Numpy and Pandas

Data Science Tools 1

Fall 2021

UNIVERSITY of DENVER

Agenda

- Numpy and Pandas
- Review Wk4 Homework
- Please wait in the zoom waiting room in the office hour
- Mid-term is next week

Week 4 Review

- Numpy
- Pandas

Numpy

- Library for vectorized computation
  - What does this mean?
  - Why is it so efficient?

Vectorized - add two arrays	Non-vectorized - add two arrays

`def add_two_vec(a, b):`
`  a = np.array(a)`
`  b = np.array(b)`
`  return a + b`
	`def add_two_slow(a, b):`
	`  c = []`
	`  for i in range(len(a)):`
	`  c.append(a[i] + b[i])`
	`  return c`

extensively in Pandas, SciPy, Matplotlib, scikit-learn, scikit-image and most other data science and scientific Python packages

In order to understand this, we need to point out the stark difference between ndarray and list -

What is the difference?

Numpy- Why is vectorized code fast?

In order to understand this, we need to point out the stark difference between ndarray and list -

Lists are non-homogenous and ndarrays are homogenous

The homogenous nature of ndarrays provides very efficient memory allocation

Numpy- Recap

Instantiation

obj = np.array(list)	
np.arange()	
np.ones(number or shape)	
np.eye()	
np.random.randn()	

Properties

obj.shape	
obj.dtype	

Slicing and boolean indexing

obj[0:2,1:3]	
obj[[obj > 0.5]]	

Mathematical Operation

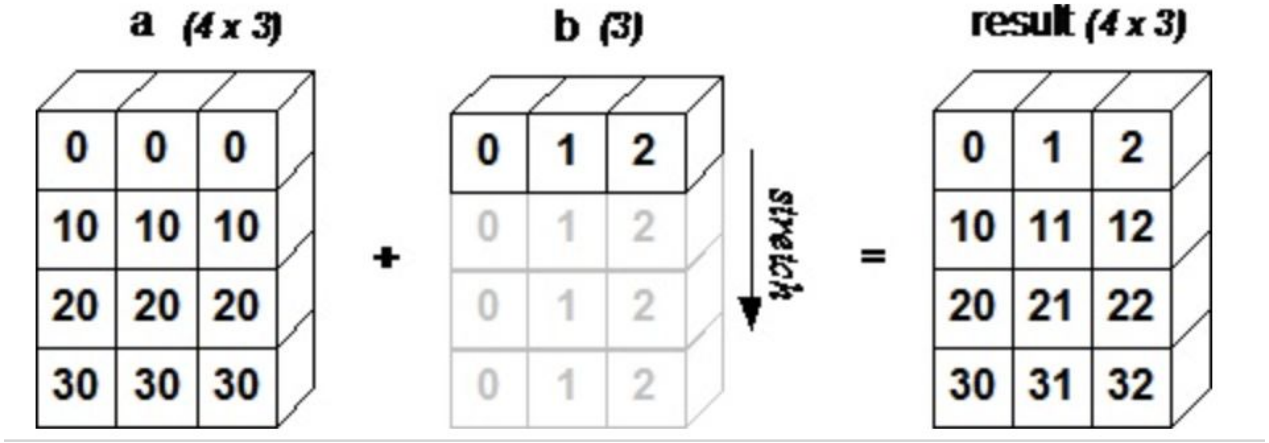
obj1 * obj2	element-wise multiplication
obj ** obj2	element-wise power
np.dot(obj1, obj2)	matrix multiplication
obj.T	transpose
np.where(cond, obj1, obj2)	
np.where(obj>0, 1, -1)	
np.mean(obj, axis=0)	0 = row, 1 = col => [[1,2], [3,4]] = [2, 3]
np.argmax	max index
np.argmin	min index
np.linalg.inv(obj)	inverse
np.linalg.svd(obj)	singular value decomposition

Numpy - what is broadcasting?

Addresses a limitation of arithmetic operation of two arrays

Broadcasting allows arithmetic operation of different shapes and sizes

Broadcasting solves the problem of arithmetic between arrays of differing shapes by in effect replicating the smaller array along the last mismatched dimension



Numpy - what is broadcasting?

```
import numpy as np
a = np.array([1, 2, 3])
b = 2
c = a + b
```

What is c?

[3, 4,5]

Numpy - what is broadcasting?

```
import numpy as np
A = np.array([[1, 2, 3], [1, 2, 3]]) # (2,3)
b = 2
C = A + b
```

What is c?

[[3, 4, 5], [3, 4, 5]]

Numpy - what is broadcasting?

```
import numpy as np
A = np.array([[1, 2, 3], [1, 2, 3]]) # (2,3)
b = np.array([1, 2, 3])
C = A + b
```

What is c?

[[2, 4, 6], [2,4,6]]

Numpy - what is broadcasting?

```
import numpy as np
A = np.array([[1, 2, 3], [1, 2, 3]])
b = np.array([1, 2])
C = A + b
```

What is c?

Error out

Numpy - what is broadcasting?

