Documentación librería Numpy

Básicos de NumPy

NumPy es una librería fundamental para el cálculo numérico en Python. Su principal estructura de datos es el **array** (arreglo), que permite almacenar y manipular datos numéricos de forma eficiente. A diferencia de las listas nativas de Python, los arrays de NumPy son más rápidos, ocupan menos memoria y permiten operaciones vectorizadas.

```
"""## Básicos"""

z = np.array([1, 2, 3], dtype="int32")
print(z)

y = np.array([[2.0, 3.0, 5.0], [6.0, 5.0, 7.0]])
print(y)

# Dimensión del array
y.ndim

# Forma del array
y.shape

# Tipo de dato de los elementos
y.dtype

# Tamaño en bytes de cada elemento
y.itemsize

# Tamaño total en bytes del array
y.nbytes

# Número total de elementos del array
y.size

```

- np.array([...]): crea un array de NumPy. Se puede especificar el tipo de dato con dtype.
- z: es un array unidimensional de enteros de 32 bits.
- y: es un array bidimensional (matriz) de números flotantes.

Atributos útiles del array:

- ndim: devuelve el número de dimensiones del array (1 para z, 2 para y).
- shape: devuelve la forma del array como una tupla (filas, columnas), por ejemplo (2, 3) para y.
- dtype: indica el tipo de dato almacenado en el array (por ejemplo, float64).
- itemsize: indica el tamaño en bytes de cada elemento del array.
- nbytes: calcula el tamaño total del array en memoria (equivalente a itemsize * size).
- size: devuelve el número total de elementos que contiene el array.

Estos conceptos básicos permiten entender cómo están estructurados los datos y son la base para trabajar eficientemente con grandes volúmenes de información en ciencia de datos, machine learning o computación científica.

Acceder, cambiar elementos y filas en NumPy

NumPy permite acceder y modificar elementos de los arrays de forma muy eficiente utilizando **índices** y **slicing** (rebanado). Esto es esencial para manipular datos en estructuras multidimensionales.

```
z = np.array([[1,2,3,4,5,6,7],[8,9,10,11,12,13,14]])
print(z)

# Recuperar un valor específico dentro del array
print(z[1,5])

# Recuperar fila entera
print(z[0,:])

# Recuperar una columna entera
print(z[:,2])

# Recuperación con saltos (del índice 1 al penúltimo, saltando de 2 en 2)
print(z[0, 1:-1:2])

# Mostrar un valor antes de modificarlo
print(z[1,5])

# Modificar un valor específico
z[1,5] = 20
print(z[1,5])

# Mostrar una columna antes de modificarla
print(z[:,2])

# Modificar todos los valores de una columna
z[:,2] = [1,2]
print(z[:,2])

# Mostrar el array actualizado
z

```

Explicación del acceso y modificación:

- z[1,5]: accede al elemento que está en la fila 1 y columna 5 (índices empiezan en 0).
- z[0,:]: accede a **toda la fila 0**.
- z[:,2]: accede a **todos los elementos de la columna 2**.
- z[0, 1:-1:2]: accede a los elementos de la fila 0 desde la posición 1 hasta la antepenúltima, saltando de 2 en 2.
- z[1,5] = 20: cambia el valor en la posición fila 1, columna 5.
- z[:,2] = [1,2]: asigna nuevos valores a la columna 2, uno por fila.

Estos métodos permiten trabajar con porciones específicas del array sin necesidad de usar bucles, lo que hace el código más limpio y más rápido.

Creación de arrays en NumPy

NumPy proporciona múltiples formas de crear arrays, ya sea manualmente, con valores constantes, valores aleatorios o estructuras específicas como la matriz identidad. Esto permite generar datos de prueba o inicializar estructuras numéricas para cálculos.

```
"""## Ejemplo array de 3 dimensiones"""

z = np.array([[[1,2],[2,3]], [[3,4],[5,6]]])
print(z)

print(z[0,0,1])

## Cambiar un valor
print(z)
z[:,1,:] = [[9,9],[8,8]]
print(z)

"""## Crear arrays de distintos valores"""

# Array de todo 0's
print(np.zeros((2,3)))

# Array de todo 1's
print(np.ones((3,2,2), dtype='int32'))

# Array de cualquier número
print(np.full((2,2), 90))

# Array del mismo tamaño que otro, rellenado con un valor
z = np.array([[1,2,3,4,5,6,7],[8,9,10,11,12,13,14]])
print(np.full_like(z, 10))

# Valores decimales aleatorios
print(np.random.rand(4,2))

# Valores enteros aleatorios entre -8 y 8
print(np.random.randint(-8,8, size=(5,5)))

# Matriz identidad
print(np.identity(5))

# Repetir un array por filas
z = np.array([[1,2,3]])
y = np.repeat(z, 3, axis=0)
print(y)

