MA TRẬN VÀ CÁC PHÉP TOÁN

Nguyễn Mạnh Hùng

AI Academy Vietnam

July, 2024

Nội dung

Khái niệm

Ma trận (matrix)

Ma trận kích thước $m \times n$ là một bảng chữ nhật gồm $m.n$ số được xếp thành m hàng và n cột:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

trong đó a_{ij} là phần tử của ma trận nằm ở hàng thứ i và cột thứ j.

Ví dụ: $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$ là một ma trận kích thước 2×3.

Khái niệm

Ma trận có thể sử dụng để mô tả dữ liệu dạng bảng:

- *Ảnh xám*: là một ma trận kích thước $m \times n$ gồm các điểm ảnh được mô tả bởi 1 số thuộc [0,255]
- *Dữ liệu về lượng mưa*: ma trận A kích thước $m \times n$ ghi lượng mưa tại m địa điểm khác nhau trong n ngày liên tiếp.
- *Lợi nhuận đầu tư*: ma trận R kích thước $T \times n$ ghi lợi nhuận của một danh mục đầu tư gồm n tài sản trong khoảng thời gian T.

Date	AAPL	GOOG	MMM	AMZN
March 1, 2016	0.00219	0.00006	−0.00113	0.00202
March 2, 2016	0.00744	−0.00894	−0.00019	−0.00468
March 3, 2016	0.01488	−0.00215	0.00433	−0.00407

Một số ma trận đặc biệt

- **Ma trận vuông** (square matrix): có số hàng = số cột = n, gọi là ma trận là vuông cấp n. Ví dụ ma trận vuông cấp 3:

$$A = \begin{pmatrix} 1 & 2 & -1 \\ 3 & 1 & 5 \\ -2 & 4 & 4 \end{pmatrix}$$

- Giả sử $A = (a_{ij})_{n\times n}$ là một ma trận vuông cấp n. Dãy số:

$$(a_{11}, a_{22}, \ldots, a_{nn})$$

được gọi là **đường chéo** (diagonal) của ma trận. Tổng tất cả các phần tử thuộc đường chéo được gọi là "**vết**" của ma trận:

$$\text{trace}(A) = a_{11} + a_{22} + \cdots + a_{nn}$$

Một số ma trận đặc biệt

- **Ma trận tam giác trên/dưới** (upper/lower triangle matrix): là ma trận vuông, có các phần tử nằm phía dưới/trên đường chéo bằng 0:

$$U = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \ddots & \cdots \\ 0 & 0 & \cdots & a_{nn} \end{pmatrix}; \quad L = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & 0 \\ \cdots & \cdots & \ddots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$

- **Ma trận đường chéo** (diagonal matrix): là ma trận vuông, có các phần tử nằm ngoài đường chéo bằng 0:

$$L = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \cdots & \cdots & \ddots & \cdots \\ 0 & 0 & \cdots & \lambda_n \end{pmatrix}$$

Một số ma trận đặc biệt

- **Ma trận đơn vị** (identity matrix): là ma trận vuông, các phần tử nằm trên đường chéo bằng 1, các phần tử còn lại bằng 0. Ví dụ:

$$I_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- **Ma trận không** (zero matrix): có các phần tử đều bằng 0. Ví dụ:

$$\theta_{2\times 3} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Các phép toán

Hai ma trận bằng nhau

Hai ma trận A và B được gọi là bằng nhau nếu chúng có cùng kích thước và các phần tử ở cùng vị trí bằng nhau. Kí hiệu: $A = B$

Chuyển vị ma trận (transpose)

Cho ma trận $A = (a_{ij})_{m\times n}$ kích thước $m \times n$. Chuyển vị của ma trận A là ma trận $A^T = (a'_{ij})_{n\times m}$ sao cho $a'_{ij} = a_{ji}$.

Ví dụ:

- Bằng nhau:

$$\begin{pmatrix} 9 & x \\ y & 5 \end{pmatrix} = \begin{pmatrix} 9 & 4 \\ 6 & 5 \end{pmatrix} \Leftrightarrow \begin{cases} x = 4 \\ y = 6 \end{cases}$$

- Chuyển vị:

$$\begin{pmatrix} 1 & 5 \\ 7 & 2 \\ -3 & 1 \end{pmatrix}^T = \begin{pmatrix} 1 & 7 & -3 \\ 5 & 2 & 1 \end{pmatrix}$$

Các phép toán

Cộng hai ma trận (addition)

Tổng của hai ma trận $A = (a_{ij})_{m\times n}$ và $B = (b_{ij})_{m\times n}$ cùng kích thước là một ma trận $C = (c_{ij})_{m\times n}$, trong đó các phần tử được xác định bởi:

$$c_{ij} = a_{ij} + b_{ij} \qquad \text{với} \quad i = 1, 2, \ldots, m;\ j = 1, 2, \ldots, n$$

Kí hiệu tổng A và B là $A + B$.

