NumPy Snippets

Query and summarize...

Published - 2016-02-17
Dan Patterson

Example 1

So the idea is...

- Obtain the data in any form.
  - Convert it if necessary.
  - Use arcpy.TableToNumPyArray to ensure the appropriate dtypes.*

- Determine the unique records in the table.

- Produce a list of sub-arrays based upon the unique values in the primary field.
  - Determine the desired parameters for each sub-array.
    These can include:
    - sum
    - min
    - mean
    - etc

    You can of course get a count for each sub array.

The rationale is that each record is essentially a unique record. This raises a concern noted by the ** flag. If you simply bring in the table, without excluding the FID/OID name then you will essentially end up with the whole table back. To prevent this, only include the field names in the export to the array. For example:

```
TableToNumPyArray (in_table, field_names, {where_clause},
                   {skip_nulls}, {null_value})
```

Simply specify the in_table name, making sure that the path to the file is correctly specified. If you are working within ArcMap, then you would specify a layer name which allows you to work with selections already made in the table.

In a perfect world, you wouldn't have any null values, but would have provided a class which represents the various incarnations of null that can exist (ie. missing, forgotten, no clue, no observation etc). The field names is where you would limit the fields. So in our example we included three fields.... [c0, c1, c2]. We didn't use a where_clause, so the data are ready to go. The 'a' sub-array in the table consists of 3 records

```
[('a', 50, 4)],
[('a', 15, 5)],
[('a', 35, 2)],
```

It should be readily apparent that the maximum is 50, the minimum is 15. A count on the number of elements in the sub-array would reveal 3 records. If a sum had been added to the for loop, then we would have the total.

Well what is the purpose of using arrays if you are intending to use loops. Arrays are for vectorizing operations.

```
>>> import numpy as np
>>> arr_data = [('a', 50, 4), ('c', 20, 1), ('a', 15, 5), ('e', 40, 4),
                ('a', 35, 2), ('b', 100, 5),('c', 80, 3), ('d', 100, 3), ('e', 60, 2)]
>>>
>>> dt =[('col_0', np.str, 5), ('col_1','<i4'), ('col_2','<i4')])
>>>
>>> a = np.array(arr_data, dtype=dt)
>>>
>>> a.reshape((-1,1))
array([[('a', 50, 4)],
       [('c', 20, 1)],
       [('a', 15, 5)],
       [('e', 40, 4)],
       [('a', 35, 2)],
       [('b', 100, 5)],
       [('c', 80, 3)],
       [('d', 100, 3)],
       [('e', 60, 2)]],
      dtype=[('col_0', 'S5'), ('col_1', '<i4'), ('col_2', '<i4')])

>>> uni_info = np.unique(a, True, True)
>>> vals, idx, inv = uni_info
>>> uni = np.unique(vals['col_0'])
>>> subs = [ vals[vals['col_0']==i] for i in uni ]
>>> for sub in subs:
...     n = len(sub['col_0']);  t = sub['col_0'][0]
...     val_max = np.max(sub['col_1'])
...     val_min = np.min(sub['col_1'])
...     frmt = "type {} -N: {} -max: {} -min: {}\n  -sub {}\n"
...     print(frmt.format(t, n, val_max, val_min, sub))

type a -N: 3 -max: 50 -min: 15
  -sub [('a', 15, 5) ('a', 35, 2) ('a', 50, 4)]

type b -N: 1 -max: 100 -min: 100
  -sub [('b', 100, 5)]

type c -N: 2 -max: 80 -min: 20
  -sub [('c', 20, 1) ('c', 80, 3)]
type d -N: 1 -max: 100 -min: 100
  -sub [('d', 100, 3)]
type e -N: 2 -max: 60 -min: 40
  -sub [('e', 40, 4) ('e', 60, 2)]
>>>
```

Here is the sub-array. The reshape method appears a bit cryptic...for now, it is just a method for fancy formatting the array so it appears in columnar format. The dtype line was created and consists of 1 string field 5 characters wide, 'S5' and two 32 bit (4 byte) integers, '<i4'. The format stuff is discussed elsewhere. It should be of no concern to you in this application of the procedure in any event.

```
>>> sub_a.reshape((-1,1))
array([[('a', 50, 4)],
       [('a', 15, 5)],
       [('a', 35, 2)]],
      dtype=[('col_0', 'S5'), ('col_1', '<i4'), ('col_2', '<i4')])
```

Perform a query if you want, just to get the data for that column out.

```
>>> sub_a['col_1']
array([50, 15, 35], dtype=int32)

Determine its length/size.

>>> len(sub_a['col_1'])
3
```

Get some answers, like the sum, minimum and maximum for example.

```
>>> ans_0 = sub_a['col_1'].sum()
>>> ans_0
100

>>> ans_1 = sub_a['col_1'].min()
>>> ans_1
15

>>> ans_2 = sub_a['col_1'].max()
>>> ans_2
50
```

You are done for now. You can export the whole summary table or parts of it using the NumPyArrayToTable function in arcpy.**

