2020

Lab 4: Data Preparation- NumPy Library

ANACONDA®

Dr. Mohammed Al-Sarem

Taibah University, Information System Department

2/15/2020

Lab 4: Data Preparation- NumPy Library

Lab Objectives:

Data mining tasks aim at extracting hidden information from the data. According to CRISP-DM, this process goes through well-defined steps. Data preparation is the most time-consuming and labor-intensive task. In this lab you will first get familiar with NumPy library and it functionalities and then get used to load dataset into Python Jupyter from scratch, including.

- How to load a CSV file?
- How to do basic statistical analysis?

Methodology

In this lab we will use the Iris Flower Species Dataset. This dataset involves the prediction of iris flower species. Your task till now is to download the dataset and save it into your current working directory with the filename **iris.csv** (details how to download the dataset and where you find it will be discussed throughout the lab).

In class task:

At the end of this lab, the student will be able to:

- Explore Numpy library and its functionality
- Load dataset to Python Jupyter.
- Get some basic statistical analysis using numpy methods

home task:

References:

- CSV File Reading and Writing, The Python Standard Library https://docs.python.org/2/library/csv.html
- CSV file format in RFC 4180: Common Format and MIME Type for Comma-Separated Values (CSV) https://tools.ietf.org/html/rfc4180
- NumPy library: https://numpy.org/

Lab 4: Data Preparation- NumPy Library

This tutorial is divided into 2 parts:

- Exploring briefly the NumPy library functionality.
- Working on the dataset.

1. NumPy Library

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

1.1 Getting Started

NumPy package is installed by default when you download anaconda. In case that is not properly installed, we strongly recommend using a *scientific Python distribution.*

- To check either the package is installed, lunch your Jupyter and write the following code:

```
In [ ]: import numpy as np
        a = np.arange(15).reshape(3, 5)
        a
```

```
Out[1]: array([[ 0,  1,  2,  3,  4],
               [ 5,  6,  7,  8,  9],
               [10, 11, 12, 13, 14]])
```

- It is expected to have the following:

Once, you get a result like the above, it means that the *NumPy* Package is properly installed and you can move ahead in reading this tutorial.

1.1. Arrays

In many algorithms, data can be represented mathematically as a *vector* or a *matrix*. Conceptually, a vector is just a list of numbers and a matrix is a two-dimensional list of numbers (a list of lists). However, even basic linear algebra operations like matrix multiplication are cumbersome to implement and slow to execute when data is stored this way. The *NumPy* module1 offers a much better solution. The basic object in NumPy is the array, which is conceptually similar to a matrix. The NumPy array class is called **ndarray** (for "n-dimensional array"). The simplest way to explicitly create a 1-D ndarray is to define a list, then cast that list as an ndarray with NumPy's ***array()*** function.

```
import numpy as np
# Create a 1-D array by passing a list into NumPy's array() function.
print(np.array([8, 4, 6, 0, 2]))
```

```
[2 0 6 4 8]
```

```
:[6] In
```

- The alias "***np***" is standard in the Python community
- An ndarray can have arbitrarily many dimensions. A 2-D array is a 1-D array of 1-D arrays

- (like a list of lists), a 3-D array is a 1-D array of 2-D arrays (a list of lists of lists), and, more generally, an n-dimensional array is a 1-D array of (n - 1)-dimensional arrays (a list of lists of lists of lists...).
- Each dimension is called an *axis*.
- For a 2-D array, the 0-axis indexes the rows and the 1-axis indexes the columns. Elements are accessed using brackets and indices, with the axes separated by commas.

```
print('Create a 2-D array by passing a list of lists into array().')
A = np.array( [ [1, 2, 3],[4, 5, 6] ] )
print(A)

print('Access elements of the array with brackets.')
print(A[0, 1], A[1, 2])

print('The elements of a 2-D array are 1-D arrays.')
print(A[0])
```

```
.Create a 2-D array by passing a list of lists into array()
[[1 2 3]
[4 5 6]]
.Access elements of the array with brackets
6 2
.The elements of a 2-D array are 1-D arrays
[3 2 1]
```

Exercise 1.1: Write a function that defines the following matrices as NumPy arrays. $A = \begin{bmatrix} 3 & -1 & 4 \\ 1 & 5 & -9 \end{bmatrix}$ $B = \begin{bmatrix} 2 & 4 & -5 & 6 \\ -1 & 7 & 9 & 3 \\ 3 & 2 & -7 & -2 \end{bmatrix}$, Return the matrix product AB. Hint: use help() function to see how np.dot() function can be used!

```
def dotproduct():
    A = np.array([[3,-1,4],[1,5,-9]])
    B = np.array([[2,4,-5,6],[-1,7,9,3],[3,2,-7,-2]])
    return np.dot(A,B)
```