Ví dụ: Tính tổng

$$\begin{pmatrix} 2 & 1 \\ -3 & 5 \\ 4 & 3 \end{pmatrix} + \begin{pmatrix} 7 & -1 \\ 4 & 2 \\ -6 & 1 \end{pmatrix} = \begin{pmatrix} 2+7 & 1+(-1) \\ -3+4 & 5+2 \\ 4+(-6) & 3+1 \end{pmatrix} = \begin{pmatrix} 9 & 0 \\ 1 & 7 \\ -2 & 4 \end{pmatrix}$$

Các phép toán

Nhân một số với ma trận (scalar multiplication)

Tích của một số λ với ma trận $A = (a_{ij})_{m\times n}$ là một ma trận $C = (c_{ij})_{m\times n}$, trong đó các phần tử được xác định bởi:

$$c_{ij} = \lambda\, a_{ij} \qquad \text{với} \quad i = 1, 2, \ldots, m;\ j = 1, 2, \ldots, n$$

Kí hiệu tích là λA.

Ví dụ: Tính tích

$$-2\begin{pmatrix} 1 & 6 \\ 2 & 5 \\ 9 & 3 \end{pmatrix} = \begin{pmatrix} (-2).1 & (-2).6 \\ (-2).2 & (-2).5 \\ (-2).9 & (-2).3 \end{pmatrix} = \begin{pmatrix} -2 & -12 \\ -4 & -10 \\ -18 & -6 \end{pmatrix}$$

Các phép toán

Nhân ma trận với ma trận (matrix multiplication)

Tích của hai ma trận $A = (a_{ij})_{m\times n}$ và $B = (b_{ij})_{n\times p}$ là một ma trận $C = (c_{ij})_{m\times p}$ được xác định bởi:

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj}$$

với $i = 1, 2, \ldots, m;\ j = 1, 2, \ldots, p$.

Lũy thừa của ma trận (power of a matrix)

$$A^k = \underbrace{A \cdots A}_{k \text{ lần}}$$

Các phép toán

Ví dụ:

$$\begin{pmatrix} 2 & -5 & 0 \\ -1 & 3 & -4 \\ 6 & -8 & -7 \\ -3 & 0 & 9 \end{pmatrix} \cdot \begin{pmatrix} 4 & -6 \\ 7 & 1 \\ 3 & 2 \end{pmatrix} = \begin{pmatrix} 2\cdot 4+(-5)\cdot 7+0\cdot 3 & 2\cdot(-6)+(-5)\cdot 1+0\cdot 2 \\ -1\cdot 4+3\cdot 7+(-4)\cdot 3 & -1\cdot(-6)+3\cdot 1+(-4)\cdot 2 \\ 6\cdot 4+(-8)\cdot 7+(-7)\cdot 3 & 6\cdot(-6)+(-8)\cdot 1+(-7)\cdot 2 \\ -3\cdot 4+0\cdot 7+9\cdot 3 & -3\cdot(-6)+0\cdot 1+9\cdot 2 \end{pmatrix}$$

$$= \begin{pmatrix} -27 & -17 \\ 5 & 1 \\ -53 & -58 \\ 15 & 36 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 4 & 5 \end{pmatrix}^3 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 4 & 5 \end{pmatrix}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 4 & 5 \end{pmatrix}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 4 & 5 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \\ 1 & 4 & 5 \end{pmatrix}\begin{pmatrix} 3 & 7 & 9 \\ 6 & 17 & 22 \\ 10 & 29 & 38 \end{pmatrix} = \begin{pmatrix} 19 & 53 & 69 \\ 45 & 128 & 167 \\ 77 & 220 & 287 \end{pmatrix}$$

Các phép toán

Tính chất: A, B, C là các ma trận, α, β là các số

- $A + B = B + A$
- $(A + B) + C = A + (B + C)$
- $A + \theta = \theta + A = A$
- $(A + B)^T = A^T + B^T$
- $\alpha(\beta A) = (\alpha\beta)A$
- $(\alpha + \beta)A = \alpha A + \beta A$
- $\alpha(A + B) = \alpha A + \alpha B$
- $(\alpha A)^T = \alpha A^T$

