Lesson N2 – Advanced NumPy

Rob Capra
INLS 570

Boolean Indexing

```
In [129]: names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'])
In [130]: names
Out[130]:
array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'],
      dtype='|S4')

In [131]: rdata = np.random.randn(7,4)

In [132]: rdata
Out[132]:
array([[-0.50616171, -0.22846826, -0.42737071, -0.63261581],
       [-0.39151879,  0.7829083 , -0.05144168,  0.16832157],
       [ 0.32729197,  0.45675639, -0.47354509,  0.59531804],
       [ 0.17286501, -0.01831906,  0.23977178, -0.47188809],
       [ 0.60561711, -0.55625868, -1.40889478, -1.24903569],
       [-1.3945014 ,  1.5493835 ,  0.41147468, -0.72185362],
       [ 0.34212125, -0.72760385,  0.58684746, -0.51088954]])
```

We associate each row with one name from names:

Bob
Joe
Will
Bob
Will
Joe
Joe

```
In [133]: names == 'Bob'
Out[133]: array([ True, False, False,  True, False, False, False], dtype=bool)

In [135]: rdata[names == 'Bob']
Out[135]:
array([[-0.50616171, -0.22846826, -0.42737071, -0.63261581],
       [ 0.17286501, -0.01831906,  0.23977178, -0.47188809]])

In [136]: rdata[names == 'Bob', 2:]
Out[136]:
array([[-0.42737071, -0.63261581],
       [ 0.23977178, -0.47188809]])
```

Boolean Indexing

```
In [129]: names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'])
In [130]: names
Out[130]:
array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'], dtype='<U4')

In [131]: rdata = np.random.randint(low=10, high=99, size=(7,4))

In [132]: rdata
Out[132]:
array([[54, 56, 60, 18],
       [71, 56, 19, 76],
       [66, 30, 48, 85],
       [43, 62, 24, 53],
       [16, 60, 68, 23],
       [34, 21, 33, 37],
       [54, 25, 28, 28]])
```

Bob
Joe
Will
Bob
Will
Joe
Joe

We associate each row with one name from names.

```
In [133]: names == 'Bob'
Out[133]: array([ True, False, False,  True, False, False, False])

In [135]: rdata[names == 'Bob']
Out[135]:
array([[54, 56, 60, 18],
       [43, 62, 24, 53]])

In [136]: rdata[names == 'Bob', 2:]
Out[136]:
array([[60, 18],
       [24, 53]])
```

Exercise N2.1 – Use Boolean indexing to select rows in one array based on values from another array

- 1 - Start with the names and rdata ndarrays as defined in the previous section:
  - names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe', 'Joe'])
  - rdata = np.random.randint(low=10, high=99, size=(7,4))

- 2 - Write code to select the rows of rdata that are associated with the name 'Joe'

- 3 - Write code to select the second and third column of rdata for rows associated with Will or Bob
  - Hint: use mask = (names == 'Bob') | (names == 'Will')

Fancy Indexing

```
In [1]: a1 = np.empty((8,4))

In [2]: for i in range(8):
   ...:     a1[i]=i
   ...:

In [3]: a1
Out[3]:
array([[ 0.,  0.,  0.,  0.],
       [ 1.,  1.,  1.,  1.],
       [ 2.,  2.,  2.,  2.],
       [ 3.,  3.,  3.,  3.],
       [ 4.,  4.,  4.,  4.],
       [ 5.,  5.,  5.,  5.],
       [ 6.,  6.,  6.,  6.],
       [ 7.,  7.,  7.,  7.]])

In [4]: a1[[7,1,4]]
Out[4]:
array([[ 7.,  7.,  7.,  7.],
       [ 1.,  1.,  1.,  1.],
       [ 4.,  4.,  4.,  4.]])
```

Pass a list of rows to select in the order specified

reshape and Transpose (T)

```
In [9]: a1 = np.arange(15)

In [10]: a1
Out[10]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])

In [11]: a2 = a1.reshape((3,5))

In [12]: a2
Out[12]:
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])

In [13]: a2.T
Out[13]:
array([[ 0,  5, 10],
       [ 1,  6, 11],
       [ 2,  7, 12],
       [ 3,  8, 13],
       [ 4,  9, 14]])

In [14]: a2
Out[14]:
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
```

