[Python] Numpy에 있는 다양한 함수 사용법 - 2(전치행렬,zeors,ones, iterator,etc..) — 나무늘보의 개발 블로그

노트북:	첫 번째 노트북
만든 날짜:	2020-10-28 오후 11:50
URL:	https://continuous-development.tistory.com/118?category=736681

Python

[Python] Numpy에 있는 다양한 함수 사용법 - 2(전치행렬,zeors,ones, iterator,etc..)

2020. 10. 9. 17:50 수정 삭제 공개

배열 변형(타입 형태에 따른 연산)

```
In [159]: x = np.array([1,2,3],dtype='f')
          aryinfo(x)

          type : <class 'numpy.ndarray'>
          shape : (3,)
          dimension : 1
          dtype : float32
          Array Data :
           [1. 2. 3.]
```

```
In [160]: x[0] + x[1]
Out[160]: 3.0
```

여기서는 float 형태여서 사칙연산으로 되었지만.

```
In [162]: x = np.array([1,2,3],dtype='U')
          aryinfo(x)

          type : <class 'numpy.ndarray'>
          shape : (3,)
          dimension : 1
          dtype : <U1
          Array Data :
           ['1' '2' '3']

In [163]: x[0] + x[1]
Out[163]: '12'
```

여기서는 string 으로 돼서 글자가 합쳐져 12로 된다.

- inf vs Nan

```
In [164]: np.array([0, 1, -1, 0])/np.array([1, 0, 0, 0]))

          C:₩Users₩hwang in beom₩Anaconda3₩lib₩site-packages₩ipykernel_launcher.py:1: RuntimeWarning: divide by zero encountered in true_divide
            """Entry point for launching an IPython kernel.
          C:₩Users₩hwang in beom₩Anaconda3₩lib₩site-packages₩ipykernel_launcher.py:1: RuntimeWarning: invalid value encountered in true_divide
            """Entry point for launching an IPython kernel.

Out[164]: array([  0.,  inf, -inf,  nan])
```

함수를 통한 배열 생성(zeros,ones,zeros_like,ones_like....)

```
- zeors, ones # 0으로 배열을 셋팅 / 1로 배열을 셋팅
- zeros_like, ones_like # shape를 참조 해서 0 으로 만든다. / shape를 참조 해서 1 으로 만든다.
- empty
- arrang
- linspace, logspace
```

- zeors: 크기가 정해져 있고 모든 값이 0인 배열을 생성

```
In [167]: z = np.zeros(5)
          aryinfo(z)

          type : <class 'numpy.ndarray'>
          shape : (5,)
          dimension : 1
          dtype : float64
          Array Data :
           [0. 0. 0. 0. 0.]
```

np.zeors - 0으로 배열을 세팅한다.
이렇게 np.zeors라는 명령어로 0 이 다섯 개가 들어간 하나의 numpy 가 생긴다.

```
In [168]: o = np.ones((2,3,4),dtype='i8')
          aryinfo(o)

type : <class 'numpy.ndarray'>
shape : (2, 3, 4)
dimension : 3
dtype : int64
Array Data :
 [[[1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]]

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

np.ones - 1으로 배열을 세팅한다.
여기서는 np.ones을 통해 2,3,4 형태의 1의 값이 들어간 numpy가 생긴다.

```
In [173]: o_like = np.ones_like(o, dtype='f') # 기존 배열의 형태를 참조한다.
          aryinfo(o_like)

type : <class 'numpy.ndarray'>
shape : (2, 3, 4)
dimension : 3
dtype : float32
Array Data :
 [[[1. 1. 1. 1.]
  [1. 1. 1. 1.]
  [1. 1. 1. 1.]]

 [[1. 1. 1. 1.]
  [1. 1. 1. 1.]
  [1. 1. 1. 1.]]]
```

ones_like 같은 경우에는 o의 형태를 가지고와 0의 값을 넣어준다.
다른 배열의 형태를 가지고와 ones / zeros를 사용한다.

