numanyilmaz / NumPyTutorial

This repo introduces the NumPy package and shows how to use some of the most common features, functions, and attributes of it.

MIT License

9 stars 6 forks

Star Watch

Code Issues Pull requests Actions Pro

master

numanyilmaz ... on 21 May 2017

View code

NumPy in 50 cells of notebook

by Numan Yilmaz, 10 May 2017 Originally posted here.

In this post I will introduce the NumPy package and show how to use some of its most common features, functions and attributes. I will describe each feature with an example.

This tutorial consists of the following parts:

- What is NumPy?
- How to create NumPy arrays
- Indexing, Fancy Indexing
- Slicing
- Universal Functions (Ufuncs)
- Broadcasting
- Masking, Sorting and Comparison

- Further learning

1- What is NumPy array?

README.md

other packages. What makes NumPy such an incredible package is its data type (ndarray). ndarray stands for n-dimensional array, which basically looks like a Python list. However, it is a lot faster than a regular Python list. A Python list can contain different kinds of data types, such as integers, strings, Boolean, True, False and even lists. On the other hand, NumPy arrays can hold only one type of data, and therefore doesn't have to check the type of data type for every single element of the array when it is doing the computations. This feature makes NumPy a great tool for data science research and projects.

Before we get started, let's check the version of NumPy and Python.

```
# import numpy
import numpy as np

# sys was imported to check the python
import sys

# check the version of python and numpy
print('NumPy version:', np.__version__)
print('Python version',sys.version)
```

```
NumPy version: 1.12.1
Python version 3.6.1 |Anaconda
custom (x86_64)| (default, Mar 22
2017, 19:25:17)
[GCC 4.2.1 Compatible Apple LLVM 6.0
(clang-600.0.57)]
```

2- How to create NumPy arrays

There are many ways to create arrays in NumPy. We will take a look at a few of them here.

```
# create one dimensional numpy array
np.array([1, 2, 3])
```

```
array([1, 2, 3])
```

```
# Array of zeros
np.zeros(3)
```

```
array([ 0.,  0.,  0.])
```

```
# Array of 1s
np.ones(3)
```

```
array([ 1.,  1.,  1.])
```

```
# array of 3 random integers between 1
np.random.randint(1,10, 3)
```

```
array([1, 4, 6])
```

```
# create linearly spaced array.
np.linspace(0, 10, 5 )
```

```
array([  0. ,   2.5,   5. ,   7.5,
10. ])
```

```
# create 2-Dimensional array
np.array([[1,2,3],
          [4,5,6],
          [7,8,9]])
```

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

```
# create 3x4 array values between 0 and
np.random.random((3,4))
```

```
array([[ 0.23669989,  0.36323724,
0.53665447,  0.39260328],
       [ 0.57680018,  0.01829588,
0.45511873,  0.44885674],
       [ 0.32340815,  0.06009959,
0.28130121,  0.55966556]])
```

```
# create 1D and 2D arrays a and b
a = np.array([1,2,3])
b = np.random.randint(0,10, (3,3))

print(a)
print(b)
```

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

```
# adding values
a = np.append(a, 4)
a
```

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

```
# print the shape and dimension of arra
print("Shape of a:", np.shape(a))
print("Shape of b:", np.shape(b))

print('Dimension of a:', np.ndim(a))
print('Dimension of b:', np.ndim(b))
```

```
Shape of a: (4,)
Shape of b: (3, 3)
Dimension of a: 1
Dimension of b: 2
```

```
#  number of elements in the arrays
print('Number of elements in a:', np.si
print('Number of elements in b:', np.si
```

```
Number of elements in a: 4
Number of elements in b: 9
```

3- Indexing and Fancy Indexing

```
# a is 1D array, we created before
a
```

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

```
# b is 2D array created in a previous c
b
```

```
array([[8, 7, 8],
       [1, 6, 3],
       [5, 4, 6]])
```

```
# get the first element of a
# these 2 print statements results the
print(a[0])
print(a[-4])
```

```
1
1
```

```
# get the last element of a
# these 2 print statements results the
print(a[-1])
print(a[3])
```

```
4
4
```

```
# get the first row of b
# these 2 print statements results the
print(b[0])
print(b[0,:])
```

```
[8 7 8]
[8 7 8]
```

```
# get the second column of b
b[:,1]
```

```
array([7, 6, 4])
```

```
# to understand the fancy indexing bett
x = np.array(['a', 'b', 'c'])
y = np.array([['d','e','f'],
              ['g', 'h', 'k']])

print(x)
print(y)
```

```
['a' 'b' 'c']
[['d' 'e' 'f']
 ['g' 'h' 'k']]
```

```
# fancy indexing on 1D array
# get the value of c in array x
ind = [2]
x[ind]
```

```
array(['c'],
      dtype='<U1')
```

```
# fancy indexing on 2D array
# get the values  e,h in array y
ind2 = [[0,1],[1]]
y[ind2]
```

```
array(['e', 'h'],
      dtype='<U1')
```

