CMM201_Week6_lab_solved

October 24, 2019

1 Week 6 Laboratory: Introduction to Numpy Arrays

As seen in the lecture, there are plenty of modules in Python that we can import to enhance the existing features. Such is the case of numpy, which is a module that allows us to use vectors and matrices, known onwards as **numpy arrays**. In this laboratory activity we will review the basic feature of numpy and create some basic functions with numpy arrays.

```
In [1]: # Dont forget to import numpy before starting!
        import numpy as np
```

1.1 Creating a numpy array

As you have probably seen during the lecture, creating a numpy array is quite confusing! You need to be very careful with all the parenthesis and brackets that you need to use. For instance, when we create a one-dimensional array (also called a vector), you can do the following:

```
In [2]: np.array([1,2,3])

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

Moreover, you can also create a two-dimensional numpy array (also called a matrix), but in order to do so, you need to define **two lists inside a list inside the *array()* function**!

```
In [3]: # List [1,2,3] and list [4,5,6] go inside a list, and then inside the array function.
        np.array([[1,2,3],[4,5,6]])

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

Keep in mind that to create a matrix, both lists need to be **of the same lenght**, otherwise the numpy array is created as an object containing that contains two lists

```
In [4]: np.array([[1,2,3],[4,5,6,7]])

Out[4]: array([list([1, 2, 3]), list([4, 5, 6, 7])], dtype=object)
```

In most cases, we want numpy arrays to be either vectors or matrices. This is because such structures allow us to perform calculations in data science and business analytics such as querying data, altering data, etc. For instance, assume that we want to create a table where each **row** represents a movie and each **column** represents a ranking (from 1 to 5) given by a different movie critic. This could be put into a numpy array as follows:

```
In [5]: ranks = np.array([[5,4,5],[4,4,3],[5,5,1],[3,4,5]])
        print(ranks)
```

```
[[5 4 5]
 [4 4 3]
 [5 5 1]
 [3 4 5]]
```

In this case, *ranks* contains **four** movies and the rankings of **three** critics.
In case that we want to add a new movie, then we can use the **append** function as follows:

```
In [6]: ranks = np.append(ranks,[[5,5,3]],axis=0)
        print(ranks)
```

```
[[5 4 5]
 [4 4 3]
 [5 5 1]
 [3 4 5]
 [5 5 3]]
```

Notice that the new movie rankings that we want to append to our *ranks* table needs to be enclosed inside a set of two square brackets. What would happen if we don't do this?

```
In [7]: np.append(ranks,[5,5,3],axis=0)
```

```
        ---------------------------------------------------------------------------

        ValueError                                Traceback (most recent call last)

        <ipython-input-7-6a7d4e9b4d35> in <module>()
    ----> 1 np.append(ranks,[5,5,3],axis=0)


        C:\ProgramData\Anaconda3\envs\foo\lib\site-packages\numpy\lib\function_base.py in apper
       5164         values = ravel(values)
       5165         axis = arr.ndim-1
    -> 5166     return concatenate((arr, values), axis=axis)


        ValueError: all the input arrays must have same number of dimensions
```

Also, notice that the third input of the append function indicates the **axis** to append (axis zero means by rows). If you don't specify the axis, then the function will "flatten" your table and add the rankings to the end.

```
In [8]: np.append(ranks.copy(),[[5,5,3]])
```

```
Out[8]: array([5, 4, 5, 4, 4, 3, 5, 5, 1, 3, 4, 5, 5, 5, 3, 5, 5, 3])
```

In this example, we used **ranks.copy()** instead of **ranks** to create a copy of the ranks table and not mess up our original data.

