Lección 3. Librerías y entornos de Python

UCO ONLINE

Para ver ejemplos de los métodos estudiados vamos a utilizar Python, que además de verse en otros cursos, dispone de una gran cantidad de librerías para poder aplicar métodos de detección de anomalías sin tener que implementarlos.

Para poder ver los ejemplos, no necesitas tener nada instalado en tu máquina, ya que lo puedes hacer desde *google colaboration* que te proporciona todo lo que vas a necesitar para ver los ejemplos, ejecutarlos y modificarlos. Para su acceso solamente necesitas un navegador, conexión a Internet y una cuenta de google.

En esta sección, os comentaré todo lo que necesitas conocer para poder comprender y ejecutar los métodos que se van a utilizar en las siguientes lecciones, así como llevar a cabo la tarea que se pedirá al final de esta semana.

1. Entorno de programación y ejecución

Como entorno utilizarás *google colaboration*, con lo que no es necesario que tengas ningún entorno específico instalado en tu máquina para poder hacer los ejemplos y ejecutarlos. Solamente necesitas un navegador y una cuenta de google (la misma que usas para Gmail). No obstante, trabajaremos con notebook de Jupiter, con lo que si tienes ya un entorno instalado para poder utilizarlo, también puedes utilizarlo sin problema.

Para acceder a google colaboration, debéis poner en el navegador:

https://colab.research.google.com

Tras indicar seleccionar el enlace, accederás a una página como la que se muestra en la figura 1. Puedes iniciar sesión con tus credenciales de la cuenta de google que utilizas en gmail.

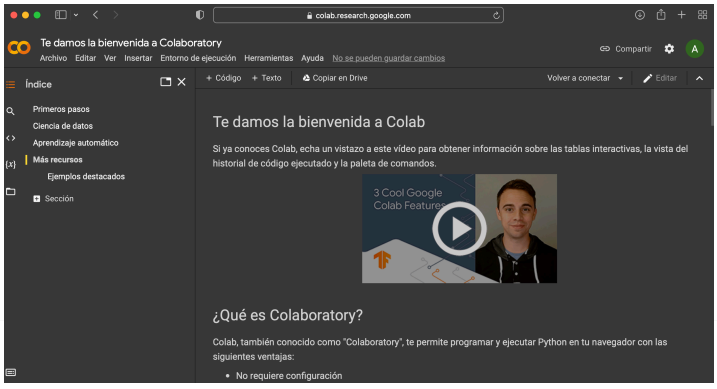


Figura 1. Entorno de Google Colab

2. Librerías de Phyton

No es necesario tener un conocimiento profundo de Python. Comentaremos las librerías y métodos que vamos a usar esta semana, que sería todo lo que necesitas para comprender los ejemplos. No todas las librerías se usan en todos los ejemplos.

2.1 Librería numpy

Es una librería de Python especializada en el cálculo numérico y el análisis de datos, especialmente para un gran volumen de datos. Para un mayor detalle puede acceder a su documentación:

https://numpy.org/

De esta librería usaremos:

- la clase array que permite representar colecciones de datos de un mismo tipo en varias dimensiones, y funciones muy eficientes para su manipulación.
  - np.array (lista) : crea un array a partir de la lista y devuelve una referencia a él. El número de dimensiones del array dependerá de las listas o listas anidadas que se hayan definido.
    - Definimos un array de una dimensión con tres elementos y lo imprimimos por pantalla:
  - np.mean (lista): Calcula la media de los elementos del lista.
  - np.std (lista): Calcula la desviación típica de los elementos de la lista.

```
#Importar librerías
import numpy as np
#Crear un array
a1 = np.array([1, 2, 3])
print(a1)
[1 2 3]
#Calcular la media
np.mean(a1)
>>2.0
#Calcular la desviación típica
np.std(a1)
>> 0.816496580927726
```

2.2 Librería pandas

Es una librería de Python especializada en el manejo y análisis de estructuras de datos. Para un mayor detalle puede acceder a su documentación:

https://pandas.pydata.org/

De esta librería usaremos:

- **dataFrame:** un objeto del tipo DataFrame define un conjunto de datos estructurado en forma de tabla donde cada columna es un objeto de tipo Series, es decir, todos los datos de una misma columna son del mismo tipo, y las filas son registros que pueden contener datos de distintos tipos. Un DataFrame contiene dos índices, uno para las filas y otro para las columnas, y se puede acceder a sus elementos mediante los nombres de las filas y las columnas. Para crear un DataFrame a partir de un array de NumPy se utiliza el siguiente método:
- `dataFrame(data=array, index=filas, columns=columnas, dtype=tipo)`: devuelve un objeto del tipo DataFrame cuyas filas y columnas son las del array array, los nombres de filas indicados en la lista filas, los nombres de columnas indicados en la lista columnas y el tipo indicado en tipo. La lista filas tiene que tener el mismo tamaño que el número de filas del array y la lista columnas el mismo tamaño que el número de columnas del array. Existe varias posibilidades para poder definirlo.

```
#Importar librerías
from numpy import random
import pandas as pd

#Generar un array de 4x3
data_array =
np.array([[27,44,77],[75,65,47],[30,84,86],[18,9,41])
fila_indices = ["fila_1", "fila_2", "fila_3", "fila_4"]
columna_nombres = ["columna_1", "columna_2", "columna_3"]

#Pasar el array a un dataframe
data_df = pd.DataFrame(data_array, index=fila_indices,
columns=columna_nombres)

#Mostrar el dataframe por pantalla
print(data_df)

>>>        columna_1  columna_2  columna_3
>>> fila_1         27         44         77
>>> fila_2         75         65         47
>>> fila_3         30         84         86
>>> fila_4         18          9         41
```

2.3 Librería scipy

Es una librería de Python de código abierto de herramientas y algoritmos matemáticos y que consiste en módulos de extensión para Python. Scipy contiene módulos para optimización y álgebra lineal, entre otros. Su documentación:

https://scipy.org/

De esta librería usaremos:

- **`z-score(data)`** es un método estadístico que ayuda a calcular cuántos valores la desviación estándar se aleja de un valor particular del valor medio.

```
#Importar librerías
import numpy as np
import scipy.stats as stats
#Creamos un array de datos con 11 elementos
input_data = np.array([5, 10, 20, 35, 25, 22, 19, 19, 50, 45,62])
#Calculamos el zscore
stats.zscore(input_data)
>> array([-1.3916106 , -1.09379511, -0.49816411,  0.39528239, -
0.20034861, -0.37903791, -0.55772721, -0.55772721,  1.28872889,
0.99091339, 2.00348608]
```

Tenga en cuenta que cada valor de z-score indica a cuántos valores de desviación estándar se aleja su valor correspondiente del valor medio. Aquí, el signo negativo representa que ese valor es esa cantidad de desviaciones estándar por debajo del valor medio, y el signo positivo representa que ese valor es esa cantidad de desviaciones estándar por encima del valor medio. Si un valor de z-score resulta ser 0, entonces ese valor está a 0 valores de desviación estándar del valor medio.

2.4 Librería Matplotlib

Es una librería de Python especializada en la creación de gráficos. Permite crear y personalizar los tipos de gráficos más comunes, entre ellos:

- Diagramas de barras.
- Histograma.
- Diagramas de sectores.
- Diagramas de caja y bigotes.
- Diagramas de violín.
- Diagramas de dispersión o puntos.
- Diagramas de líneas.
- Diagramas de áreas.
- Diagramas de contorno.
- Mapas de color.

Para crear un gráfico con matplotlib es habitual seguir los siguientes pasos:

1. Importar el módulo pyplot.
2. Definir la figura que contendrá el gráfico, que es la región (ventana o página) donde se dibujará y los ejes sobre los que se dibujarán los datos. Para ello se utiliza la función subplots().
3. Dibujar los datos sobre los ejes. Para ello se utilizan distintas funciones dependiendo del tipo de gráfico que se quiera.
4. Personalizar el gráfico. Para ello existen multitud de funciones que permiten añadir un título, una leyenda, una rejilla, cambiar colores o personalizar los ejes.
5. Mostrar el gráfico. Para ello se utiliza la función show().