- Nói chung, $AB \neq BA$
- $(AB)C = A(BC)$
- $A(B + C) = AB + AC$
- $(B + C)A = BA + CA$
- $\alpha(AB) = (\alpha A)B = A(\alpha B)$
- $I_m A = A I_n = A$
- $(AB)^T = B^T A^T$

Định thức (determinant)

Định thức của một ma trận A vuông cấp n là một số, kí hiệu $\det A$ hoặc $|A|$, được xác định như sau:

- Định thức cấp 2: $\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21}$
- Định thức cấp $n > 2$: $\det A = a_{11}A_{11} + a_{12}A_{12} + \cdots + a_{1n}A_{1n}$, trong đó $A_{ij} = (-1)^{i+j}M_{ij}$ là phần bù đại số (cofactor) của a_{ij}, với M_{ij} là định thức của ma trận thu được từ A sau khi xóa hàng i và cột j.

Ví dụ:

$$\begin{vmatrix} 1 & 4 & -2 \\ 3 & 2 & 1 \\ -6 & 0 & 3 \end{vmatrix} = 1(-1)^{1+1}\begin{vmatrix} 2 & 1 \\ 0 & 3 \end{vmatrix} + 4(-1)^{1+2}\begin{vmatrix} 3 & 1 \\ -6 & 3 \end{vmatrix} + (-2)(-1)^{1+3}\begin{vmatrix} 3 & 2 \\ -6 & 0 \end{vmatrix}$$

$$= 1(6-0) - 4(9+6) - 2(0+12) = -78$$

Hạng (rank) của ma trận

Định nghĩa

Hạng của ma trận A, kí hiệu bởi **rank**(A), là số chiều của không gian cột của ma trận A.

- **Ví dụ:**

$$A = \begin{bmatrix} 2 & 5 & -3 & -4 \\ 4 & 7 & -4 & -3 \\ 6 & 9 & -5 & 2 \\ 0 & -9 & 6 & 5 \end{bmatrix} \Rightarrow v_1 = \begin{bmatrix} 2 \\ 4 \\ 6 \\ 0 \end{bmatrix}, v_2 = \begin{bmatrix} 5 \\ 7 \\ 9 \\ -9 \end{bmatrix}, v_3 = \begin{bmatrix} -3 \\ -4 \\ -5 \\ 6 \end{bmatrix}, v_4 = \begin{bmatrix} -4 \\ -3 \\ 2 \\ 5 \end{bmatrix}$$

$$\Rightarrow \text{rank}(A) = \dim(\text{span}(v_1, v_2, v_3, v_4))$$

- Để tính hạng của ma trận A, ta có thể sử dụng phương pháp khử Gauss (Gaussian Elimination) để đưa A về dạng bậc thang. Khi đó $\text{rank}(A)$ = số hàng khác 0 của ma trận bậc thang.

Hạng của ma trận

Ma trận dạng bậc thang (echelon form)

Ma trận được gọi là có dạng bậc thang nếu có các tính chất sau đây:

- Các hàng khác 0 nằm trên các hàng bằng 0.
- Phần tử khác 0 đầu tiên của một hàng lệch về bên phải so với phần tử khác 0 của các hàng nằm trên nó.

Ví dụ: các ma trận sau có dạng bậc thang

$$\begin{bmatrix} \textcircled{2} & -3 & 2 & 1 \\ 0 & \textcircled{1} & -4 & 8 \\ 0 & 0 & 0 & \textcircled{5} \end{bmatrix} ; \quad \begin{bmatrix} \textcircled{1} & 0 & 0 & 29 \\ 0 & \textcircled{1} & 0 & 16 \\ 0 & 0 & \textcircled{1} & 3 \end{bmatrix}$$

Hạng của ma trận

Phương pháp khử Gauss:

$$\begin{pmatrix} \mathbf{2} & 5 & -3 & -4 \\ 4 & 7 & -4 & -3 \\ 6 & 9 & -5 & 2 \\ 0 & -9 & 6 & 5 \end{pmatrix} \cdot(-2) \overset{?}{\sim} R_2 - 2\cdot R_1 \to R_2 \begin{pmatrix} \mathbf{2} & 5 & -3 & -4 \\ 0 & -3 & 2 & 5 \\ 6 & 9 & -5 & 2 \\ 0 & -9 & 6 & 5 \end{pmatrix} \cdot(-3) \overset{?}{\sim} R_3 - 3\cdot R_1 \to R_3$$