4- Slicing

use : for slicing

```
# create an array integer from 1 to 10
X = np.arange(1, 11, dtype=int)
X
```

```
array([ 1,  2,  3,  4,  5,  6,  7,
8,  9, 10])
```

```
# get the first two elements of X
X[:2]
```

```
array([1, 2])
```

```
# get the number 3,4 and 5
X[2:5]
```

```
array([3, 4, 5])
```

```
# get the odd numbers
X[::2]
```

```
array([1, 3, 5, 7, 9])
```

```
# get the even numbers
X[1::2]
```

```
array([ 2,  4,  6,  8, 10])
```

```
# create 2D array
Y= np.arange(1,10).reshape(3,3)
Y
```

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

```
# get the first and second row
Y[:2,:]
```

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

```
# get the second and third column
Y[:, 1:]
```

```
array([[2, 3],
       [5, 6],
       [8, 9]])
```

```
#get the element of 5 and 6
Y[1,1:]
```

```
array([5, 6])
```

5- Universal Functions(Ufuncs)

press TAB after np. to see list of available ufuncs. np.{TAB}

Allow fast computation in NumPy arrays.

```
# use the same array we created earlier
X
```

```
array([ 1,  2,  3,  4,  5,  6,  7,
8,  9, 10])
```

```
#find the maximum element of X
np.max(X)
```

```
10
```

```
#mean of values in the X
np.mean(X)
```

```
5.5
```

```
# get the 4th power of each value
np.power(X, 4)
```

```
array([    1,    16,    81,   256,
625,  1296,  2401,  4096,  6561,
10000])
```

```
# trigonometric functions
print(np.sin(X))
print(np.tan(X))
```

```
[ 0.84147098  0.90929743  0.14112001
-0.7568025  -0.95892427 -0.2794155
  0.6569866   0.98935825  0.41211849
-0.54402111]
[ 1.55740772 -2.18503986 -0.14254654
1.15782128 -3.38051501 -0.29100619
  0.87144798 -6.79971146 -0.45231566
0.64836083]
```

```
# x2 + y2 = 1
np.square(np.sin(X)) + np.square(np.cos
```

```
array([ 1.,  1.,  1.,  1.,  1.,  1.,
1.,  1.,  1.,  1.])
```

```
# same rules applies for 2D array
Y
```

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

```
np.multiply(Y, 2)
```

```
array([[ 2,  4,  6],
       [ 8, 10, 12],
       [14, 16, 18]])
```

```
# split Y into 3 subarrays
np.split(Y, 3)
```

```
[array([[1, 2, 3]]), array([[4, 5,
6]]), array([[7, 8, 9]])]
```

6- Broadcasting

Broadcasting is being able to use ufuncs and many other operations on different size of arrays

```
X
```

```
array([ 1,  2,  3,  4,  5,  6,  7,
8,  9, 10])
```

```
# add 5 to each element
X + 5
```

```
array([ 6,  7,  8,  9, 10, 11, 12,
13, 14, 15])
```

```
# or
np.add(X, 5)
```

```
array([ 6,  7,  8,  9, 10, 11, 12,
13, 14, 15])
```

```
# create new array Z
Z = np.arange(3)[:, np.newaxis]
Z
```

```
array([[0],
       [1],
       [2]])
```

```
# multiple Y and Z
np.multiply(Y, Z)
```

```
array([[ 0,  0,  0],
       [ 4,  5,  6],
       [14, 16, 18]])
```

7- Sorting, Comparising and Masking

```
# create array of 10 elements between 1
x = np.random.randint(1,5, 10)
x
```

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

```
# create (3,3) size of array elements 1
y = np.random.randint(1,5, (3,3))
y
```

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

```
# sort elements in array x
np.sort(x)
```

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

```
# sort values along the rows
np.sort(y, axis=0)
```

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

```
# sort values along the columns
np.sort(y, axis=1)
```

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

```
# == , !=, < , >, >=, <= operations on
#This returns a boolean
x > 3
```

```
array([False, False,  True, False,
True, False, False, False, False,
False], dtype=bool)
```

```
# use masking feature to get the values
x[x>3]
```

```
array([4, 4])
```

```
# more example
x[(x <= 3) & (x>1)]
```

```
array([3, 2, 2, 3, 3])
```

Releases

No releases published

Packages

No packages published

Languages

- **Jupyter Notebook** 100.0%