微 學 習 array

# Example 1 : NumPy Array性能測試

# 1 先建立空的list再一一擺入元素

start = timer() # 計時開始

for x in range(100): # 重複做100次

j = [] # 產生一個空的list

for i in range(10000):

j.append(i\*\*2) # 將平方數一個個擺入list

end = timer() # 計時結束

print(end - start) # 計算時間差

# 2 以list comprehensions方式直接建立平方數list

start = timer() # 計時開始

for x in range(100): # 重複做100次

j = [i\*\*2 for i in range(10000)] # 直接建立平方數list

end = timer() # 計時結束

print(end - start) # 計算時間差

# 3 直接建立Array

start = timer() # 計時開始

for x in range(100): # 重複做100次

i = np.arange(10000) # 建立一個0~9999的Array

j = i\*\*2 # 將Array裡面的每個元素平方

end = timer() # 計時結束

print(end - start) # 計算時間差

# %%

# %%

# Example 2 : NumPy Array基本運算

# step 1：1維的等差陣列

a = np.arange(0, 4.0, 0.5) # 類似python內建的range，只是輸出是array的資料型態

print(a) # 輸出為：[ 0. 0.5 1. 1.5 2. 2.5 3. 3.5]

print(type(a)) # 輸出為：<class 'numpy.ndarray'>

# step 2：修改陣列中的內容

a[0] = 5 # 將array a中index為0的元素重新指定數值為5

a[-1] = 100 # 將array b中的最後一個元素改成100

print('array a = ', a) # 輸出為：[ 5. 0.5 1. 1.5 2. 2.5 3. 100.]

# step 3：先建立list再將轉為array

b = np.array(range(10)) # range(10)為list的資料型態，利用array( )指令就可轉換為array

print(type(range(10))) # 輸出為：<class 'range'>

print(type(b)) # 輸出為：<class 'numpy.ndarray'>

print('array b = ', b) # 輸出為：[0 1 2 3 4 5 6 7 8 9]

# step 4：對array中的元素進行運算

c = a[1:-1]\*\*2 # 將array中index為1~-1取出來，然後每個元素平方

print(c) # 輸出為：[ 0.25 1. 2.25 4. 6.25 9.]

d = a[5:]\*0.5 # 將array中index為5以後的元素取出來，然後每個元素乘0.5

print(d) # 輸出為：[1.25 1.5 50.]

# step 5：兩個array之間的運算

e = a[:-1] + b[-7:] # 將array a和array b的index 0~-1與-7~最後元素取出相加

print(e) # 各取出a和b七個元素後相加，請注意取出的個數要一樣多

# 這裡我們接著介紹2維的array，或者你暫時把它理解為矩陣也沒什麼問題。以下我們給幾個簡單的範例：

# %%

# Example 3 : 2維的Array

a = np.array([[1, 2], [3, 4]], dtype=float) # 建立一個2\*2的array，裡面元素型態為float

print(a) # 印出array a

print(type(a)) # 印出a的資料型態

print(a.shape) # 印出a的形狀，輸出結果為(2,2)

# # 從程式執行的結果會發現a是一個 2×2 的矩陣，在數學上的表示法為：

# a=(1324)

# 下指令a.shape可給出a形狀，得到它是一個2列（row）、2行（column）的矩陣。其內容為，第1個 row 為 (12)、第2個 row 為 (34)；第1個 column 為 (13)、第2個 column 為 (24)。

# 除此之外，如果要將這個陣列中的元素更改、取出或做任何運算，則必須知道這個 array 的 index。如果是2維的就會有兩組 index 來表示各個元素的位置。如下圖所示：紅色數字為兩組 index，axis 為 array 預設的軸方向

# %%

# Example 4 : 2維Array的索引值

a = np.array([[1, 2], [3, 4]], dtype=float) # 建立一個2\*2的array，裡面元素型態為float

print('the element at 1st row and 1st column = ', a[0][0]) # 第1列，第1行 = 1.0

print('the element at 1st row and 2nd column = ', a[0][1]) # 第1列，第2行 = 2.0

print('the element at 2nd row and 1st column = ', a[1][0]) # 第2列，第1行 = 3.0

print('the element at 2nd row and 2nd column = ', a[1][1]) # 第2列，第2行 = 4.0

# 另外在科學上也常常利用到求和的指令 sum()，就是將矩陣中的元素加總，這常用在求平均值、向量量值、質心或重心位置…等。以下我們作簡單練習：

# %%

# Example 5 : 2維Array內的元素求和

a = np.array([[1, 2], [3, 4]], dtype=float) # 建立一個2\*2的array，裡面元素型態為float

b = np.sum(a, axis=0) # 將矩陣的元素沿著直的的方向加起來，並以1維array儲存結果

c = np.sum(a, axis=1) # 將矩陣的元素沿著橫的的方向加起來，並以1維array儲存結果

print(b, b.shape) # 可得array b為[4. 6.]的1維array，b.shape = (2,)

print(c, c.shape) # 可得array c為[3. 7.]的1維array，c.shape = (2,)