$$\begin{pmatrix} 2 & 5 & -3 & -4 \\ 0 & \mathbf{-3} & 2 & 5 \\ 0 & -6 & 4 & 14 \\ 0 & -9 & 6 & 5 \end{pmatrix} \cdot(-2) \overset{?}{\sim} R_3 - 2\cdot R_2 \to R_3 \begin{pmatrix} 2 & 5 & -3 & -4 \\ 0 & \mathbf{-3} & 2 & 5 \\ 0 & 0 & 0 & 4 \\ 0 & -9 & 6 & 5 \end{pmatrix} \cdot(-3) \overset{?}{\sim} R_4 - 3\cdot R_2 \to R_4$$

$$\begin{pmatrix} 2 & 5 & -3 & -4 \\ 0 & -3 & 2 & 5 \\ 0 & 0 & 0 & \mathbf{4} \\ 0 & 0 & 0 & -10 \end{pmatrix} \cdot\left(\frac{5}{2}\right) \overset{?}{\sim} R_4 - \left(\frac{-5}{2}\right)\cdot R_3 \to R_4 \begin{pmatrix} 2 & 5 & -3 & -4 \\ 0 & -3 & 2 & 5 \\ 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$rank\begin{pmatrix} 2 & 5 & -3 & -4 \\ 4 & 7 & -4 & -3 \\ 6 & 9 & -5 & 2 \\ 0 & -9 & 6 & 5 \end{pmatrix} = rank\begin{pmatrix} 2 & 5 & -3 & -4 \\ 0 & -3 & 2 & 5 \\ 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 \end{pmatrix} = 3$$

Khởi tạo ma trận

- Tạo ma trận bằng mảng ndarray:

```
[1]: import numpy as np
a=np.array([[1,-2,1],[2,4,-3]])
b=np.arange(3)
print("a = ",a)
print("b = ",b)
```

```
a =  [[ 1 -2  1]
 [ 2  4 -3]]
b =  [0 1 2]
```

- Ghép nối, xóa mảng:

```
In [2]: c=np.vstack([a,b])
d=np.delete(c,1,axis=0)
e=np.delete(c,1,axis=1)
print("Ghép ma trận",c,sep="\n")
print("Xóa hàng",d,sep="\n")
print("Xóa cột",e,sep="\n")
```

```
Ghép ma trận
[[ 1 -2  1]
 [ 2  4 -3]
 [ 0  1  2]]
Xóa hàng
[[ 1 -2  1]
 [ 0  1  2]]
Xóa cột
[[ 1  1]
 [ 2 -3]
 [ 0  2]]
```

Khởi tạo ma trận

Tạo ma trận bằng hàm tích hợp sẵn trong NumPy:

- **np.empty**(*shape*, *dtype*, *order*): trả về một ma trận mà các phần tử chưa được khởi tạo,

  shape : kích thước của ma trận,
  dtype : (optional) kiểu dữ liệu phần tử,
  order : (optional) 'C' hoặc 'F'.

  Giá trị của các phần tử của ma trận là các số sẵn có trong các ô nhớ được cấp phát.
- **np.zeros**(*shape*): trả về ma trận không.
- **np.ones**(*shape*) : trả về ma trận có phần tử bằng 1.

Khởi tạo ma trận

- **np.eye**(n, M, k, *dtype*): trả về một ma trận mà các phần tử thuộc đường chéo bằng 1, các phần tử còn lại bằng 0,

  n : số hàng của ma trận,
  M : số cột của ma trận, giá trị mặc định bằng n,
  k : chỉ số đường chéo,
  dtype : kiểu dữ liệu phần tử.

- **np.random.rand**(*shape*): trả về ma trận mà giá trị phần tử được sinh ngẫu nhiên trong khoảng [0,1).

Một số ma trận đặc biệt

- **np.identity**(n): trả về ma trận đơn vị cấp n.

```
In [17]: np.identity(3)

Out[17]: array([[1., 0., 0.],
                [0., 1., 0.],
                [0., 0., 1.]])
```

- **np.diag**(v): trả về một ma trận đường chéo, trong đó v là một danh sách hoặc một kiểu tương đương.

```
In [18]: a=np.random.randint(-5,5,3)
         np.diag(a)

Out[18]: array([[ 3,  0,  0],
                [ 0, -3,  0],
                [ 0,  0,  3]])
```

Một số ma trận đặc biệt

- **np.tril**(A,k): tạo ra ma trận tam giác dưới.
- **np.triu**(A,k): tạo ra ma trận tam giác trên.

