Index, Slice and Reshape NumPy Arrays

October 12, 2020

1 From List to Arrays

1.0.1 One-Dimensional List to Arrays

Cuando cargas tus datos, es posible que tengas acceso a ellos mediante una lista. Se puede convertir una lista de una dimensión a un arreglo a través de `array()`.

[15]:
```
# Creando arreglo de una dimensión

from numpy import array
# Lista de datos
data = [11, 22, 33, 44, 55]
# Arreglo de datos
data = array(data)
print(data)
print(type(data))
```

```
[11 22 33 44 55]
<class 'numpy.ndarray'>
```

1.0.2 Two-Dimensional List of Lists to Array

Los datos bidimensionales son tablas con datos que cada fila representa una observación y una columa, una característica. El cargar este tipo de datos puede dar entrada a que tengamos una **lista de listas** donde cada lista representa una observación. Se puede conevritir este tipo de listas a arreglos con `array()`.

[2]:
```
# Creando un arreglo bidimensional

from numpy import array
# Lista de datos
data = [[11, 22],
        [33, 44],
        [55, 66]]
# Arreglo de datos
data = array(data)
print(data)
print(type(data))
```

```
[[11 22]
 [33 44]
 [55 66]]
<class 'numpy.ndarray'>
```

2 Array Indexing

2.0.1 One-Dimensional Indexing

Los índices en Python nos servirán para acceder a los datos contenidos en arreglos. Para hacerlo, se usarán los corchetes [] para especificar la posición i a la que queremos acceder, recordando que i empieza a tomar valores en 0.

[1]:
```
# Acceder a un dato en un arreglo unidimensional

from numpy import array
# Definiendo el arreglo
data = array([11, 22, 33, 44, 55])
# Accediendo a los datos
print(data[0])
print(data[3])
```

```
11
44
```

Si queremos acceder a una posición que se sale del límite del arreglo, el código nos enviará un error.

[2]:
```
# Acceder a un dato en un arreglo unidimensional

from numpy import array
# Definiendo el arreglo
data = array([11, 22, 33, 44, 55])
# Accediendo a los datos
print(data[5])
```

```
␣
↪---------------------------------------------------------------------------

IndexError                                Traceback (most recent call␣
↪last)

<ipython-input-2-9f963988d8fe> in <module>
  5
  6 # Accediendo a los datos
----> 7 print(data[5])
```

```
IndexError: index 5 is out of bounds for axis 0 with size 5
```

Es posible acceder a los últimos elementos del arreglo utilizando un índice o posición negativa. Por ejemplo, el índice -1 hace referencia al último elemento del arreglo. Es importante recalcar que cuando accedemos de esta manera al arreglo, el "primer" índice no empieza en 0 sino en -1.

[4]:
```
# Índice negativo

from numpy import array
# Definiendo el arreglo
data = array([11, 22, 33, 44, 55])
# Accediendo a los datos
print(data[-1])
print(data[-5])
```

```
55
11
```

2.0.2 Two-Dimensional Indexing

Es similar que el índice unidimensional pues el único cambio que hay es que separaremos el índice para cada dimensión. Por ejemplo, `data[(0,0)]`.

[6]:
```
# Accediendo a un dato en un arreglo bidimensional

from numpy import array
# Definiendo el arreglo
data = array([
    [11, 22],
    [33, 44],
    [55, 66]])
# Accediendo a los datos
print(data[2,1])
```

```
66
```

Si solo nos interesa en los elementos contenidos en alguna fila, podemos dejar el índice para la segunda dimensión vacío.

[8]:
```
# Accediendo a un dato en un arreglo bidimensional

from numpy import array
# Definiendo el arreglo
data = array([
    [11, 22],
    [33, 44],
    [55, 66]])
# Accediendo a los datos
print(data[1,])
```

```
[33 44]
```

3 Array Slicing

Las estructuras como las listas o los arreglos pueden ser rebanadas. Esto nos quiere decir que una subsecuencia de la estructura puede ser indexada. Este proceso es muy útil en Machine Learning para especificar las variables de entrada y de salida, o también para dividir las filas para entrenamiento y prueba. La rebanada es especificada con el operador : con un índice *from-to*, la cual, se extiende desde el índice *from* y termina en un elemento antes del indicado por el índice *to*.

3.0.1 One-Dimensional Slicing

[9]:
```
# Rebanar un arreglo unidimensional

from numpy import array
# Definiendo el arreglo
data = array([11, 22, 33, 44, 55])
print(data[:]) # Si no especificamos el from-to, nos imprimirá todo el arreglo
```

```
[11 22 33 44 55]
```

El primer elemento de un arreglo puede ser rebanado especificando que la rebanada empieza en el índice 0 y termina en el índice 1 (un elemento antes del índice *to*).

[10]:
```
# Rebanar un subconjunto de un arreglo unidimensional

from numpy import array
# Definiendo el arreglo
data = array([11, 22, 33, 44, 55])
print(data[0:1])
```

```
[11]
```

De igual manera, podemos usar índices negativos. Por ejemplo, podemos rebanar los últimos dos elementos empezando la rebanada en -2 (el segundo último elemento) y no especificamos el índice *to*.

