Table of Contents

Práctica No. 1. Introducción a Machine Learning y Redes Neuronales

Esta sesión cubrirá los conceptos básicos de Scikit-Learn, un paquete popular que contiene una colección de herramientas para el aprendizaje automático escritas en Python. Ver más en http://scikit-learn.org.

Finalidad de la práctica

Esta práctica tiene como finalidad introducir los conceptos centrales del aprendizaje automático y cómo se pueden aplicar en Python utilizando el paquete Scikit-Learn.

- **Familiarizarse con el manejo de un entorno Python, Jupyter Notebook (ficheros con extensión `.ipynb`) y las principales librerías de cálculo numérico como numpy.**
- **Afianzar las definiciones de Machine Learning a través de ejemplo muy sencillos.**
- **Introducción a la representación de los datos en Scikit-Learn.**
- **Introducción al uso de Scikit-Learn API.**

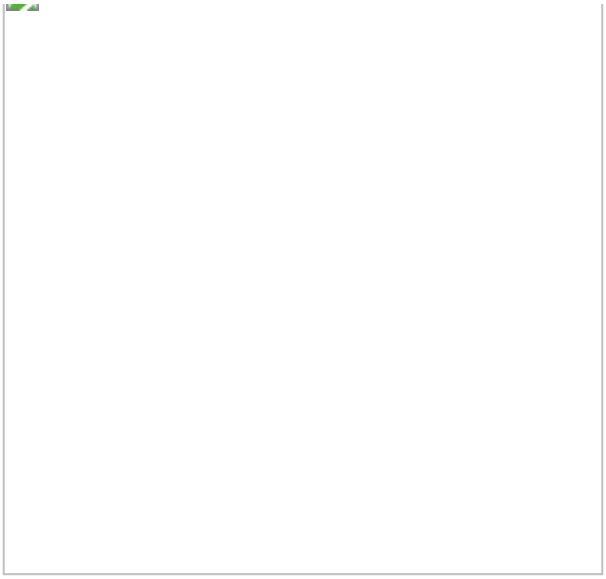
Sobre Scikit-Learn

Scikit-Learn es un paquete de Python diseñado para dar acceso a algoritmos de Aprendizaje Automático bien conocidos dentro del código Python, a través de una API limpia y bien pensada. Ha sido construido por cientos de colaboradores de todo el mundo y se usa en la industria y el mundo académico.

Scikit-Learn esá basada en las librerías NumPy (Numerical Python) y SciPy (Scientific Python), que permiten la computación numérica y científica eficiente en Python.

Para esta breve introducción, nos centraremos en visualzación y análisis de conjuntos de datos pequeños o medianos con Scikit-Learn.

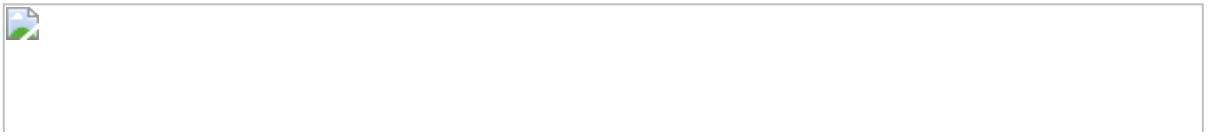
```
from IPython.core.display import HTML
```



```
print("Scikit-learn Logo\n")
```

¿Qué es el Machine Learning?

En esta sección comenzaremos a explorar los principios básicos del aprendizaje automático. El aprendizaje automático consiste en crear programas con parámetros ajustables (normalmente una matriz de valores numéricos) que se ajustan automáticamente para mejorar su comportamiento mediante la adaptación a los datos vistos anteriormente.



```
print("Artificial Intelligence, Machine Learning and Deep Learning relationships\n")
```

```
Artificial Intelligence, Machine Learning and Deep Learning relationships
```

El aprendizaje automático puede considerarse un subcampo de Inteligencia Artificial desde que los algoritmos se pueden ver como bloques de construcción para que las máquinas aprendan a comportarse de una manera más inteligente a través de la generalización en lugar de simplemente almacenar y recuperar elementos de datos como haría un sistema de base de datos.