Question: What if we want to add the rankings of a new movie critic? Is there any function in the numpy module that can allow us to include a new column in the *ranks* table? Try to add [1,2,3,4,5] as a new column of the table in the following cell:

```
In [9]: ## OPTION 1: Use this cell to add a new column to ranks using np.append
        ranks = np.append(ranks,np.array([[1],[2],[3],[4],[5]],axis=1)
        print(ranks)

[[5 4 5 1]
 [4 4 3 2]
 [5 5 1 3]
 [3 4 5 4]
 [5 5 3 5]]
```

```
In [9]: ## OPTION 2: Use this cell to add a new column to ranks using c_
        ranks = np.c_[ranks,np.array([1,2,3,4,5])]
        print(ranks)

[[5 4 5 1]
 [4 4 3 2]
 [5 5 1 3]
 [3 4 5 4]
 [5 5 3 5]]
```

Now you should have a table with five movies and four rankings per movie.

```
In [10]: ranks.shape

Out[10]: (5, 4)
```

1.2 Querying and changing data in a numpy array

Numpy arrays allow us to extract and access certain data by using square brackets after the name of the array. For instance, we can print the first ranking of the first movie critic by running the following code:

```
In [11]: ranks[0,0]

Out[11]: 5
```

Once again, keep in mind that Python starts its numeration in zero. Moreover, consider that the first zero corresponds to the row, and the second one to the column.

Question: Use the next cell to print the second movie's third ranking (it should be a 3):

```
In [12]: ## Use this cell to print the second movie's third ranking:
         ranks[1,2]

Out[12]: 3
```

What if a critic changes his mind? Can he access the table and change a ranking? Say movie critic 4 wants to change his 1 ranking in the first movie for a four. We can execute the following code to do so!

```
In [13]: ranks[0,3] = 4
         print(ranks)

[[5 4 5 4]
 [4 4 3 2]
 [5 5 1 3]
 [3 4 5 4]
 [5 5 3 5]]
```

Could we change a ranking for a "float" one? Try to change the first ranking of the fourth movie into a 3.5 (HINT: You need to change the type of the numpy array as it is currently defined for integers only!):

```
In [14]: ## Use this cell to change the first ranking of the fourth movie into a 3.5
         ranks = ranks.astype('float64')
         ranks[3,0] = 3.5
         print(ranks)

[[5.  4.  5.  4. ]
 [4.  4.  3.  2. ]
 [5.  5.  1.  3. ]
 [3.5 4.  5.  4. ]
 [5.  5.  3.  5. ]]
```

As said in the lecture, numpy arrays allow us to perform operations with the data. For instance, we can multiply the numpy array by 2 to double all rankings:

```
In [15]: ranks*2

Out[15]: array([[10.,  8., 10.,  8.],
               [ 8.,  8.,  6.,  4.],
               [10., 10.,  2.,  6.],
               [ 7.,  8., 10.,  8.],
               [10., 10.,  6., 10.]])
```

1.3 Slicing a numpy array

We can also extract the rankings of one movie or one critic. To do so, we use the : symbol to express that we want **ALL** entries. Say that we want to create a variable called *movie0* to store the rankings of the first movie. To do so, we can execute the following code:

```
In [16]: movie0 = ranks[0,:]
         print(movie0, type(movie0),movie0.shape)

[5. 4. 5. 4.] <class 'numpy.ndarray'> (4,)
```

Use the following cell to create a numpy array called *critic1* and store the rankings provided by the second critic. Is critic1 the same type and shape as movie0?

```
In [17]: critic1 = ranks[:,1]
         print(critic1, type(critic1),critic1.shape)

[4. 4. 3. 2.] <class 'numpy.ndarray'> (4,)
```

Notice that even if *movie0* was extracted form a row and *critic1* from a column, they become horizontal vectors. We know this because of the shape (4,), which means that the array has four entries and no columns. Nonetheless sometimes we explicitly want to create numpy arrays that have an explicit number of rows and columns, like a shape (4,1) or (1,4). To do so, we can use the **reshape** command as follows:

```
In [18]: movie0.reshape(1,4) # movie0 now has 1 row and 4 columns

Out[18]: array([[5., 4., 5., 4.]])

In [19]: critic1.reshape(4,1)# critic1 now has 4 row and 1 columns

Out[19]: array([[4.],
                [4.],
                [3.],
                [2.]])
```

