Python: NumPy Course Notes

Klein
carlj.klein@gmail.com

1 Learning Objectives

Concepts

- Creating and Saving NumPy ndarrays
- Using Built-in Functions to Create ndarrays
- Accessing, Deleting, and Inserting Elements Into ndarrays
- Slicing ndarrays
- Boolean Indexing, Set Operations, and Sorting
- Exercise: Manipulating ndarrays
- Arithmetic Operations and Broadcasting
- Exercise: Creating ndarrays with Broadcasting

Commands

- np.array
- .shape
- .size
- np.save and np.load
- np.zeros
- np.ones
- np.eye
- np.diag
- np.arange
- np.linspace
- np.reshape

- np.random
  - np.random.random
  - np.random.randint
  - np.random.normal
  - np.random.permutation
- np.delete
- np.append
- np.insert
- np.vstack
- np.hstack
- np.unique
- np.intersect1d
- np.setdiff1d
- np.union1d
- np.sort
- Mathematical Functions
  - np.add
  - np.subtract
  - np.multiply
  - np.divide
  - np.sqrt
  - np.exp
  - np.power
- Statistical Functions
  - mean
  - std
  - median
  - max
  - min

NumPy is useful in dealing with arrays of data, which can be thought of in a mathematical sense as vectors and matrices. NumPy is a powerful tool in itself, however an even more useful Python data tool, the Pandas library, is built on top of NumPy. A basic understanding of NumPy is necessary, and we'll cover some of the essential functions and uses in these notes.

2 Creating and Saving NumPy ndarrays

Not needing too much of an explanation, here's a beginning look at creating basic NumPy arrays.

Covered in this section:

- np.array: turn a list, or list of lists, into a NumPy array object
- shape: return the dimensions of an array object
- size: return the total number of elements in an array object
- save & load: arrays can be saved for future use

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

x.shape
# (5,)

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

Y.shape
# (4, 3)
Y.size
# 12

# Notes Section:
# dtype changes with mixed data, and will uptype (int + floats ->
                                 all floats)
# numpy arrays can be saved for later use

# save
np.save('my_array', x)
# load
y = np.load('my_array.npy')
```

3 Using Built-in Functions to Create ndarrays

There are several function in the NumPy library which build generalized arrays, or arrays populated by data generated following rules. For example, build an array with dimensions nxm that consist of all 0s, or 1s along the diagonal, etc.

Here are some of the essential generating functions:

- zeros: create an array populated with all 0s
- ones: create an array populated with all 1s
- full: create an array populated with a specified input
- eye: creates an array populated with the identiy matrix (s1 on the diagonals, 0s for the rest)
- diag: takes a list, and places the list elements on the diagnoal of a matrix
- arange: will return a 1d array with increasing values based upon specified start, stop and spacing where the endpoint is non-inclusive: [start, stop)
- linspace: will return a 1d array with increasing values based upon start, stop and number of values desired where the endpoint is inclusive: [start, stop]
- reshape: will return an array with the elements reshaped into desired dimensions (given that the two arrays have a compatible number of elements)
- np.random.random: will return an array of random numbers filling specified dimensions
- np.random.randint: will return an array of random integers based upon a range, and filling specified dimensions
- np.random.normal: will return an array of normally distributed values based upon a range, and filling specified dimensions (note that this concept transfers to other common distributions)

```
# numpy zeros function
x = np.zeros((3, 4))
# can change dtype: np.zeros((3, 4), dtype = int)

# Below is what x, the object, look like:
"""
array([[0., 0., 0., 0.],
[0., 0., 0., 0.],
[0., 0., 0., 0.]])
"""
# For the rest of the examples, we'll be showing the print(x)
                                version:
"""
[[0. 0. 0. 0.]
[0. 0. 0. 0.]
[0. 0. 0. 0.]]
"""
```

```
x = np.ones((4, 5))
"""
[[1. 1. 1. 1. 1.]
[1. 1. 1. 1. 1.]
[1. 1. 1. 1. 1.]
[1. 1. 1. 1. 1.]]
"""

x = np.full((4, 3), 5)
"""
[[5 5 5]
[5 5 5]
[5 5 5]
[5 5 5]]
"""

x = np.eye(5)
"""
[[1. 0. 0. 0. 0.]
[0. 1. 0. 0. 0.]
[0. 0. 1. 0. 0.]
[0. 0. 0. 1. 0.]
[0. 0. 0. 0. 1.]]
"""

x = np.diag([1, 2, 3, 4, 5])
"""
[[1 0 0 0 0]
[0 2 0 0 0]
[0 0 3 0 0]
[0 0 0 4 0]
[0 0 0 0 5]]
"""

x = np.arange(10)
"""
# [0 1 2 3 4 5 6 7 8 9]
"""

x = np.arange(4, 10)
"""
# [4 5 6 7 8 9]
"""

x = np.arange(1, 14, 3)
"""
# [ 1  4  7 10 13]
"""