A : mảng 2 chiều hoặc tương đương,

k : (optional) chỉ số đường chéo, mặc định $k = 0$.

```
In [19]: a=np.random.randint(-5,5,9).reshape(3,3)
         print(a)
         print(np.tril(a))
         print(np.triu(a,1))
```

```
[[ 3 -2  4]
 [-4 -2  4]
 [-2  3  3]]

[[ 3  0  0]
 [-4 -2  0]
 [-2  3  3]]

[[ 0 -2  4]
 [ 0  0  4]
 [ 0  0  0]]
```

Các phép toán với ma trận

- Cộng hai ma trận:

```
In [6]: a=np.array([[-1,4,2],[3,1,-5]])
        b=np.arange(6).reshape(2,3)
        print(a," +",b," =",a+b,sep="\n")
```

```
[[-1  4  2]
 [ 3  1 -5]]
 +
[[0 1 2]
 [3 4 5]]
 =
[[-1  5  4]
 [ 6  5  0]]
```

- Nhân một số với ma trận:

```
In [7]: a=np.array([[-1,4,2],[3,1,-5]])
        -2*a
```

```
Out[7]: array([[ 2, -8, -4],
               [-6, -2, 10]])
```

Các phép toán với ma trận

- Nhân ma trận với ma trận: hàm **np.matmul**()

```
In [8]: a=np.array([[-1,4,2],[3,1,-5]])
        b=np.random.randint(-10,10,(3,2))
        print(b)
        print("ab=",np.matmul(a,b),sep="\n")
        print("ba=",np.matmul(b,a),sep="\n")
```

```
[[-4  4]
 [ 3  6]
 [-1  1]]
ab=
[[14 22]
 [-4 13]]
ba=
[[ 16 -12 -28]
 [ 15  18 -24]
 [  4  -3  -7]]
```

Các phép toán với ma trận

- Nhân ma trận với ma trận: hàm **np.dot**()

```
In [9]: a=np.array([[-1,4,2],[3,1,-5]])
        b=np.random.randint(-10,10,(3,2))
        print(b)
        print("ab=",np.dot(a,b),sep="\n")
```

```
[[-9  8]
 [-7 -3]
 [ 6 -3]]
ab=
[[ -7 -26]
 [-64  36]]
```

Các phép toán với ma trận

Chú ý: Phép toán $A * B$ sẽ được thực hiện bằng cách nhân các phần tử của ma trận A với phần tử của ma trận B ở vị trí tương ứng. Do đó tích $A * B$ không phải là phép nhân ma trận với ma trận như được định nghĩa trong đại số ma trận.

```
In [10]: a=np.array([[3,2],[-4,1]])
         b=np.arange(4).reshape(2,2)
         print(a,"*",b,"=",a*b,sep="\n")
```

```
[[ 3  2]
 [-4  1]]
*
[[0 1]
 [2 3]]
=
[[ 0  2]
 [-8  3]]
```

Các phép toán với ma trận

- Lũy thừa ma trận: hàm **np.linalg.matrix_power()**

```
In [11]: x=np.arange(9).reshape(3,3)
         print(np.matmul(x,x))
         np.linalg.matrix_power(x,2)

[[ 15  18  21]
 [ 42  54  66]
 [ 69  90 111]]

Out[11]: array([[ 15,  18,  21],
                [ 42,  54,  66],
                [ 69,  90, 111]])
```

Các phép toán với ma trận

- Chuyển vị ma trận: hàm **np.transpose**()

```
In [12]: a=np.array([[-1,4,2],[3,1,-5]])
         np.transpose(a)

Out[12]: array([[-1,  3],
                [ 4,  1],
                [ 2, -5]])
```

```
In [13]: a.T

Out[13]: array([[-1,  3],
                [ 4,  1],
                [ 2, -5]])
```

- Tính "vết" của ma trận vuông: hàm **np.trace**()

```
In [14]: A=np.random.randint(-5,5,(3,3))
         print(A)
         print("trace(A) =",np.trace(A))

[[ 1  4 -5]
 [ 4 -1 -3]
 [ 2 -5 -1]]
trace(A) = -1
```

Các phép toán với ma trận

- Định thức của ma trận: hàm **np.linalg.det()**

```
In [15]: a=np.arange(9).reshape(3,3)
         print(a)
         print("det(a)=",np.linalg.det(a))
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
det(a)= 0.0
```

- Hạng của ma trận: hàm **np.linalg.matrix_rank()**

```
In [16]: a=np.array([[-3,1,1,1],[1,-3,1,1],[1,1,-3,1],[1,1,1,-3]])
         print(a)
         print("rank(a) =",np.linalg.matrix_rank(a))
```

```
[[-3  1  1  1]
 [ 1 -3  1  1]
 [ 1  1 -3  1]
 [ 1  1  1 -3]]
rank(a) = 3
```

