Numpy

December 16, 2020

0.0.1 Working with Numpy (Used For Mathematical Operation)

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

[2]:
```
a = np.array([1,2,3,4])
```

[3]:
```
print(a)
print(type(a))
print(a.shape)
```

```
[1 2 3 4]
<class 'numpy.ndarray'>
(4,)
```

[4]:
```
b = np.array([[1],[2],[3],[4],[5]])
```

[5]:
```
print(b)
print(b.shape)
```

```
[[1]
 [2]
 [3]
 [4]
 [5]]
(5, 1)
```

[6]:
```
 c = np.array([[1,2,3],[4,5,6]])
```

[7]:
```
print(c)
print(c.shape)
print(c[1][1])
```

```
[[1 2 3]
 [4 5 6]]
(2, 3)
5
```

```
[8]: #Create Zeroes , Ones Array
a = np.zeros((3,3))
print(a)

a = np.ones((3,3))
print(a)
```

```
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

```
[9]: #Array of some constants
#First value is a tuple giving shape of matrix
c = np.full((3,2),5)
print(c)
```

```
[[5 5]
 [5 5]
 [5 5]]
```

```
[10]: #Identity matrix
d = np.eye(4)
print(d)
```

```
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

```
[11]: #random matrix
randomMatrix=np.random.random((2,3))
print(randomMatrix)
```

```
[[0.36866702 0.86576119 0.74815607]
 [0.91859001 0.01864204 0.4557789 ]]
```

```
[12]: # Starting Point : Ending Point
# Starting point is included but not ending point
print(randomMatrix[:,1:2])
```

```
[[0.86576119]
 [0.01864204]]
```

```
[13]: print(randomMatrix[:,2])
```

```
[0.74815607 0.4557789 ]
```

[14]:
```
randomMatrix[0:,2:]=2
print(randomMatrix)
```

```
[[0.36866702 0.86576119 2.        ]
 [0.91859001 0.01864204 2.        ]]
```

[15]:
```
#Selecting columns
z = np.zeros((3,3))
z[1,:]=1
z[:,2]=1
z[:,-1]=-1
print(z)
```

```
[[ 0.  0. -1.]
 [ 1.  1. -1.]
 [ 0.  0. -1.]]
```

[16]:
```
#Datatype of Numpy Array
x = np.ones((5,5))
```

[17]:
```
print(x)
print(type(x))
print(x.dtype)
```

```
[[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
<class 'numpy.ndarray'>
float64
```

[43]:
```
x = np.ones((5,5),dtype=np.int64)
```

[44]:
```
print(x)
print(type(x))
print(x.dtype)
```

```
[[1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]]
<class 'numpy.ndarray'>
int64
```

[18]:
```
# Data type conversion
k = np.ones((5,5),dtype=np.int64)
print(k.dtype)
```

```
int64
```

0.1 Mathematical Operation

[19]:
```
x = np.array([[1,2],[5,6]])
y= np.array([[3,4],[2,2]])
print(x+y)
print(np.add(x,y))
```

```
[[4 6]
 [7 8]]
[[4 6]
 [7 8]]
```

[20]:
```
#Matrix Multiplaction (Dot Product)
print(x.dot(y))
print(np.dot(x,y))
```

```
[[ 7  8]
 [27 32]]
[[ 7  8]
 [27 32]]
```

[21]:
```
#Multiplication Of (Dot Product) of Vectors - Scalar
a = np.array([1,2,6])
b= np.array([3,4,2])
print(a.dot(b))

# 1*3+2*4+6*2
```

```
23
```

[22]:
```
#Sum of array
su = np.array([1,2,3,4,5])
print(sum(su))
print(np.sum(su))
```

```
15
15
```

[23]:
```
#Sum Along X-Axis
mat=np.array([[1,2,3],[4,5,6],[7,8,9]])
print(np.sum(mat,axis=0)) #Along column
print(np.sum(mat,axis=1)) #Along rows
```

```
[12 15 18]
[ 6 15 24]
```

0.2 Stacking of arrays

[24]:
```
a=np.array([1,2,3])
b=np.array([4,5,6])
b=b**2
print(b)
```

```
[16 25 36]
```

[25]:
```
np.stack((a,b),axis=0)
```

[25]:
```
array([[ 1,  2,  3],
       [16, 25, 36]])
```

[26]:
```
np.stack((a,b),axis=1)
```

[26]:
```
array([[ 1, 16],
       [ 2, 25],
       [ 3, 36]])
```

0.2.1 Reshape Numpy Array

[45]:
```
re = np.array([[1,2,3,4],[5,6,7,8]])
#First define rows and second define columns
re = re.reshape((4,2))
print(re)
#If we give -1 , it automatically decides according to row the
#number of columns
ree = re.reshape((4,-1))
print(ree)
#For automatic column
pree = re.reshape((-1,8))
print(pree)
```

```
[[1 2]
 [3 4]
 [5 6]
 [7 8]]
[[1 2]
 [3 4]
 [5 6]
 [7 8]]
[[1 2 3 4 5 6 7 8]]
```

0.3 Numpy Random Module

- rand : Random values in a given shape
- randn : Return a sample from the 'standard normal' distribution
- randint : Return random integers from low to high
- random : Return random folats in the half open interval
- choice : Generates a random sample from given 1-d Array
- shuffle : Shuffle the contents of a sequence

[28]:
```python
a = np.arange(10)+5
print(a)
```

```
[ 5  6  7  8  9 10 11 12 13 14]
```