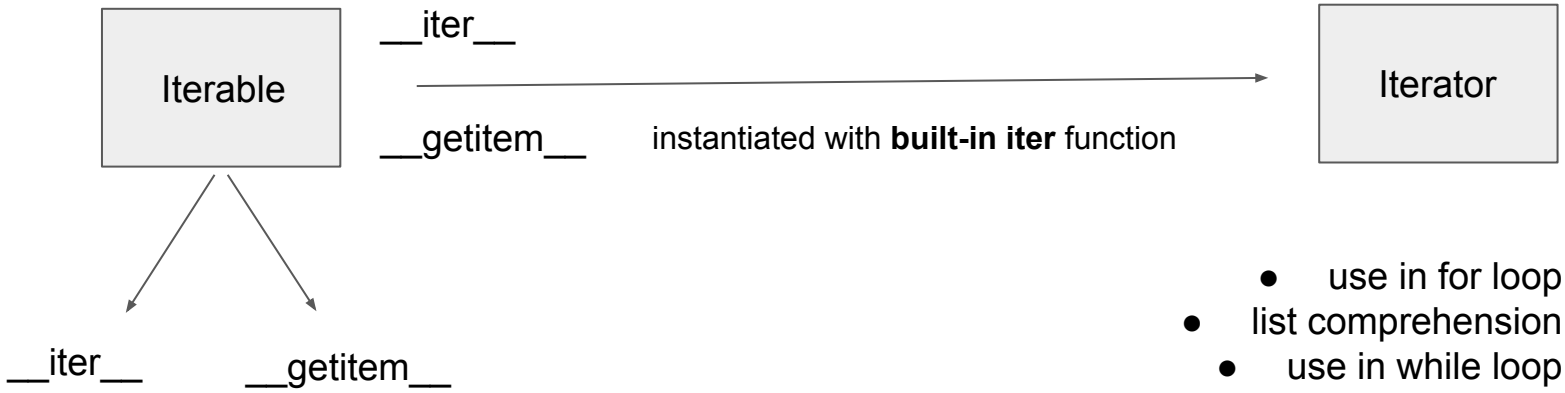
```
import numpy as np
A = np.array([[1, 2, 3], [1, 2, 3]])
b = np.array([1, 2]) # b.shape => (2,)
C = A + b
```

What is c?

Broadcasting can only be performed when the shape of each dimension in the arrays are equal or one has the dimension size of 1

Note: for broadcasting purpose, 1-d ndarray can be thought of as a row vector

- Iterable
- Iterators
- Generators



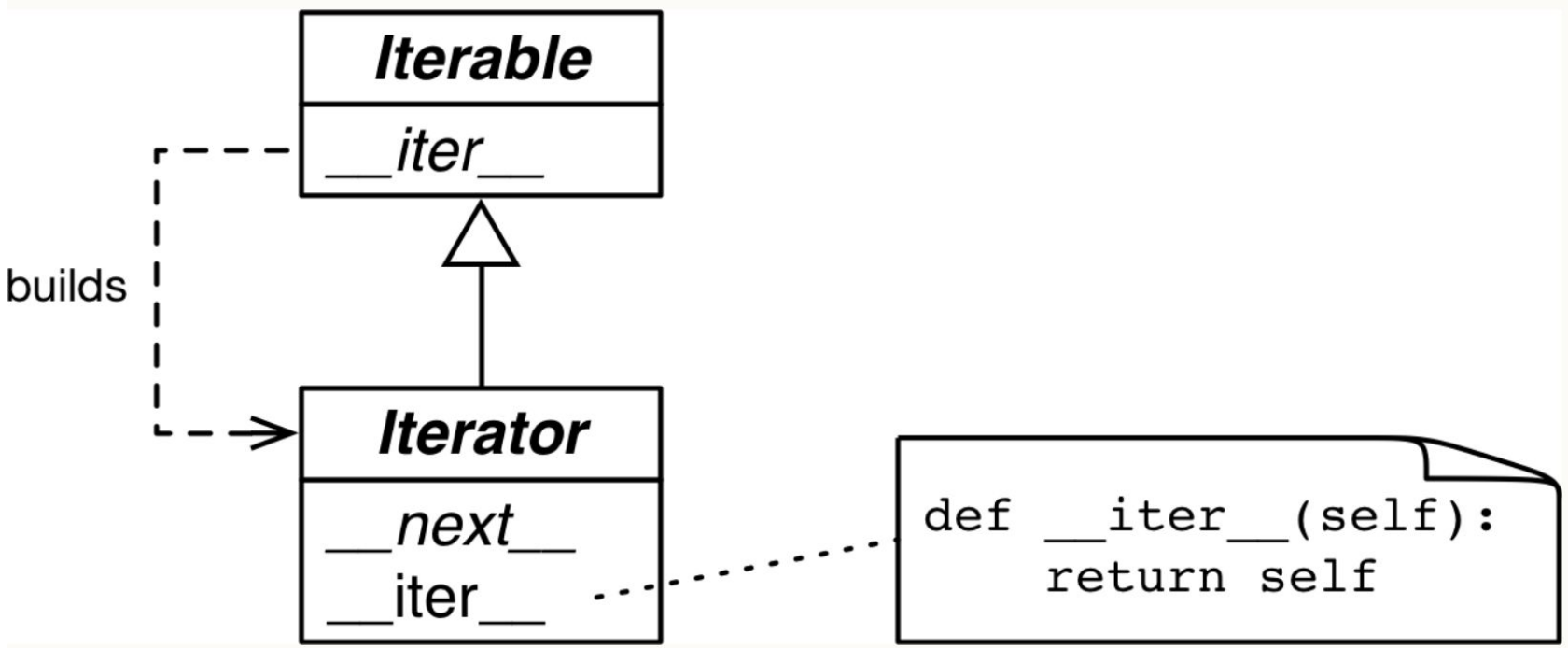
- use in for loop
- list comprehension
- use in while loop
- call next(iterator)
- The iterator raises StopIteration when there are no further items

```
import re
import reprlib

RE_WORD = re.compile(r'\w+')


class Sentence:

    def __init__(self, text):
        self.text = text
        self.words = RE_WORD.findall(text)

    def __repr__(self):
        return f'Sentence({reprlib.repr(self.text)})'

    def __getitem__(self, index):
        return self.words[index]

    def __len__(self):
         return len(self.words)
```

```
s = Sentence("Hello world")
it = iter(s)
for i in it:
    print(i)
```

```
import re
import reprlib

RE_WORD = re.compile(r'\w+')


class Sentence:

    def __init__(self, text):
        self.text = text
        self.words = RE_WORD.findall(text)

    def __repr__(self):
        return f'Sentence({reprlib.repr(self.text)})'

    def __iter__(self):
        return SentenceIterator(self.words)
```

```
class SentenceIterator:

    def __init__(self, words):
        self.words = words
        self.index = 0

    def __next__(self):
        try:
            word = self.words[self.index]
        except IndexError:
            raise StopIteration()
        self.index += 1
        return word

    def __iter__(self):
        return self
```



