Assignment 4 Advanced Programming

TRANSFORMERS FOR SCIKIT-LEARN

Authors: Rafaela Becerra & Marta Cortés

Import some libraries:

In [6]:
```
import numpy as np
from numpy.random import randint
import pandas as pd
from sklearn.base import TransformerMixin
```

Create a random matrix with some NA values:

In [7]:
```
data=np.array([1,10,np.NaN,50,4,np.NaN,12,6,-2,np.NaN,22,41,np.NaN,
33,np.NaN,-5,np.NaN,0,-25,-17,1,-1,np.NaN,0,np.NaN,2,5,6,7,2,9,1])

X=(data.reshape(8,4))
X
y=X[:,3]
y
```

Out[7]:
```
array([ 50.,   6.,  41.,  -5., -17.,   0.,   6.,   1.])
```

Defining the new transformer:

Some times you want to preprocess data in order to work better with it and the operation you need to compute is not in the pipeline.

Then, you can create your onwn pre-process, programing your own transformer function.

In this case, we want to replace the NAs with a random number which follows a normal distribution with parameters column mean and column standard deviation of the matrix.

the '.fit' function is the operation that trains the transformer, and the '.transformer' function is the operation that transform the data.

In [8]:
```
class no_nas(TransformerMixin):
    def __init__(self):
        pass

    def fit(self,X,y=None):
        self.mean_ = np.nanmean(X,axis=0)
        self.sd_ = np.nanstd(X,axis=0)
        self.statistics_ = np.random.normal(self.mean_,self.sd_)
        return(self)

    def transform(self, X):
        for j in range(X.shape[1]):
            for i in range(X.shape[0]):
                if(np.isnan(X[i,j])):
                    X[i,j]=self.statistics_[j]
        return(X)
```

Once we have executed the function, we apply it among the matrix defined previously:

In [9]:
```
my_nonas_imputer = no_nas()
my_nonas_imputer = my_nonas_imputer.fit(X,y)

print(my_nonas_imputer.statistics_)

XX = my_nonas_imputer.transform(X)
print(XX)
```

```
[-0.90334705 -4.40937181  9.08445424 57.96168016]
[[  1.          10.           9.08445424  50.        ]
 [  4.          -4.40937181  12.           6.        ]
 [ -2.          -4.40937181  22.          41.        ]
 [ -0.90334705  33.           9.08445424  -5.        ]
 [ -0.90334705   0.         -25.         -17.        ]
 [  1.          -1.           9.08445424   0.        ]
 [ -0.90334705   2.           5.           6.        ]
 [  7.           2.           9.           1.        ]]
```

As we can observed in the result, all of the NAs values have been replaced with the same number for each column.

We can also use the new transformer in a pipeline:

In [10]:
```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.pipeline import Pipeline

knn = KNeighborsRegressor()
qi_knn = Pipeline([('my_nonas_imputer', my_nonas_imputer), ('knn',
knn)])

pipe = qi_knn.fit(X,y)
y_pred = pipe.predict(X)

print(y_pred)
```

```
[20.8 10.8 20.6  1.6 -0.8  1.6  1.6  1.6]
```

Now, We have a vector with the predictions of the missing values.