Object Oriented Programming in Python

Alejandro Ribeiro

When developing Python code for data processing we eventually encounter the need to use object oriented programming. There is nothing inherently difficult about object oriented programming, except that it requires a somewhat convoluted thinking process. We illustrate the idea here with a simple example.

1 Least Squares

We are given observation vectors $\mathbf{y} \in \mathbb{R}^m$ and matrices $\mathbf{A} \in \mathbb{R}^{m \times n}$ and we are asked to find the vector $\mathbf{x} \in \mathbb{R}^n$ that solves the least squares problem

$$\mathbf{x}^* = \underset{\mathbf{x}}{\text{argmin}} \frac{1}{2} \|\mathbf{y} - \mathbf{A}\mathbf{x}\|^2. \tag{1}$$

The solution to this problem is to make $\mathbf{x}^* = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{y}$, which we can verify is true by taking the gradient of (1) and setting it to zero.

In order to solve least squares we write a function that takes $\mathbf{A}$ and $\mathbf{y}$ as inputs, and computes the solution $\mathbf{x}^* = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{y}$ which it returns as an output:

```
import numpy as np

def ls(A, y):
    AT  = np.transpose(A)        # Compute A^T
    ATA = np.matmul(A,AT)        # Compute A^T A
    iATA = np.linalg.inv(ATA)    # Compute (A^T A)^-1
    iATAAT = np.matmul(AT,iATA) # Compute (A^T A)^-1 A^T
```

```
    x = np.matmul(y,iATAAT)      # Compute (A^T A)^{-1} A^T y
    return x
```

Once we have programmed this function we can use it to solve least squares for any matrix-vector pair that we are given. For instance, consider a least squares problem in which the number of columns of `A` is $n = 3$ and the number of rows of `A`, which is also the number of columns of `y`, is $m = 13$. The entries of both are random, drawn from a Gaussian distribution with zero mean and variance 1. The following code generates the matrix `A` and the vector `y` according to this specification

```
m = 13
n = 3
A = np.random.normal(0, 1, size=(n, m))
y = np.random.normal(0, 1, size=(1, m))
```

And we can solve the corresponding least squares problem by invoking the function `ls`:

```
x = ls(A, y)
```

If we want to solve least squares for a different matrix and different vector, we call the function `ls` with different arguments:

```
AnotherA = np.random.normal(0, 1, size=(n, m))
Anothery = np.random.normal(0, 1, size=(1, m))
x = ls(AnotherA, Anothery)
```

There is nothing wrong with using this approach to code the solution of a least squares problem. But in data science circles developers are more accustomed to object oriented programming. This requires that we create objects that instantiate classes where we specify the operations that are to be performed. This results in code that can look weird and complicated but that, most argue, is easier to modify. And while it may look complicated, it is not, in reality that much more complicated than just writing a function.

2 Classes: Attributes and Methods

The first concept to understand is that of a class. A class is an abstract entity that contains *atrtibutes* and *methods*. We will see in the next section that objects are specific instantiations of a class. To solve the least squares problem we introduced in the previous section using object oriented programming, we define a class to store the matrix $\mathbf{A}$ and the vector $\mathbf{y}$ and create a method in the class that solves the least squares problem:

```
import numpy as np

class LeastSquares():

    def __init__(self, A, y):
        self.A = A
        self.y = y

    def solve(self):
        A = self.A
        y = self.y
        AT  = np.transpose(A)        # Compute A^T
        ATA = np.matmul(A,AT)        # Compute A^T A
        iATA = np.linalg.inv(ATA)    # Compute (A^T A)^-1
        iATAAT = np.matmul(AT,iATA) # Compute (A^T A)^-1 A^T
        x = np.matmul(y,iATAAT)      # Compute (A^T A)^-1 A^T y
        return x
```

The class definition contains two methods. The method `__init__` plays a special role in the creation of objects which we will explain soon. At this point, observe how it specifies the attributes that are part of the class. In this specific example, the class contains two attributes, the matrix `A` and the vector `y`. When we define a class, the `__init__` function has to be specified always and `self` has to always be the first parameter of the `__init__` method. This is Python syntax.

The other function, `solve`, is a function proper, which in object oriented programming we call a method. This method returns the solution of the least squares problem `x` associated with the matrix `A` and the vector `y`. The matrix `A` and the vector `y` are not inputs to this function. They are attributes that belong to the class. Further notice that `self` is the first parameter of the `evaluate` method. Any method that is defined in a class

has to take `self` as the first parameter. This is just Python syntax as well.

