)
- Fast row-wise linear algebra operations

Cons:

- A bit less readable
- <u>Very slow for column access</u>, like COO

Compressed Sparse Column (CSC) format

The Compressed Sparse Column format works exactly like the CSR except that it compresses rows.

How to use COO, CSC, CSR in practice

In practice, one usually creates the matrix using the simple COO format and the uses the library functions to get the CSC or CSR

The format is not set in stone, if for an operation you need column access and for another you need row access, it is likely faster to change the format of the matrix than to use only one format.

Typical mistake

What happens if I use a CSR when I need fast column access, or a CSC for row access?

Nothing really…

the result will be correct (unless you use the internal data structures directly), but the script will be <u>tens or hundreds of times slower</u>