Introduzione a NumPy

Python

NumPy (Numerical Python) è una libreria fondamentale per il calcolo scientifico in Python.

Fornisce supporto per array e matrici multidimensionali, insieme a una vasta raccolta di funzioni matematiche per operare su questi array.

È utilizzato estensivamente in vari campi come l'analisi dei dati, la modellazione scientifica e il machine learning.

NumPy è una libreria open-source che offre le seguenti funzionalità principali:

- **Supporto per array multidimensionali (ndarray), che sono più efficienti in termini di memoria e prestazioni rispetto alle liste native di Python.**
- **Funzioni matematiche avanzate per operare sugli array, inclusi operazioni vettoriali e matriciali, algebra lineare, trasformate di Fourier e funzioni statistiche.**
- **Strumenti per integrare codice C/C++ e Fortran, per ottimizzare ulteriormente le prestazioni delle operazioni.**

Ecco alcune delle keyword e dei concetti di base di NumPy:

ndarray: L'oggetto array multidimensionale principale di NumPy. Gli array di NumPy sono più veloci e più efficienti in termini di memoria rispetto alle liste native di Python.

dtype: Specifica il tipo di dato degli elementi di un array. I tipi di dato comuni includono int, float, bool, etc.

shape: Una proprietà che restituisce le dimensioni dell'array. Per esempio, un array con 3 righe e 4 colonne avrà una shape di (3, 4).

arange: Una funzione per creare array con valori sequenziali. È simile alla funzione range() di Python, ma restituisce un array invece di una lista.

reshape: Cambia la shape di un array senza modificarne i dati.

linspace: Genera un array di numeri equamente distribuiti tra un valore iniziale e un valore finale.

random: Modulo per generare array con valori casuali, incluse distribuzioni normali e uniformi.

sum, mean, std: Funzioni per calcolare rispettivamente la somma, la media e la deviazione standard degli elementi di un array.

Per utilizzare NumPy, è necessario prima installarlo. Si può fare facilmente utilizzando pip, il gestore dei pacchetti di Python:

- pip install numpy

Una volta installato, puoi importare NumPy nel tuo script Python:

- import numpy as np

Questo è il convenzionale alias utilizzato per importare NumPy, e consente di risparmiare tempo nella digitazione delle funzioni della libreria.

L'ndarray è l'elemento fondamentale di NumPy.
È un array multidimensionale che può contenere dati di un singolo tipo.

Esempio:

```
import numpy as np

# Creazione di un array unidimensionale
arr = np.array([1, 2, 3, 4, 5])

# Creazione di un array bidimensionale
arr2d = np.array([[1, 2, 3], [4, 5, 6]])
```

L'ndarray è l'elemento fondamentale di NumPy.
È un array multidimensionale che può contenere dati di un singolo tipo.

Esempio:

```
import numpy as np

# Creazione di un array unidimensionale
arr = np.array([1, 2, 3, 4, 5])

# Creazione di un array bidimensionale
arr2d = np.array([[1, 2, 3], [4, 5, 6]])
```

È possibile creare array NumPy utilizzando funzioni come np.array(), np.zeros(), np.ones(), np.arange(), e np.linspace()

Questo esempio ne spiega alcuni:

```
1. import numpy as np
2.
3. # Creazione di un array
4. arr = np.array([1, 2, 3, 4, 5])
5.
6. # Utilizzo di alcuni metodi
7. print("Forma dell'array:", arr.shape) # Output: (5,)
8. print("Dimensioni dell'array:", arr.ndim) # Output: 1
9. print("Tipo di dati:", arr.dtype) # Output: int64 (varia a seconda della piattaforma)
10. print("Numero di elementi:", arr.size) # Output: 5
11. print("Somma degli elementi:", arr.sum()) # Output: 15
12. print("Media degli elementi:", arr.mean()) # Output: 3.0
13. print("Valore massimo:", arr.max()) # Output: 5
14. print("Indice del valore massimo:", arr.argmax()) # Output: 4
```

Il dtype specifica il tipo di dati contenuti nell'array.
Può essere int, float, bool, etc.

Esempio:

1. **arr = np.array([1, 2, 3], dtype='int32')**
2. **print(arr.dtype) # Output: int32**

La shape di un array indica le sue dimensioni.
È una tupla che rappresenta il numero di elementi in ciascuna dimensione.

Esempio:

1. **arr = np.array([[1, 2, 3], [4, 5, 6]])**
2. **print(arr.shape) # Output: (2, 3)**

Es1: ndarray, dtype, shape, arange

Crea un array NumPy utilizzando arange e verifica il tipo di dato (dtype) e la forma (shape) dell'array.

Esercizio:

- **Utilizza la funzione np.arange per creare un array di numeri interi da 10 a 49.**
- **Verifica il tipo di dato dell'array e stampa il risultato.**
- **Cambia il tipo di dato dell'array in float64 e verifica di nuovo il tipo di dato.**
- **Stampa la forma dell'array.**

Es1: ndarray, dtype, shape, arange

```
import numpy as np

# Creazione dell'array
array = np.arange(10, 50)

# Verifica e stampa del tipo di dato
print("Tipo di dato iniziale:", array.dtype)

# Cambia il tipo di dato in float64
array = array.astype(np.float64)

# Verifica e stampa del tipo di dato dopo la conversione
print("Tipo di dato dopo la conversione:", array.dtype)

# Stampa della forma dell'array
print("Forma dell'array:", array.shape)

```

Passiamo ora al concetto di Indexing e Slicing, gliarray NumPy possono essere indicizzati e affettati in modo simile alle liste Python, ma con funzionalità aggiuntive.

Esempio:

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

# Indexing
print(arr[0]) # Output: 1

# Slicing
print(arr[1:3]) # Output: [2 3]

# Boolean Indexing
print(arr[arr > 2]) # Output: [3 4 5]
```

Gli array NumPy supportano il slicing e il fancy indexing, permettendo di estrarre porzioni di array e modificare il loro contenuto in modo efficiente.

Esempio:

```
arr_2d = np.array([[1, 2, 3, 4],
 [5, 6, 7, 8],
 [9, 10, 11, 12]
])
# Slicing sulle righe
print(arr_2d[1:3]) # Output: [[ 5 6 7 8]
 # [ 9 10 11 12]]

# Slicing sulle colonne
print(arr_2d[:, 1:3]) # Output: [[ 2 3]
 # [ 6 7]
 # [10 11]]

# Slicing misto
print(arr_2d[1:, 1:3]) # Output: [[ 6 7]
 # [10 11]]
```

Slicing è una tecnica utilizzata per estrarre una parte di un array o di una sequenza.

In NumPy, lo slicing è simile a quello delle liste in Python, ma è molto più potente e versatile.
Consente di ottenere subarray di un array esistente senza copiare i dati, il che è efficiente in termini di memoria.

La sintassi base per lo slicing in NumPy è:

array[start:stop:step]

- **start: L'indice di inizio dello slicing (inclusivo). Se omesso, il valore predefinito è 0.**
- **stop: L'indice di fine dello slicing (esclusivo). Se omesso, il valore predefinito è la dimensione dell'array.**
- **step: Il passo tra un indice e l'altro. Se omesso, il valore predefinito è 1.**

```
import numpy as np
arr = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

# Slicing di base
print(arr[2:7]) # Output: [2 3 4 5 6]

# Slicing con passo
print(arr[1:8:2]) # Output: [1 3 5 7]

# Omettere start e stop
print(arr[:5]) # Output: [0 1 2 3 4]
print(arr[5:]) # Output: [5 6 7 8 9]

# Utilizzare indici negativi
print(arr[-5:]) # Output: [5 6 7 8 9]
print(arr[:-5]) # Output: [0 1 2 3 4]
```

Fancy indexing è una tecnica che permette di selezionare elementi di un array utilizzando array di indici interi.
Questo consente una selezione complessa e flessibile di elementi rispetto allo slicing normale.

Sintassi della Fancy Indexing