Exercise 1.2: Explore how np.ndarray() can be used! Give an example?

```
C = np.ndarray(shape=(2,2), dtype=float, order='F')
```

1.2. Basic Array Operations

NumPy arrays behave differently with respect to the binary arithmetic operators + and * than Pythonlists do. For lists, + concatenates two lists and * replicates a list by a scalar amount (strings also behave this way).

```
print('Addition concatenates lists together')
print([1, 2, 3] + [4, 5, 6])

print('Mutliplication concatenates a list with itself a given number of times')
print([1, 2, 3] * 4)
```

```
Addition concatenates lists together
[1, 2, 3, 4, 5, 6]
Mutliplication concatenates a list with itself a given number of times
[3 ,2 ,1 ,3 ,2 ,1 ,3 ,2 ,1 ,3 ,2 ,1]
```

Lab 4: Data Preparation- NumPy Library

NumPy arrays act like mathematical vectors and matrices: + and * perform component-wise addition or multiplication.

Exercise 1.3: Assume X, Y matrices are given as follows: $X = [3 \quad -4 \quad 1]$ $Y = [5 \quad 2 \quad 3]$, what is the result of X+10? Y*4? X+Y? and X*Y? Write code down!

```
X+10 = [13  6 11]
Y*4 = [20  8 12]
X+Y = [ 8 -2  4]
X*Y = [15 -8  3]
```

1.4. Array Attributes

An ndarray object has several attributes, some of which are listed below.

Attribute	Description
dtype	The type of the elements in the array.
ndim	The number of axes (dimensions) of the array.
shape	A tuple of integers indicating the size in each dimension.
size	The total number of elements in the array.

Exercise 1.4: What is the output of A.ndim, A.shape, A.size? if A = np.array([[1, 2, 3],[4, 5, 6]])

```
A.ndim = 2
A.shape = (2, 3)
A.size = 6
```

Unlike native Python data structures, **all elements of a NumPy array must be of the same data type**. To change an existing array's data type, use the array's *astype()* method.

Exercise 1.5: Given A = np.array([[1, 2, 3],[4, 5, 6]]) what is the current data type of A?_____
Change A type to **float64**

```
current data type is int64
A = np.float64(A)
```

1.5. Data Access

1.5.1 Array Slicing

Indexing for a 1-D *NumPy* array uses the slicing syntax **x[start:stop:step]**. If there is no colon, a single entry of that dimension is accessed. With a colon, a range of values is accessed. For multidimensional arrays, use a comma to separate slicing syntax for each axis.

What does x = np.arange(10) do? generates an array starts from zero which stops at ten whre ~~ten is excluded and stores it~~ in variable called x

Exercise 1.6: Using the code above, what is the output of x[3]?___3_____

x[: 4] [0 1 2 3]

x[4 :] [4 5 6 7 8 9]

x[4:8] [4 5 6 7]

Exercise 1.7: Now, let x= np.array([[0,1,2,3,4],[5,6,7,8,9]]), what is the output of x[1, 2]? 7

x[: , 2 :]? [[2 3 4]
[7 8 9]]

Note: Indexing and slicing operations return a view of the array. Changing a view of an array also changes the original array. In other words, arrays are mutable. To create a copy of an array, use np.copy() or the array's copy() method. Changes to a copy of an array does not affect the original array, but copying an array uses more time and memory than getting a view.

1.5.2 Fancy Indexing

So-called fancy indexing is a second way to access or change the elements of an array. Instead of using slicing syntax, provide either an array of indices or an array of *boolean* values (called a mask) to extract specific elements.

```
x = np.arange(0, 50, 10)
index = np.array([3, 1, 4])
print(x[index])
# A boolean array extracts the elements of 'x' at the same places as 'True'
mask = np.array([True, False, False, True, False])
print(x[mask])
```

```
[40 10 30]
[30 0 ]
```

Fancy indexing is especially useful for extracting or changing the values of an array that meet some sort of criterion. Use comparison operators like < and == to create masks.

```
# Fancy indexing with condition
y = np.arange(10, 20, 2) # Every other integers from 10 to 20.
print(y)
mask = y > 15 # Same as np.array([i > 15 for i in y]).
print (mask)
print(y[mask])

# Change the values of 'y' that are larger than 15 to 100.
y[mask] = 100
print(y)
```

```
[18 16 14 12 10]
[False False False  True  True]
[18 16]
[100 100 14  12  10 ]
```

1.6. Numerical Computing with *NumPy*

1.6.1 Universal Functions

A universal function is one that operates on an entire array element-wise. Universal functions are significantly more efficient than using a loop to operate individually on each element of an array.

