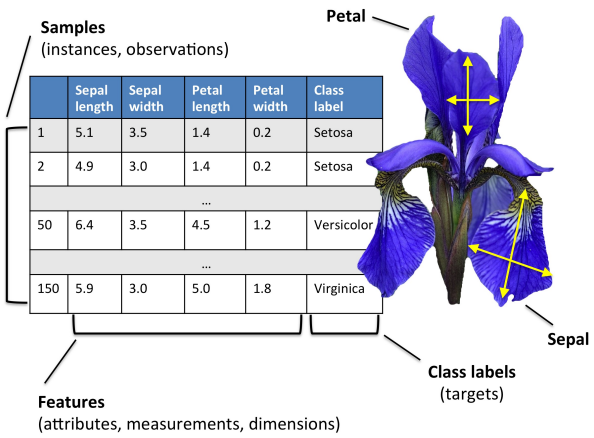
ML_CLASIFICACION_SVM_02	Modelo de Clasificación SVM para predecir el tipo de flor	ML

Esta práctica consiste en construir un modelo de aprendizaje maquina (machine learning), basado en el algoritmo de **SVM (Support Vector Machines)** que aprenda a clasificar el tipo de especie de la flor en función de las medidas conocidas, la longitud y anchura de los pétalos (petal), y la longitud y anchura de los sépalos (sepal). Todas las medidas están en centímetros.

Se realiza la clasificación para dos de las especies (versicolor y virginica) de la base de datos 'iris' de la librería sklearn.datasets, y se utilizan sólo las dimensiones de los sépalos.



	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica


Iris Versicolor


Iris Virginica


Iris Setosa

SOLUCIÓN

Importar las librerías necesarias para realizar la práctica.

```
# Librerías
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets, svm
```

Cargar la base de datos (incluida ya en la librería 'sklearn'.

```
iris = datasets.load_iris()
```

Generar X e y.

```
# Generar X e y
X = iris.data
y = iris.target

# Coger como X las dimensiones del sépalo de la planta de iris. Y que la
etiqueta 'y' sea distinto de 0
# Como etiqueta 'y' las que no son 0 (setosa). Cogemos el tipo de flor
'versicolor' y 'virginica'
X = X[y != 0, :2]
y = y[y != 0]

# Determinar la longitud de la base de datos
n_sample = len(X)

# Ordenar el conjunto de datos
np.random.seed(0)
order = np.random.permutation(n_sample)
X = X[order]
y = y[order].astype(np.float)
```

Dividir en entrenamiento y test

```
# Dividir en entrenamiento y test (90 % entrenamiento y 10 % test)
X_train = X[:int(.9 * n_sample)]
y_train = y[:int(.9 * n_sample)]
X_test = X[int(.9 * n_sample):]
y_test = y[int(.9 * n_sample):]
```

Iterar para diferentes tipos de 'kernel'. Crear, ajustar el clasificador y visualizar

```
# Ajustar el modelo con diferentes 'kernel'
for fig_num, kernel in enumerate(('linear', 'rbf', 'poly')):

    # crear el clasificador SVM para cada kernel
    clf = svm.SVC(kernel=kernel, gamma=10)
```

```
    # Ajustar el modelo a los datos de entrenamiento
    clf.fit(X_train, y_train)

    # Visualizar figura, clasificador y puntos
    plt.figure(fig_num)
    plt.clf()
    plt.scatter(X[:, 0], X[:, 1], c=y, zorder=10, cmap=plt.cm.Paired,
                edgecolor='k', s=20)

    # Marcar con un círculo los datos de test
    plt.scatter(X_test[:, 0], X_test[:, 1], s=80, facecolors='none',
                zorder=10, edgecolor='k')

    # Dibujar los ejes
    plt.axis('tight')
    x_min = X[:, 0].min()
    x_max = X[:, 0].max()
    y_min = X[:, 1].min()
    y_max = X[:, 1].max()

    # Construir la rejilla para la visualización del gráfico
    XX, YY = np.mgrid[x_min:x_max:200j, y_min:y_max:200j]
    # Definir la función de borde de decisión 'Z'
    Z = clf.decision_function(np.c_[XX.ravel(), YY.ravel()])

    # Dibujar el resultado en una gráfica de color
    Z = Z.reshape(XX.shape)
    plt.pcolormesh(XX, YY, Z > 0, cmap=plt.cm.Paired)
    plt.contour(XX, YY, Z, colors=['k', 'k', 'k'],
                linestyles=['--', '-', '--'], levels=[-.5, 0, .5])

    # Poner el 'kernel' como título de la gráfica
    plt.title(kernel)
```

Visualizar los gráficos

plt.show()