* http://desktop.arcgis.com/en/arcmap/latest/analyze/arcpy-data-access/tabletonumpyarray.htm

** http://desktop.arcgis.com/en/arcmap/latest/analyze/arcpy-data-access/numpyarraytotable.htm

Example 2

Procedure

Take some data from either lists or tables and convert them to a numpy ndarray. The columns in the example that follows specifies field names and data types. If you are converting a featureclass, shapefile or table, then you can specify the fields to bring over for the analysis. In this example, the geometry is of no interest, so we will just bring over the ID, Main, Sub and Counts field.

```
"""
Script:  crosstab_statistics.py
Author:  Dan.Patterson@carleton.ca
Modified: 2016-02-27
Purpose:
  To demonstrate how to determine values (counts, etc) using two fields.
  The values are converted to an ndarray if in list format.
  The unique values based upon the fields of interest are used for the
  classification.
  The above returns the unique combinations in the fields and the indices
  where those combinations occur in the main table.
    Counts are determined from a bincount of the indices.
    Sums, etc  are determined by bincount on another field.
"""
```

```
import numpy as np
data = [[0, 'A', 'a', 10], [1, 'A', 'a', 10],
        [2, 'A', 'b', 2],  [3, 'A', 'c', 3],
        [4, 'C', 'c', 50], [5, 'B', 'a', 4],
        [6, 'B', 'b', 40], [7, 'B', 'c', 5],
        [8, 'C', 'c', 50], [9, 'B', 'a', 4]]
d = [tuple(i) for i in data]
dt = [("ID","int32"),("Main", "S10"),("Sub","S15"),("Counts","int32")]
a = np.array(d, dtype=dt)
```

The array, a, is shown above. The data originated as a list of lists. In order to convert data in that format, you need to provide a list of tuples instead. The regular expression will expedite the conversion for the purposes here.

Tasks

The task is to perform a simple cross-tabulation between the Main and Sub field to determine the number of combinations there are in the dataset and the number in each pairing. As a side trip, the sum of the Counts field for each pairing will also be determined.

The numpy function **unique** is used o obtain two lists

a_u A list of the unique pairings in the array, based upon the **Main** and **Sub** fields

idx The indices in the table where the unique pairings occur.

Two or more repeats of an index indicates there are multiple observation classes for the class.

bin_cnt This is the result of the counts on the idx values.

sum_fld In this instance, **np.bincount** performed a count on the idx values, but performed a weighted count on the values on the **Counts** field.

```
#
# get the unique pairs for the columns a0, a1
a_u, idx = np.unique(a[["Main","Sub"]], return_inverse=True)
a_all = a[idx]
ord = np.argsort(a,order=("Main","Sub"))
b = a[ord]
headers = a.dtype.names
bin_cnt = np.bincount(idx)
sum_fld = np.bincount(idx,weights=a["Counts"])
final = zip(a_u,bin_cnt,sum_fld)
out_arr = np.array(final)

frmt = """
Input array
{}
Field names  {}
Unique combos
- {}
Indices to original
- {}
Final order
{}
Now the real stuff... the bin's then the counts
- {:<25}  : bins
- {:<25}  : counts
- {:^25}  : sum Counts field
And the final array...
{}
"""
#
print(frmt.format(a, headers, a_u, idx, b, idx, bin_cnt, sum_fld, out_arr))
```

The results are as follows:

```
Input array
[(0, 'A', 'a', 10) (1, 'A', 'a', 10) (2, 'A', 'b', 2) (3, 'A', 'c', 3)
 (4, 'C', 'c', 50) (5, 'B', 'a', 4) (6, 'B', 'b', 40) (7, 'B', 'c', 5)
 (8, 'C', 'c', 50) (9, 'B', 'a', 4)]
Field names  ('ID', 'Main', 'Sub', 'Counts')
Unique combos
- [('A', 'a') ('A', 'b') ('A', 'c') ('B', 'a') ('B', 'b') ('B', 'c')
 ('C', 'c')]
Indices to original
- [0 0 1 2 6 3 4 5 6 3]
Final order
[(0, 'A', 'a', 10) (1, 'A', 'a', 10) (2, 'A', 'b', 2) (3, 'A', 'c', 3)
 (5, 'B', 'a', 4) (9, 'B', 'a', 4) (6, 'B', 'b', 40) (7, 'B', 'c', 5)
 (4, 'C', 'c', 50) (8, 'C', 'c', 50)]
Now the real stuff... the bin's then the counts
- [0 0 1 2 6 3 4 5 6 3]      : bins
- [2 1 1 2 1 1 2]            : counts
- [  20.    2.    3.    8.   40.    5.  100.]  : sum Counts field
-
And the final array...
[[('A', 'a') 2 20.0]
 [('A', 'b') 1 2.0]
 [('A', 'c') 1 3.0]
 [('B', 'a') 2 8.0]
 [('B', 'b') 1 40.0]
 [('B', 'c') 1 5.0]
 [('C', 'c') 2 100.0]]
```