Universal Functions

- Fast element-wise array functions

```
In [15]: a1 = np.arange(10)

In [16]: a1
Out[16]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

In [17]: a2 = np.sqrt(a1)
In [18]: a2
Out[18]:
array([ 0.        ,  1.        ,  1.41421356,  1.73205081,  2.        ,
        2.23606798,  2.44948974,  2.64575131,  2.82842712,  3.        ])

In [19]: a3 = a1 + a2
In [20]: a3
Out[20]: array([ 2,  3,  4,  5,  6,  7,  8,  9, 10, 11])

In [21]: a4 = np.add(a1,a2)
In [22]: a4
Out[22]:
array([ 0.        ,  2.        ,  3.41421356,  4.73205081,  6.        ,
        7.23606798,  8.44948974,  9.64575131, 10.82842712, 12.        ])
```

Table 4-3. Unary ufuncs

Function	Description
`abs, fabs`	Compute the absolute value element-wise for integer, floating point, or complex values. Use `fabs` as a faster alternative for non-complex-valued data
`sqrt`	Compute the square root of each element. Equivalent to `arr ** 0.5`
`square`	Compute the square of each element. Equivalent to `arr ** 2`
`exp`	Compute the exponent e^x of each element
`log, log10, log2, log1p`	Natural logarithm (base *e*), log base 10, log base 2, and log(1 + x), respectively
`sign`	Compute the sign of each element: 1 (positive), 0 (zero), or -1 (negative)
`ceil`	Compute the ceiling of each element, i.e. the smallest integer greater than or equal to each element
`floor`	Compute the floor of each element, i.e. the largest integer less than or equal to each element
`rint`	Round elements to the nearest integer, preserving the `dtype`
`modf`	Return fractional and integral parts of array as separate array
`isnan`	Return boolean array indicating whether each value is NaN (Not a Number)
`isfinite, isinf`	Return boolean array indicating whether each element is finite (non-`inf`, non-NaN) or infinite, respectively
`cos, cosh, sin, sinh, tan, tanh`	Regular and hyperbolic trigonometric functions
`arccos, arccosh, arcsin, arcsinh, arctan, arctanh`	Inverse trigonometric functions
`logical_not`	Compute truth value of `not x` element-wise. Equivalent to `-arr`.

Table 4-4. Binary universal functions

Function	Description
`add`	Add corresponding elements in arrays
`subtract`	Subtract elements in second array from first array
`multiply`	Multiply array elements
`divide, floor_divide`	Divide or floor divide (truncating the remainder)
`power`	Raise elements in first array to powers indicated in second array
`maximum, fmax`	Element-wise maximum. `fmax` ignores NaN
`minimum, fmin`	Element-wise minimum. `fmin` ignores NaN
`mod`	Element-wise modulus (remainder of division)
`copysign`	Copy sign of values in second argument to values in first argument

Exercise N2.2 – Use universal functions to transform data in an array

1. Create a 2-dimensional array (2x3) called x: $\begin{bmatrix} -3 & 5 & -1 \\ 8 & -4 & -7 \end{bmatrix}$

2. Create a 2-dimensional array (2x3) called y: $\begin{bmatrix} 2 & 4 & 9 \\ -3 & 8 & 3 \end{bmatrix}$

3. Use universal function(s) to create a new array z that contains each element of y subtracted from each elemnt of x (i.e., x - y).