3 Objects: Concrete Instances of Abstract Classes

The class is an abstract entity with methods that specify how to manipulate its attributes. If we want to *actually* process data, we create a specific instance. This is an object. For example, if we are interested in the same least squares problem we consider in the first section, namely, a random matrix and a random vector with Gaussian entries and dimensions $m = 13$ and $n = 3$, we create the following object as an instance of the class `LeastSquares`,

```
m = 13
n = 3
A = np.random.normal(0, 1, size=(n, m))
y = np.random.normal(0, 1, size=(1, m))
LSobject = LeastSquares(A, y)
```

When creating the object `LSobject` we are implicitly calling the method `LeastSquares.__init__`. In doing so we instantiate the attributes that belong to the object. If we now want to implement the solution of a least squares problem associated with this matrix and this vector we invoke the method `solve` of the object `LSobject`

```
x = LSobject.solve()
```

The creation of an object and the invocation of a method are slightly more complicated than the single invocation of a function. But in the end the difference is minimal.

Whenever the object `LSobject` is referenced in the code, we are referring to the least squares problem associated with the specific matrix `A` and the specific vector `y` that we passed during the creation of the object. If we wanted to have a different least squares problem, we could do so by instantiating another object of the `LeastSquares` class,

```
AnotherA = np.random.normal(0, 1, size=(n, m))
```

```
Anothery = np.random.normal(0, 1, size=(1, m))
AnotherLSobject = ls.LeastSquares(AnotherA, Anothery)
```

If we now want to implement the solution of the least squares for this matrix-vector pair we invoke the `solve` method of this specific object:

```
x = AnotherLSobject.solve()
```

The main difference between the use of classes, as we do in this section, and functions, as we did in the previous one, is one of frame of mind. When writing functions, the parameters that we pass are separate from the function. When writing a class, the parameters and the methods are integral parts of the object. Thinking in terms of objects helps our human brains in several ways. For example, having `A` and `y` encapsulated inside the object `LSobject` while having `AnotherA` and `Anothery` encapsulated inside the object `AnotherLSobject` reduces the likelihood that we mix up `A` and `y` with `AnotherA` and `Anothery`. There are less things to remember. This advantage is minimal in this problem with two attributes and one method. However, it is not difficult to appreciate the advantage of this frame of mind when we have classes with large numbers of attributes and methods. This is particularly advantageous when we want to modify code an old piece of code or when we want to share code with other developers.

In any event, discussing the relative merits of using object oriented programming is outside of scope for us. The main reason for us to use it is that it is customary in data sciences and we will follow custom.

4 Inheritance

A third concept of object oriented programming we have to introduce is inheritance. This is the possibility of defining a "child" class that inherits methods from a "parent" class. As an example, suppose that we intend to create several random Gaussian least squares problems. Thus, instead of generating several matrices and vectors to pass as arguments in the creation of several different objects, it is more convenient to encapsulate the generation of the Gaussian matrix and vector inside of an object. To do that,

create a class `GaussianLeastSquares` which we define as a child of the `LeastSquares` class,

```
class GaussianLeastSquares(LeastSquares):

    def __init__(self, m, n):
        self.A = np.random.normal(0, 1, size=(n, m))
        self.y = np.random.normal(0, 1, size=(n, m))
```

The specification of *theclass* `GaussianLeastSquares` as a child of the class `LeastSquares` is done by making the latter an argument in the `class` statement. The use of inheritance allows us to reuse our hard work in the creation of the `LeastSquares` class. We do not need to specify the `solve` function for the `GaussianLeastSquares` because we are reusing from the parent class `LeastSquares`. We are inheriting, to use the more technical term. If at some point in the future we update the `solve` method in the `LeastSquares` class, that updated method is automatically inherited by the child class.

With this new class, the creation of least squares problem with a random matrix and vector simplifies to the code

```
m = 13
n = 3
LSobject = GaussianLeastSquares(m,n)
```

The code for the evaluation of the solution of the least squares problem is still the same because it has been inherited.

The most important advantage of defining a new class is that modifications to the class will now propagate to all the places where a Gaussian least squares problem is defined. If at some point in the future we decide that variance 2 is more appropriate, it's just a matter of changing the definition of the `GaussianLeastSquares.__init__` method. The change will propagate to all the places where we instantiate an object belonging to the `GaussianLeastSquares` class.

5 Code Links

The code described here can be downloaded from the folder oop_python.zip. This folder contains the following three files:

`least_squares.py`: The class `LeastSquares` and the child class `GaussianLeastSquares` are specified in this file. This is where the important code is written, but the file itself does not perform any computation. The other two files are the ones that are executable.

`least_squares_main.py`: This file instantiates an object of the class `LeastSquares` and executes the `solve` method. The matrix `A` and the vectors `y` and `x` are printed.

`gaussian_least_squares_main.py`: This files instantiates an object of the class `GaussianLeastSquares` and executes the `solve` method inherited from the parent class. We print the matrix `A` and the vectors `y` and `x`.