MACHINE LEARNING

Numpy

Trainer : Sujata Mohite
Email: sujata.mohite@sunbeaminfo.com

Package

- Collection of python files which you want to make it reusable libraries is called as a **package**.
- When you want to make a huge library which may not be fitting into a single file and huge logic required. You can put it into multiple python files.
- these multiple python files combined together make a package
- package = library (with multiple python files)

Package

- Python has multiple built-in packages(eg: math,os,sys) and third party packages(eg: numpy , pandas , matplotlib , sklearn)

Eg: syntax --> python -m pip install package-name

python -m pip install numpy

python -m pip install pandas

python -m pip uninstall package-name

Numpy

- third party package (need to be installed explicitly using pip3)
- developed for optimizing memory while creating collections
- mainly used while performing statistical calculations
- faster alternative for python collections
- free and open source
- developed in C++

Installation

- to install the numpy package on linux/macOS
  - sudo pip3 install numpy
- to install the numpy package on windows
  - pip install numpy

Usage

- to import numpy
  - import numpy as np

ndarray

- n-dimensional array
- collection of similar values
- the memory will get allocated contiguously
- immutable in nature

Rules

- every value in an array must be of same type
- in multi-dimensional array, every row must be having same number of columns

Ndarray- Data Types

- data types
  - Integer
    - int8
    - int16
    - int32
    - Int64
  - float
    - float16
    - float32
    - Float64
  - string
    - unicode characters

Ndarray- Data Types

- to change the data type, pass dtype as second parameters

```
a1 = np.array([10, 20, 30, 40, 50])
a2 = np.array([10, 20, 30, 40, 50], dtype=np.int32)
a3 = np.array([10, 20, 30, 40, 50], dtype=np.int16)
a4 = np.array([10, 20, 30, 40, 50], dtype=np.int8)
```

Ndarray-dtype

- returns the data type of value(s) stored in the array
- e.g.

```
a1 = np.array([10, 20, 30, 40, 50])
print(a1.dtype)
# int64
```

Ndarray- size

- number of values in the array
- similary to len(array)
- e.g.

```
a1 = np.array([10, 20, 30, 40, 50])
print(a1.size)
# 5
```

Ndarray- itemsize

- number of bytes needed to store every value
- e.g.

```
a1 = np.array([10, 20, 30, 40, 50])
print(a1.itemsize) # 8

a2 = np.array([10, 20, 30, 40, 50], dtype=np.int32)
print(a2.itemsize) # 4

a3 = np.array([10, 20, 30, 40, 50], dtype=np.int16)
print(a3.itemsize) # 2

a4 = np.array([10, 20, 30, 40, 50], dtype=np.int8)
print(a4.itemsize) # 1
```

Ndarray- nbytes

- return the size requirement of the array
- e.g

```
a1 = np.array([10, 20, 30, 40, 50])

print(a1.nbytes)      # 40 => 8 bytes * 5
```

Ndarray- ndim

- returns the dimensions of the array
- e.g

```
a1 = np.array([10, 20, 30, 40, 50])
print(a1.ndim)          # 1

a2 = np.array([[10, 20], [30, 40]])
print(a2.ndim)          # 2
```

Indexing

- Array (operations / functions)
- array used to create an array
- e.g.

```
a1 = np.array([10, 20, 30])
print(a1)   # [10 20 30]
```

Indexing - positive indexing

- indexing using positive index values
- e.g:

```
a1 = np.array([10, 20, 30, 40, 50])

print(f"value at index 0: {a1[0]}")          # 10

print(f"value at index 1: {a1[1]}")          # 20

print(f"value at index 2: {a1[2]}")          # 30

print(f"value at index 3: {a1[3]}")          # 40

print(f"value at index 4: {a1[4]}")          # 50
```

Indexing - negative indexing

- ndexing using negative index values
- -1 always represents the last value in an array
- -(len(array)) always represents the first value in an array
- e.g.

```
a1 = np.array([10, 20, 30, 40, 50])

print(f"value at index -1: {a1[-1]}") # 50

print(f"value at index -2: {a1[-2]}") # 40

print(f"value at index -3: {a1[-3]}") # 30

print(f"value at index -4: {a1[-4]}") # 20

print(f"value at index -5: {a1[-5]}") # 10
```