We can even reshape into new shapes (as long as these are coherent)

```
In [20]: movie0.reshape(2,2) # The last two numbers become a second row

Out[20]: array([[5., 4.],
                [5., 4.]])

In [21]: movie0.reshape(1,3) # Won't work since we need to accomodate four entries


        ---------------------------------------------------------------------------

        ValueError                                Traceback (most recent call last)
```

```
    <ipython-input-21-2aeae7139408> in <module>()
----> 1 movie0.reshape(1,3) # Won't work since we need to accomodate four entries


    ValueError: cannot reshape array of size 4 into shape (1,3)
```

We can also access to certain "regions" of the table. For instance, with the following command we can show only the rankings of the first two movies:

```
In [22]: ranks[0:2,:]

Out[22]: array([[5., 4., 5., 4.],
                [4., 4., 3., 2.]])
```

Notice that we are doing two things: * Before the comma, we are defining a range from 0 (inclusive) to 2 (exclusive). Therefore, we indicate that we want the row 0 and 1, but not the 2. * After the comma we have put : which means all.

```
In [23]: ## Use the following cell to access only the first two columns of the ranks table
         ranks[:,0:2]

Out[23]: array([[5. , 4. ],
                [4. , 4. ],
                [5. , 5. ],
                [3.5, 4. ],
                [5. , 5. ]])

In [24]: ## Use the following cell to access only the last three movies
         ranks[2:,:]

Out[24]: array([[5. , 5. , 1. , 3. ],
                [3.5, 4. , 5. , 4. ],
                [5. , 5. , 3. , 5. ]])

In [25]: ## Use the following cell to access all rankings from only the first and third movies
         ## Hint: you can use a list to specify that you want row 0 and row 2.
         ranks[[0,2],:]

Out[25]: array([[5., 4., 5., 4.],
                [5., 5., 1., 3.]])

In [26]: ## Use the following cell to access the first, second and third rankings from the fir
         ## You need to get as a result array([[5.,4.,5.],[5.,5.,1.]])
         ranks[[0,2],0:3]

Out[26]: array([[5., 4., 5.],
                [5., 5., 1.]])
```

```
In [27]: ## Use the following cell to access only the first and third rankings from the first
         ## You need to get as a result array([[5.,5.],[5.,1.]])
         ## Hint: You may need to slice twice!
         ranks[[0,2],:][:,[0,2]]

Out[27]: array([[5., 5.],
                [5., 1.]])
```

1.4 Adding strings to numpy arrays

Why don't we add some row and column names to our *ranks* table? With the functions that you have explored so far, you should be able to add an extra row on top of the numpy array which indicates the critic names (e.g. critic0, critic1,...) and an extra column in the left-hand side of your numpy array which indicates the movie names (e.g. movie0, movie1,...)

```
In [28]: ## OPTION 2: Use this cell to update ranks with row and column names with append
         ranks = ranks.astype('object')
         ranks = np.append(np.asarray([['movie0'],['movie1'],['movie2'],['movie3'],['movie4']]
         ranks = np.append([['X','critic0','critic1','critic2','critic3']],ranks,axis=0)
         print(ranks)

[['X' 'critic0' 'critic1' 'critic2' 'critic3']
 ['movie0' 5.0 4.0 5.0 4.0]
 ['movie1' 4.0 4.0 3.0 2.0]
 ['movie2' 5.0 5.0 1.0 3.0]
 ['movie3' 3.5 4.0 5.0 4.0]
 ['movie4' 5.0 5.0 3.0 5.0]]
```

```
In [28]: ## OPTION 1: Use this cell to update ranks with row and column names with c_
         ranks = ranks.astype('object')
         ranks = np.c_[np.asarray(['movie0','movie1','movie2','movie3','movie4']),ranks]
         ranks = np.append([['X','critic0','critic1','critic2','critic3']],ranks,axis=0)
         print(ranks)

[['X' 'critic0' 'critic1' 'critic2' 'critic3']
 ['movie0' 5.0 4.0 5.0 4.0]
 ['movie1' 4.0 4.0 3.0 2.0]
 ['movie2' 5.0 5.0 1.0 3.0]
 ['movie3' 3.5 4.0 5.0 4.0]
 ['movie4' 5.0 5.0 3.0 5.0]]
```