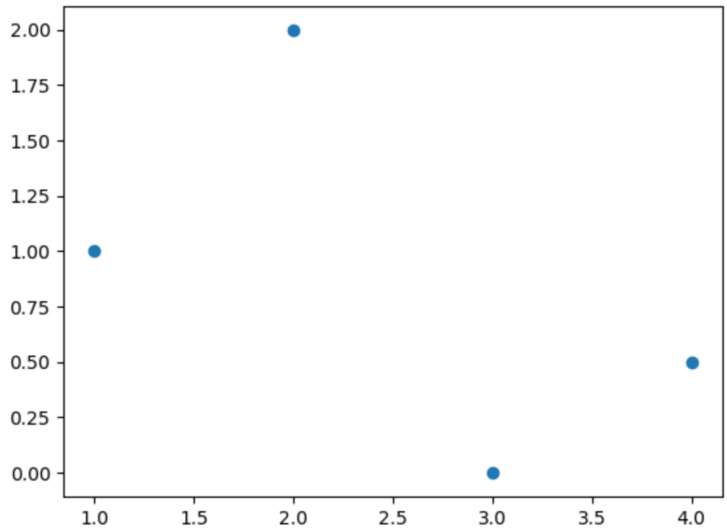
Su documentación está disponible en:

https://matplotlib.org/

De esta librería usaremos:

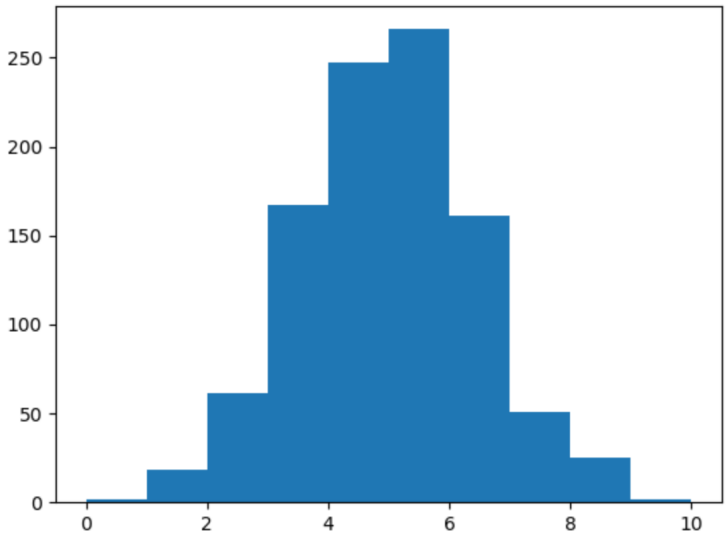
- **`scatter(x, y):`** dibuja un diagrama de puntos con las coordenadas de la lista `x` en el eje X y las coordenadas de la lista `y` en el eje Y.
    - Dibujamos un diagrama de puntos importando la librería como plt, creamos la figura, dibujamos los puntos y lo mostramos.

```
# Importar el módulo pyplot con el alias plt
import matplotlib.pyplot as plt
# Crear la figura y los ejes
fig, ax = plt.subplots()
# Dibujar puntos
ax.scatter(x = [1, 2, 3], y = [3, 2, 1])
# Mostrar el gráfico
plt.show()
```



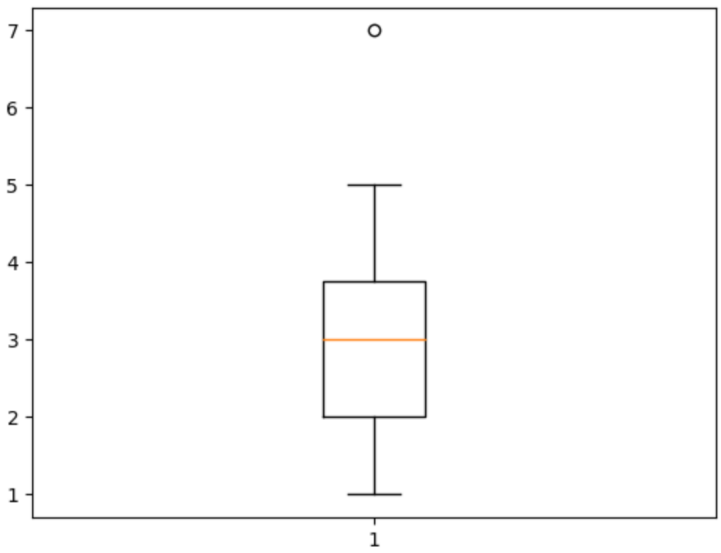
- **`hist(x, bins):`** dibuja un histograma con las frecuencias resultantes de agrupar los datos de la lista x en las clases definidas por la lista bins.

```
# Importar la librería numpy
import numpy as np
# Importar el módulo pyplot con el alias plt
import matplotlib.pyplot as plt
# Crear la figura y los ejes
fig, ax = plt.subplots()
#Generar los números
x = np.random.normal(5, 1.5, size=1000)
#Dibujar el histograma
ax.hist(x, np.arange(0, 11))
#Mostrar el gráfico
plt.show()
```



