numpy_professor

September 19, 2019

1 Exercises about Numpy

```
Author: Jerónimo Arenas García (jeronimo.arenas@uc3m.es)

Notebook version: 1.1 (Sep 20, 2017)

Changes: v.1.0 (Mar 15, 2016) - First version
         v.1.1 (Sep 20, 2017) - Compatibility with python 2 and python 3
                                Display messages in English

Pending changes:
    * Add a section 7.4. representing f_poly as a function of x
```

[ ]:

```
# Import some libraries that will be necessary for working with data and␣
 ↪displaying plots

import numpy as np
import hashlib

# Test functions

def hashstr(str1):
    """Implements the secure hash of a string"""
    return hashlib.sha1(str1).hexdigest()

def test_arrayequal(x1, x2, err_msg, ok_msg='Test passed'):
    """Test if all elements in arrays x1 and x2 are the same item by item
    :param x1: First array for the comparison
    :param x2: Second array for the comparison
    :param err_msg: Display message if both arrays are not the same
    :param ok_msg: Display message if arrays are the same (optional)
    """
    try:
        np.testing.assert_array_equal(x1, x2)
        print(ok_msg)
    except:
        print(err_msg)
```

```
def test_strequal(str1, str2, err_msg, ok_msg='Test passed'):
    """Test if str1 and str2 are the same string
    :param str1: First string for the comparison
    :param str2: Second string for the comparison
    :param err_msg: Display message if both strings are not the same
    :param ok_msg: Display message if strings are the same (optional)
    """
    try:
        np.testing.assert_string_equal(str1, str2)
        print(ok_msg)
    except:
        print(err_msg)

def test_hashedequal(str1, str2, err_msg, ok_msg='Test passed'):
    """Test if hashed(str1) and str2 are the same string
    :param str1: First string for the comparison
                 str1 will be hashed for the comparison
    :param str2: Second string for the comparison
    :param err_msg: Display message if both strings are not the same
    :param ok_msg: Display message if strings are the same (optional)
    """
    try:
        np.testing.assert_string_equal(hashstr(str1), str2)
        print(ok_msg)
    except:
        print(err_msg)
```

This notebook reviews some of the Python modules that make it possible to work with data structures in an easy an efficient manner. We will review Numpy arrays and matrices, and some of the common operations which are needed when working with these data structures in Machine Learning.

1.1 1. Create numpy arrays of different types

The following code fragment defines variable `x` as a list of 4 integers, you can check that by printing the type of any element of `x`. Use python command `map()` to create a new list with the same elements as x, but where each element of the list is a float. Note that, since in Python 3 `map()` returns an iterable object, you need to call function `list()` to populate the list.

[ ]:
```
x = [5, 4, 3, 4]
print(type(x[0]))

# Create a list of floats containing the same elements as in x
# x_f = list(map(<FILL IN>))
x_f = list(map(float, x))
```

[ ]:
```
test_arrayequal(x, x_f, 'Elements of both lists are not the same')
if ((type(x[-2])==int) & (type(x_f[-2])==float)):
    print('Test passed')
else:
    print('Type conversion incorrect')
```

Numpy arrays can be defined directly using methods such as `np.arange()`, `np.ones()`, `np.zeros()`, as well as random number generators. Alternatively, you can easily generate them from python lists (or lists of lists) containing elements of numeric type.

You can easily check the shape of any numpy vector with the property `.shape`, and reshape it with the method `reshape()`. Note the difference between 1-D and N-D numpy arrays (`ndarrays`). You should also be aware of the existence of another numpy data type: Numpy matrices (http://docs.scipy.org/doc/numpy-1.10.1/reference/generated/numpy.matrix.html) are inherently 2-D structures where operators `*` and `**` have the meaning of matrix multiplication and matrix power.

In the code below, you can check the types and shapes of different numpy arrays. Complete also the exercise where you are asked to convert a unidimensional array into a vector of size 4×2.