```

Explicación:

- np.array(...): crea un array directamente desde listas anidadas.
- z[0,0,1]: accede a un valor específico en un array tridimensional (fila 0, subfila 0, elemento 1).
- z[:,1,:] = [[9,9],[8,8]]: modifica todas las filas en la segunda subfila.
- np.zeros((2,3)): array de 2x3 lleno de ceros.
- np.ones((3,2,2), dtype='int32'): array 3x2x2 lleno de unos, con tipo entero.
- np.full((2,2), 90): array 2x2 lleno del número 90.
- np.full_like(z, 10): array del mismo tamaño que z, relleno con 10.
- np.random.rand(4,2): array 4x2 con valores decimales aleatorios entre 0 y 1.
- np.random.randint(-8,8, size=(5,5)): array 5x5 con enteros aleatorios entre -8 y 7.
- np.identity(5): matriz identidad de tamaño 5x5.

- np.repeat(z, 3, axis=0): repite z 3 veces por filas.

Estas funciones cubren los casos más comunes de creación de arrays, muy útiles en procesamiento de datos, simulaciones, machine learning y otras aplicaciones numéricas.

Operaciones matemáticas básicas en NumPy

NumPy permite realizar operaciones matemáticas y algebraicas de manera eficiente sobre arrays y matrices. Las operaciones se aplican de forma vectorizada, lo que significa que se ejecutan sobre todos los elementos sin necesidad de usar bucles.

```
z = np.array([1,2,3,4])
print(z)

# Suma
print(z + 3)

# Resta
print(z - 2)

# Multiplicación
print(z * 2)

# División
print(z / 2)

# Suma de arrays
y = np.array([1,0,1,0])
print(z)
print(z + y)

# Resta de arrays
print(z - y)

# Elevar al cuadrado
print(z**2)

# Seno y coseno
print("Seno")
print(np.sin(z))
print("Coseno")
print(np.cos(z))

```

Explicación:

- Operaciones como +, -, *, / se aplican **elemento a elemento**.
- Se pueden combinar arrays del mismo tamaño directamente.
- También se pueden usar funciones matemáticas como np.sin() o np.cos() sobre arrays completos.

Álgebra lineal en NumPy

NumPy también permite realizar operaciones de álgebra lineal como producto escalar, producto de matrices, norma de un vector, resolver sistemas de ecuaciones, calcular determinantes y transpuestas.

```
"""## Álgebra"""

z = np.array([1,2,3])
y = np.array([4,5,6])

# Suma de vectores
print(z + y)

# Producto escalar
print(np.dot(z,y))

# Norma de un vector
print(np.linalg.norm(z))

A = np.array([[1,2],[3,4]])
B = np.array([[5,6],[7,8]])

# Multiplicación de matrices
print(np.dot(A,B))

# Sistema de ecuaciones lineales
A = np.array([[2,1],[1,3]])
b = np.array([8,18])

# Resolver sistema Ax = b
print(np.linalg.solve(A,b))

# Determinante de una matriz
A = np.array([[1,2],[3,4]])
print(np.linalg.det(A))

# Transpuesta de una matriz
A = np.array([[1,2,3],[4,5,6]])
A_T = A.T
print(A_T)

```

Explicación:

- np.dot(): realiza producto escalar o producto matricial.
- np.linalg.norm(): calcula la norma (longitud) de un vector.
- np.linalg.solve(A, b): resuelve un sistema lineal del tipo Ax = b.
- np.linalg.det(A): calcula el determinante de la matriz A.
- A.T: devuelve la transpuesta de una matriz.

Estas funciones son esenciales en aplicaciones de álgebra lineal, ingeniería, ciencia de datos y machine learning.