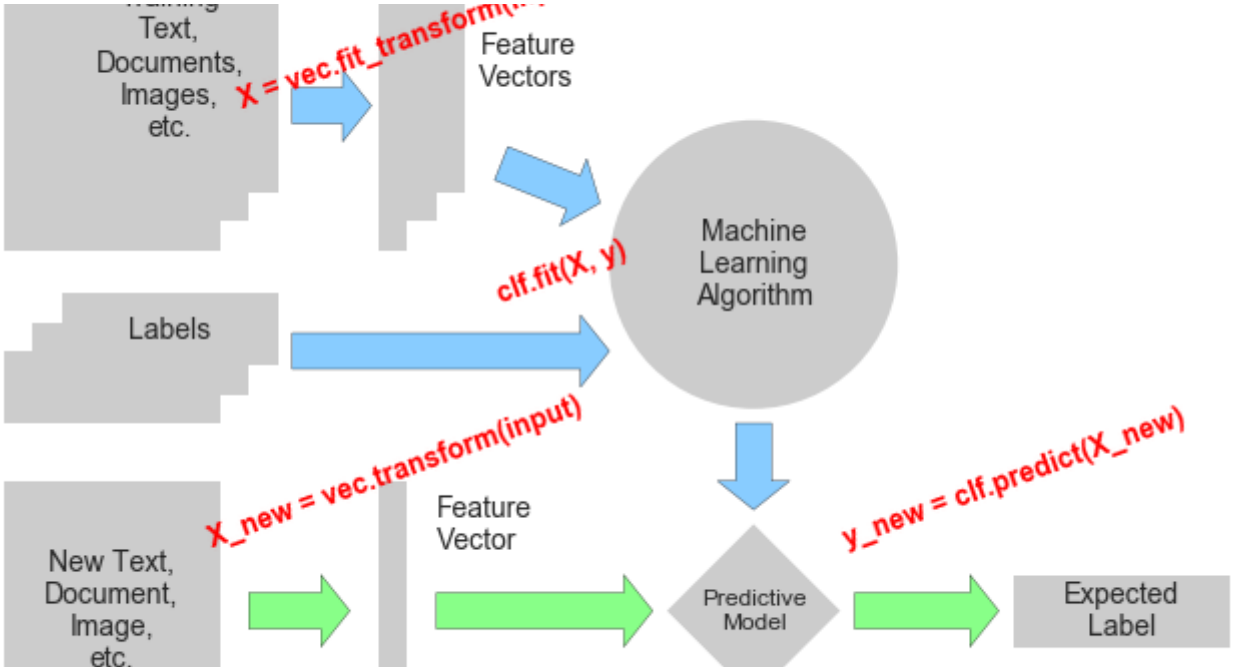
Vamos a echar un vistazo a dos tareas muy simples de aprendizaje automático aquí. La primera es una tarea de clasificación: la figura muestra una recopilación de datos bidimensionales, coloreados según dos clases diferentes. Se puede usar un algoritmo de clasificación para dibujar un límite de división entre los dos grupos de puntos:

```
# En esta celda se incluye un script que representa las pipelines de los principales

import numpy as np
import pylab as pl
from matplotlib.patches import Circle, Rectangle, Polygon, Arrow, FancyArrow
```

```
%matplotlib inline

def create_base(box_bg = '#CCCCCC',
                arrow1 = '#88CCFF',
                arrow2 = '#88FF88',
                supervised=True):
    fig = pl.figure(figsize=(9, 6), facecolor='w')
    ax = pl.axes((0, 0, 1, 1),
                 xticks=[], yticks=[], frameon=False)
    ax.set_xlim(0, 9)
    ax.set_ylim(0, 6)

    patches = [Rectangle((0.3, 3.6), 1.5, 1.8, zorder=1, fc=box_bg),
               Rectangle((0.5, 3.8), 1.5, 1.8, zorder=2, fc=box_bg),
               Rectangle((0.7, 4.0), 1.5, 1.8, zorder=3, fc=box_bg),

               Rectangle((2.9, 3.6), 0.2, 1.8, fc=box_bg),
               Rectangle((3.1, 3.8), 0.2, 1.8, fc=box_bg),
               Rectangle((3.3, 4.0), 0.2, 1.8, fc=box_bg),

               Rectangle((0.3, 0.2), 1.5, 1.8, fc=box_bg),

               Rectangle((2.9, 0.2), 0.2, 1.8, fc=box_bg),

               Circle((5.5, 3.5), 1.0, fc=box_bg),

               Polygon([[5.5, 1.7],
                        [6.1, 1.1],
                        [5.5, 0.5],
                        [4.9, 1.1]], fc=box_bg),

               FancyArrow(2.3, 4.6, 0.35, 0, fc=arrow1,
                          width=0.25, head_width=0.5, head_length=0.2),

               FancyArrow(3.75, 4.2, 0.5, -0.2, fc=arrow1,
                          width=0.25, head_width=0.5, head_length=0.2),

               FancyArrow(5.5, 2.4, 0, -0.4, fc=arrow1,
                          width=0.25, head_width=0.5, head_length=0.2),

               FancyArrow(2.0, 1.1, 0.5, 0, fc=arrow2,
                          width=0.25, head_width=0.5, head_length=0.2),

               FancyArrow(3.3, 1.1, 1.3, 0, fc=arrow2,
                          width=0.25, head_width=0.5, head_length=0.2),

               FancyArrow(6.2, 1.1, 0.8, 0, fc=arrow2,
                          width=0.25, head_width=0.5, head_length=0.2)]

    if supervised:
        patches += [Rectangle((0.3, 2.4), 1.5, 0.5, zorder=1, fc=box_bg),
```

```
                    Rectangle((0.5, 2.6), 1.5, 0.5, zorder=2, fc=box_bg),
                    Rectangle((0.7, 2.8), 1.5, 0.5, zorder=3, fc=box_bg),
                    FancyArrow(2.3, 2.9, 2.0, 0, fc=arrow1,
                               width=0.25, head_width=0.5, head_length=0.2),
                    Rectangle((7.3, 0.85), 1.5, 0.5, fc=box_bg)]
    else:
        patches += [Rectangle((7.3, 0.2), 1.5, 1.8, fc=box_bg)]

    for p in patches:
        ax.add_patch(p)

    pl.text(1.45, 4.9, "Training\nText,\nDocuments,\nImages,\netc.",
            ha='center', va='center', fontsize=14)

    pl.text(3.6, 4.9, "Feature\nVectors",
            ha='left', va='center', fontsize=14)

    pl.text(5.5, 3.5, "Machine\nLearning\nAlgorithm",
            ha='center', va='center', fontsize=14)

    pl.text(1.05, 1.1, "New Text,\nDocument,\nImage,\netc.",
            ha='center', va='center', fontsize=14)

    pl.text(3.3, 1.7, "Feature\nVector",
            ha='left', va='center', fontsize=14)

    pl.text(5.5, 1.1, "Predictive\nModel",
            ha='center', va='center', fontsize=12)

    if supervised:
        pl.text(1.45, 3.05, "Labels",
                ha='center', va='center', fontsize=14)

        pl.text(8.05, 1.1, "Expected\nLabel",
                ha='center', va='center', fontsize=14)
        pl.text(8.8, 5.8, "Supervised Learning Model",
                ha='right', va='top', fontsize=18)

    else:
        pl.text(8.05, 1.1,
                "Likelihood\nor Cluster ID\nor Better\nRepresentation",
                ha='center', va='center', fontsize=12)
        pl.text(8.8, 5.8, "Unsupervised Learning Model",
                ha='right', va='top', fontsize=18)



def plot_supervised_chart(annotate=False):
    create_base(supervised=True)
    if annotate:
        fontdict = dict(color='r', weight='bold', size=14)
        pl.text(1.9, 4.55, 'X = vec.fit_transform(input)',
```

```
                fontdict=fontdict,
                rotation=20, ha='left', va='bottom')
        pl.text(3.7, 3.2, 'clf.fit(X, y)',
                fontdict=fontdict,
                rotation=20, ha='left', va='bottom')
        pl.text(1.7, 1.5, 'X_new = vec.transform(input)',
                fontdict=fontdict,
                rotation=20, ha='left', va='bottom')
        pl.text(6.1, 1.5, 'y_new = clf.predict(X_new)',
                fontdict=fontdict,
                rotation=20, ha='left', va='bottom')

def plot_unsupervised_chart():
    create_base(supervised=False)


if __name__ == '__main__':
    plot_supervised_chart(True)
    plot_unsupervised_chart()
    pl.show()
```



