Short Course: Python for Data Science

Day 2 Python & Its Scientific Computing Features

Author

Heru Praptono

December 2018

Introduction

This exercise is about to help you get familiar with scietific programming in Python Environment, by doing some given mini-exercises. Please do not skip as this is a **very compulsory** stage for you to get the sense on working with Python.

Goal

You **will** need to be able to write some Python code and utilise its related libraries to solve the problem on **learning from data** (later we call as "data science") tasks. As this is an introductory material, it will be rather a "chestpass" for you to get the idea on its usage.

1 Utilisation of Scientific Python

1.1 Vectorisation

Vectorisation is the way we optimise mathematical manipulation. The central idea is to change primitive loop into matrix operation. This trick will optimise the computation as matrix calculation is internally well deserved by Numpy.

Consider the mathematical equation below:

$$\sigma_{x,y} = \frac{1}{N}\sum_{i=1}^{N}(x-\bar{x})(y-\bar{y}) \tag{1}$$

That is, the formula to calculate covariance between x and y. At the glance, it looks a very simple calculation. It consists of some operations, including:

1. averaging the samples
2. subtracting each sample with the average
3. then multiply with other variable,

on the corresponding dataset. Thus, this will need the primitive loop operation such as

Listing 1: A simple very primitive FOR-LOOP example

```
N = 200000000
mean_x = sum(x)/len(x)
mean_y = sum(x)/len(x)
sig_x_y = 0
for i in range(0,N):
    sig_x_y = sig_x_y + (x[i] - mean_x[i])* (y[i] - mean_y[i])
sig_x_y = sig_x_y/N
```

If we have large N, then, when we use primitive FOR-LOOP calculation, then it would take very long time!

Fortunately, we may be able to represent the mathematical formulation above with matrix operation. Moreover, there have been several scientific computing tools such as for example Python, Matlab (and Octave, of course), and R. So, if we would to optimise the above operation, then we need to change that into matrix form.

In Python, there is a library that enable array and matrix calculation: Numpy.

Listing 2: A simple vectorisation (suppose that we want to create our own cov function)

```
import numpy as np
x = np.asarray(x) #if x is list of random variables
y = np.asarray(y) #if y is list of random variables
mean_x = np.mean(x)
mean_y = np.mean(y)
cov_xy = ((x-mean_x).dot((y-mean_y)))/N
```

and that is it!

Task 1 Now, try to simulate a task below, given a template.

$$y = \frac{1}{\sqrt{(2\pi)^D|\mathbf{\Sigma}|}} e^{-\frac{1}{2}(\mathbf{x}-\bar{\mathbf{x}})'\mathbf{\Sigma}^{-1}(\mathbf{x}-\bar{\mathbf{x}})}. \tag{2}$$

It is a multivariate Gaussian distribution, where $\mathbf{x} \in \mathbb{R}^{4x1}$. The $\mathbf{\Sigma}$ is positive definite matrix, and $|\mathbf{\Sigma}|$ is the determinant of $\mathbf{\Sigma}$ Now we have N=1000 data, so we would like to represent all $\mathbf{x}$ into $\mathbf{X}$, so that $\mathbf{X} \in \mathbb{R}^{1000x4}$. Consequently it yields $\mathbf{y} \in \mathbb{R}^{1000x1}$.

Listing 3: A simple vectorisation exercise

```
import numpy as np

def my_mvn(x):
    #-
    #your code goes here
    #-

N = 1000
D = 4
X = np.random.rand(N,D)
x = X

y = my_mvn(x)
```

1.2 Lambda Function

Lambda function, on the other hand is the way Python simplify ad hoc function, as we have seen that alot of mathematical expression needs to be represented as a function, on each own. This helps give function name as intuitive as possible, following the arguments needed to be provided.

An example of how lambda function works is as follows: Given a mathematical expression as below, create Python function, by utilising lambda function.

$$y = x^2 + 34x \tag{3}$$

The corresponding Python function representation is given by:

Listing 4: A simple function that calculates y

```
#define the function
def calculateY(x):
    y = x**2 + 34 * x

#then call the function
x = 3
y = calculateY(x)
print(y)
```

With lambda function, we can create the function just in one line as follows:

Listing 5: A simple function that calculates y

```
#define the function
```

```python
y = lambda x: x**2 + 34 * x

#then call the function
x = 3
print(y(x))
```

Thus, we do not need to explicitly define function (`def`) on every creation. We may also say that the lambda function is a nameles function, concerning on the output given any input.

Task 1 Now create some lambda function (with vectorisation, if needed) for some mathematical expressions below:

1. $r = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$
2. $y = \frac{\lambda^k e^\lambda}{k!}$

1.3 The use of Random Seed

Random data that is generated by Python is not even fully random, in the scientific perspective. It is rather a pseudorandom: number that is generated with a pseudorandom number generator. A pseudorandom generator is essentially any algorithm for generating seemingly random but still reproducible data.

We often use as it makes subsequent calls in order to make generate random numbers becomes deterministic: input A always produces output B. This very fortunate condition can also be a curse if it is used maliciously.

Task 1 Now, try some pieces of code below, demonstrating how random seed works in our experiments. See what happens!

Listing 6: A simple function that calculates y

```python
#import the library needed
import numpy as np

x1 = np.random.rand(4,1)
y = x1 + 2
print(y)
#what is the value of y?

x1 = np.random.rand(4,1)
y = x1 + 2
print(y)
#now what is the value of y? and why? what is the cause, x or y?

#implement random seed
np.random.seed(123)
y = x1 + 2
print(y)
#what is the value of y?

x1 = np.random.rand(4,1)
y = x1 + 2
print(y)
```

```
    #now what is the value of y? and why?
```

Question: So what is the difference between both two scenario? Why should we need to use random seed?