Slicing

- getting a slice using range of values
- e.g.

```
a1 = np.array([10, 20, 30, 40, 50])

print(a1[2:4])          # [30 40]
print(a1[2:])           # [30 40 50]
print(a1[:4])           # [10 20 30 40]
print(a1[:])            # [10 20 30 40 50]
```

Slicing – array indexing

- getting a slice (portion) of an array by using array of index values
- e.g.

```
a1 = np.array([10, 20, 30, 40, 50])

print(a1[[2, 3]])                    # [30 40]
```

Slicing-Boolean indexing

- getting a slice (portion) of an array by using array of boolean values representing required positions
- e.g

```
a1 = np.array([10, 20, 30, 40, 50])

print(a1[[False, False, True, True, False]])        # [30 40]

print(a1[[False, True, False, True, True]])         # [20 40 50]
```

Filtering

- getting the required values based on certain conditions
- e.g.

```
salaries = np.array([10000, 15000, 12000, 30000, 45000, 50000, 55000,
120000, 150000])

print(salaries[salaries > 100000])          # [120000 150000]
```

Ndarray- shape

- returns number of values in the respective dimension
- e.g

```
a1 = np.array([10, 20, 30, 40, 50])
print(a1.shape) # (5,)

a2 = np.array([[10, 20], [30, 40]])
print(a2.shape) # (2, 2)

a3 = np.array([[10, 20, 30], [40, 50, 60]])
print(a3.shape) # (2, 3)
```

Ndarray- reshape

- used to convert shape of an array
- e.g.

```
a1 = np.array([[10, 20, 30], [40, 50, 60]])
print(f"shape of a1 = {a1.shape}") # (2, 3)
# [
#   [10 20 30]
#   [40 50 60]
# ]

a2 = a1.reshape([3, 2])
print(f"shape of a2 = {a2.shape}") # (3, 2)
# [
#   [10 20]
#   [30 40]
#   [50 60]
# ]

a3 = a1.reshape([1, 6])
print(f"shape of a3 = {a3.shape}") # (1, 6)
# [
#   [10 20 30 40 50 60]
# ]

a4 = a1.reshape([6, 1])
print(f"shape of a4 = {a4.shape}") # (6, 1)
# [
#   [10]
#   [20]
#   [30]
#   [40]
#   [50]
#   [60]
# ]
```

Ndarray- arange

- used to create an array with consecutive numbers starting from first to the last value
- note: the upper bound will never be included in the array
- e.g.

```
a1 = np.arange(1, 5)
print(a1)            # [1 2 3 4]

a1 = np.arange(1, 5, 2)
print(a1)            # [1 3]
```

Ndarray- ones

- used to create an array with all values set to one
- e.g.

```
a1 = np.ones(5, dtype=int8)
print(a1)

# [1 1 1 1 1]

a2 = np.ones([2, 2], dtype=int8)
print(a2)

# [
#   [1 1]
#   [1 1]
# ]
```

Ndarray - zeros

- used to create an array with all values set to zero
- e.g.

```
a1 = np.zeros(5)
print(a1)

# [0 0 0 0 0]

a2 = np.zeros([2, 2], dtype=int8)
print(a2)

# [
#   [0 0]
#   [0 0]
# ]
```

Ndarray - random

- used to create a random array
- e.g.

```
a1 = np.random.random(5)
print(a1)       # [0.344 0.1234 0.42 0.5223 0.7777]

a2 = np.random.randint(1, 10, 5)
        # any five random numbers from 1 to 10

print(a2)       # [1 5 2 7 4]
```

Mathematical Operations

- all mathematical operations are supported on numpy array
- e.g.

```
a1 = np.array([1, 2, 3])
a2 = np.array([1, 2, 3])

print(a1 + a2)  # [2 4 6]
print(a1 - a2)  # [0 0 0]
print(a1 / a2)  # [1. 1. 1.]
print(a1 // a2) # [1 1 1]
print(a1 * a2)  # [1 4 9]
print(a1 % a2)  # [0 0 0]
print(a1 ** 3)  # [1 8 27]
```

Broadcast operations

- performing an operation on every member of an array
- e.g.

```
a1 = np.array([1, 2, 3])

print(a1 + 10)      # [11 12 13]
print(a1 - 10)      # [-9 -8 -7]
print(a1 / 10)      # [0.1 0.2 0.3]
print(a1 // 10)     # [0 0 0]
print(a1 * 10)      # [10 20 30]
print(a1 ** 2)      # [1 4 9]
print(a1 % 10)      # [1 2 3]
```

Thank You!!