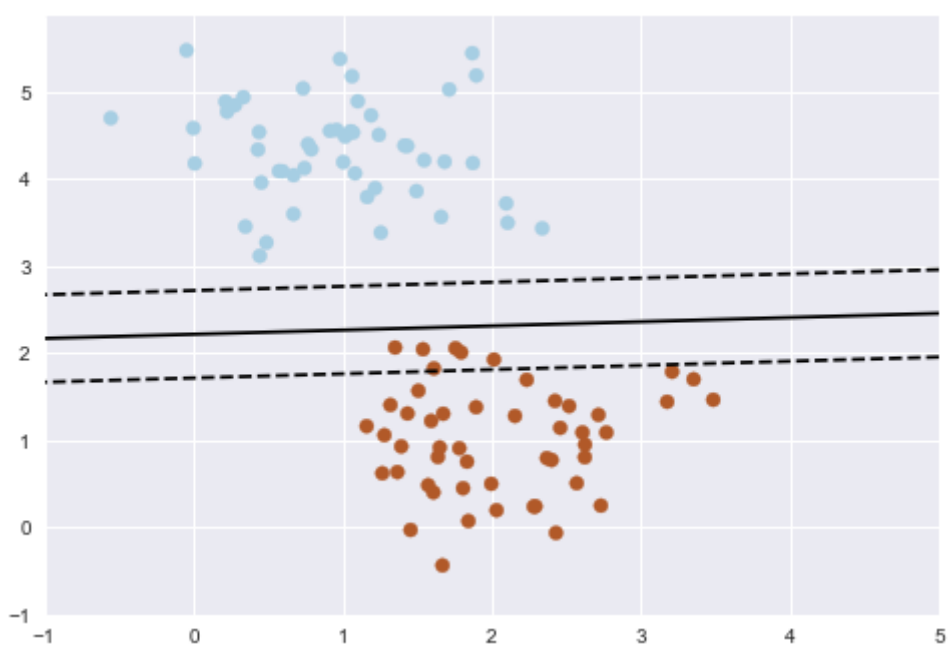

```
# Importamos las librerías

%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import SGDClassifier
from sklearn.datasets.samples_generator import make_blobs

plt.style.use('seaborn')
```

Definimos una función de clasificación:

```
def plot_sgd_separator(num, groups, sep):
    # We create num separable points
    X, Y = make_blobs(n_samples=num, centers=groups,
                      random_state=0, cluster_std=sep)

    # Fit the model
    clf = SGDClassifier(loss="hinge", alpha=0.01,
                        max_iter=200, fit_intercept=True)
    clf.fit(X, Y)

    # Plot the line, the points, and the nearest vectors to the plane
    xx = np.linspace(-1, 5, 10)
    yy = np.linspace(-1, 5, 10)

    X1, X2 = np.meshgrid(xx, yy)
    Z = np.empty(X1.shape)
    for (i, j), val in np.ndenumerate(X1):
        x1 = val
        x2 = X2[i, j]
        p = clf.decision_function(np.array([x1, x2]).reshape(1, -1))
```

```
            Z[i, j] = p[0]
    levels = [-1.0, 0.0, 1.0]
    linestyles = ['dashed', 'solid', 'dashed']
    colors = 'k'

    ax = plt.axes()
    ax.contour(X1, X2, Z, levels, colors=colors, linestyles=linestyles)
    ax.scatter(X[:, 0], X[:, 1], c=Y, cmap=plt.cm.Paired)

    ax.axis('tight')
```

```
# Ejecutamos la anterior función para 2 grupos de 100 puntos, por ejemplo:
plot_sgd_separator(num=100, groups=2, sep=0.6)
```



Esto puede parecer una tarea trivial, pero es una versión simple de un concepto muy importante. Al dibujar esta línea de separación, hemos aprendido un modelo que puede generalizar nuevos datos: si tuviera que colocar otro punto en el plano sin etiquetar, este algoritmo ahora podría predecir si es un punto azul o rojo.

La siguiente tarea simple que veremos es una tarea de regresión: una línea simple que se adapta mejor a un conjunto de datos:

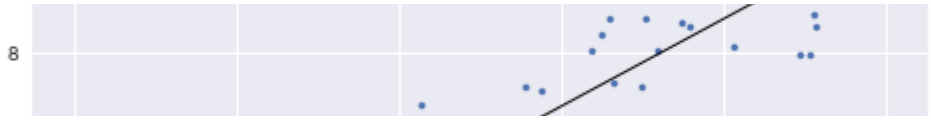
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
```

Definimos una función de regresión lineal:

```
def plot_linear_regression(a, b, num_points):
    # The slope (a) and the interception with y-axis (bias) are defined as a function

    # x from 0 to 30
    x = 10 * np.random.random(50)

    # y = a * x + b with gaussian noise
    y = a * x + b + np.random.normal(size=x.shape)

    # Create a linear regression classifier
    clf = LinearRegression()
    clf.fit(x[:, None], y)

    # Predict y from the data
    x_new = np.linspace(0, 10, 100)
    y_new = clf.predict(x_new[:, None])

    # Plot the results
    ax = plt.axes()
    ax.scatter(x, y, s=10)
    ax.plot(x_new, y_new, c='k', linewidth=1.)

    ax.set_xlabel('x')
    ax.set_ylabel('y')

    ax.axis('tight')
```

Por ejemplo, para ejecutar la función de arriba con una pendiente $a = 1.0$ y un corte de ordenadas $b = 1.0$ para 50 puntos:

```
plot_linear_regression(a=1.0, b=1.0, num_points=50)
plt.show()
```



De nuevo, este es un ejemplo de ajuste de un modelo a los datos, de manera que el modelo puede hacer generalizaciones sobre nuevos datos. El modelo ha aprendido a partir del training dataset, y se puede utilizar para predecir el resultado de los datos de testeo.

Aquí, se nos puede dar un valor de x, y el modelo nos permitirá predecir el valor de y. Una vez más, esto puede parecer un problema trivial, pero es un ejemplo básico de un tipo de operación que resulta fundamental para Tareas de Machine Learning.

Primera pregunta (3 puntos)

a) Habiendo definido la función que representa la regresión lineal, representar una regresión lineal a `50 puntos` con una pendiente de `1.5` y un punto de corte con ordenadas de `2.0`. (1 punto)

[ ] ↳ *1 celda oculta*

b) Modifique el ruido gausiano incluido a los datos sobre los que se realiza la regresión lineal en la función `plot_linear_regression` (2 puntos)

[ ] ↳ *3 celdas ocultas*

¿Qué observa en los puntos aleatoriamente generados si la desviación típica es $\sigma = 5.0$ manteniendo la desviación de la media en $\mu = 0$? ¿Qué efectos tiene en la recta ajustada por el estimador `LinearRegressor` ?

[ ] ↳ *1 celda oculta*

¿Qué efecto en los puntos aleatoriamente generados si la desviación típica es $\sigma = 0$ cambiando la desviación de la media a $\mu = 5$? ¿Qué efectos tiene en la recta ajustada por el estimador `LinearRegressor` ?

[ ] ↳ *2 celdas ocultas*

Representación de datos con Scikit-Learn

El Aprendizaje Automático consiste en crear modelos a partir de datos: por esa razón, comenzaremos por analizar cómo se pueden representar los datos para que nuestro modelo pueda comprenderlos. Junto con esto, desarrollaremos nuestros ejemplos de matplotlib de la sección anterior y mostraremos algunos ejemplos de cómo visualizar datos.

La mayoría de los algortimos de Machine Learning esperan que los datos estén almacenados en tensores, generalmente de orden 2 u oreden 3. En este caso trabajaremos con matrices bidimensionales. Los arrays, por lo general, si se trabaja con librerías como Scikit-Learn se pueden contruir con `numpy` arrays o `scipy.sparse`. La dimensaionalidad de esta matriz es `[n_samples, n_features]`

- **n_samples: La cantidad de muestras: cada muestra es un elemento para procesar (por ejemplo, clasificar). Una muestra puede ser un documento, una imagen, un sonido, un video, un objeto astronómico, una fila en la base de datos o un archivo CSV, o lo que pueda describir con un conjunto fijo de rasgos cuantitativos.**
- **n_features: La cantidad de características o rasgos distintos que se pueden usar para describir cada elemento de manera cuantitativa. Las características son generalmente de valor real, pero pueden ser booleanas o de valor discreto en algunos casos.**

