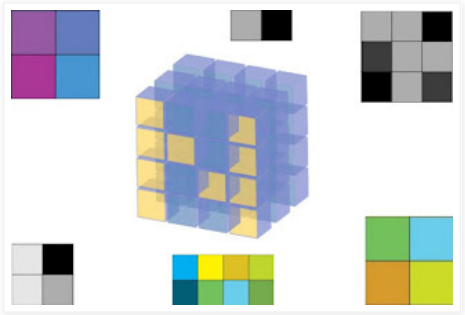
jueves, 31 de octubre de 2019

Tipos de datos en arrays Numpy (dtype)



Formas de definir el tipo de datos en un array

Básicamente, los datos (elementos) que puede contener un array Numpy son cadenas de caracteres, números enteros de distintos tamaños, números reales y booleanos. (A partir de Numpy 1.7 es posible declarar arrays para fechas y horas con el tipo de dato *datetime64* pero de momento esta opción se considera experimental y puede sufrir cambios).

La propiedad **dtype** se utiliza para definir y acceder al tipo de datos de un array:

```
a = np.array([0, 0, 1, 1, 0])
a.dtype  # dtype('int64')
```

En este ejemplo dtype ha devuelto que los datos del array son enteros de 64 bits (8 bytes).

Existen distintos modos de definir el tipo de datos de un array Numpy:

1. Implícita

Numpy analiza los datos y establece el tipo, automáticamente.

```
a = np.array([0, 0, 1, 1, 0])  # Entero (int)
print(a)  # [0 0 1 1 0]
a.dtype  # dtype('int64')

a = np.array([0.1, 0.4, 1.0, 1.5, 0.9])  # Flotante (float)
print(a)  # [0.1 0.4 1.  1.5 0.9]
a.dtype  # dtype('float64')

a = np.array(['0', '0', '1', '1', '0'])  # Cadena Unicode (str)
print(a)  # ['0' '0' '1' '1' '0']
a.dtype  # dtype('<U1')

a = np.array([False, False, True, True, False])  # Booleano (bool)
print(a)  # [False False  True  True False]
a.dtype  # dtype('bool')
```

2. Con los tipos genéricos

int, float, str, bool

Se establece asignando un tipo genérico Python al argumento **dtype**.

Buscar

Buscar

Python para impacientes

Python
IPython
EasyGUI
Tkinter
JupyterLab
Numpy

Anexos

Guía urgente de MySQL
Guía rápida de SQLite3

Entradas + populares

Dar color a las salidas en la consola
En Python para dar color a las salidas en la consola (o en la terminal de texto) existen varias posibilidades. Hay un método basado ...

Instalación de Python, paso a paso
Instalación de Python 3.6 A finales de 2016 se produjo el lanzamiento de Python 3.6 . El propósito de esta entrada es mostrar, pas...

Añadir, consultar, modificar y suprimir elementos en Numpy
Acceder a los elementos de un array. [], [,], ... Acceder a un elemento de un array. Para acceder a un elemento se utiliz...

Variables de control en Tkinter
Variables de control Las variables de control son objetos especiales que se asocian a los widgets para almacenar sus valore...

Cálculo con arrays Numpy
Numpy ofrece todo lo necesario para obtener un buen rendimiento cuando se trata de hacer cálculos con arrays. Por como está concebid...

Operaciones con fechas y horas. Calendarios
Los módulos datetime y calendar amplían las posibilidades del módulo time que provee funciones para manipular expresiones de ti...

Tkinter: interfaces gráficas en Python
Introducción Con Python hay muchas posibilidades para programar una interfaz gráfica de usuario (GUI) pero Tkinter es fácil d...

El módulo random
El módulo random de la librería estándar de Python incluye un conjunto de funciones que permiten obtener de distintos modos números a...

Threading: programación con hilos (I)
En programación, la técnica que permite que una aplicación ejecute simultáneamente varias operaciones en el mismo espacio de proceso se...

Cadenas, listas, tuplas, diccionarios y conjuntos (set)
Las cadenas , listas y tuplas son distintos tipos de secuencias . Una secuencia es un

```
a = np.array([0, 0, 1, 1, 0], dtype=int)  # Entero (int)
print(a)  # [0 0 1 1 0]
a.dtype  # dtype('int64')

a = np.array([0.1, 0.4, 0.9], dtype=float)  # Flotante (float)
print(a)  # [0.1 0.4 0.9]
a.dtype  # dtype('float64')

a = np.array(['0', '0', '1', '1', '0'], dtype=str)  # Unicode (str)
print(a)  # ['0' '0' '1' '1' '0']
a.dtype  # dtype('<U1')

a = np.array([0, 0, 1, 1, 0], dtype=bool)  # Booleano (bool)
print(a)  # [False False  True  True False]
a.dtype  # dtype('bool')
```

3. Con los tipos de Numpy

np.int, np.float, np.str, np.bool, etc.