Estadísticas con NumPy

NumPy proporciona funciones para realizar análisis estadísticos básicos sobre arrays. Estas funciones son útiles para obtener medidas de tendencia central, dispersión y relaciones entre variables.

```
z = np.array([1,2,3,4,5])

print("Media:", np.mean(z))
print("Mediana: ", np.median(z))
print("Desviación estándar", np.std(z))
print("Var: ", np.var(z))

print("MIN: ", np.min(z))
print("MAX: ", np.max(z))

print("25%: ", np.percentile(z, 25))
print("75%: ", np.percentile(z, 75))

z = np.array([1,2,3,4,5])
y = np.array([5,4,3,2,1])

print("Matriz de correlación:", np.corrcoef(z,y))

```

Explicación:

- np.mean(z): calcula la **media** (promedio) del array z.
- np.median(z): obtiene la **mediana**, el valor central del array ordenado.
- np.std(z): calcula la **desviación estándar**, medida de dispersión.
- np.var(z): obtiene la **varianza**, el cuadrado de la desviación estándar.
- np.min(z): valor **mínimo** del array.
- np.max(z): valor **máximo** del array.
- np.percentile(z, 25): valor del **percentil 25** (cuartil inferior).
- np.percentile(z, 75): valor del **percentil 75** (cuartil superior).
- np.corrcoef(z, y): devuelve la **matriz de correlación** entre z y y, que mide la relación lineal entre ambos vectores.

Estas herramientas permiten realizar análisis descriptivos básicos, útiles en exploración de datos, ciencia de datos y análisis numérico.

Reorganizar arrays en NumPy

NumPy permite **reorganizar la forma** de los arrays sin cambiar los datos. Esto es útil cuando se necesita ajustar las dimensiones para operaciones específicas, concatenaciones o entradas de modelos de machine learning.

```
z = np.array([[1,2,3,4],[5,6,7,8]])

print(z)
print(z.shape)

y = z.reshape((4,2))
print(y)
print(y.shape)

y = z.reshape((8,1))
print(y)
print(y.shape)
```

```
13.
14. y = z.reshape((1,8))
15. print(y)
16. print(y.shape)
17.
18. # Concatenación vertical
19. z = np.array([1,2,3,4])
20. y = np.array([5,6,7,8])
21.
22. print(np.vstack([z,y, z, y]))
23.
24. # Concatenación horizontal
25. h1 = np.ones((2,4))
26. h2 = np.zeros((2,2))
27.
28. print(np.hstack((h1,h2)))
29.
```

Explicación:

- reshape((filas, columnas)): reorganiza la forma del array sin modificar sus valores.
    - De (2, 4) a (4, 2), (8, 1), o (1, 8) según el caso.
- np.vstack([...]): **concatena verticalmente** (por filas). Todos los arrays deben tener la misma cantidad de columnas.
- np.hstack(...): **concatena horizontalmente** (por columnas). Todos los arrays deben tener el mismo número de filas.
- np.ones((2,4)): crea un array 2x4 con todos los valores iguales a 1.
- np.zeros((2,2)): crea un array 2x2 con todos los valores iguales a 0.

Estas operaciones son fundamentales para preparar datos en tareas de análisis, transformación de estructuras y procesamiento previo a modelado.

Carga de ficheros con NumPy

NumPy permite leer datos desde archivos de texto de forma rápida y directa utilizando la función np.genfromtxt(). Esta función es útil cuando se trabaja con archivos CSV o similares, permitiendo transformar automáticamente el contenido en un array numérico.

```
1. array_datos = np.genfromtxt('datos.txt', delimiter=',')
2. array_datos = array_datos.astype('int32')
3. print(array_datos)
4.
```

Explicación:

- np.genfromtxt('datos.txt', delimiter=','): carga los datos del archivo datos.txt, separando los valores por comas (,) y convirtiéndolos en un array de NumPy. Cada fila del archivo será una fila en el array.
- .astype('int32'): convierte los valores del array al tipo entero de 32 bits. Esto es útil para asegurar que todos los datos sean del tipo deseado y optimizar el uso de memoria.

- print(array_datos): muestra por consola el contenido del array resultante.

Este enfoque es ideal para importar conjuntos de datos estructurados desde archivos de texto para análisis, limpieza o procesamiento posterior.