El número de características debe estar fijado de antemano. Debe haber un número fijo de características o features para cada muestra, mientras que cada número `i` de una característica debe ser de un orden de magnitud similar para todas las muestras. Sin embargo, puede tener dimanesionalidad muy elevada (por ejemplo, millones de características) y la mayoría de ellas son ceros para una muestra determinada. Este es un caso en el que las matrices `scipy.sparse` pueden ser útiles, ya que son mucho más eficientes en memoria que las matrices `numpy`.

 **Data Layout**

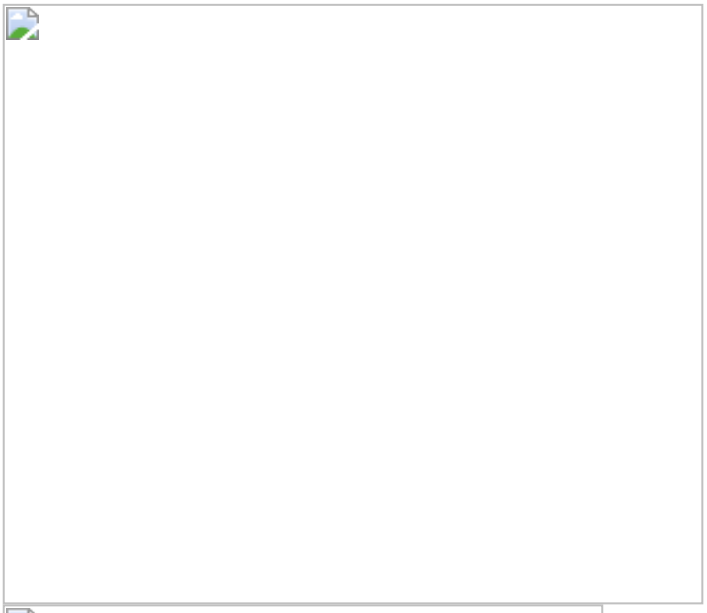
(Imagen obtenida de [Python Data Science Handbook])

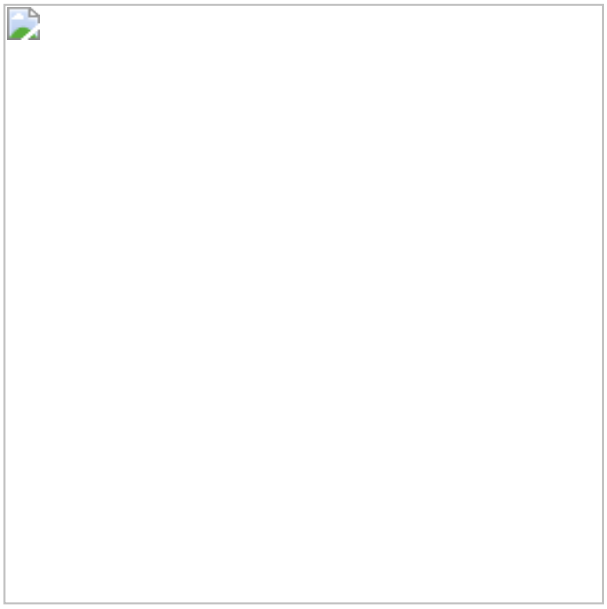
Un ejemplo sencillo: el Iris flower data set

Como ejemplo de un conjunto de datos simple, vamos a echar un vistazo a los datos del iris almacenados por scikit-learn. Los datos consisten en mediciones de tres especies diferentes de lirios. Hay tres especies de iris en el conjunto de datos, que podemos ver aquí:

lirios. Hay tres especies de iris en el conjunto de datos, que podemos ver aquí.

A continuación, puede ver de izquierda a derecha muestras de lirios setosa, versicolor y virgínica respectivamente.





Cargando el Iris Dataset con Scikit-Learn

Scikit-Learn cuenta con un gran número de datasets que se pueden importar de manera rápida y directa. entre ellos se encuentra el dataset de las diferentes especies de lirios. Nuestros datos consisten en:

- **Características del Iris dataset:**
    1. **Longitud del sépalo en cm**
    2. **Anchura del sépalo en cm**
    3. **Longitud del pétalo en cm**
    4. **Anchura del pétalo en cm**

4. Anchura del pétalo en cm

- **Las clases soportadas en este dataset son:**

Apuntamos a los dos primeros números naturales y al 0.

1. **Iris Setosa** $\rightarrow 0$
2. **Iris Versicolour** $\rightarrow 1$
3. **Iris Virginica** $\rightarrow 2$

`scikit-learn` **embebe una copia del archivo** `csv` **de iris junto con una función auxiliar para cargarlo en matrices numpy:**

[ ] *17 celdas ocultas*

Segunda pregunta (4 puntos):

a) Si queremos diseñar un algoritmo de clasificación que reconozca las especies de lirios, ¿qué tipos de datos de nuestro dataset vamos a necesitar? (1 punto)

[ ] *1 celda oculta*

b) ¿Dentro de qué paradigma de aprendizaje caería la clasificación por mapeo características-etiquetas del dataset Iris Setosa? Justifique brevemente su respuesta (1 punto)

[ ] *1 celda oculta*

c) ¿Se podría hacer uso de aprendizaje no-supervisado para clasificar este dataset? De ser así, ¿qué familia de algoritmos se podría usar para ese fin? (1 punto)

[ ] *2 celdas ocultas*

d) ¿A qué se refiere `n_samples` con el caso del dataset de los lirios? (0,5 puntos)

[ ] ↳ *1 celda oculta*

e) ¿Con qué se equipara `n_features` en el caso del dataset de los lirios? (0,5 puntos)

[ ] ↳ *2 celdas ocultas*

Tercera pregunta (3 puntos)

¿Cuál parece el mejor gráfico para diferenciar las muestras del Iris flower data set? Cambie `x_index` e `y_index` en la última pieza de código para encontrar una combinación de parámetros que maximice la separación entre clases. Incluya abajo el código necesario para representar las diferentes clases lo más separadas posible y ejecute para representar la gráfica.

[ ] ↳ *6 celdas ocultas*