PYTHON FOR DATA ANALYSIS

Third Edition

Alvarado Becerra Ludwig
El ingeniero más lamentable

Índice

1. Numpy Basics: Arrays and Vectorized Computation

Numpy proporciona:

- Operaciones basadas en arrays haciendo que todo el proceso se lleve a otro tipo de computación.
- Proporciona algoritmos comúnes como ordenar, único, y operaciones de conjunto.
- Operaciones estadísticas rápidas.
- Arreglos de manipuladores de datos para unir y fusionar datasets heterogéneos.
- Expresar condiciones lógicas como arrays en lugar de ciclos con if-elif-else.
- Manipulaciones de grupos (agregación, transformación, y aplicaciones de función).

Numpy da una fundación computacional para un procesamiento general de los datos. Pandas, da algunas funciones más específicas como manipulación de series de tiempo, algo que no está en Numpy.

Numpy almacena los datos en un bloque de memoria continuo, independiente de otros objetos construidos en Python. Sus arrays también utilizan mucho menos memoria que las construidas en Python.

Se va a ejecutar el siguiente código para probar esta diferencia de velocidad.

```
import numpy as np

my_arr = np.arange(1_000_000)

my_list = list(range(1_000_000))
```

Ahora, multiplicando cada lista/array por dos, se obtiene el siguiente resultado:

```
In [4]: %timeit my_arr2 = my_arr*2
888 µs ± 43 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)

In [5]: %timeit my_list2 = [x * 2 for x in my_list]
31.3 ms ± 1.45 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Se obtiene una diferencia de escalas de miles a micro segundos, por lo tanto, se comprueba que las arrays en Numpy son muy eficientes.

1.1. La ndarray: Una array multidimensional

Las arrays permiten realizar operaciones matemáticas de bloques de datos utilizando sintaxis similar a las equivalentes entre elementos escalares.

Una ndarray es un contenedor multidimensional genérico para datos homogéneos, eso tiene que ser así, todos los elementos deben tener el mismo *data type*. Todo dato tiene una *shape*, una tupla que indica el tamaño de cada dimensión, y un *dtype*, un objeto describiendo el *data type* de la array.

```
In [1]: import numpy as np

In [2]: data = np.array([[1.5, -0.1, 3], [0, -3, 6.5]])

In [3]: data
Out[3]:
array([[1.5, -0.1, 3. ],
       [0. , -3. , 6.5]])

In [4]: data.shape
Out[4]: (2, 3)

In [5]: data.dtype
Out[5]: dtype('float64')
```

Las filas son lo que le da dimensión a las arrays, por ejemplo:

```
In [1]: data2 = [[1, 2, 3, 4], [5, 6, 7, 8]]

In [2]: arr2 = np.array(data2)

In [3]: arr2
Out[3]:
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])

In [4]: arr2.ndim
Out[4]: 2

In [5]: arr2.shape
Out[5]: (2, 4)
```

La primera posición del *shape* es la dimensionalidad, o las filas del array, mientras que la segunda posición son las columnas del array.

1.2. Tipos de datos

A continuación, se presentan los tipos de datos que se pueden manejar en Numpy

Type	**Código del *type***	**Descripción**
`int8, uint8`	`i1, u1`	Enteros asignados y sin asignar de 8-bits (1 byte)
`int16, uint16`	`i2, u2`	Enteros asignados y sin asignar de 16-bits (2 bytes)
`int32, uint32`	`i4, u4`	Enteros asignados y sin asignar de 32-bits (4 bytes)

`int64, uint64`	`i8, u8`	Enteros asignados y sin asginar de 64-bits (8 bytes)
`float16`	`f2`	Mitad de precisión de un punto flotante
`float32`	`f4` o `f`	Punto flotante con precisión estándar individual; compatible con el flotante de C
`float64`	`f9` o `d`	Punto flotante con precisión doble; compatible con el *double* de C y el objeto flotante de Python.