```
In [175]: e = np.empty((4,3))
          aryinfo(e)

type : <class 'numpy.ndarray'>
shape : (4, 3)
dimension : 2
dtype : float64
Array Data :
 [[0.0078125 0.0078125 0.0078125]
 [0.0078125 0.0078125 0.0078125]
 [0.0078125 0.0078125 0.0078125]
 [0.0078125 0.0078125 0.0078125]]
```

np.empty는 비어있는 배열의 형태를 만들어준다.
여기서 값이 들어가 있는 것은 numpy 구조상 비어있는 배열을 만든다는 것은 데이터가 초기화되지 않는다라는 뜻이다.
여기 나오는 값은 주소 값이다.

arrange로 numpy 생성하기

```
In [176]: a = np.arange(10)
          aryinfo(a)
```

```
type : <class 'numpy.ndarray'>
shape : (10,)
dimension : 1
dtype : int32
Array Data :
 [0 1 2 3 4 5 6 7 8 9]
```

```
In [178]: a = np.arange(3,21,2)
          aryinfo(a)
```

```
type : <class 'numpy.ndarray'>
shape : (9,)
dimension : 1
dtype : int32
Array Data :
 [ 3  5  7  9 11 13 15 17 19]
```

arrange를 통해 해당 범위까지의 배열을 만든다.

전치 행렬

- 전치 행렬(transpose matirx)이란? 행렬의 행은 열로, 열은 행으로 바꾼 행렬을 의미한다.
- (). T
- transpose opertion

```
In [184]: vec_transpose = vec.reshape(1,10)
          aryinfo(vec_transpose)
```

```
type : <class 'numpy.ndarray'>
shape : (1, 10)
dimension : 2
dtype : int32
Array Data :
 [[0 1 2 3 4 5 6 7 8 9]]
```

위에는 기본적인 np값이 있다.

```
In [185]: vec_transpose = vec.reshape(1,10).T
          aryinfo(vec_transpose)
```

```
type : <class 'numpy.ndarray'>
shape : (10, 1)
dimension : 2
dtype : int32
Array Data :
 [[0]
 [1]
 [2]
 [3]
 [4]
 [5]
 [6]
 [7]
 [8]
 [9]]
```

이것을

. T라는 명령어로 행렬의 행과 열을 바꿔준다. 이렇게 행렬을 전치해준다.

배열의 원소를 순차적으로 접근(for , iterator)

- 배열의 원소를 순차적으로 access 하고자 한다면? 2가지 방법이 있다.
- **for** (Vector, Matrix)
- **iterator**(internext(), finished 속성을 이용해서 ndarray 모든 요소를 순차적으로 접근할 수 있다. 이 방법이 속도가 더 빠르다.)

```
In [187]: arr = np.array([1,2,3,4,5])

In [188]: for tmp in arr:
              print(tmp, end=" ")
1 2 3 4 5

In [190]: for idx in range(len(arr)):
              print(arr[idx],end=" ")
1 2 3 4 5
```

1차원 배열들은 for 문을 통해 접근할 수 있다.

```
In [194]: ite = np.nditer(arr, flags=['c_index'])
          while not ite.finished:# 순환 반복 속성이다.
              print(arr[ite.index], end = " ")
              ite.iternext()
1 2 3 4 5
```

이 문법은 일반적인 for 문보다 연산속도가 더 빠르다.

2차원 배열들에 접근해보자.

```
In [195]: arr = np.array([[1,2,3],[4,5,6]])
          aryinfo(arr)

type : <class 'numpy.ndarray'>
shape : (2, 3)
dimension : 2
dtype : int32
Array Data :
 [[1 2 3]
 [4 5 6]]
```

```
In [207]: # 2차원 배열에 대한 순차적 접근 코드 작성
          # print(arr.shape[0] : row)
          # print(arr.shpae[1]) : col

          for i in range(arr.shape[0]):
              for j in range(arr.shape[1]):
                  print(arr[i,j], end="₩t")
              print("₩n")

1       2       3

4       5       6
```

for 문을 쓴다면 위와 같이 쓰인다.

```
In [209]: ite = np.nditer(arr, flags=['multi_index'])
          while not ite.finished:# 순환 반복 속성이다.
              print(arr[ite.multi_index], end = " ")
              ite.iternext()

1 2 3 4 5 6
```

이걸 **iterator**를 쓰게 된다면 아래와 같은 형태를 가지게 된다.

'Python' 카테고리의 다른 글

[Python] Numpy 배열 합치기(concatenate)

[Python] Numpy의 reshape 통한 차원 변경(재배열)

[Python] Numpy에 있는 다양한 함수 사용법 - 2(전치행렬,zeors,ones, iterat...

[Python] Numpy를 통한 배열 indexing(Boolen indexing, fancy indexing)

[Python] Numpy에 대한 기초 정리와 사용법 정리

[Python] python 에서 Seleium을 통한 동적 크롤링 - 4

Iterator python iterator python ones python zeors python 전치행렬

나무늘보스

혼자 끄적끄적하는 블로그 입니다.