Basic Python 6 - NumPy

December 12, 2020

1 NumPy Library

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays

1.1 Use case

Calculate mean

[ ]:
```
number_list = [1,2,3,4,5,6,7,8,9,10]

sum_num = 0
n = len(number_list)

for num in number_list:
    sum_num = sum_num + num

result = sum_num/n
print(result)
```

[ ]:
```
import numpy as np
np.mean(number_list)
```

1.2 Importing

To use Numpy, we first need to import the `numpy` package:

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

1.3 Array creation

NumPy is used to work with arrays. The array object in NumPy is called ndarray.

1.4 array() function

[ ]:
```
# Create an array 1x3 Dimension from Python list
a = np.array([1,2,3])
print(a)
```

[ ]:
```
type(a)
```

[ ]:
```
print(a.shape)
print(np.shape(a))
```

[ ]:
```
# Create an array 2x3 Dimension from Python list
b = np.array([[1, 2, 3], [4, 5, 6]])
print(b)
```

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

- array
- credit: https://medium.com/datadriveninvestor/artificial-intelligence-series-part-2-numpy-walkthrough-64461f26af4f

1.4.1 NumPy array VS Python list

[ ]:
```
a_list = [1,2,3]
b_list = [4,5,6]
print(a_list+b_list)
```

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

[ ]:
```
print(a+a_list)
```

[ ]:
```
a_list = [1,2,3]
c_np = np.array([[1, 2, 3], [4, 5, 6]])
print(a_list)
print('---')
print(c_np)
print('---')
print(c_np+a_list)
```

1.4.2 Others functions to create NumPy arrays

[ ]:
```
# Create an array of all zeros
a = np.zeros((3, 3))
print(a)
```

[ ]:
```
# Create an array of all ones
b = np.ones((2,3))
print(b)
```

[ ]:
```
# Create a constant array
c = np.full((5,5), 10)
print(c)
```

[ ]:
```
c = np.ones((5,5)) * 3
print(c)
```

[ ]:
```
# Create a 5x5 identity matrix
d = np.eye(5)
print(d)
```

[ ]:
```
# Create an array filled with random values
e1 = np.random.random((4,4))
e2 = np.random.random((4,4))
print(e1)
print('---')
print(e2)
```

[ ]:
```
np.random.seed(0)
e1 = np.random.random((4,4))
np.random.seed(0)
e2 = np.random.random((4,4))
print(e1)
print('---')
print(e2)
```

[ ]:
```
np.random.seed(1)
e = np.random.random((4,4))
print(e)
```

[ ]:
```
np.random.seed(1)
e = np.random.random((4,4))
print(e)
```

[ ]:
```
# Create an array from a number range
f = np.arange(start=1, stop=10, step=2) # Stop(excluding from range)
print(f)
```

2 Array indexing

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

```
[ ]: print(a[0])
     print(a[1][1])
```

3 Array slicing

Similar to Python lists, numpy arrays can be sliced. Since arrays may be multidimensional, you must specify a slice for each dimension of the array.

`array[start_row : stop_row (exclude), start_column : stop_column(exclude), ...]`

```
[ ]: print(a[0:2, 0:2])
     print('---')
     print(a[1: , 2: ])
     print('---')
     print(a[:2 , 1:3])
```

3.1 View concept

A slice of an array is a view into the same data, so modifying it will modify the original array.

```
[ ]: print(a)
```

```
[ ]: b = a[1:,1:]
     print(b[0, 0])
     print(a[1, 1])
```

```
[ ]: b[0,0] = 70
```

```
[ ]: print(b)
     print('---')
     print(a)
```

4 Boolean array indexing

Boolean array indexing lets you pick out arbitrary elements of an array. Frequently this type of indexing is used to select the elements of an array that satisfy some condition. Here is an example:

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

```
[ ]: # Find the elements of a that are bigger than 2;
     bool_idx = (a > 2)
     print(bool_idx)
```

[ ]:
```
# We use boolean array indexing to construct a rank 1 array
# consisting of the elements of a corresponding to the True values
# of bool_idx
print(a[bool_idx])
```

[ ]:
```
# We can do all of the above in a single concise statement:
print(a[a>2])
```

4.1 Reshaping array

[ ]:
```
a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10 , 11, 12]])
print(a.shape)
print(a)
```

[ ]:
```
print(a.reshape(2,6))
print(a.reshape(2,6).shape)
```

[ ]:
```
print(a[a>2].reshape(2,5))
```

[ ]:
```
# One shape dimension can be -1. In this case,
# the value is inferred from the length of the array and remaining dimensions.
print(a.reshape(-1,3))
```

5 Datatypes

Every numpy array is a grid of elements of the same type. Numpy provides a large set of numeric datatypes that you can use to construct arrays. Numpy tries to guess a datatype when you create an array, but functions that construct arrays usually also include an optional argument to explicitly specify the datatype. You can read all about numpy datatypes in the documentation link1 or link2.

[ ]:
```
x = np.array([1, 2])  # Let numpy choose the datatype
y = np.array([1.0, 2.0])  # Let numpy choose the datatype
z = np.array([1, 2], dtype=np.int64)  # Force a particular datatype
o = np.array([1, "1", "1.2"])

print(x.dtype)
print(y.dtype)
print(z.dtype)
print(o.dtype)
```

6 Array math

Basic mathematical functions operate elementwise on arrays, and are available both as operator overloads and as functions in the numpy module:

```
[ ]: x = np.array([[3,3],[3,3]], dtype=np.float64)
     y = np.array([[2,2],[2,2]], dtype=np.float64)
     print(x)
     print(y)
```

```
[ ]: # Elementwise operation
     print(x + y)
     print('---')
     print(x - y)
     print('---')
     print(x * y)
     print('---')
     print(x / y)
```

```
[ ]: # Elementwise operation
     print(np.add(x, y))
     print('---')
     print(np.subtract(x, y))
     print('---')
     print(np.multiply(x, y))
     print('---')
     print(np.divide(x, y))
```

```
[ ]: # Inner product of vectors
     v = np.array([1,2])
     w = np.array([10, 20])

     #(1*10)+(2*20) = 50
     print(np.dot(v, w))
     print(v.dot(w))
     print(v @ w)
```

```
[ ]: # Matrix / matrix product; both produce the rank 2 array
     x = np.array([[1,2],[3,4]])
     y = np.array([[5,6],[7,8]])
     print(np.dot(x, y))
```

7 Arrays functions

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

```
[ ]: # Compute sum of all elements
     print(x.sum())
```

```
[ ]: # Compute sum of each column
     print(x.sum(axis=0))
```

[ ]:
```python
# Compute sum of each row
print(x.sum(axis=1))
```

[ ]:
```python
print(np.sum(x, axis=1))
```

[ ]:
```python
print(x.mean())
```

[ ]:
```python
print(x.max())
```

[ ]:
```python
print(x.min())
```