# 1維和2維用shape指令取出陣列形狀的差異：

# 同學們會發現在這段段程式中，如果有一個1維的array裡面含有N個元素，則該陣列的shape為(N,)，這裡的N指的是元素的個數。雖然它看起來和1列N行的矩陣很像，但是它們實際上差了一個維度，1列N行的矩陣是2維的！舉例來說：

# 1維5個元素的array為：a = np.array([1, 2, 3, 4, 5]) → a.shape = (5,)

# 2維1列N行的矩陣為：b = np.array([[1, 2, 3, 4, 5]]) → b.shape = (1,5)

# 同學該可以明顯看出其差異，1維的array只要用1層中括號，而2維的array需要用到兩層中括號。

# 這裡還有一些常見NumPy內建可以快速製造array的指令們：

# %%

# Example 6 : Array其他常用內建指令

print(np.zeros((5, 3))) # 建立shape為(5,3)，每個元素都是0的Array

print(np.ones((5, 3))) # 建立shape為(5,3)，每個元素都是1的Array

print(np.diag([1, 2, 3])) # 建立主對角線元素依序為1,2,3的方陣，shape為(3,3)

print(np.random.rand(5, 5)) # 建立shape為(5,5)的Array，每個元素介於0~1之間均勻隨機產生

print(np.random.randn(5, 5)) # 建立shape為(5,5)的Array，所有元素的產生呈常態分佈，平均值為0、標準差為1

# 更多 NumPy Array 的內建指令請參考 NumPy 官方網站：

# Array creation routines : 初階的陣列產生與控制指令。

# Random sampling : 以陣列產生隨機亂數的各種指令，包含眾多分佈函數。

# Linear algebra : 以陣列進行線性代數等矩陣相關的運算指令。

# 最後我們介紹更進階的用法－array 的維度擴充，例如：將1維 array 擴充成2維、2維擴充為3維。這裡只用到一個簡單的指令叫做「newaxis」，就可以輕易達成這件事。以下我們看看範例程式：

# %%

# Example 7 : Array的維度擴充 - 1維到2維

a = np.array([1, 2, 3, 4]) # 建立1維的array

b = a[:, np.newaxis] # 意義等同array([[1], [2], [3], [4]])

c = a[np.newaxis, :] # 意義等同array([[1, 2, 3, 4]])

print('array b = ', b, 'shape of b is', b.shape) # 印出b和其形狀

print('array c = ', c, 'shape of c is', c.shape) # 印出c和其形狀

# 以上程式執行之後你會發現，增加一個維度事實上就是多了一層中括號，只是有兩種括的方式。[:, newaxis]是將第一層中括號內的每個元素括起來；而[newaxis, :]是將第一層中括號內的所有元素整個括起來。這就使得原來1維4個元素的array分別變成shape是(4,1)和(1,4)的2維array，示意圖如下：

# 根據前面的原理，我們試試將array的維度由2維擴充到3維，請執行以下程式碼：

# %%

# Example 8 : Array的維度擴充 - 2維到3維

# 建立shape為(5,3)的array

d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])

# 意義同array([[[1, 2, 3]], [[4, 5, 6]], [[7, 8, 9]], [[10, 11, 12]]])

e = d[:, np.newaxis]

# 意義同array([[[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]]])

f = d[np.newaxis, :]

print(d, d.shape) # 印出d和其形狀

print(e, e.shape) # 印出d[:, newaxis]和其形狀

print(f, f.shape) # 印出d[newaxis, :]和其形狀

print(e[1][0][2]) # 印出d[:, newaxis]的第2列、第1行、第3排之元素 = 6

print(f[0][1][1]) # 印出d[newaxis, :]的第1列、第2行、第2排之元素 = 5

# 以下我們用圖示來表示3維array的樣子，同學對照立體圖形看會比較具體：

# 這裡的規則我們從程式碼拆解來看，請切記中括號要一層一層拆下去，每個逗點隔開的就是一個元素，看完一層後再繼續往內。以下舉例說明：

# 紅色中括號[…]，稱為第一層，這裡面放的元素會往axis = 0的方向擺，因此成為第1列、第2列…。藍色中括號[…]，稱為第二層，這裡面放的元素會往axis = 1的方向擺，故成為第1行、第2行…。綠色中括號[…]，稱為第三層，這裡面放的元素會往axis = 2的方向擺，故成為第1排、第2排…。所以你可以看出來，d[:, newaxis]會是一個共4列、1行、3排的array。如果是以下情況，原理也相同：

# 可看出d[newaxis, :]將會是一個共1列、4行、3排的array。同學再對照上一頁的立體圖形，概念上將會清楚很多！

# 如果繼續往更高維度探討，可能很容易超出人腦的想像空間，因為我們活在3維空間。但是電腦不會，因為他只要一層一層作中括號下去就好了！

# 不過我們還是可以再想一下，

# 例如你在圖書館裡也會看到類似的結構，如上圖。

# 接著下個維度可能是「第X區」，再來是「第X樓」，再來是「第X棟」，

# 只是我們不可能一直命名下去，同學腦海中記得這種一層一層的觀念即可！