[ ]:
```
# Numpy arrays can be created from numeric lists or using different numpy methods
y = np.arange(8)+1
x = np.array(x_f)

# Check the different data types involved
print('Variable x_f is of type', type(x_f))
print('Variable x is of type ', type(x))
print('Variable y is of type', type(y))

# Print the shapes of the numpy arrays
print('Variable y has dimension', y.shape)
print('Variable x has dimension', x.shape)

#Complete the following exercises
# Convert x into a variable x_matrix, of type `numpy.matrixlib.defmatrix.matrix`␣
 ↪using command
# np.matrix(). The resulting matrix should be of dimensions 4x1
# x_matrix = <FILL IN>
x_matrix = np.matrix(x).T

# Convert x into a variable x_array, of type `ndarray`, and shape (4,1)
# x_array = <FILL IN>
x_array = x[:,np.newaxis]

# Reshape array y into a numpy array of shape (4,2) using command np.reshape()
# y = <FILL IN>
y = y.reshape((4,2))
```

[ ]:
```
test_strequal(str(type(x_matrix)), "<class 'numpy.matrixlib.defmatrix.matrix'>",␣
 ↪'x_matrix is not defined as a matrix')
test_hashedequal(x_matrix.tostring(),␣
 ↪'1215ced5d82501bf03e04b30f16c45a4bdcb8838', 'Incorrect variable x_matrix')
test_strequal(str(type(x_array)), "<class 'numpy.ndarray'>", 'x_array is not␣
 ↪defined as numpy ndarray')
test_hashedequal(x_array.tostring(), '1215ced5d82501bf03e04b30f16c45a4bdcb8838',␣
 ↪'Incorrect variable x_array')
test_strequal(str(type(y)), "<class 'numpy.ndarray'>", 'y is not defined as a␣
 ↪numpy ndarray')
test_hashedequal(y.tostring(), '0b61a85386775357e0710800497771a34fdc8ae5',␣
 ↪'Incorrect variable y')
```

Some other useful Numpy methods are:

- `np.flatten()`: converts a numpy array or matrix into a vector by concatenating the elements in the different dimension. Note that the result of the method keeps the type of the original variable, so the result is a 1-D `ndarray` when invoked on a numpy array, and a numpy matrix (and necessarily 2-D) when invoked on a matrix.
- `np.tolist()`: converts a numpy array or matrix into a python list.

These uses are illustrated in the code fragment below.

[ ]:
```
print('Applying flatten() to matrix x_matrix (of type matrix)')
print('x_matrix.flatten():', x_matrix.flatten())
print('Its type:', type(x_matrix.flatten()))
print('Its dimensions:', x_matrix.flatten().shape)

print('\nApplying flatten() to matrix y (of type ndarray)')
print('y.flatten():', y.flatten())
print('Its type:', type(y.flatten()))
print('Its dimensions:', y.flatten().shape)

print('\nApplying tolist() to x_matrix (of type matrix) and to the 2D vector y␣
 ↪(of type ndarray)')
print('x_matrix.tolist():', x_matrix.tolist())
print('y.tolist():', y.tolist())
```

1.2 2. Products and powers of numpy arrays and matrices

- `*` and `**` when used with **Numpy arrays** implement **elementwise** product and exponentiation
- `*` and `**` when used with **Numpy matrices** implement **matrix** product and exponentiation
- Method np.dot() implements matrix multiplication, and can be used both with numpy arrays and matrices.