Function	Description
abs() or absolute()	Calculate the absolute value element-wise.
exp() / log()	Exponential (ex) / natural log element-wise.
maximum() / minimum()	Element-wise maximum / minimum of two arrays.
sqrt()	The positive square-root, element-wise.
sin(), cos(), tan(), etc.	Element-wise trigonometric operations

1.6.2 Other Array Methods

The np.ndarray class itself has many useful methods for numerical computations.

Method	Return
all()	True if all elements evaluate to True.
any()	True if any elements evaluate to True.
argmax()	Index of the maximum value.

argmin()	Index of the minimum value.
argsort()	Indices that would sort the array.
clip()	restrict values in an array to fit within a given range
max()	The maximum element of the array.
mean()	The average value of the array.
min()	The minimum element of the array.
sort()	Return nothing; sort the array in-place.
std()	The standard deviation of the array.
sum()	The sum of the elements of the array.
var()	The variance of the array.

Demonstration of how the aforementioned method are used will present in section 2 of this lab.

2. Working on Iris.csv file

To work with iris.csv file, we have to ways:

- load data set from https://archive.ics.uci.edu/ml/datasets/Iris
- Use the code below to get the file path in your Anaconda environment. Then pass the path to *numpy.genfromtxt()* function.

```
from sklearn import datasets
# import some data to play with
iris = datasets.load_iris()
print(iris.filename)
```

```
C:\Users\HP\Anaconda3\lib\site-packages\sklearn\datasets\data\iris.csv
```

```
import numpy as np
iris_data = np.genfromtxt('C:\\Users\\HP\\Anaconda3\\lib\\site-packages\\sklearn\\datasets\\data\\iris.csv',
                          delimiter=",", skip_header=1)
print(iris_data)
```

```
[ .0 0.2 1.4 3.5 5.1]]
[ .0 0.2 1.4  .3 4.9]
[ .0 0.2 1.3 3.2 4.7]
[ .0 0.2 1.5 3.1 4.6]
[ .0 0.2 1.4 3.6  .5]
[ .0 0.4 1.7 3.9 5.4]
[ .0 0.3 1.4 3.4 4.6]
[ .0 0.2 1.5 3.4  .5]
[ .0 0.2 1.4 2.9 4.4]
[ .0 0.1 1.5 3.1 4.9]
[ .0 0.2 1.5 3.7 5.4]
[ .0 0.2 1.6 3.4 4.8]
[ .0 0.1 1.4  .3 4.8]
[ .0 0.1 1.1  .3 4.3]
[ .0 0.2 1.2  .4 5.8]
[ .0 0.4 1.5 4.4 5.7]
[ .0 0.4 1.3 3.9 5.4]
[ .0 0.3 1.4 3.5 5.1]
[ .0 0.3 1.7 3.8 5.7]
```

Note: Iris data set has the following attributes:

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class:

-- Iris Setosa
-- Iris Versicolour
-- Iris Virginica

We can use the methods presented in Section 1.6.2 to show the descriptive features of the iris data set. The code below gives the mean value of the first attribute of iris data set.

```
import numpy as np
iris_data = np.genfromtxt('C:\\Users\\HP\\Anaconda3\\lib\\site-packages\\sklearn\\datasets\\data\\iris.csv',
                    delimiter=",", skip_header=1)
#print(iris_data)
print("mean of {} is {}".format(iris.feature_names[0], iris_data[:,0].mean()))
```

```
Mean of sepal length (cm) is 5.843333333333334
```

Exercise 2: Do the same thing for other attributes including the following methods (mean(), std(), var(), max(), and min())? Comment your observation!

```
print("mean of {} is {}".format(iris.feature_names[1],iris.data[:,1].mean()))
print("std of {} is {}".format(iris.feature_names[1],iris.data[:,1].std()))
print("var of {} is {}".format(iris.feature_names[1],iris.data[:,1].var()))
print("max of {} is {}".format(iris.feature_names[1],iris.data[:,1].max()))
print("min of {} is {}".format(iris.feature_names[1],iris.data[:,1].min()))
```

Upload YOUR ANSWER using your account in turnitin/ by email due to the beginning of next meeting

All the Best!!!