"Potencial de NumPy"

Téllez Gerardo Rubén

2/5/2021

Funciones universales (ufunc)

Son funciones que realizan operaciones sobre cada elemento en el array sin necesitad de declara bucles (como las de R)

Operaciones unarias

Operan elemento a elemento + sqrt + sin + cos + **2

```
import numpy as np
x = np.arange(10)

print(x + 3)
```

```
## [ 3  4  5  6  7  8  9 10 11 12]
```

```
print(x*3)
```

```
## [ 0  3  6  9 12 15 18 21 24 27]
```

```
print(x/3, "\n")
```

```
## [0.         0.33333333 0.66666667 1.         1.33333333 1.66666667
##  2.         2.33333333 2.66666667 3.        ]
```

```
alpha = np.linspace(0, 2*np.pi, 4)
print("Seno:" ,np.sin(alpha))
```

```
## Seno: [ 0.00000000e+00  8.66025404e-01 -8.66025404e-01 -2.44929360e-16]
```

```
print("Coseno:"  ,np.cos(alpha))
```

```
## Coseno: [ 1.  -0.5 -0.5  1. ]
```

```
print("Tangenete:", np.tan(alpha), "\n")
```

```
## Tangenete: [ 0.00000000e+00 -1.73205081e+00  1.73205081e+00 -2.44929360e-16]
```

```
print("Exponencial:", np.exp(x))
```

```
## Exponencial: [1.00000000e+00 2.71828183e+00 7.38905610e+00 2.00855369e+01
##  5.45981500e+01 1.48413159e+02 4.03428793e+02 1.09663316e+03
##  2.98095799e+03 8.10308393e+03]
```

```
print("Base 2:", np.exp2(x))
```

```
## Base 2: [  1.   2.   4.   8.  16.  32.  64. 128. 256. 512.]
```

```
print("Potencia", np.power(3, x))
```

```
## Potencia [    1     3     9    27    81   243   729  2187  6561 19683]
```

```
print("Potencia", np.power(x, 3), "\n")
```

```
## Potencia [  0   1   8  27  64 125 216 343 512 729]
```

```
print("Logaritmo base 10:", np.log10(x))
```

```
## Logaritmo base 10: [      -inf 0.         0.30103    0.47712125 0.60205999 0.69897
##  0.77815125 0.84509804 0.90308999 0.95424251]
##
## C:/Users/ruben/AppData/Local/r-miniconda/envs/r-reticulate/python.exe:1: RuntimeWarning: divide by z
```

Operaciones binarias

Reciben dos arrays y devuelven arrays (casi siempre uno) como resultado + maximum + minimum

Manejo estadístico (ufunc, unarias)

Es recomendable la librería NumPy. NP tiene dos tipos de funciones estadísticas, aquellas que funcionan de manera regular, y aquellas que se libran de los valores NA.

Suma:

$$\sum_{i=1}^{n} Xi$$

```
#Sumar los valores de la lista

y = np.array([10, 20, 30, 40, 50, np.nan])
print(np.sum(y))

#Sin NAs
```

```
## nan
```

```
print(np.nansum(y)) #Ignora los nan
```

```
## 150.0
```

Producto:

$$X_1 \cdot X_2 \cdot X_3 \ldots X_n$$

```
print(np.prod(x))
```

```
## 0
```

```
print(np.nanprod(y))
```

```
## 12000000.0
```

Media:

$$\overline{X} = \frac{\sum_{i=1}^{n} Xi}{n}$$

```
print(np.mean(x))
```

```
## 4.5
```

```
print(np.nanmean(y))
```

```
## 30.0
```

Mediana:

Valor a la mitad de los datos

```
print(np.median(x))
```

```
## 4.5
```

```
print(np.nanmedian(y))
```

```
## 30.0
```

Máximo y mínimo:

```
print(np.max(x))
```

```
## 9
```

```
print(np.nanmax(y))
```

```
## 50.0
```

```
print(np.min(x))
```

```
## 0
```

```
print(np.nanmin(y))
```

```
## 10.0
```

Desviación estándar:

$$S = \sqrt{\frac{\sum (X_1 - \overline{X})^2 \dots (X_n - \overline{X})^2}{N-1}}$$

```
print(np.std(x))
```

```
## 2.8722813232690143
```

```
print(np.nanstd(y))
```

```
## 14.142135623730951
```

Varianza:

$$S^2 = \frac{\sum (X_1 - \overline{X})^2 \dots (X_n - \overline{X})^2}{N-1}$$

```
print(np.var(x))
```

```
## 8.25
```

```
print(np.nanvar(y))
```

```
## 200.0
```

Posición en la que se encuentra el index más pequeño y más grande:

```
np.argmin(x)
```

```
## 0
```

```
np.argmax(x)
```

```
## 9
```

```
np.nanargmin(y)
```

```
## 0
```

```
np.nanargmax(y)
```

```
## 4
```

Percentil:

$$P = L_i + \frac{1}{f_i} \cdot \left(\frac{kN}{100} - f_a \right)$$

```
#Percentil 50
np.percentile(x, q=0.5)
```

```
## 0.045
```

```
np.nanpercentile(y, q=0.5)
# Percentil 95
```

```
## 10.200000000000001
```

```
np.percentile(x, q=0.95)
```

```
## 0.08549999999999999
```

```
np.nanpercentile(y, q=0.95)
```

```
## 10.379999999999999
```

any y all

```
tf = [True, False, True, False]
# ¿Alguno es verdaderos?
np.any(tf)
# ¿Todos son verdaderos?
```

```
## True
```

```
np.all(tf)
```

```
## False
```

Parámetros axis

```
np.random.seed(seed=0)
r = np.random.random((3, 5))
print(r)
```

```
## [[0.5488135  0.71518937 0.60276338 0.54488318 0.4236548 ]
##  [0.64589411 0.43758721 0.891773   0.96366276 0.38344152]
##  [0.79172504 0.52889492 0.56804456 0.92559664 0.07103606]]
```

Se puede agregar una *axis* para especificar la dimensión.

```
# Regresa una suma por columnas
r.sum(axis=0)

#Regresa una suma por filas
```

```
## array([1.98643266, 1.6816715 , 2.06258094, 2.43414258, 0.87813238])
```

```
r.sum(axis=1)
```

```
## array([2.83530423, 3.3223586 , 2.88529722])
```

NumPy Random

Se basa según las diferentes distribuciones de probabilidad que existen.

```
#Números aleatorios
print(np.random.random((4, 4)))

#Números pseudoaleatorios a partir de una semilla
# Sembrar una semilla para desaleatorizar los datos
```

```
## [[0.0871293  0.0202184  0.83261985 0.77815675]
##  [0.87001215 0.97861834 0.79915856 0.46147936]
##  [0.78052918 0.11827443 0.63992102 0.14335329]
##  [0.94466892 0.52184832 0.41466194 0.26455561]]
```

```
np.random.seed(seed=0)
```