4. Use universal functions to compute a new array w that contains the absolute values of all the elements of z.

5. Combine #3 and #4 above into **one** line of Python code.

np.where

- Can be used like a fast for + if

```
In [50]: a1 = np.random.randn(4,4)

In [51]: a1
Out[51]:
array([[-1.23761788, -1.91331672, -0.30916365, -0.4493096 ],
       [-1.68793919,  1.01367457, -1.56939479,  0.96812584],
       [ 0.54251421, -1.12670291,  0.73271873, -0.3220519 ],
       [-0.36038843,  1.08876332,  0.18622184,  0.81105757]])

In [52]: np.where(a1>0, 9, 0)
Out[52]:
array([[0, 0, 0, 0],
       [0, 9, 0, 9],
       [9, 0, 9, 0],
       [0, 9, 9, 9]])
```

np.where

- Can be used like a fast for + if

```
In [50]: a1 = np.random.randn(4,4)

In [51]: a1
Out[51]:
array([[-1.23761788, -1.91331672, -0.30916365, -0.4493096 ],
       [-1.68793919,  1.01367457, -1.56939479,  0.96812584],
       [ 0.54251421, -1.12670291,  0.73271873, -0.3220519 ],
       [-0.36038843,  1.08876332,  0.18622184,  0.81105757]])

In [52]: np.where(a1>0, 9, 0)
Out[52]:
array([[0, 0, 0, 0],
       [0, 9, 0, 9],
       [9, 0, 9, 0],
       [0, 9, 9, 9]])

In [54]: np.where(a1>0, a1, 0)
Out[54]:
array([[ 0.        ,  0.        ,  0.        ,  0.        ],
       [ 0.        ,  1.01367457,  0.        ,  0.96812584],
       [ 0.54251421,  0.        ,  0.73271873,  0.        ],
       [ 0.        ,  1.08876332,  0.18622184,  0.81105757]])
```

np.where

- Can be used like a fast for + if

```
import numpy as np

x = np.array([1.1, 1.2, 1.3, 1.4, 1.5])
y = np.array([9.1, 9.2, 9.3, 9.4, 9.5])
cond = np.array([True, False, True, True, False])

result = []

for i in range(len(cond)):
    if cond[i] == True:
        result.append(x[i])
    else:
        result.append(y[i])

result2 = np.where(cond, x, y)
```

Exercise N2.3 – Use np.where to selectively transform data in an array

1. Create a 2-dimensional array (3x3) called x: $\begin{bmatrix} 4 & -2 & 9 \\ 5 & -6 & -3 \\ -1 & 4 & 7 \end{bmatrix}$

2. Use np.where() to create a new array in which all the negative numbers in x are replaced with zeros, and all the positive numbers are replaced with ones.

3. Use np.where() to create a new array in which all the even numbers in x are kept as they are, but all the odd numbers are replaced with zeros.

4. Create another 2-dimensional array (3x3) called y: $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$

5. Use np.where() to create a new array. If the value of the cell in y is 1, the new array should contain the value from the corresponding cell of x. If the value of the cell in y is 0, the new array should contain zero.

Aggregation

```
In [20]: a1 = np.array([[1, 1, 2, 2],[2, 2, 4, 4]])

In [21]: a1
Out[21]:
array([[1, 1, 2, 2],
       [2, 2, 4, 4]])

In [22]: a1.mean()
Out[22]: 2.25

In [23]: np.mean(a1)
Out[23]: 2.25

In [24]: a1.sum()
Out[24]: 18

In [25]: a1.mean(axis=1)
Out[25]: array([ 1.5,  3. ])
```

Computes the mean over the selected axis

Reading and Writing Arrays to Disk

- Quick and easy way to read/write numpy arrays
- Save & Load
- Stored in a binary format

```
import numpy as np

a1 = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8], dtype=np.uint8)

np.save('a1_data.npy',a1)

a2 = np.load('a1_data.npy')
```

Exercise N2.4 – Write and read an array from disk, use aggregate funcs

1. Use np.random.randint to create a 1000 row by 20 column array called x. The array should contain random two digit numbers (i.e. 10 to 99). Hint: See Exercise 2.1.

2. Use np.save to save the array ` to a file called bigarray.npy.

3. Use np.load to load the array from the file bigarray.npy into a new array called y.

4. Find the min and max number across of all the cells in the array y.
5. Find the min and max number across the first row of cells in the array y.
6. Find the min and max number across the first column of cells in the array y.