VINBIGDATA VINGROUP Academy Vietnam

Hệ thống gợi ý

- Hệ thống gợi ý (Recommender System - RS) là một dạng hệ thống lọc thông tin tìm kiếm dự báo "*đánh giá*" hoặc "*sở thích*" của người dùng (user) với một sản phẩm hoặc đối tượng nào đó (item).
- Các hệ thống gợi ý được ứng dụng trong nhiều lĩnh vực:
  1. Netflix, YouTube và Spotify sẽ nhận diện sở thích của người dùng và gợi ý danh sách phát nhạc, dựa trên mức độ tương tác của người dùng với một bản nhạc nào đó.
  2. Amazon đưa ra gợi ý mua sắm dựa trên tần suất tìm kiếm sản phẩm và lịch sử mua hàng của người dùng.
  3. Facebook, Zalo đưa ra gợi ý kết bạn dựa trên các mối liên kết chung.
- Các hệ thống gợi ý được chia thành 2 nhóm chính:
  1. *Content based system*: đánh giá các thuộc tính của các *item* mà *user* "ưa thích", từ đó gợi ý các *item* có chung thuộc tính đó.
  2. *Collaborative filtering*: dựa trên sự tương quan giữa các *user* và/hoặc *item*, một nhóm *item* được gợi ý tới một *user* dựa trên các *user* có hành vi tương tự.

VINBIGDATA VINGROUP Academy Vietnam

Ma trận utility

- Trong một hệ thống gợi ý, người ta quan tâm đến 3 thông tin chính là *item*, *user*, và *rating* (đánh giá) của *user* về *item*. Giá trị *rating* phản ánh mức độ quan tâm của *user* đối với *item*.
- Ma trận utility: biểu diễn các thông tin về *item*, *user*, và *rating*.

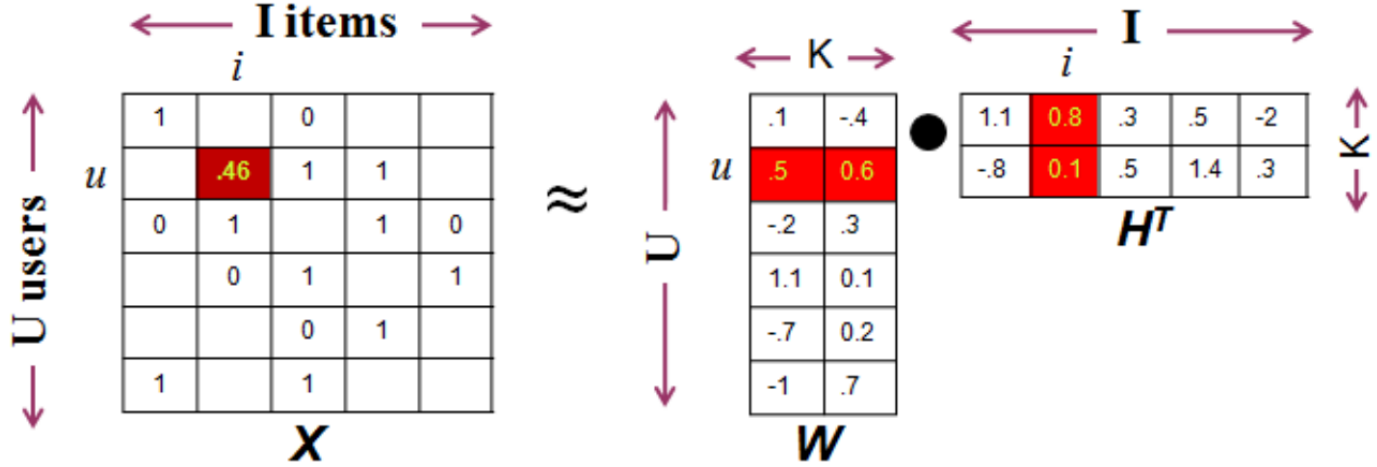
	item 1	*item* 2	*item* 3	*item* 4	*item* 5	*item* 6
user 1	4	5	1	?	2	1
user 2	?	4	?	3	?	2
user 3	3	?	1	?	3	?
user 4	2	?	?	4	?	1
user 5	5	?	3	4	3	?

- Các ô màu xanh không có đánh giá của *user* về *item*. Hệ thống gợi ý cần phải tự điền các giá trị này.