Therefore, iterators are also iterable, but iterables are not iterators.

```
import re
import reprlib

RE_WORD = re.compile(r'\w+')


class Sentence:

    def __init__(self, text):
        self.text = text
        self.words = RE_WORD.findall(text)

    def __repr__(self):
        return 'Sentence(%s)' % reprlib.repr(self.text)

    def __iter__(self):
        for word in self.words:
                yield word
```

Python Generators

Definition:

a short-cut way to defining an iterator. All you need to do is define a function with at least 1 call to ***yield*** *and now when you call that function it will return "****something****" which will act like an iterator (you can call* ***next*** *method and use it in a* ***for*** *loop)*

```
>>> def gen_123():
...     yield 1
...     yield 2
...     yield 3
...
>>> gen_123  # doctest: +ELLIPSIS
<function gen_123 at 0x...>
>>> gen_123()   # doctest: +ELLIPSIS
<generator object gen_123 at 0x...>
>>> for i in gen_123():
...     print(i)
1
2
3
>>> g = gen_123()
>>> next(g)
1
>>> next(g)
2
>>> next(g)
3
>>> next(g)
Traceback (most recent call last):
  ...
    StopIteration
```

```
>>> def gen_AB():
...     print('start')
...     yield 'A'
...     print('continue')
...     yield 'B'
...     print('end.')
...
>>> for c in gen_AB():
...     print('-->', c)
...
start
--> A
continue
--> B
end.
>>>
```

5 mins Break

Homework 4 review

- Twitter API needs developer verification, so start early
- Announcement
  - Late submissions are not graded
  - One exception can be provided upon 48 hours prior notice

What is Pandas?

Wrapper around ndarray with more information such as column names, index etc

What is the difference between ndarray and list?

- ndarray - homogeneous, list - non-homogenous

What is the method to get the underlying ndarray of a pandas dataframe or series?

- df.values()

Pandas

Data Structure

- Series -> 1D
- Dataframe -> 2D

Series

s = pd.Series(ndarray or dict, index=[], name=[])	
s[[‘idx1’, ‘idx2’]]	
s[[s > 0.5]]	

Properties

s.index.name	
s.name	

Dataframe

df = pd.DataFrame(ndarray or dict, index=[], columns=[])	
df.head()	
df.tail()	
df.sample(10)	
df.loc(first arg: row index, sec arg: col names)	
df.loc[df['col1'] > 0.5]	
df.iloc(0, 1)	
apply(axis=[0 or 1])	computation along rows or columns
applymap()	element-wise

I/O

read_csv / to_csv	
read_html / to_html	
read_excel / to_excel	
read_hdf / to_hdf	
many more...	

General

df.describe	
df.info()	
df.isnull()	
df.dropna()	
df.fillna()	

Documentation is your friend!

https://pandas.pydata.org/docs/reference/io.html

Feel the pain!

Class exercise:

Write a python program to do matrix multiplication of two multi-dimensional without using np.dot and check your answer with it in the end:

```
a = [[1, 2, 3],
     [4, 5, 6]]
b = [[7, 8],
     [9, 10],
     [11, 12]]
res = [[-1, -1],
       [-1, -1]]
```