LA LIBRERÍA NUMPY

M3: OBTENCIÓN Y PREPARACIÓN DE DATOS

|AE1: MANIPULAR ESTRUCTURAS DE DATOS VECTORIALES Y MATRICIALES UTILIZANDO BIBLIOTECA NUMPY PARA RESOLVER UN PROBLEMA.

alkemy

Introducción👩‍💻

NumPy es una biblioteca esencial para el manejo eficiente de datos numéricos en Python. Su estructura optimizada y su capacidad para realizar operaciones matemáticas de manera eficiente la convierten en una herramienta indispensable en campos como la ciencia de datos, la inteligencia artificial y la computación científica.

El presente manual tiene como objetivo proporcionar una guía completa sobre el uso de NumPy, desde la creación de arreglos y matrices hasta su manipulación avanzada mediante indexación, selección y operaciones matemáticas. Se abordarán conceptos fundamentales, funciones predefinidas y prácticas comunes para optimizar el procesamiento de datos en Python.

Al finalizar este manual, el lector habrá adquirido conocimientos sólidos sobre el funcionamiento de NumPy y podrá aplicarlos en diversos contextos, desde la investigación académica hasta la implementación de modelos de machine learning.

Aprendizaje esperado

Cuando finalices la lección serás capaz de:

- Comprender la importancia y funcionalidad de NumPy dentro del ecosistema de Python para el procesamiento numérico.
- Crear y manipular arreglos unidimensionales (vectores) y bidimensionales (matrices) de manera eficiente.
- Utilizar funciones predefinidas para la creación de arreglos con valores específicos, como secuencias numéricas, matrices de ceros, unos e identidad.
- Aplicar técnicas de indexación y selección para extraer subconjuntos de datos y realizar modificaciones en arreglos existentes.
- Realizar operaciones matemáticas básicas y avanzadas entre arreglos y escalares, optimizando el rendimiento computacional.
- Implementar funciones matemáticas de NumPy para el análisis y transformación de datos, incluyendo operaciones trigonométricas, logarítmicas y algebraicas.
- Diferenciar entre referencias y copias de arreglos para evitar modificaciones no deseadas en los datos.

LA LIBRERÍA NUMPY

1. RESEÑA DE LA LIBRERÍA NUMPY

NumPy (Numerical Python) es una de las bibliotecas más utilizadas en Python para el cálculo numérico. Proporciona una estructura de datos eficiente para manejar grandes volúmenes de datos numéricos en forma de arreglos multidimensionales y matrices. Su uso es fundamental en campos como la inteligencia artificial, la ciencia de datos y la estadística.

Una característica clave de NumPy es su integración con otras bibliotecas populares como Pandas, Matplotlib y Scikit-learn. Estas librerías dependen de NumPy para manejar datos de manera eficiente, lo que lo convierte en un pilar fundamental dentro del ecosistema de ciencia de datos en Python.

Otra ventaja de NumPy es su capacidad para manejar operaciones vectorizadas. Esto significa que se pueden realizar cálculos matemáticos sobre conjuntos de datos completos sin necesidad de escribir bucles, lo que mejora la legibilidad del código y la eficiencia computacional.

CREACIÓN DE ARREGLOS DE NUMPY

2. VECTORES

Los vectores en NumPy son arreglos unidimensionales que pueden almacenar múltiples valores numéricos. Se pueden definir utilizando la función numpy.array() a partir de listas de Python.

Un vector puede ser de cualquier tipo de dato numérico, como enteros, flotantes o booleanos. NumPy permite la conversión automática del tipo de datos para optimizar la memoria utilizada en la ejecución de cálculos.

Los vectores son especialmente útiles en el álgebra lineal, donde representan coordenadas, velocidades y fuerzas en el espacio. También se utilizan en modelos matemáticos y de machine learning.