Bài toán phân tích ma trận

Kĩ thuật phân tích ma trận (Matrix Factorization - MF) có thể được sử dụng để điền các giá trị còn thiếu vào ma trận utility:

- *Phân tích ma trận utility* X *thành hai ma trận có kích thước nhỏ hơn là* W *và* H *sao cho tích ma trận* $W H^T$ *xấp xỉ* X *với độ chính xác cao:*



- *Ma trận* W *mô tả* K *thuộc tính ẩn (latent factor) của các user và ma trận* H *mô tả* K *thuộc tính ẩn của các item.*

Hàm mất mát (loss function)

- Kí hiệu $R = (r_{ui})$ là ma trận mô tả việc *user* đánh giá *item* hay không, $r_{ui} = 1$ nghĩa là *user u* có đánh giá *item i*, $r_{ui} = 0$ nếu ngược lại.
- Hai ma trận W và H được tìm bằng cách cực tiểu hóa hàm mất mát. Chẳng hạn, một dạng đơn giản của hàm mất mát như sau:

$$\mathcal{L}(W, H) = \sum_{(u,i):r_{ui}=1} \left(x_{ui} - \sum_{j=1}^{K} w_{uj} h_{ij} \right)^2$$

Chẳng hạn

$$\begin{pmatrix} 0.1 & -(0.4) \\ 0.5 & 0.6 \\ -(0.2) & 0.3 \\ 1.1 & 0.1 \\ -(0.7) & 0.2 \\ -1 & 0.7 \end{pmatrix} \cdot \begin{pmatrix} 1.1 & 0.8 & 0.3 & 0.5 & -2 \\ -(0.8) & 0.1 & 0.5 & 1.4 & 0.3 \end{pmatrix} = \begin{pmatrix} \boxed{0.43} & 0.04 & \boxed{-0.17} & -0.51 & -0.32 \\ 0.07 & 0.46 & \boxed{0.45} & \boxed{1.1} & -0.82 \\ \boxed{-0.46} & \boxed{-0.13} & 0.09 & \boxed{0.32} & \boxed{0.49} \\ 1.1 & \boxed{0.89} & \boxed{0.38} & 0.69 & \boxed{-2.2} \\ -0.93 & -0.54 & \boxed{-0.11} & \boxed{-0.07} & 1.5 \\ \boxed{-1.7} & -0.73 & \boxed{0.05} & 0.48 & 2.2 \end{pmatrix}$$

$$\mathcal{L}(W, H) = (1 - 0.43)^2 + (0 + 0.17)^2 + \cdots = 23.6233$$

Hiệu chỉnh hàm mất mát (Regularization)

Để ngăn chặn hiện tượng quá khớp (overfitting), người ta thay đổi hàm mất mát bằng cách thêm vào một đại lượng gọi là hiệu chỉnh (regularization) để điều khiển độ lớn của các giá trị trong W và H:

$$\mathcal{L}(\mathrm{W},\mathrm{H}) = \sum_{(u,i):r_{ui}=1} \left(x_{ui} - \sum_{j=1}^{K} w_{uj} h_{ij} \right)^2 + \lambda \left(\|\mathrm{W}\|_F^2 + \|\mathrm{H}\|_F^2 \right)$$

trong đó λ là hệ số chuẩn hóa và $\|A\|_F$ là chuẩn Frobenius của ma trận A,

$$\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2}$$

Tìm nghiệm tối ưu

Để tìm nghiệm tối ưu cho W và H, ta lần lượt tối ưu một ma trận trong khi cố định ma trận còn lại cho tới khi hội tụ.

- Khi cố định W, ta tối ưu hóa H với hàm mất mát:

$$\mathcal{L}(\mathrm{H}) = \sum_{(u,i):r_{ui}=1} \left(x_{ui} - \sum_{j=1}^{K} w_{uj} h_{ij} \right)^2 + \lambda \|\mathrm{H}\|_F^2$$

- Khi cố định H, ta tối ưu hóa W với hàm mất mát:

$$\mathcal{L}(\mathrm{W}) = \sum_{(u,i):r_{ui}=1} \left(x_{ui} - \sum_{j=1}^{K} w_{uj} h_{ij} \right)^2 + \lambda \|\mathrm{W}\|_F^2$$

Tìm nghiệm tối ưu

Sử dụng thuật toán tối ưu **Gradient descent**, ta thu được công thức cập nhật các phần tử của hai ma trận W và H từ hệ phương trình như sau:

$$w_{uk}^{new} = w_{uk} + \beta \left[2 \left(x_{ui} - \sum_{j=1}^{K} w_{uj} h_{ij} \right) h_{ik} - \lambda w_{uk} \right]$$

$$h_{ik}^{new} = h_{ik} + \beta \left[2 \left(x_{ui} - \sum_{j=1}^{K} w_{uj} h_{ij} \right) w_{uk} - \lambda h_{ik} \right]$$

trong đó β là tốc độ học ($0 < \beta < 1$). Lúc ban đầu, hai ma trận W và H sẽ được khởi tạo ngẫu nhiên.

