OOB

January 22, 2021

1 Module 0 - Object-Oriented Programming

A PDF version of this notebook is available at Module 0 - OOB

Using classes (a cornerstone in OOB) provides a way to group information together into a single unit (also known as an object). We can create functions inside a class to have multiple functionalities.

[1]:
```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age
```

We can then **instantiate** the class by assigning it to an object

[2]:
```
p1 = Person("John", 36)
```

Instantiating a class is creating a copy of the class which inherits all class variables and methods.

[3]:
```
p1.age
```

[3]:
```
36
```

[4]:
```
p1.name
```

[4]:
```
'John'
```

Classes are highly used in Machine Learning, specially when developing new algorithms. In this class, we will not necessarily be creating many classes, however it is still important that we understand how object-oriented programming, and classes work.

1.0.1 Example 1

Let's create a class that contains two functions. 1) computes mean and 2) computes the variance of a list

[5]:
```
import numpy as np

class stats:
    def __init__(self, list):
        self.list = list
    def compute_mean(self):
        print(np.mean(self.list))
    def compute_var(self):
        print(np.var(self.list))
```

[6]:
```
## let's instantiate the class by assigning it to the object my_stats. Now␣
↪my_stats has inherited all the functions of the class stats
my_stats = stats([1,2,3,4,5])
```

[7]:
```
my_stats.compute_mean()
```

```
3.0
```

[8]:
```
my_stats.compute_var()
```

```
2.0
```

1.1 An implementation from scratch of Linear Regression

When we are finding the slope and y-intercept estimates in linear regression, we are solving a system of equations. The estimates are solved such that the sums of squared errors (SSE) are minimized.

source: https://towardsdatascience.com/linear-regression-from-scratch-977cd3a1db16

The estimates can be obtained with the equation: $\hat{\theta} = (X^\top X)^{-1}(X^\top y)$. Where X is the design matrix with the first column of ones.

[9]:
```
import numpy as np

class ISA630_LinearRegression:

    def __init__(self):

        self.X = X
        self.y = y

    def fit(self, X, y):

        Xt = np.transpose(X)
        XtX = np.dot(Xt,X) # X transpose X
        Xty = np.dot(Xt,y) # X tranpose y
        self.beta_hat = np.linalg.solve(XtX,Xty) # betahat calculation
        print(f'The y-intercept is {self.beta_hat[0]} and the slope is {self.
↪beta_hat[1]}')
```

The `ISA630_LinearRegression` class has a `fit` method, so let's check it out.

[10]:
```python
## Let's create a design matrix X
X = np.array([[1,1],[1,1],[1,2], [1,2], [1,3]])
X
```

[10]:
```
array([[1, 1],
       [1, 1],
       [1, 2],
       [1, 2],
       [1, 3]])
```

[11]:
```python
## a the response vector y
y = np.array([[2.2, 2.3, 4.0, 3.9, 5.5]]).T
```

[12]:
```python
## let's instantiate the LinearRegression class
my_reg = ISA630_LinearRegression()
```

[13]:
```python
## let's call the fit method
my_reg.fit(X, y)
```

```
The y-intercept is [0.63571429] and the slope is [1.63571429]
```

[14]:
```python
import numpy as np

class ISA630_LinearRegression:

    def __init__(self):

        self.X = X
        self.y = y

    def fit(self, X, y):

        Xt = np.transpose(X)
        XtX = np.dot(Xt,X) # X transpose X
        Xty = np.dot(Xt,y) # X tranpose y
        self.beta_hat = np.linalg.solve(XtX,Xty) # betahat calculation
        print(f'The y-intercept is {self.beta_hat[0]} and the slope is {self.
 ↪beta_hat[1]}')


    def predict(self, X):
        return np.dot(X, self.beta_hat)
```

[15]:
```python
## let's instantiate the ISA630_LinearRegression class again
my_reg = ISA630_LinearRegression()
```

[16]:
```
## Let's call the fit method with X and y
my_reg.fit(X, y)
```

```
The y-intercept is [0.63571429] and the slope is [1.63571429]
```

[17]:
```
## Let's call the predict method on X
my_reg.predict(X)
```

[17]:
```
array([[2.27142857],
       [2.27142857],
       [3.90714286],
       [3.90714286],
       [5.54285714]])
```

1.2 Sklearn's Regression Module

All the ML modules are constructed using object-oriented programming in the same way we built our `ISA630_LinearRegression` class. They contain many more methods and have numerous extra functionalities. We can better understand what goes on under the hood with classes. Let's import sklearn's `linear_model` module. This is a class that contains, among many others, a `fit` and a `predict` method.

[18]:
```
from sklearn import datasets, linear_model
```

[19]:
```
## We also need to instantiate the Sklearn's LinearRegression
regr = linear_model.LinearRegression(fit_intercept = False) ## let's use the␣
 ↪option fit_intercept = False b/c our design matrix already has the first␣
 ↪columns of 1s
```

[20]:
```
## We also call the fit function
regr.fit(X, y)
```

[20]:
```
LinearRegression(copy_X=True, fit_intercept=False, n_jobs=None, normalize=False)
```

[21]:
```
print('Coefficients: \n', regr.coef_)
```

```
Coefficients:
 [[0.63571429 1.63571429]]
```

[22]:
```
## The linear_model class has a predict method
regr.predict(X)
```

[22]:
```
array([[2.27142857],
       [2.27142857],
       [3.90714286],
       [3.90714286],
       [5.54285714]])
```