Carga de Datos

March 31, 2020

0.1 Importacion Datos de OpenML

```
In [11]: from sklearn.datasets import fetch_openml
         iris = fetch_openml(name="iris")
         type(iris)
```

```
Out[11]: sklearn.utils.Bunch
```

0.2 Transformacion de los Datos Importados a DataFrame

Antes un ejemplo del uso de `c_[]` del modulo de `Numpy` que sirve para concatenar vectores verticalmente

```
In [117]: X = np.arange(1,7).reshape(6,1)
          Y = np.arange(7,13).reshape(6,1)
          Z = np.arange(13,25).reshape(2,6).transpose()
          np.c_[x,y,z] #Concatena X, Y y Z
```

```
Out[117]: array([[ 1,  7, 13, 19],
                 [ 2,  8, 14, 20],
                 [ 3,  9, 15, 21],
                 [ 4, 10, 16, 22],
                 [ 5, 11, 17, 23],
                 [ 6, 12, 18, 24]])
```

La importacion de datos de la libreria `fetch_openml` devuelve los datos en formato `sklearn.utils.Bunch` ahora se va a transformar de ese tipo de datos a DataFrame

Para ver el nombre de los atributos del conjunto de datos que se importo `iris`, se puede usar la funcion `keys()` que se usa para acceder a las llaves de un diccionario, ya que `sklearn.utils.Bunch` esta implemetado usando diccionarios.

```
In [24]: iris.keys()
```

```
Out[24]: dict_keys(['data', 'target', 'feature_names', 'DESCR', 'details', 'categories', 'url'])
```

```
In [70]: import numpy as np
         import pandas as pd
         iris_data = pd.DataFrame(data=np.c_[iris.data,iris.target],
                                  columns= iris.feature_names + ['target'])
         iris_data.head()
```

```
Out[70]:   sepallength sepalwidth petallength petalwidth       target
         0         5.1        3.5         1.4        0.2  Iris-setosa
         1         4.9          3         1.4        0.2  Iris-setosa
         2         4.7        3.2         1.3        0.2  Iris-setosa
         3         4.6        3.1         1.5        0.2  Iris-setosa
         4           5        3.6         1.4        0.2  Iris-setosa
```

Ahora para cambiarle el nombre a las columnas del DataFrame se accede al atributo `columns` de `iris_data` y se ingresa una lista con los nombres de las columnas en orden y entre comillas

```
In [79]: iris_data.columns = ['Sepal.Length','Sepal.Width','Petal.Length',
                              'Petal.Width','Species']
         iris_data.head()
```

```
Out[79]:   Sepal.Length Sepal.Width Petal.Length Petal.Width      Species
         0          5.1         3.5          1.4         0.2  Iris-setosa
         1          4.9           3          1.4         0.2  Iris-setosa
         2          4.7         3.2          1.3         0.2  Iris-setosa
         3          4.6         3.1          1.5         0.2  Iris-setosa
         4            5         3.6          1.4         0.2  Iris-setosa
```

Ahora por ejemplo si se quiere cambiar el nombre de los datos del atributo `Species` para que el nombre no sea **Iris-setosa** sino solo **setosa** se puede usar la funcion `replace()` del DataFrame y cambiar los datos. Luego de usar la funcion `replace()` el resultado devuelve un DataFrame con los datos modificados pero no se guarda en la variable original `iris_data`, entonces para que se guarde asignamos el resultado de esta funcion a la variable del dataframe original `iris_data`

```
In [83]: iris_data = iris_data.replace({"Iris-setosa":"setosa",
                                        "Iris-versicolor":"versicolor",
                                        "Iris-virginica":"virginica"})
         iris_data.head()
```

```
Out[83]:    Sepal.Length  Sepal.Width  Petal.Length  Petal.Width Species
         0           5.1          3.5           1.4          0.2  setosa
         1           4.9          3.0           1.4          0.2  setosa
         2           4.7          3.2           1.3          0.2  setosa
         3           4.6          3.1           1.5          0.2  setosa
         4           5.0          3.6           1.4          0.2  setosa
```

Ahora por ejemplo se quiere una muestra de 15 elementos aleatorios del DataFrame `iris_data`, para esto se puede usar la función `sample` del modulo `random`, para generar una muestra aleatoria de 15 numeros entre 0 y 149, para luego acceder al DataFrame usando estos numeros como índices con la función `iloc([[]])`

```
In [100]: import random as rd
          rd.seed(155)
          l = (rd.sample(range(150),15))
          l.sort()
          iris_data.iloc[l]
```

```
Out[100]:      Sepal.Length  Sepal.Width  Petal.Length  Petal.Width     Species
          6             4.6          3.4           1.4          0.3      setosa
          15            5.7          4.4           1.5          0.4      setosa
          22            4.6          3.6           1.0          0.2      setosa
          44            5.1          3.8           1.9          0.4      setosa
          45            4.8          3.0           1.4          0.3      setosa
          49            5.0          3.3           1.4          0.2      setosa
          70            5.9          3.2           4.8          1.8  versicolor
          73            6.1          2.8           4.7          1.2  versicolor
          106           4.9          2.5           4.5          1.7   virginica
          128           6.4          2.8           5.6          2.1   virginica
          134           6.1          2.6           5.6          1.4   virginica
          136           6.3          3.4           5.6          2.4   virginica
          138           6.0          3.0           4.8          1.8   virginica
          146           6.3          2.5           5.0          1.9   virginica
          148           6.2          3.4           5.4          2.3   virginica
```

0.3 Matrices

Creacion de matrices usando `Numpy`.

```
In [1]: import numpy as np

In [122]: M = np.arange(1,10).reshape(3,3)
          M

Out[122]: array([[1, 2, 3],
                 [4, 5, 6],
                 [7, 8, 9]])

In [132]: np.zeros((4,3)) #Matriz de ceros de 4X3

Out[132]: array([[0., 0., 0.],
                 [0., 0., 0.],
                 [0., 0., 0.],
                 [0., 0., 0.]])

In [169]: np.ones((3,5)) #Matriz de unos de 3X5

Out[169]: array([[1., 1., 1., 1., 1.],
                 [1., 1., 1., 1., 1.],
                 [1., 1., 1., 1., 1.]])

In [136]: np.shape(M)

Out[136]: (3, 3)

In [137]: M.shape

Out[137]: (3, 3)
```

Calcular $A \cdot B$ con las siguientes matrices

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad B = \begin{pmatrix} 3 & 0 \\ 1 & -1 \end{pmatrix}$$

```
In [150]: A = np.arange(1,5).reshape(2,2)
          B = np.array([3,0,1,-1]).reshape(2,2)
          A.dot(B)

Out[150]: array([[ 5, -2],
                 [13, -4]])

In [152]: A.transpose() #matriz transpuesta de A

Out[152]: array([[1, 3],
                 [2, 4]])

In [154]: A.trace() #Traza de A

Out[154]: 5

In [160]: np.linalg.matrix_power(A,5) #A*A*A*A*A

Out[160]: array([[1069, 1558],
                 [2337, 3406]])

In [161]: np.linalg.det(A) #Determinante de A

Out[161]: -2.0000000000000004

In [166]: np.linalg.inv(A) #MAtriz inversa de A

Out[166]: array([[-2. ,  1. ],
                 [ 1.5, -0.5]])

In [167]: np.round(A.dot(np.linalg.inv(A)),2) #Comprobación de resultado A*A^-1=I

Out[167]: array([[1., 0.],
                 [0., 1.]])
```