Máscara de booleanos e indexación avanzada en NumPy

NumPy permite seleccionar, filtrar y modificar elementos de un array utilizando condiciones booleanas y técnicas de indexación avanzada. Esto facilita trabajar con grandes volúmenes de datos de forma eficiente y sin necesidad de bucles explícitos.

```
datos = np.array([1,2,3,4,5,6,7,8,9,10])

mask = datos > 5
print("Mask", mask)

array_filtrada = datos[mask]
print(array_filtrada)

array_filtrada = datos[(datos > 5) & (datos < 9)]
print(array_filtrada)

datos[datos > 5] = 0
print(datos)

matriz = np.array([[10,20,30], [40,50,60], [70,80,90]])
print(matriz)

indices = [0,1,2]
columnas = [1,2,0]

print("Elementos seleccionados", matriz[indices,columnas])

print("Subselección: ", matriz[1:, [0,2]])

matriz[[0,1], [1,2]] = [999, 888]
print(matriz)

array_simple = np.array([10,15,20,25,30])

array_simple[array_simple > 20] = array_simple[array_simple > 20] ** 2
print(array_simple)

```

Explicación:

- datos > 5: crea una **máscara booleana**, es decir, un array del mismo tamaño con valores True o False según se cumpla la condición.
- datos[mask]: **filtra** los valores del array que cumplen la condición.
- Se pueden combinar condiciones con & (AND), | (OR), y ~ (NOT).
- datos[datos > 5] = 0: **modifica** directamente los elementos que cumplen la condición.

- matriz[indices, columnas]: usa **indexación avanzada** para seleccionar elementos específicos de una matriz, combinando listas de índices de filas y columnas.
- matriz[1:, [0,2]]: selecciona un **subconjunto** de filas y columnas específicas.
- matriz[[0,1], [1,2]] = [999, 888]: modifica múltiples posiciones específicas de forma directa.
- array_simple[array_simple > 20] ** 2: aplica una operación matemática solo a los elementos que cumplen la condición.

Estas técnicas son fundamentales en análisis de datos, ya que permiten aplicar transformaciones y filtros de forma eficiente y clara.

Ejemplo final completo con NumPy

Este ejemplo muestra un flujo completo de trabajo con NumPy, incluyendo generación de datos, análisis estadístico, normalización, filtrado, operaciones matriciales y guardado de resultados en un archivo de texto. Es una demostración práctica de cómo combinar múltiples funcionalidades de NumPy en un único proceso.

```
 1. np.random.seed(42)
 2.
 3. datos = np.random.rand(1000, 3) * 100
 4.
 5. print(datos.shape)
 6.
 7. media = np.mean(datos, axis=0)
 8. dst_std = np.std(datos, axis=0)
 9. max = np.max(datos, axis=0)
10. min = np.min(datos, axis=0)
11.
12. print("Características básicas")
13. print("Media", media)
14. print("Dst std", dst_std)
15. print("Máximo", max)
16. print("Mínimo", min)
17.
18. datos_normalizados = (datos - min) / (max - min)
19. print("Cinco primeras columnas normalizadas")
20. print(datos_normalizados[:5])
21.
22. datos_normalizados = datos_normalizados[datos_normalizados[:, 0] > 0.5]
23. print(datos_normalizados.shape)
24.
25. resultados = np.dot(datos_normalizados, datos_normalizados.T)
26. print(resultados.shape)
27.
28. diagonal = np.diag(resultados)
29. print(diagonal[:5])
30.
31. np.savetxt("dataset_custom.txt", datos, delimiter=',', header="Columna 1, Columna 2,
Columna 3", comments="", fmt="%.2f")
32.
```

Explicación:

- np.random.seed(42): asegura que los valores generados sean siempre los mismos (reproducibilidad).

- np.random.rand(1000, 3) * 100: genera una matriz de 1000 filas y 3 columnas con valores aleatorios entre 0 y 100.
- np.mean, np.std, np.max, np.min: calculan estadísticas básicas de cada columna (por eje axis=0).
- Normalización: (datos - min) / (max - min) escala los valores entre 0 y 1 por columna.
- Filtrado: datos_normalizados[datos_normalizados[:,0] > 0.5] selecciona las filas donde el valor de la primera columna normalizada sea mayor que 0.5.
- np.dot(datos_normalizados, datos_normalizados.T): calcula el **producto escalar entre todas las filas**, generando una matriz cuadrada.
- np.diag(resultados): extrae la diagonal principal de la matriz resultante (autoproducto de cada fila).
- np.savetxt(...): guarda el dataset original como archivo .txt separado por comas, con un encabezado y formato de dos decimales.

Este ejemplo reúne muchas funciones esenciales de NumPy, aplicadas de forma integrada para trabajar con datos simulados de forma eficiente.