Podemos realizar operaciones matemáticas directamente sobre vectores sin necesidad de escribir bucles.

```
import numpy as np
vector = np.array([1, 2, 3, 4, 5])
print(vector * 2)  # Multiplica cada elemento por 2
```

También se pueden realizar operaciones entre dos vectores, como la suma, la resta y la multiplicación elemento a elemento.

```
vector1 = np.array([1, 2, 3])
vector2 = np.array([4, 5, 6])
suma = vector1 + vector2
print(suma)  # Output: [5 7 9]
```

El producto punto entre dos vectores es una operación común en machine learning y álgebra lineal, y se puede calcular fácilmente con np.dot().

```
producto_punto = np.dot(vector1, vector2)
print(producto_punto)  # Output: 32
```

3. MATRICES

Las matrices en NumPy son arreglos bidimensionales que almacenan datos en filas y columnas. Se pueden definir como listas de listas dentro de numpy.array(). Son esenciales en muchas áreas, como la estadística, el procesamiento de imágenes y la simulación de datos.

Las matrices permiten la aplicación de múltiples operaciones matemáticas de manera eficiente y rápida. Podemos crear una matriz de la siguiente forma:

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

Para acceder a un elemento dentro de la matriz, utilizamos índices.

```
valor = matriz[1, 2]  # Elemento en la fila 1, columna 2
print(valor)  # Output: 6
```

Es posible realizar operaciones matemáticas directamente sobre matrices sin necesidad de iteraciones.

```
matriz_cuadrada = matriz ** 2
print(matriz_cuadrada)
```

Las matrices también pueden multiplicarse entre sí utilizando la función np.dot() o el operador @.

```
matriz2 = np.array([[1, 0], [0, 1], [1, 1]])
resultado = np.dot(matriz, matriz2)
print(resultado)
```

En machine learning, las matrices se utilizan para representar conjuntos de datos, imágenes y operaciones en redes neuronales.

FUNCIONES PRECONSTRUIDAS DE CREACIÓN

4. ARREGLO CON VALORES (ARANGE)

La función np.arange() genera secuencias de números con intervalos definidos. Es una alternativa eficiente a la función range() de Python.

```
secuencia = np.arange(0, 10, 2)  # Valores de 0 a 10
print(secuencia)
```

Esta función es especialmente útil en la generación de datos de prueba o en simulaciones matemáticas.

5. MATRICES DE CEROS Y UNOS

NumPy permite la creación de matrices de ceros y unos mediante np.zeros() y np.ones(), respectivamente.

```
matriz_ceros = np.zeros((3, 3))
matriz_unos = np.ones((2, 4))
print(matriz_ceros)
print(matriz_unos)
```

Estas matrices se utilizan frecuentemente en la inicialización de pesos en redes neuronales y en algoritmos de optimización.

6. VECTOR CON DISTRIBUCIÓN DE PUNTOS

La función linspace() genera un vector con valores equidistantes en un rango dado.

```
vector_puntos = np.linspace(0, 1, 5)
print(vector_puntos)
```

Este tipo de vector es utilizado en la generación de datos para gráficos y modelos matemáticos.

7. MATRIZ IDENTIDAD

La matriz identidad es un concepto clave en álgebra lineal y se puede crear con np.eye().

```
matriz_identidad = np.eye(4)
print(matriz_identidad)
```

8. MATRIZ ALEATORIA

Las matrices aleatorias se generan con np.random.rand() o np.random.randint(), dependiendo de si se necesitan valores flotantes o enteros.

```
matriz_aleatoria = np.random.rand(3, 3)
print(matriz_aleatoria)
```

9. REDIMENSIONADO DE UN ARREGLO

La función reshape() permite cambiar la estructura de un arreglo sin modificar sus valores.

INDEXACIÓN Y SELECCIÓN

10. SELECCIÓN DE ELEMENTOS DE UN ARREGLO

NumPy permite acceder a elementos individuales de un arreglo utilizando índices, similar a las listas en Python. En un vector unidimensional, el acceso se realiza indicando la posición del elemento.

```
import numpy as np
vector = np.array([10, 20, 30, 40, 50])
print(vector[2])  # Output: 30
```

En arreglos bidimensionales (matrices), se deben especificar dos índices: el de la fila y el de la columna.

```
matriz = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print(matriz[1, 2])  # Output: 6
```

También se pueden seleccionar porciones del arreglo utilizando el operador : (slicing).

```
subvector = vector[1:4]  # Elementos desde 1 hasta el 3
print(subvector)

submatriz = matriz[0:2, 1:3]  # Filas 0-1 y columnas 1-2
print(submatriz)
```