Phân tích ma trận - Ví dụ giải số

Bài toán: Phân tích ma trận utility

$$X = \begin{bmatrix} 5 & 3 & 0 & 1 \\ 4 & 0 & 0 & 1 \\ 1 & 1 & 0 & 5 \\ 1 & 0 & 0 & 4 \\ 0 & 1 & 5 & 4 \end{bmatrix}$$

thành tích $W H^T$, với $W \in \mathbb{R}^{5\times 2}$, $H \in \mathbb{R}^{4\times 2}$. Sử dụng hàm mất mát dạng:

$$\mathcal{L}(W, H) = \sum_{(u,i):x_{ui}\neq 0} \left(x_{ui} - \sum_{j=1}^{K} w_{uj} h_{ij} \right)^2$$

Ví dụ giải số

Xây dựng hàm thực hiện phân tích ma trận:

```
import numpy as np
```

```
"""
@INPUT:
    A     : a matrix to be factorized, dimension M x N
    W     : an initial matrix of dimension M x K
    H     : an initial matrix of dimension N x K
    K     : the number of latent features
    steps : the maximum number of steps to perform the optimisation
    beta  : the learning rate
@OUTPUT:
    the final matrices W and H
"""
```

Ví dụ giải số

```
def matrix_factor(A, W, H, K, steps=5000, beta=0.0002):
    H = H.T
    for step in range(steps):
        for i in range(len(A)):
            for j in range(len(A[i])):
                if A[i][j] > 0:
                    eij = A[i][j] - np.dot(W[i,:],H[:,j])
                    for k in range(K):
                        W[i][k] = W[i][k] + beta*(2*eij*H[k][j])
                        H[k][j] = H[k][j] + beta*(2*eij*W[i][k])
        e = 0
        for i in range(len(A)):
            for j in range(len(A[i])):
                if A[i][j] > 0:
                    e = e + pow(A[i][j]-np.dot(W[i,:],H[:,j]), 2)
        if e < 0.0001:
            break
    return W, H.T
```

Ví dụ giải số

```
# Khởi tạo ma trận
A = np.array([[5,3,0,1],
              [4,0,0,1],
              [1,1,0,5],
              [1,0,0,4],
              [0,1,5,4]])

M = len(A)
N = len(A[0])
K = 2

W = np.random.rand(M,K)
H = np.random.rand(N,K)
```

Ví dụ giải số

```
nW, nH = matrix_factor(A, W, H, K)
print(nW)
print(nH)
np.dot(nW,nH.T)
```

```
[[0.6543035  2.30374006]
 [0.60303339 1.80998947]
 [2.11695809 0.24661478]
 [1.69575326 0.23855388]
 [1.77211928 0.7276636 ]]
[[ 0.26725979  2.10564034]
 [ 0.22926983  1.20501075]
 [ 2.06948627  1.70549746]
 [ 2.38631769 -0.24609887]]
```

```
A = np.array([[5,3,0,1],
              [4,0,0,1],
              [1,1,0,5],
              [1,0,0,4],
              [0,1,5,4]])
```

```
array([[5.02571703, 2.92604359, 5.28309495, 0.99442821],
       [3.97235342, 2.31931413, 4.33490177, 0.99359289],
       [1.0850598 , 0.78252809, 4.8016166 , 4.99104293],
       [0.95551534, 0.67624506, 3.91619114, 3.98789817],
       [2.00581405, 1.28313595, 4.90840494, 4.0497624 ]])
```

Tài liệu tham khảo

1. Charu C. Aggarwal; Linear Algebra and Optimization for Machine Learning, Springer, 2020.
2. Stephen Boyd, Lieven Vandenberghe; Introduction to Applied Linear Algebra: Vectors, Matrices, and Least Squares, Cambridge University Press, 2018.
3. David C. Lay, Steven R. Lay, Judi J. McDonald; Linear Algebra and Its Applications, Fifth edition, Pearson, 2016
4. Tom Lyche; Numerical Linear Algebra and Matrix Factorizations, Springer, 2020.
5. Gilbert Strang; Linear Algebra and Learning from Data, Wellesley- Cambridge Press, 2019.