- **`boxplot(x)`:** dibuja un diagrama de caja y bigotes con los datos de la lista x.

```
# Importar la librería numpy
import matplotlib.pyplot as plt
#Crear la figura y los ejes
fig, ax = plt.subplots()
#Dibujar el boxplot
ax.boxplot([1, 2, 1, 2, 3, 4, 3, 3, 5, 7])
#Mostrar el gráfico
plt.show()
```



2.5 Librería scikit-learn

Es una librería de Python especializada en Machine Learning. Contiene algoritmos y funciones que nos ayudan a la resolución de diferentes problemas con métodos de aprendizaje automático. Tiene disponible funcionalidad para preprocesamiento, algoritmos y métricas. La documentación de la librería puede ser consultada en:

https://scikit-learn.org/stable/

De esta librería usaremos:

sklearn.metrics.confusion_matrix(*y_true*, *y_pred*, *, *labels=None*, *sample_weight=None*, *normalize=None*): calcula la matriz de confusión dados los valores reales de las ejemplos, los valores predichos y las etiquetas. También, se puede dar peso a los ejemplos y normalizar los resultados de la matriz de confusión.

```
#Importar las librerías
from sklearn.metrics
import confusion_matrix
#Valores reales y predichos
y_true = [2, 0, 2, 2, 0, 1]
y_pred = [0, 0, 2, 2, 0, 2]
#Calcula la matriz de confusión
confusion_matrix(y_true, y_pred)
>>> array([[2, 0, 0],
          [0, 0, 1],
          [1, 0, 2]])
```

En el caso binario, también se puede usar

```
tn, fp, fn, tp = confusion_matrix([0, 1, 0, 1], [1, 1, 1,
0]).ravel()
(tn, fp, fn, tp)
>>>(0, 2, 1, 1)
```

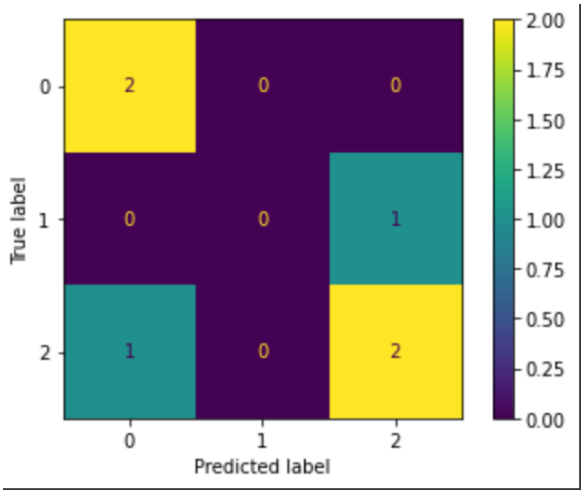
sklearn.metrics.ConfusionMatrixDisplay.from_predictions(*y_true*, *y_pred*, *, *labels=None*, *sample_weight=None*, *normalize=None*, *display_labels=None*, *include_values=True*, *xticks_rotation='horizontal'*, *values_format=None*, *cmap='viridis'*, *ax=None*, *colorbar=True*): partiendo de los valores reales y predichos, podemos visualizar la matriz de confusión. Se pueden normalizar os valores, mostrar las etiquetas, mostrar los ejes, dar peso a los ejemplos y determinar los colores, entre algunos de los parámetros que permite la función.

```
#Importar las librerías
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay

#Valores reales y predichos
y_true = [2, 0, 2, 2, 0, 1]
y_pred = [0, 0, 2, 2, 0, 2]

#Mostramos la matriz de confusión
ConfusionMatrixDisplay.from_predictions(y_true, y_pred)

plt.show()
```



2.6 Librería PyOD

Es una librería de Python con un conjunto completo de algoritmos para detectar anomalías. PyOD cuenta con un conjunto de más de 30 algoritmos de detección y una API unificada para todos los algoritmos, documentación técnica y ejemplos. PyOD es fácil de usar y todos los métodos utilizan las mismas funciones. Todos los métodos trabajan en un entorno no supervisado.

https://pyod.readthedocs.io/_/downloads/en/stable/pdf/

Veremos tanto la preparación de los datos que vamos a usar, en los ejemplos trabajaremos con una generación de datos sintéticas, como los algoritmos de la librería que utilizaremos.

