Indexar, partir y reescala

October 8, 2020

1 De listas a arreglos

1.1 De Lista Unidimensional a un Arreglo

[4]:
```
import numpy as np

# Creamos los datos
data = [11, 22, 33, 44, 55]

#Lo convertimos en un arreglo
data = np.array(data)

# Imprimimos la información y el tipo de información
print(data)
print(type(data))
```

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

1.2 De Lista Bidimensionales a un Arreglo

[2]:
```
# Creamos los datos
list = [[11, 22],
        [33, 44],
        [55, 66]]

# Lo ponemos en un arreglo
data = np.array(list)

# Imprimimos lo guardado
print(data)
print(type(data))
```

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

1.3 Índice del Arreglo

Para acceder a un elemento del arreglo, se siga la notación que se utiliza en otros lenguajes como Java o C#, para acceder al primer elemento, se usa el 0; para acceder al segundo, se usa el 1 y así suscesivamente.

[4]:
```
# Por ejemplo si utilizamos un arreglo unidimensional\
data = np.array([11, 22, 33, 44, 55, 66])

# Ahora imprimimos un par de elementos
print(data[0])
print(data[1])

# También podemos usar índices negativos, empezando por -1 para obtener el␣
 ↪último elemento
print(data[-1])
print(data[-2])
```

```
11
22
66
55
```

1.4 Índices Bidimensionales

Parecido a los índice unidimensionales, para los bidimensionales se necesitan dos entradas, separadas por coma. Ahora se mostrará un ejemplo.

[11]:
```
# Creamos un arreglo bidimensional
data = np.array([[11, 22],
                 [33, 44],
                [ 55, 66]])

# Vamos a pedir un par de ejemplos
print(data[0, 0])
print(data[1,0])

# También podemos pedir un renglón entero, que sería el n-ésimo elemento del␣
 ↪arreglo
print(data[0,])
```

```
11
33
[11 22]
```

1.5 Partir un arreglo

Otra propiedad de los arreglos es que se puede seleccionar únicamente una porción del mismo. En inglés se le conoce como “slice”. Es muy útil en Machine Learning porque de esta manera, se pueden

especificar las variables de input y de output. Para partir el arreglo, se utiliza el operador ":".

[17]:
```
# Creamos un arreglo unidimensional de datos
data = np.array([11, 22, 33, 44, 55, 66])

# Utilizamos la notación para obtener todos los elementos del arreglo
print(data[:])

# Utilizamos la notación para obtener los primeros dos elementos
# La primera entrada representa el primer valor que se requiere obtener␣
 ↪elementos
# La segunda entrada representa el valor a partir del cual ya no se requiere␣
 ↪obtener elementos
print(data[0:1])

#También existen los índices negativos, por ejemplo si se quieren los últimso␣
 ↪dos elementos
print(data[-3:-2])
```

```
[11 22 33 44 55 66]
[11]
[44]
```

1.6 Particiones Bidimensionales

En Machine Learning, se utilizará con más frecuencia una partición bidimensional. Es común en el oficio, partir la información en distintas variables, por ejemplo, se parten las variables de entrada (X) y las variables de salida (Y).

[23]:
```
# Definimos lo que serán las variables de entrada
# X = [:, :-1] esto quiere decir que tomaremos de todos los renglones, todas␣
 ↪las columnas menos la última

#Definimos lo que serán las variables de salida
# y = [:, -1] esto quiere decir que tomaremos de todos los renglones,␣
 ↪únicamente se tomará la última columna

# Creamos un arreglo bidimensional para partir
data2 = np.array([[11, 22, 33],
                  [44, 55, 66],
                  [77, 88, 99]])

# Imprimimos las variables de entrada y la de salida
X, y = data2[:, :-1], data2[:, -1]
print(X)
print(y)
```

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

2 Renglones de entrenamiento y prueba

También se acostumbra partir la información, de tal manera que una parte sea utilizada para el entrenamiento del modelo y la otra parte para probar qué tan preciso es el modelo de aprendizaje de máquina.

[24]:
```
# Creamos una variable que determine cuántos datos queremos para la parte de␣
 ↪entrenamiento
split = 2

# Separamos los datos
train, test = data2[:split, :], data2[split:, :]

# Imprimimos los datos
print(train)
print(train.shape)
print(test)
print(test.shape)
```

```
[[11 22 33]
 [44 55 66]]
(2, 3)
[[77 88 99]]
(1, 3)
```

2.1 Reescalamiento de arreglos

Después de partir los datos, quizás es necesario hacer una rescala. Por ejemplo, algunas librerías como scikit-learn requieren un que un arreglo unidimensional para la variable de salida (y) tenga el formato de un arreglo bidimensional con una columna. Otros algoritmos como el Long Short-Memory recurrent neural network en Keras, necesita que la entrada sea un arreglo tridimensional compuesta por muestras, tiempos y características. Es importante saber cómo reescalar los arreglos con Numpy.

[27]:
```
# Nota: Para la función print, existe la manera de imprimir texto con variables␣
 ↪sin tener que estar poniendo y quitando comillas
nombre = 'André'
edad = 24
IMC = 0.13

# Se escribe de la siguiente manera
print("Su nombre es %s, tiene %d años de edad y su IMC es de %.3f" % (nombre,␣
 ↪edad, IMC))
```

```
Su nombre es André, tiene 24 años de edad y su IMC es de 0.130
```

[31]:
```
# Creamos datos unidimensionales y los imprimimos
data1 = np.array([11, 22, 33, 44, 55])
print(data1)

# Imprimimos las características del tamaño
print(data1.shape)
print("Elementos: %d" % data1.shape[0])

# Imprime otro arreglo bidimensional y su tamaño
print(data2)
print(data2.shape)
print("Renglones: %d" % data2.shape[0])
print("Columnas: %d" % data2.shape[0])
```

```
[11 22 33 44 55]
(5,)
Elementos: 5
[[11 22 33]
 [44 55 66]
 [77 88 99]]
(3, 3)
Renglones: 3
Columnas: 3
```

3 De Unidimensional a Bidimensional

Como ya se vio, es común que se tenga que cambiar la dimensionalidad de un arreglo, por ejemplo, cambiar un arreglo unidmensional a un arreglo bidimensional con una columna y múltiples arreglos. NumPy, para esto, nos da la función *reshape()*, ésta toma como argumento una tupla con los valores a los que se quiere convertir el arreglo.

[35]:
```
# Imprimimos el arreglo que teníamos
print(data1)
print(data1.shape)

# Cambiamos la escala con la función reshape()
data3 = data1.reshape((data1.shape[0], 1))
print(data3)
print(data3.shape)
```

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

```
 [55]]
(5, 1)
```

4 Reescala de bidimensional a tridimensional

Pensemos que tenemos un arreglo en donde cada secuencia tiene registrados mútliples tiempos con una observación. Entonces podemos utilizar los tamaños para especificar el números de muestras (queremos que sean los renglones) y columnas (los pasos del tiempo) y fijar el número de características a 1.

[38]:
```
# Tomamos el vector antes utilizado de data2
print(data2)
print(data2.shape)

# Hacemos la reescala
data4 = data2.reshape((data2.shape[0], data2.shape[1], 1))
print(data4)
print(data4.shape)
print(data4[0,0,])
```

```
[[11 22 33]
 [44 55 66]
 [77 88 99]]
(3, 3)
[[[11]
  [22]
  [33]]

 [[44]
  [55]
  [66]]

 [[77]
  [88]
  [99]]]
(3, 3, 1)
[11]
```