[50]:
```python
np.random.randint(50,60,10)
```

[50]:
```
array([59, 50, 50, 51, 57, 59, 52, 50, 53, 50])
```

[30]:
```python
#To random shuffle of array
np.random.shuffle(a)
print(a)
```

```
[ 6  5 14 13 10  7  9  8 12 11]
```

[31]:
```python
#Returns Values from a Standard Normal Distributions
a = np.random.randn(2,3)
print(a)
```

```
[[-1.19853043 -0.24379676  1.00680161]
 [-0.67026228 -1.26712389 -0.87927845]]
```

[32]:
```python
#Randomly pick one element from a array
element = np.random.choice([1,4,3,2,11,27])
print(element)
```

```
11
```

0.4 Numpy Functions :- Statistics

- min,max
- mean
- median
- average
- variance
- standard deviation

0.4.1 MIN

[33]:
```
a = np.array([[1,2,3,4],[5,6,7,8]])
print(a)
print(np.min(a))
#Along columns
print(np.min(a,axis=0))
#Along Rows
print(np.min(a,axis=1))
```

```
[[1 2 3 4]
 [5 6 7 8]]
1
[1 2 3 4]
[1 5]
```

0.4.2 MAX

[51]:
```
a = np.array([[1,2,3,4],[5,6,7,8]])
print(a)
print(np.max(a))
#Along columns
print(np.max(a,axis=0))
#Along Rows
print(np.max(a,axis=1))
```

```
[[1 2 3 4]
 [5 6 7 8]]
8
[5 6 7 8]
[4 8]
```

0.4.3 MEAN & MEDIAN

[52]:
```
#Mean (Average of all elements)
b = np.array([1,2,3,4,5])
m = sum(b)/5
print(m)
print(np.mean(b))
print(np.median(b))

a = np.array([[1,2,3,4],[5,6,7,8]])
#Along columns
print(np.mean(a,axis=0))
#Along Rows
print(np.mean(a,axis=1))
```

```
3.0
3.0
```

```
3.0
[3. 4. 5. 6.]
[2.5 6.5]
```

0.4.4 STANDARD DEVIATION

[55]:
```python
#Standard deviation and Variance
c = np.array([1,5,4,2,0])
print(np.median(c))

u = np.mean(c)
myStd = np.sqrt(np.mean(abs(c-u)**2))
print(myStd)
print(myStd**2)
dev = np.std(c)
var = np.var(c) # Square of standard deviation
print(dev)
print(var)
```

```
2.0
1.854723699099141
3.440000000000001
1.854723699099141
3.4400000000000004
```

0.5 Statistics :- Interview Question

Given a running stream of numbers , compute mean and variance at any given point

We will spend O(n) time computing the new mean and variance each time for each new number added.

[36]:
```python
import cv2
import matplotlib.pyplot as plt
```

[37]:
```python
img = cv2.imread('numpy.jfif')
```

[38]:
```python
print(img)
```

```
[[[175 182 177]
  [172 179 174]
  [170 177 172]
  …
  [167 181 177]
  [162 176 170]
  [159 173 167]]

 [[176 183 178]
  [174 181 176]
```

```
 [171 178 173]
 …
 [160 174 170]
 [155 169 163]
 [152 166 160]]

[[177 184 179]
 [175 182 177]
 [173 180 175]
 …
 [166 180 176]
 [161 175 169]
 [158 172 166]]

…

[[151 161 155]
 [154 164 158]
 [157 167 161]
 …
 [175 186 184]
 [175 186 184]
 [175 186 184]]

[[127 137 131]
 [134 144 138]
 [142 152 146]
 …
 [175 186 184]
 [175 186 184]
 [175 186 184]]

[[ 63  73  67]
 [ 74  84  78]
 [ 88  98  92]
 …
 [175 186 184]
 [175 186 184]
 [175 186 184]]]
```

[39]:
```
print(img.shape)
```

```
(1280, 860, 3)
```

[40]:
```
plt.figure(figsize=(20,20))
plt.imshow(img)
plt.axis("off")
```

```
plt.show()
```

Date _______ Page _______

Given a running stream of numbers compute mean & variance at any given point

$X_1, X_2 \ldots\ldots X_{n-1}$

$E(X) = \text{Mean} = \frac{1}{n} \sum_{i=1}^{n} X_i$

Also known as Expected value of X

Variance $= \sigma^2$

σ, Standard deviation $= \sqrt{\frac{1}{n} \sum_{i=1}^{n} (X_i - \mu)^2}$

So, we try to derive other formula because this is time-consuming. Suppose we X_{n-1} numbers if we add a new number than it will take $O(n)$

$$\sigma^2 = \frac{1}{n}\left[\sum_{i=1}^{n} (X_i - \mu)^2\right]$$

$$= \frac{1}{n}\left[\sum_1^n X_i^2 + \sum_1^n \mu^2 - 2\mu \sum_{i=1}^{n} X_i\right.$$

$$= \frac{1}{n}\left[\sum X_i^2 + n\mu^2 - 2\mu \sum_{i=1}^{n} X_i\right]$$

$$= \frac{\sum X_i^2}{n} + \mu^2 - 2\mu \left(\frac{\sum_{i=1}^{n} X_i}{n}\right) \text{ i.e. } \mu \text{ or } E(X)$$

$$= \frac{\sum X_i^2}{n} - \mu^2$$

$$= E(X^2) - E(X)^2$$

So can be done easily.

Mean: $\frac{\text{Sum}}{N}$

$E(X^2)$: $\frac{\text{Sum}^2}{N}$

We can get this values easily

Variance & standard deviation is $O(1)$ time.

GURUKUL

```
[ ]:
```