[12]:
```
# Rebanar un subconjunto de un arreglo unidimensional con un índice negativo

from numpy import array
# Definiendo el arreglo
data = array([11, 22, 33, 44, 55])
print(data[-2:])
```

```
[44 55]
```

3.0.2 Two-Dimensional Slicing

Algunos ejemplos de la rebanda bidimensional que pueden ser ocupados en Machine Learning son:

Split Input and Output Features Es común dividir los datos cargados en variables de entrada (X) y variables de salida (y). Esto es posible hacerlo si rebanamos antes de la última columna, y a ésta, la indexamos de manera separada. Para los input, especificamos las filas y columnas de la siguiente manera: `X = [:, :-1]`, en donde la primer entrada indica que ocuparemos todas las filas y la segunda entrada indica que ocuparemos todas las columnas excepto la última; para el output, especificamos que ocuparemos todas las filas y sólo la última columna de la siguiente manera: `y = [:, -1]`. Como ejemplo, podemos separar un dataset bidimensional con 3 columnas en datos de entrada y salida:

[14]:
```
# Separar los datos en datos de entrada y salida

from numpy import array
# Definiendo el arreglo
data = array([
    [11, 22, 33],
    [44, 55, 66],
    [77, 88, 99]])
# Separando los datos
X, y = data[:, :-1], data[:, -1]
print(X)
print(y)
```

```
[[11 22]
 [44 55]
 [77 88]]
[33 66 99]
```

Notemos que `X` es un arreglo bidimensional y `y`, unidimensional.

Split Train and Test Rows Es igualmente común separar el dataset en datos de entrenamiento y prueba, los cuales se ocupan para entrenar el modelo y estimar la predición de éste, respectivamente. Esto conlleva a rebanar las columnas especificando `:` en la segunda entrada. Para los datos de entrenamiento se especifica que tomarán todas las filas desde el inicio hasta el punto de separación (`train = data[:split, :]`) y, para los datos de prueba, serán todas las filas que comiencen en el punto de separación hasta el final de la dimensión (`test = data[split:, :]`).

[17]:
```
# Separar los datos en datos de entrenamiento y prueba

from numpy import array
# Definiendo el arreglo
data = array([
    [11, 22, 33],
    [44, 55, 66],
    [77, 88, 99]])
# Separando los datos
split = 2
train, test = data[:split, :], data[split:, :]
print(train)
print(test)
```

```
[[11 22 33]
 [44 55 66]]
[[77 88 99]]
```

4 Array Reshaping

Después de rebanar los datos, es posible que se necesiten reescalar, pues algunas librerías suelen pedir que un arreglo unidimensional sea reescalado a un arreglo bidimensional, por ejemplo. Algunos algoritmos, como Long Short-Term Memory recurrent neural network en keras, requiere que el input sea especificado como un arreglo tridimensional.

4.0.1 Data Shape

En NumPy, los arreglos tienen el atributo `shape` que regresa una tupla (conjunto ordenado de elementos que pueden ser del mismo tipo o no,lal cual, no puede ser modificada), la cual, contiene el largo de cada dimensión del arreglo.

[18]:
```
# Forma de un arreglo unidimensional

from numpy import array
# Definiendo el arreglo
data = array([11, 22, 33, 44 ,55])
print(data.shape)
```

```
(5,)
```

Ahora, si consideramos un arreglo bidimensional:

[21]:
```
# Forma de un arreglo bidimensional

from numpy import array
# Definiendo el arreglo
data = array([
    [11, 22],
    [33, 44],
    [55, 66]])
print(data.shape) # La primer entrada indica el número de filas,
                  # y la segunda, el número de columnas
```

```
(3, 2)
```

Podemos acceder a los elementos de una tupla de la misma manera en como accedemos a los de un arreglo:

[28]:
```
# Acceder a los elementos de una tupla

from numpy import array
# Definiendo el arreglo
data = array([
```

```
    [11, 22],
    [33, 44],
    [55, 66]])
print('Filas: %i' % data.shape[0])
print('Columnas: %i' % data.shape[1])
```

```
Filas: 3
Columnas: 2
```

4.0.2 Reshape 1D to 2D Array

En NumPy existe la función `reshape()` para reescalar los datos. ésta toma un sólo argumento que especifica la nueva forma que queremos darle al arreglo. Si queremos reescalar unn arreglo unidimensioanl a uno bidimensional con una columna, la tupla tendrá como primera entrada la forma del arreglo (`data.shape[0]`) y como segunda entrada, un 1.

[34]:
```
# Reescalando un arreglo unidimensional a uno bidimensional

from numpy import array
# Definiendo el arreglo
data = array([11, 22, 33, 44, 55])
print(data.shape)
# Reescalando
data = data.reshape((data.shape[0], 1)) # Obtendremos un arreglo con cinco␣
 ↪filas y
                                        # una columna
print(data.shape)
print(data)
```

```
(5,)
(5, 1)
[[11]
 [22]
 [33]
 [44]
 [55]]
```

4.0.3 Reshape 2D to 3D Array

Es común reescalar datos bidimensionales (en donde cada fila representa una secuencia) a un arreglo tridimensional para algoritmos que necesitan múltiples muestras que constan de uno o más pasos en el tiempo y de una o más características. Para reescalarlo, ocupamos `data.reshape((data.shape[0]), data.shape[1], 1))`.

[38]:
```
# Reescalando un arreglo bidimensional a uno tridimensional

from numpy import array
# Definiendo el arreglo
data = array([[11, 22],
```

```
              [33, 44],
              [55, 66]])
print(data.shape)
# Reescalando
data = data.reshape((data.shape[0], data.shape[1], 1))
print(data.shape)
print(data)
```

```
(3, 2)
(3, 2, 1)
[[[11]
  [22]]

 [[33]
  [44]]

 [[55]
  [66]]]
```