# numpy linspace function (default = 50)
x = np.linspace(0, 25, 10)
"""
[ 0.          2.77777778  5.55555556  8.33333333 11.11111111 13.
                                    88888889
16.66666667 19.44444444 22.22222222 25.        ]
"""
```

```
x = np.linspace(0, 25, 10, endpoint=False)
"""
[ 0.   2.5  5.   7.5 10.  12.5 15.  17.5 20.  22.5]
"""

# numpy array methods
x = np.arange(20).reshape((4, 5))
"""
[[ 0  1  2  3  4]
[ 5  6  7  8  9]
[10 11 12 13 14]
[15 16 17 18 19]]
"""

# numpy random functions
x = np.random.random((3, 3))
"""
[[0.82591877 0.69755492 0.27402617]
[0.13409014 0.66530258 0.10349078]
[0.22057372 0.17607417 0.98691068]]
"""

x = np.random.randint(4, 15, (3, 2))
"""
[[ 9  4]
[14  8]
[ 4  5]]
"""

# the 1000x1000 example is not illustrated, for formatting
                                reasons
x = np.random.normal(0, 0.1, size = (1000, 1000))
```

4 Accessing, Deleting, and Inserting Elements Into ndarrays

NumPy arrays are mutable objects, meaning they can be altered following creation. We'll be going over several ways to access and change existing arrays. Some of the essential functions covered are:

- delete: specify the array and the indices for elements to be deleted
- append: add elements to the end of an array
- insert: add elements at a specified position of an array
- vstack: stack two NumPy arrays vertically
- hstack: stack two NumPy arrays horizontally

```python
# Can access and change elements based on indices
x = np.arange(1, 10).reshape(3, 3)
"""
[[1 2 3]
[4 5 6]
[7 8 9]]
"""
# Access element
x[0, 0] = 1
# 1

# Change element
x[0, 0] = 99
"""
[[99  2  3]
[ 4  5  6]
[ 7  8  9]]
"""

# deleting elements
x = np.arange(1, 6)
# [1 2 3 4 5]
# delete first and last elements
x = np.delete(x, [0, -1])
# [2 3 4]

x = np.arange(1, 10).reshape(3, 3)
# for rank 2+ arrays, axis refers to dimension (row = 0, column =
                                 1)
W = np.delete(x, 0, axis = 0)
"""
W is x with first row deleted
[[4 5 6]
[7 8 9]]
"""
W = np.delete(x, [0, 2], axis = 1)
"""
W is x with first and last columns deleted
[[2]
[5]
[8]]
"""

# adding (appending) elements
x = np.arange(1, 6)
x = np.append(x, [6, 7])
# [1 2 3 4 5 6 7]
x = np.arange(1, 10).reshape(3, 3)
W = np.append(x, [[10 11 12]], axis = 0) # append row
Y = np.append(x, [[10], [11], [12]], axis = 1) # append column

# inserting elements
x = np.arange(1, 6)
# insert elements in between elements in rank 1 array
x = np.insert(x, 2, [3 ,4])
print(x)
# [1 2 3 4 3 4 5]
# insert row between rows in rank 2 array
x = np.arange(1, 6)
Y = np.arange(1, 10).reshape(3, 3)
W = np.insert(Y, 1, [4, 5, 6], axis = 0)
print(W)
"""
[[1 2 3]
[4 5 6]
[4 5 6]
[7 8 9]]
"""

# insert column between columns in rank 2 array
W = np.insert(Y, 1, 5, axis = 1)
print(W)
"""
[[1 5 2 3]
[4 5 5 6]
[7 5 8 9]]
"""

# stack arrays using vertical or horizontal stacking (pay
                                attention to dimension)
x = np.array([1, 2])
Y = np.array([[3, 4], [5,6]])
z = np.vstack((x, Y))
print(z)
"""
[[1 2]
[3 4]
[5 6]]
"""

z = np.hstack((Y, x.reshape(2, 1)))
print(z)
"""
[[3 4 1]
[5 6 2]]
"""
```

5 Slicing ndarrays

6 Boolean Indexing, Set Operations, and Sorting

7 Exercise: Manipulating ndarrays

8 Arithmetic Operations and Broadcasting

9 Exercise: Creating ndarrays with Broadcasting