So you have to be careful about the types you are using for each variable

[ ]:
```
# Try to run the following command on variable x_matrix, and check what happens
print(x_array**2)
```

[ ]:
```
print('Remember that the shape of x_array is', x_array.shape)
print('Remember that the shape of y is', y.shape)

# Complete the following exercises. You can print the partial results to␣
 ↪visualize them

# Multiply the 2-D array `y` by 2
# y_by2 = <FILL IN>
y_by2 = y * 2

# Multiply each of the columns in `y` by the column vector x_array
# z_4_2 = <FILL IN>
z_4_2 = x_array * y

# Obtain the matrix product of the transpose of x_array and y
# x_by_y = <FILL IN>
x_by_y = x_array.T.dot(y)

# Repeat the previous calculation, this time using x_matrix (of type numpy␣
 ↪matrix) instead of x_array
# Note that in this case you do not need to use method dot()
# x_by_y2 = <FILL IN>
x_by_y2 = x_matrix.T * y

# Multiply vector x_array by its transpose to obtain a 4 x 4 matrix
#x_4_4 = <FILL IN>
x_4_4 = x_array.dot(x_array.T)

# Multiply the transpose of vector x_array by vector x_array. The result is the␣
 ↪squared-norm of the vector
#x_norm2 = <FILL IN>
x_norm2 = x_array.T.dot(x_array)
```

[ ]:
```
test_hashedequal(y_by2.
 ↪tostring(),'1b54af8620657d5b8da424ca6be8d58b6627bf9a','Incorrect result for␣
 ↪variable y_by2')
test_hashedequal(z_4_2.
 ↪tostring(),'0727ed01af0aa4175316d3916fd1c8fe2eb98f27','Incorrect result for␣
 ↪variable z_4_2')
test_hashedequal(x_by_y.
 ↪tostring(),'b33f700fec2b6bd66e76260d31948ce07b8c15d3','Incorrect result for␣
 ↪variable x_by_y')
test_hashedequal(x_by_y2.
 ↪tostring(),'b33f700fec2b6bd66e76260d31948ce07b8c15d3','Incorrect result for␣
 ↪variable x_by_y2')
```

```
test_hashedequal(x_4_4.
  ↪tostring(),'832c97cc2d69298287838350b0bae66deec58b03','Incorrect result for␣
  ↪variable x_4_4')
test_hashedequal(x_norm2.
  ↪tostring(),'33b80b953557002511474aa340441d5b0728bbaf','Incorrect result for␣
  ↪variable x_norm2')
```

1.3 3. Numpy methods that can be carried out along different dimensions

Compare the result of the following commands:

[ ]:
```
print(z_4_2.shape)
print(np.mean(z_4_2))
print(np.mean(z_4_2,axis=0))
print(np.mean(z_4_2,axis=1))
```

Other numpy methods where you can specify the axis along with a certain operation should be carried out are:

- `np.median()`
- `np.std()`
- `np.var()`
- `np.percentile()`
- `np.sort()`
- `np.argsort()`

If the axis argument is not provided, the array is flattened before carriying out the corresponding operation.

1.4 4. Concatenating matrices and vectors

Provided that the necessary dimensions fit, horizontal and vertical stacking of matrices can be carried out with methods `np.hstack()` and `np.vstack()`.

Complete the following exercises to practice with matrix concatenation:

[ ]:
```
# Previous check that you are working with the right matrices
test_hashedequal(z_4_2.
  ↪tostring(),'0727ed01af0aa4175316d3916fd1c8fe2eb98f27','Incorrect result for␣
  ↪variable z_4_2')
test_hashedequal(x_array.tostring(), '1215ced5d82501bf03e04b30f16c45a4bdcb8838',␣
  ↪'Incorrect variable x_array')

# Vertically stack matrix z_4_2 with itself
# ex1_res = <FILL IN>
ex1_res = np.vstack((z_4_2,z_4_2))

# Horizontally stack matrix z_4_2 and vector x_array
# ex2_res = <FILL IN>
ex2_res = np.hstack((z_4_2,x_array))
```

```
# Horizontally stack a column vector of ones with the result of the first␣
↪exercise (variable ex1_res)
# X = <FILL IN>
X = np.hstack((np.ones((8,1)),ex1_res))
```