11. SELECCIÓN CONDICIONAL DE ELEMENTOS DE UN ARREGLO

NumPy permite seleccionar elementos de un arreglo que cumplan ciertas condiciones, generando arreglos booleanos.

```
numeros = np.array([3, 7, 2, 8, 5])
mayores_a_5 = numeros[numeros > 5]
print(mayores_a_5)  # Output: [7 8]
```

También se pueden combinar múltiples condiciones usando operadores lógicos como & (AND) y | (OR).

```
numeros_filtrados = numeros[(numeros > 3) & (numeros < 8)]
print(numeros_filtrados)  # Output: [7 5]
```

12. REFERENCIA Y COPIA DE ARREGLOS

Cuando se asigna un arreglo a una nueva variable en NumPy, no se crea una copia, sino una referencia al mismo objeto en memoria.

```
original = np.array([1, 2, 3, 4])
copia_ref = original  # No es una copia real
copia_ref[0] = 99
print(original)  # Output: [99  2  3  4]
```

Para crear una copia real, se debe usar copy().

```
copia_real = original.copy()
copia_real[0] = 1
print(original)  # Output: [99  2  3  4]
print(copia_real)  # Output: [1  2  3  4]
```

OPERACIONES

13. OPERACIONES ENTRE ARREGLOS

NumPy permite realizar operaciones aritméticas directamente entre arreglos del mismo tamaño.

```
array1 = np.array([1, 2, 3])
array2 = np.array([4, 5, 6])

suma = array1 + array2
multiplicacion = array1 * array2
print(suma)  # Output: [5 7 9]
print(multiplicacion)  # Output: [4 10 18]
```

Si los arreglos tienen dimensiones compatibles, NumPy aplica broadcasting, replicando los valores de menor dimensión para ajustarlos.

14. OPERACIONES CON ESCALARES

Cuando se realizan operaciones con un escalar, esta se aplica a todos los elementos del arreglo.

```
array = np.array([2, 4, 6])
print(array * 3)  # Output: [6 12 18]
print(array - 1)  # Output: [1 3 5]
```

15. APLICANDO FUNCIONES A UN ARREGLO

NumPy proporciona funciones matemáticas para aplicar a cada elemento de un arreglo sin necesidad de bucles.

```
array = np.array([1, 4, 9, 16])
raiz_cuadrada = np.sqrt(array)
logaritmo = np.log(array)
seno = np.sin(array)

print(raiz_cuadrada)  # Output: [1. 2. 3. 4.]
print(logaritmo)  # [0. 1.38629436 2.19722458 2.77258872]
print(seno)  # [0.84147098  -0.7568025  0.41211849 -0.28790332]
```

Estas funciones optimizan el procesamiento de datos en cálculos matemáticos avanzados.

Cierre ✍️

Este manual ha proporcionado un recorrido completo por las funcionalidades más importantes de NumPy, desde la creación y manipulación de arreglos hasta la aplicación de operaciones matemáticas avanzadas. A lo largo de este documento, hemos explorado cómo NumPy optimiza el procesamiento de datos y facilita el trabajo con grandes volúmenes de información en Python.

El conocimiento adquirido permitirá a los usuarios aplicar NumPy en distintos campos, como el análisis de datos, la inteligencia artificial, la estadística y la simulación científica. Además, este manual sienta las bases para profundizar en otras bibliotecas del ecosistema de Python, como Pandas para el análisis de datos, Matplotlib para visualización y Scikit-learn para machine learning.

Se recomienda seguir practicando con los ejemplos presentados, explorar la documentación oficial de NumPy y combinar su uso con otras herramientas para maximizar su potencial. Con la práctica y la aplicación en proyectos reales, NumPy se convertirá en una herramienta fundamental en cualquier flujo de trabajo de programación científica y análisis de datos.

Referencias📚

1. NumPy Documentation - https://numpy.org/doc/
2. Python Data Science Handbook - Jake VanderPlas (2016).
3. Machine Learning with Python - Andreas C. Müller, Sarah Guido (2017).
4. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow - Aurélien Géron (2019).

¡Muchas gracias!

Nos vemos en la próxima lección