```
1.
2. arr = np.array([10, 20, 30, 40, 50])
3.
4. # Utilizzo di un array di indici
5. indices = np.array([1, 3])
6. print(arr[indices]) # Output: [20 40]
7.
8. # Utilizzo di una lista di indici
9. indices = [0, 2, 4]
10. print(arr[indices]) # Output: [10 30 50]
```

Differenze tra Slicing e F.Indexing

Slicing:

- **È limitato a selezioni rettangolari.**
- **Restituisce una vista dell'array originale (non crea una copia).**
- **Utilizza indici di inizio, fine e passo.**

Fancy Indexing:

- **Può selezionare elementi non contigui e in ordine arbitrario.**
- **Crea sempre una copia dei dati selezionati.**
- **Utilizza array di indici interi.**

Esercizio Slicing e F.Indexing

Esercizio su NumPy Slicing

Consegna:

- **Crea un array NumPy 1D di 20 numeri interi casuali compresi tra 10 e 50.**
- **Utilizza lo slicing per estrarre i primi 10 elementi dell'array.**
- **Utilizza lo slicing per estrarre gli ultimi 5 elementi dell'array.**
- **Utilizza lo slicing per estrarre gli elementi dall'indice 5 all'indice 15 (escluso).**
- **Utilizza lo slicing per estrarre ogni terzo elemento dell'array.**
- **Modifica, tramite slicing, gli elementi dall'indice 5 all'indice 10 (escluso) assegnando loro il valore 99.**
- **Stampa l'array originale e tutti i sottoarray ottenuti tramite slicing.**

Esercizio Slicing e F.Indexing

Esercizio su NumPy Slicing

```
import numpy as np  # Importa la libreria numpy e la assegna all'alias np

# Crea un array NumPy 1D di 20 numeri interi casuali compresi tra 10 e 50
array = np.random.randint(10, 51, 20)

# Utilizza lo slicing per estrarre i primi 10 elementi dell'array
primi_10 = array[:10]

# Utilizza lo slicing per estrarre gli ultimi 5 elementi dell'array
ultimi_5 = array[-5:]

# Utilizza lo slicing per estrarre gli elementi dall'indice 5 all'indice 15 (escluso)
indice_5_15 = array[5:15]

# Utilizza lo slicing per estrarre ogni terzo elemento dell'array
ogni_terzo = array[::3]

# Modifica, tramite slicing, gli elementi dall'indice 5 all'indice 10 (escluso) assegnando loro il valore 99
array[5:10] = 99

# Stampa l'array originale (dopo la modifica)
print("Array originale (modificato):", array)

# Stampa i sottoarray ottenuti tramite slicing
print("Primi 10 elementi:", primi_10)
print("Ultimi 5 elementi:", ultimi_5)
print("Elementi dall'indice 5 all'indice 15 (escluso):", indice_5_15)
print("Ogni terzo elemento dell'array:", ogni_terzo)
```

Funzioni Matematiche e Statistiche

NumPy fornisce una vasta gamma di funzioni per eseguire operazioni matematiche e statistiche su array mantenendo la coerenza funzionale.

- **Operazioni Aritmetiche: np.add(), np.subtract(), np.multiply(), np.divide().**
- **Funzioni Matematiche: np.sin(), np.cos(), np.exp(), np.log().**
- **Statistica: np.mean(), np.median(), np.std(), np.var().**

La funzione linspace genera un array di numeri equidistanti tra un valore iniziale e uno finale.

Esempio:

```
1. arr = np.linspace(0, 1, 5)
2. print(arr) # Output: [0. 0.25 0.5 0.75 1. ]
```

Il modulo random di NumPy permette di generare array di numeri casuali.

Esempio:

```
1. random_arr = np.random.rand(3, 3)
2. # Matrice 3x3 con valori casuali uniformi tra 0 e 1
3. print(random_arr)
```

NumPy fornisce diverse funzioni per eseguire operazioni matematiche sugli array.

Esempio:

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

sum_value = np.sum(arr)
mean_value = np.mean(arr)
std_value = np.std(arr)

print("Sum:", sum_value) # Output: Sum: 15
print("Mean:", mean_value) # Output: Mean: 3.0
print("Standard Deviation:", std_value)
# Output: Standard Deviation: 1.4142135623730951
```

Es1: linspace, random, sum

Esercizio:

- **Crea un array di 12 numeri equidistanti tra 0 e 1 usando linspace.**
- **Cambia la forma dell'array a una matrice 3x4.**
- **Genera una matrice 3x4 di numeri casuali tra 0 e 1.**
- **Calcola e stampa la somma degli elementi di entrambe le matrici.**

Es1: linspace, random, sum

Esercizio:

```
import numpy as np  # Importa la libreria numpy e la assegna all'alias np

# Crea un array di 12 numeri equidistanti tra 0 e 1 usando linspace
array_linspace = np.linspace(0, 1, 12)

# Cambia la forma dell'array a una matrice 3x4
matrix_linspace = array_linspace.reshape(3, 4)

# Genera una matrice 3x4 di numeri casuali tra 0 e 1
matrix_random = np.random.random((3, 4))

# Calcola la somma degli elementi della matrice generata con linspace
sum_linspace = np.sum(matrix_linspace)

# Calcola la somma degli elementi della matrice generata casualmente
sum_random = np.sum(matrix_random)

# Stampa la somma degli elementi della matrice generata con linspace
print("Somma degli elementi della matrice linspace:", sum_linspace)

# Stampa la somma degli elementi della matrice generata casualmente
print("Somma degli elementi della matrice random:", sum_random)
```

Algebra Lineare

NumPy include un modulo per l'algebra lineare (numpy.linalg) che supporta operazioni come:

- **Prodotto Matriciale: np.dot(), np.matmul().**
- **Determinante: np.linalg.det().**
- **Autovalori e Autovettori: np.linalg.eig().**
- **Risoluzione di Sistemi Lineari: np.linalg.solve().**

1. **Norma di un Vettore**
2.
3. **import numpy as np**
4.
5. **# Creazione di un vettore**
6. **v = np.array([3, 4])**
7.
8. **# Calcolo della norma del vettore**
9. **norm_v = np.linalg.norm(v)**
10. **print("Norma di v:", norm_v) # Output: 5.0**

Broadcasting

Il broadcasting è una potente funzione di NumPy che permette di eseguire operazioni aritmetiche su array di forme diverse.

Questo riduce la necessità di creare array esplicitamente di dimensioni compatibili per le operazioni.

Regole di Broadcasting

- **Aggiungere Dimensioni Leading: Se gli array hanno un numero diverso di dimensioni, aggiungere dimensioni leading di 1 all'array con meno dimensioni.**
- **Allineare da Destra a Sinistra: Allineare gli array partendo dalla dimensione più a destra.**
- **Compatibilità delle Dimensioni: Due dimensioni sono compatibili se sono uguali o se una delle due è 1.**
- **Espansione Automatica: Espandere le dimensioni di 1 per adattarsi alla dimensione dell'altro array.**

NumPy è ampiamente utilizzato in molte aziende per una varietà di applicazioni che richiedono il calcolo numerico, la manipolazione di dati e l'analisi.

Ecco alcuni esempi di lavorazioni comuni:

- **Analisi dei Dati**
- **Machine Learning e Data Science**
- **Elaborazione di Immagini**
- **Finanza e Analisi dei Rischi**
- **Ricerca Scientifica e Ingegneria**
- **Big Data e Analisi delle Serie Temporali**
- **Sviluppo di Applicazioni desktop e mobile**

NumPy è una libreria essenziale per la computazione scientifica e l'elaborazione di dati in Python.

Offre strumenti potenti per lavorare con array e matrici multidimensionali, fornendo numerose funzioni matematiche, logiche e di algebra lineare per eseguire calcoli complessi in modo efficiente.


Buon MasterD a tutti