Numpy

Tuesday October 8, 2019

Work together in pairs on these problems, but be sure you each write your own code.

Problem 1:
One of the reasons people use Numpy is for the way it improves processing speed over the standard Python library for operations over arrays like lists. In this problem you'll compare how long it takes to calculate the mean of a set of numbers in an array using three methods: (1) writing your own function; (2) the Python built-in mean function; and (3) numpy's mean function.

Part 1: Write your own mean function
Write a function `def mymean(data):` that takes in an array of numbers (`data`) and calculates the mean of the numbers in `data`. The mean of a set of numbers is defined as the sum of the set divided by the size of the set. Use a `for` loop to step through each element of `data` and compute the sum, while keeping track of the number of elements in data using a counter. Have your function return `sum/counter` which is the mean.

Now, let's test your function to see how long it takes to calculate the mean of an array with 10,000 entries. Generate a numpy array of length 10,000 using the numpy random number generator `test_data = np.random.randn(10000)`. Then, you can use the Jupyter magic command `%timeit` to get a report on how long your function takes to run on average:

```
%timeit mymean(test_data)
```

Next, use a combination of the built-in `sum()` and `len()` functions to calculate the mean. Again, use `%timeit` to see how long this takes to run.

Finally, use the numpy `.mean()` function, along with `%timeit` to get an estimate of computation time.

Problem 2:
Create a 5x5 array with 1's around the border and 0's on the inside (see the desired output below). Some potentially useful information: numpy has both `.zeros()` and `.ones()`. methods. Also, using slicing/indexing will probably be useful here.

```
[[1 1 1 1 1]
 [1 0 0 0 1]
 [1 0 0 0 1]
 [1 0 0 0 1]
 [1 1 1 1 1]]
```

Problem 3:
Create a 10x10 array with a checkerboard pattern (alternating 1's and 0's--see desired output below).

```
[[1 0 1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1 0 1]
 [1 0 1 0 1 0 1 0 1 0]
 [0 1 0 1 0 1 0 1 0 1]]
```

Now, create the reverse checkerboard, where there is a 0 in the `[0,0]` index position instead of a 1.

Hint: remember how indexing works in python: `x[start:stop:step]`.