Se establece asignando uno de los tipos de Numpy al argumento **dtype**.

```
a = np.array([0, 0, 1, 1, 0], dtype=np.int)
print(a)  # [0 0 1 1 0]
a.dtype  # dtype('int64')
```

Para optimizar el uso de los recursos del sistema es importante definir el tipo de dato apropiado de los arrays en función a los datos y al tamaño que se prevean van a contener y procesar.

Los modos implícitos o de tipos genéricos con números enteros pequeños reservan demasiado espacio de memoria para cada dato: 8 bytes. Esto no es ningún problema cuando son pocos los datos a tratar pero con arrays de gran tamaño es un factor que afecta al rendimiento que hay que tener presente.

Para ajustar el tamaño o la longitud de los datos definir los tipos con los siguientes tipos Numpy:

- np.int8: (1 byte) - Para enteros entre -128 y 127.
- np.int16: (2 bytes) - Para enteros entre -32768 y 32767.
- np.int32: (4 bytes) - Para enteros entre -2147483648 y 2147483647.
- np.int64: (8 bytes) - Para números enteros entre -9223372036854775808 y 9223372036854775807.

Si los arrays no van a contener enteros negativos se recomienda utilizar los tipos sin signo: np.uint8, np.uint16, np.uint32 y np.uint64. Así, aunque el espacio que ocupan en memoria sea el mismo, los enteros positivos pueden llegar a 255, 65535, 4294967295 y 18446744073709551615, respectivamente.

Para conocer el valor máximo o mínimo posible de un tipo:

```
np.iinfo(np.uint8).max  # 255
np.iinfo(np.uint8).min  # 0
```

Existe el problema de que cuando se asigna un dato numérico que está por encima o por debajo de los límites que impone el tipo de dato del array, numpy a priori no genera ninguna excepción y almacena un número que no se corresponde con el real. Por tanto, es conveniente elegir bien el tipo de dato en un array e implementar soluciones que eviten errores como el siguiente:

```
a = np.array([10, 10, 30, 110, 20], dtype=np.int8)  # Entero (int8)
a = a * 2
print(a)  # [ 20  20  60 -36  40]
```

El valor 110 al multiplicarlo por 2 supera el límite superior para los enteros int8, que está en 127.

4. Con los tipos abreviados

La mejor opción muchas veces es la más simple. Los tipos abreviados son cadenas formadas por un carácter que representa el tipo y un número que expresa el tamaño o longitud del dato:

- Booleanos: 'b1'.
- Enteros: 'i1' (int8), 'i2' (int16), 'i4' (int32), 'i8' (int64).
- Cadenas Unicode con distintas longitudes: 'u1', 'u2', 'u4', 'u8'.
- Cadenas Bytes con distintas longitudes: 'S1', 'S2', 'S3', 'S4'.

tipo de objeto que almacena datos y que permite ...

Archivo

octubre 2019 (2)

python.org

pypi.org

Sitios

- ActivePython
- Anaconda
- Bpython
- Django
- Flask
- Ipython
- IronPython
- Matplotlib
- MicroPython
- Numpy
- Pandas
- Pillow
- PortablePython
- PyBrain
- PyCharm
- PyDev
- PyGame
- Pypi
- PyPy
- Pyramid
- Python.org
- PyTorch
- SciPy.org
- Spyder
- Tensorflow
- TurboGears

- Flotantes: 'f2' (float16), 'f4' (float32), 'f8' (float64).
- Complejos: 'c8' (complex64), 'c16' (complex128).

```
a = np.array([0, 0, 1, 1, 0], dtype='b1')  # Booleano (bool)
print(a)  # [False False  True  True False]
a.dtype  # dtype('bool')

a = np.array([0, 0, 1, 1, 0], dtype='i1')  # Entero (int8)
print(a)  # [0 0 1 1 0]
a.dtype  # dtype('int8')
```

Para ampliar la información sobre tipos de datos que puede contener un array se recomienda visitar la página oficial de Numpy.

Publicado por Pherkad en 3:42

Etiquetas: Numpy

Entrada más reciente Inicio Entrada antigua