- **Generando datos sintéticos utilizando la librería**

Como primer paso generaremos datos para la detección de valores atípicos. Se recuerda que estos modelos trabajan con un conjunto de datos de entrenamiento que son usados para ajustar los modelos y luego utilizaremos otro conjunto de datos que será más reducido para probar los algoritmos y que denominaremos test. Así, veremos cómo los algoritmos funcionan cuando reciben un conjunto de datos que sería nuevo para él, ya que no lo había usado mientras diseñaba el modelo.

Tener en cuenta que esta es la idea que siempre planteamos cuando queremos resolver un problema:

- Tenemos un conjunto de datos que le permiten a mi modelo aprender a diferenciar lo normal de la anomalías.
- Llegado nuevos datos, en función de lo que se ha aprendido se va a determinar si se trata de un valor normal o no.

Usaremos la función `generate_data` de PyOD para generar un conjunto de datos sintéticos con 200 muestras de entrenamiento y 100 muestras de prueba. Las muestras se generan mediante una distribución gaussiana multivariante y se indica el porcentaje de valores atípicos que se desea introducir. Tanto los conjuntos de datos de entrenamiento como los de prueba tienen 2 características para que podamos visualizarlas en un gráfico bidimensional y el 10% de las filas están etiquetadas como anomalías.

En esta librería los algoritmos trabajan con datos no etiquetados. No obstante, nosotros guardaremos las etiquetas para ver en el paso final cómo es el funcionamiento de estos algoritmos y comprobar las anomalías que han podido detectar tanto gráficamente como utilizando medidas que se vieron en la semana anterior. Es importante resaltar que los algoritmos no han tenido esa información durante el proceso de entrenamiento

```
from pyod.utils.data import generate_data
import numpy as np
X_train, y_train, X_test, y_test = \
        generate_data(n_train=200,
                      n_test=100,
                      n_features=5,
                      contamination=0.1,
                      random_state=3)
```

- **Aplicando los algoritmos de la librería**

Todos los métodos tienen las siguientes funciones:

- `fit(data-training):` aplicado sobre un método concreto utiliza los datos de entrenamiento para realizar su procesamiento y obtener las anomalías. Para realizar su procesamiento solamente utiliza los datos de entrenamiento sin etiqueta, no se requiere del atributo de salida.
- `decision_function(data-test):` una vez ya hemos entrenado nuestro modelo, asignamos las puntuaciones de anomalías a los datos sin procesar. *(Nota: algunos métodos no necesitan un ajuste previo).*
- `predict(data):` devuelve una etiqueta binaria (anomalía / normal) correspondiente a cada instancia, se puede aplicar tanto a los datos de entrenamiento (train) como a los de test para analizar ambos resultados. Esta función aplica un umbral utilizando las puntuaciones de anomalía devueltas por la función `decision_function`.
- `decision_scores_`: asigna las puntuaciones atípicas de los datos de entrenamiento. Las puntuaciones más altas indican que los datos son más anómalos.
- `threshold_:` el umbral que se ha determinado para establecer las anomalías.
- `labels_` : las etiquetas de anomalías binarias para los datos de entrenamiento. 0 indica que una observación es un valor normal y 1 indica un valor anómalo. Se genera aplicando el `threshold_` to `decision_scores_`.

- `evaluate_print(clf_name, y, y_pred):` función de utilidad para evaluar e imprimir los resultados de los ejemplos. Las métricas predeterminadas incluyen ROC y Precisión. Se le pasa el nombre del método, los valores reales de los ejemplos (si son anómalos o no) y los valores predichos de los ejemplos (si el algoritmo ha determinado si es una anomalía o no).