[ ]:
```
test_hashedequal(ex1_res.
↪tostring(),'e740ea91c885cdae95499eaf53ec6f1429943d9c','Wrong value for␣
↪variable ex1_res')
test_hashedequal(ex2_res.
↪tostring(),'d5f18a630b2380fcae912f449b2a87766528e0f2','Wrong value for␣
↪variable ex2_res')
test_hashedequal(X.tostring(),'bdf94b49c2b7c6ae71a916beb647236918ead39f','Wrong␣
↪value for variable X')
```

1.5 5. Slicing

Particular elements of numpy arrays (both unidimensional and multidimensional) can be accessed using standard python slicing. When working with multidimensional arrays, slicing can be carried out along the different dimensions at once

[ ]:
```
# Keep last row of matrix X
# X_sub1 = <FILL IN>
X_sub1 = X[-1,]

# Keep first column of the three first rows of X
# X_sub2 = <FILL IN>
X_sub2 = X[:3,0]

# Keep first two columns of the three first rows of X
# X_sub3 = <FILL IN>
X_sub3 = X[:3,:2]

# Invert the order of the rows of X
# X_sub4 = <FILL IN>
X_sub4 = X[::-1,:]
```

[ ]:
```
test_hashedequal(X_sub1.
↪tostring(),'51fb613567c9ef5fc33e7190c60ff37e0cd56706','Wrong value for␣
↪variable X_sub1')
test_hashedequal(X_sub2.
↪tostring(),'12a72e95677fc01de6b7bfb7f62d772d0bdb5b87','Wrong value for␣
↪variable X_sub2')
test_hashedequal(X_sub3.
↪tostring(),'f45247c6c31f9bcccfcb2a8dec9d288ea41e6acc','Wrong value for␣
↪variable X_sub3')
```

```
test_hashedequal(X_sub4.
↪tostring(),'1fd985c087ba518c6d040799e49a967e4b1d433a','Wrong value for␣
↪variable X_sub4')
```

Extracting columns and rows from multidimensional arrays Something to be aware of when extracting rows or columns from numpy arrays is that if you specify just the index of the row or column you want to extract, the result will be a 1-D numpy array in any case. For instance, the following code prints the second column and third row of the numpy array X, and shows its dimensions. Notice that in both cases you get arrays with 1 dimension only.

[ ]:
```
X_col2 = X[:,1]
X_row3 = X[2,]

print('Matrix X is\n', X)
print('Second column of matrix X:', X_col2, '; Dimensions:', X_col2.shape)
print('Third row of matrix X:', X_row3, '; Dimensions:', X_row3.shape)
```

If you wish that the extracted row or column is still a 2-D row or column vector, it is important to specify an interval instead of a single value, even if such interval consists of just one value.

Many numpy functions will also return 1-D vectors. It is important to be aware of such behavior to avoid and detect bugs in your code that may give place to undesired behaviors.

[ ]:
```
X_col2 = X[:,1:2]
X_row3 = X[2:3,]

print('Second column of matrix X:', X_col2, '; Dimensions:', X_col2.shape)
print('Third row of matrix X:', X_row3, '; Dimensions:', X_row3.shape)
```

1.6 6. Matrix inversion

Non singular matrices can be inverted with method `np.linalg.inv()`. Invert square matrices $X \cdot X^\top$ and $X^\top \cdot X$, and see what happens when trying to invert a singular matrix. The rank of a matrix can be studied with method `numpy.linalg.matrix_rank()`.

[ ]:
```
print(X.shape)
print(X.dot(X.T))
print(X.T.dot(X))

print(np.linalg.inv(X.T.dot(X)))
#print np.linalg.inv(X.dot(X.T))
```

1.7 7. Exercises

In this section, you will complete three exercises where you will carry out some common operations when working with data structures. For this exercise you will work with the 2-D numpy array `X`, assuming that it contains the values of two different variables for 8 data patterns. A first column of ones has already been introduced in a previous exercise: