




Maestría en Inteligencia Artificial Aplicada (MNA)

Matriz Dispersa : Sparse Matrix

Inteligencia Artificial y Aprendizaje Automático

Luis Eduardo Falcón Morales



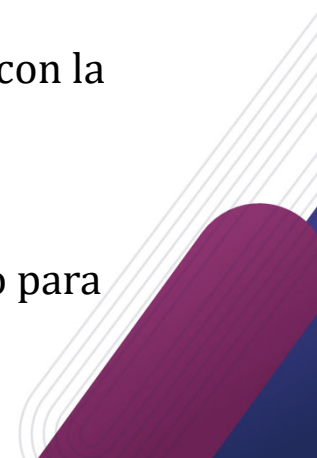
Una **matriz dispersa** (*sparse matrix*) es aquella en la que la mayoría de sus entradas son cero.


Una **matriz densa** (*dense matrix*) es aquella en la que la mayoría de sus entradas son diferentes de cero.

Al cociente del total de entradas cero de una matriz, dividida por el total de entradas de dicha matriz se le llama **dispersión** de la matriz (*sparsity*).


El problema con las matrices dispersas es que pueden llegar a ser matrices muy grandes con la mayoría de sus entradas iguales a cero, es decir, haciendo uso de recursos de espacio de memoria y tiempo para guardar o procesar solamente valores u operaciones de ceros.

Así, la importancia de estas matrices radicarán en poder tomar ventaja de sus entradas cero para optimizar su uso y recursos en su manipulación y operaciones.





Existen muchos tipos de problemas de donde pueden surgir las matrices dispersas:

- Catálogo de compras de productos por parte de clientes.
 - Listado de las recomendaciones dada por usuarios a películas/productos.
 - Listado de las canciones escuchadas por los usuarios de Spotify.
 - Conteo de frecuencias de las palabras que aparecen en un documento/diccionario.
 - Páginas web conectadas entre sí mediante algún hipervínculo.
 - ...
- 




Los problemas anteriores nos llevan a obtener matrices dispersas y de ahí a llevar a cabo operaciones matriciales con ellas.

En particular, se deseará resolver sistemas de ecuaciones de la forma $AX = b$, donde la matriz A es dispersa.

De acuerdo a la forma en que se multiplican las matrices, existirían muchas operaciones de multiplicaciones de ceros y suma de ceros, los cuales obviamente no aportan nada a la solución del sistema.

Así, una representación de las matrices dispersas requiere además que todas las operaciones matriciales se definan nuevamente para estos casos. Existen librerías que ya integran todas estas operaciones. De hecho en la mayoría de los casos todo este tratamiento como matrices dispersas es transparente para nosotros y no nos damos cuenta de que así se están llevando a cabo estas operaciones.



En ocasiones, se dice que una matriz es dispersa cuando su cantidad de elementos no cero es aproximadamente igual a su cantidad de renglones o columnas. Así, por ejemplo, en una matriz A de tamaño 1000×1000 , considerando este criterio tendría en este caso un valor de dispersión de 99.9% aproximadamente: $1000/(1000 \times 1000)$.

Veamos con un ejemplo cómo se reflejaría esta diferencia en memoria:

$A_{1000 \times 1000}$

Sea una matriz dispersa.

Supongamos que su valor de dispersión es del 99.9%.
Y que el tipo de dato no cero es de 8 bytes.

Memoria requerida en este caso: 8,000 bytes = 0.008 MB

$B_{1000 \times 1000}$

Sea una matriz densa.

Supongamos que el tipo de dato no cero es de 8 bytes.

Memoria en este caso:
8'000,000 bytes = 8 MB



Existen diferentes formatos para guardar la información de una matriz dispersa.

Veamos algunos de los principales utilizados en el área de aprendizaje automático:

- COO: Coordinate format : Formato de Coordenadas – triplet format ijk
- CSR: Compressed Sparse Row : Comprimido por filas
- CSC: Compressed Sparse Column : Comprimido por columnas
- Existen otros tipos de matrices cuyos valores de cero tienen cierta distribución particular, como la matriz por bloques, por bandas, la diagonal, entre muchas otras.

Por el momento solo estudiaremos las matrices dispersas. Existen librerías especializadas en el manejo y tratamiento de estas matrices.

En particular SciPy es de las más completas:

<https://scipy.org/>

<https://docs.scipy.org/doc/scipy/tutorial/sparse.html>



Sparse Matrix → Simple-Triplet-Matrix : Caso 1: COO

```
import numpy as np
from scipy import sparse
```

```
A = np.array([[1, 0, 0, 3, 0, 0], [0, 0, 7, 0, 0, 0],
              [0, -1, 0, 0, 0, 9], [2, 0, 0, 8, 0, 0]], dtype=np.int64)
```

```
A
array([[ 1,  0,  0,  3,  0,  0],
       [ 0,  0,  7,  0,  0,  0],
       [ 0, -1,  0,  0,  0,  9],
       [ 2,  0,  0,  8,  0,  0]])
```

Sparsity=17/24≈ 0.71

COO : Coordinate format

Formato para
matrices COO:

```
print(D.row)
print(D.col)
print(D.data)
```

```
renglón: [0 0 1 2 2 3 3]
columna: [0 3 2 1 5 0 3]
valor:   [ 1  3  7 -1  9  2  8]
```

SciPy nos proporciona esta
salida en una matriz COO:

```
D = sparse.coo_matrix(A)
print(D)
```

(0, 0)	1
(0, 3)	3
(1, 2)	7
(2, 1)	-1
(2, 5)	9
(3, 0)	2
(3, 3)	8

En general, podemos decir que
en una COO sus entradas se van
describiendo por su posición
renglón-columna en la matriz.

Sparse Matrix → Simple-Triplet-Matrix : Caso 2: CSR

```
import numpy as np
from scipy import sparse
```

```
A = np.array([[1, 0, 0, 3, 0, 0], [0, 0, 7, 0, 0, 0],
              [0, -1, 0, 0, 0, 9], [2, 0, 0, 8, 0, 0]], dtype=np.int64)
```

```
A
array([[ 1,  0,  0,  3,  0,  0],
       [ 0,  0,  7,  0,  0,  0],
       [ 0, -1,  0,  0,  0,  9],
       [ 2,  0,  0,  8,  0,  0]])
```

Sparsity=17/24≈ 0.71

También podemos obtener la posición (renglón, columna) de cada elemento no-cero de la matriz, mediante la siguiente instrucción:

```
B = sparse.csr_matrix(A)
print(B)
```

```
(0, 0)      1
(0, 3)      3
(1, 2)      7
(2, 1)     -1
(2, 5)      9
(3, 0)      2
(3, 3)      8
```

CSR : Compressed Sparse Row

Formato para matrices CSR:

Son los índices del primer elemento no cero de cada renglón.

índice ptr: [0 2 3 5 7]
 columna: [0 3 2 1 5 0 3]
 valor: [1 3 7 -1 9 2 8]

```
print(B.indptr)
print(B.indices)
print(B.data)
```

Observa que su usa una dimensión igual a $rows + 1$

En general, podemos decir que en una CSR sus entradas se van describiendo por renglón.

Ejemplo: Obtengamos la representación CSR de la siguiente matriz A:

$$A = \begin{bmatrix} 0 & 4 & 0 & 6 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 8 & 0 & 0 \end{bmatrix}$$

Para ello requerimos obtener los valores de los siguientes 3 vectores:

índice ptr : $[\dots]$

columna: $[\dots]$

valor: $[\dots]$

En CSR la inspección se realiza por renglón.

Iniciamos buscando en el primer renglón de la matriz A el valor del primer valor no cero, en este ejemplo es el 4 como se indica a continuación:


$$A = \begin{bmatrix} 0 & 4 & 0 & 6 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 8 & 0 & 0 \end{bmatrix}$$

índice ptr : $[\dots]$

columna: $[\dots]$

valor: $[\dots]$

$$A = \begin{bmatrix} 0 & \text{idx}(0) \circledast 4 & 0 & 6 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 8 & 0 & 0 \end{bmatrix}$$

Como es el primer valor no cero encontrado en A , su índice asociado será 0:

índice ptr: [0]
columna: [...]
valor: [...]

Y el cual se encuentra en la segunda columna con índice 1 y de valor 4:

índice ptr: [0]
columna: [1]
valor: [4]

NOTA: Recordemos que las columnas de A se indexan a partir de 0.

Ejemplo

Continuamos buscando, por renglón, el segundo dato no cero y al cual le asociaremos el índice 1:

$$A = \begin{bmatrix} 0 & 4 & 0 & \text{idx}(1) \circledast 6 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 8 & 0 & 0 \end{bmatrix}$$

Como encontramos el segundo dato no cero en el mismo renglón que el dato anterior, no registramos su índice (1) y solo registramos su columna (3) y valor (6), es decir, ahora tendremos:

índice ptr: [0]
columna: [1 3]
valor: [4 6]

Continuamos buscando el siguiente valor no cero por renglón. De encontrarlo será el tercer valor y por lo tanto tendría índice 2. Observamos que en el primer renglón ya no hay más valores no cero, por lo que continuamos la búsqueda en el segundo renglón. Como cambiamos de renglón, esto implica que si encontramos este tercer dato no cero, deberemos registrar su índice 2 en el primer vector de índices. Y efectivamente, encontramos dicho valor no cero en este renglón. Registramos entonces su índice (2), columna (2) y valor (1):

$$\longrightarrow A = \begin{bmatrix} 0 & 4 & 0 & 6 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 8 & 0 & 0 \end{bmatrix}$$

índice ptr: [0 2]

columna: [1 3 2]

valor: [4 6 1]

Ejemplo

Continuamos la búsqueda por renglón del siguiente valor no cero: sería el cuarto y tendría índice 3. Lo encontramos en el mismo renglón: 9. Como está en el mismo renglón, no registramos su índice (3), pero sí su columna (5) y valor (9):

$$\longrightarrow A = \begin{bmatrix} 0 & 4 & 0 & 6 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 8 & 0 & 0 \end{bmatrix}$$

índice ptr: [0 2]

columna: [1 3 2 5]

valor: [4 6 1 9]

Continuemos con la búsqueda del siguiente valor no cero: sería el quinto y por lo tanto con índice 4.

Ya terminamos con el segundo renglón y la nueva búsqueda se lleva a cabo ahora en el tercer renglón. Esto implicará de nuevo que al encontrar el siguiente valor no cero, deberemos registrar su índice (4 en este caso).

$$\xrightarrow{A} \begin{bmatrix} 0 & 4 & 0 & 6 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 8 & 0 & 0 \end{bmatrix}$$

Sin embargo, vemos que en el tercer renglón no existen valores no cero. En estos casos lo único que registramos es el valor del índice que estamos buscando (4). Esto ayudará a identificar que hubo un renglón de ceros:

índice ptr: [0 2 4]

columna: [1 3 2 5]

valor: [4 6 1 9]

Continuamos nuestra búsqueda entonces en el último renglón de nuestra matriz A:

$$\xrightarrow{A} \begin{bmatrix} 0 & 4 & 0 & 6 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 8 & 0 & 0 \end{bmatrix}$$

idx(4)

Como es el primer valor no cero encontrado en ese renglón, registramos su índice (4), su columna (0) y valor (2):

índice ptr: [0 2 4 4]

columna: [1 3 2 5 0]

valor: [4 6 1 9 2]

Continuamos nuestra búsqueda de otro valor no cero en el último renglón y encontramos el 8:

$$A = \begin{bmatrix} 0 & 4 & 0 & 6 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 8 & 0 & 0 \end{bmatrix}$$

idx(5)

Como no es el primer valor no cero encontrado en dicho renglón, solamente registramos su columna (3) y valor (8):

índice ptr: [0 2 4 4]
 columna: [1 3 2 5 0 3]
 valor: [4 6 1 9 2 8]

Continuamos nuestra búsqueda de otro valor no cero en el último renglón. De encontrarlo sería el séptimo dato no cero y por lo tanto con índice 6. Sin embargo vemos que ya no hay más valores no cero, además de que ahí termina la matriz:

$$A = \begin{bmatrix} 0 & 4 & 0 & 6 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 8 & 0 & 0 \end{bmatrix}$$

Cuando terminamos la búsqueda en la matriz, registramos solamente el último índice del dato que estábamos buscando, en este caso el 6:

índice ptr: [0 2 4 4 6]
 columna: [1 3 2 5 0 3]
 valor: [4 6 1 9 2 8]

Esta es finalmente la representación CSR de la matriz A dada.

$$A = \begin{bmatrix} 0 & 4 & 0 & 6 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 8 & 0 & 0 \end{bmatrix}$$

Representación dispersa CSR de la matriz A:

índice ptr: [0 2 4 4 6]

columna: [1 3 2 5 0 3]

valor: [4 6 1 9 2 8]

idx(0)

idx(1)

idx(2)

idx(3)

idx(4)

idx(5)

Sparsity = $18/24 \approx 0.75$

En resumen:

Se incluye el valor del índice en el vector de “salto de renglón” cada vez que se cumple cualquiera de las siguientes condiciones:

- si el número encontrado es la primer entrada no-cero del renglón (leído de izquierda a derecha).
- si el renglón que se inspecciona no tiene entradas no-cero, se anota el índice que se está buscando en ese momento.
- cuando se termina de inspeccionar la matriz se anota el valor del índice que se estaba buscando en ese momento. Dicho valor será además igual al total de entradas no cero que tiene la matriz.

Ejemplo

Encontrar la matriz de dimensión 5×6 cuya representación simple-triplet-matrix en su formato CSR (compressed sparse row) está dada como sigue:

```
print(B.indptr)
print(B.indices)
print(B.data)
```

```
[0 2 2 2 4 5]
[0 3 1 4 3]
[ 5  9  7 -2  4]
```

Solución:

```
array([[ 5,  0,  0,  9,  0,  0],
       [ 0,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0],
       [ 0,  7,  0,  0, -2,  0],
       [ 0,  0,  0,  4,  0,  0]])
```

Sparsity= $5/30 \approx 0.83$

Sparse Matrix → Simple-Triplet-Matrix : Caso 3: CSC

```
import numpy as np
from scipy import sparse
```

```
A = np.array([[1, 0, 0, 3, 0, 0], [0, 0, 7, 0, 0, 0],
              [0, -1, 0, 0, 0, 9], [2, 0, 0, 8, 0, 0]], dtype=np.int64)
```

```
A
array([[ 1,  0,  0,  3,  0,  0],
       [ 0,  0,  7,  0,  0,  0],
       [ 0, -1,  0,  0,  0,  9],
       [ 2,  0,  0,  8,  0,  0]])
```

Sparsity=17/24 ≈ 0.71

También podemos obtener la posición de cada elemento no-cero de la matriz:

```
C = sparse.csc_matrix(A)
print(C)
```

(0, 0)	1
(3, 0)	2
(2, 1)	-1
(1, 2)	7
(0, 3)	3
(3, 3)	8
(2, 5)	9

CSC : Compressed Sparse Column

Formato para matrices CSC:

```
print(C.indptr)
print(C.indices)
print(C.data)
```

Observa que su usa una dimensión igual a $cols + 1$

índice ptr: [0 2 3 4 6 6 7]
 renglón: [0 3 2 1 0 3 2]
 valor: [1 2 -1 7 3 8 9]

En general, podemos decir que en una CSC sus entradas se van describiendo por columnas.



D.R.© Tecnológico de Monterrey, México, 2022.
Prohibida la reproducción total o parcial
de esta obra sin expresa autorización del
Tecnológico de Monterrey.