Perceptron Clasificacion Binaria

Este modelo está inspirado en el funcionamiento de las neuronas biológicas que forman las redes neuronales de nuestros cerebros, recibiendo una serie de señales de entrada y devolviendo un resultado a la salida, calculando una suma ponderada de todos los inputs y aplicando una función de activación. Para realizar el ejemplo se necesita un conjunto de datos formado por un número determinado de elementos con varias características acompañados de sus correspondiente clase. Se va a utilizar un dataset muy utilizado en el ámbito académico muy útil para aprender a desarrollar modelos de clasificación: El dataset Iris. Este dataset contiene ejemplos de flores que tendremos que clasificar en 3 grupos diferentes a partir del ancho y longitud de sus pétalos y sépalos (en total 4 características). Este dataset está disponible a través de la librería Scikit Learn.

Se buscara determinar si un flor es de un tipo o de otra.

Cargar las librerias a utilizar

In [33]:
```
from sklearn.datasets import load_iris
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation,rc
from matplotlib.colors import ListedColormap
```

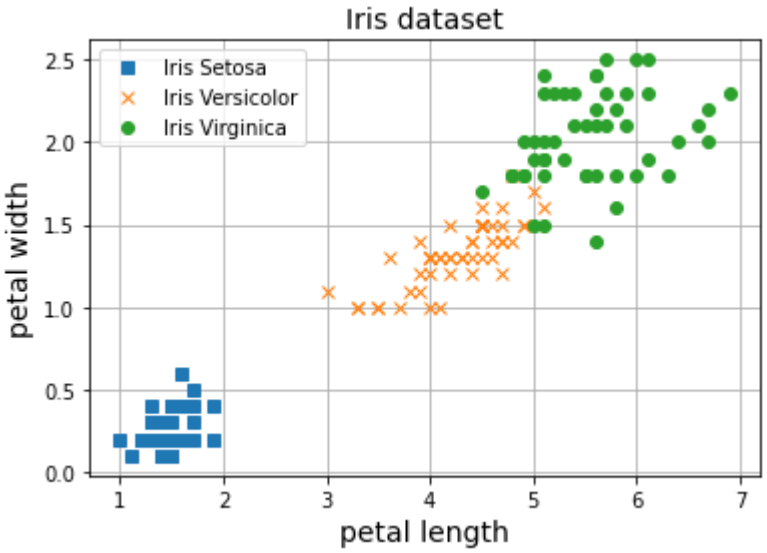
Carga de los Datos

In [5]:
```
iris = load_iris()
X= iris.data[:,(2,3)]
y = iris.target

X.shape,y.shape
```

Out[5]:
```
((150, 2), (150,))
```

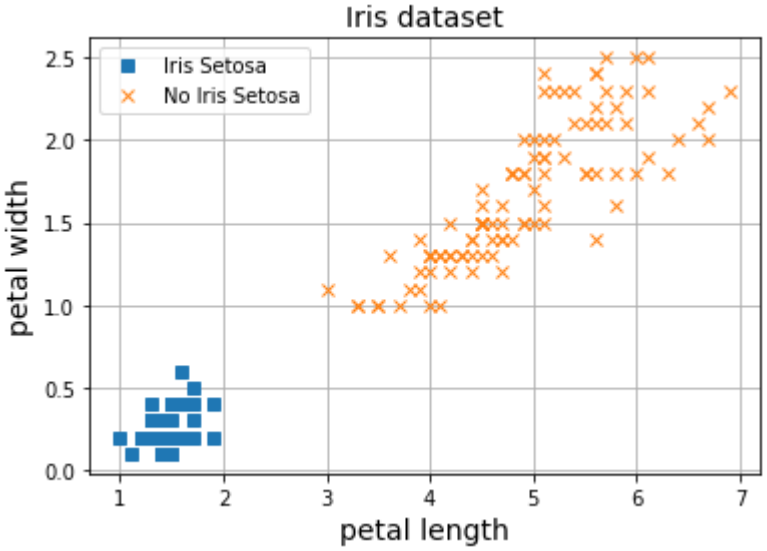
In [7]:
```
plt.plot(X[y==0, 0], X[y==0, 1], 's', label="Iris Setosa")
plt.plot(X[y==1, 0], X[y==1, 1], 'x', label="Iris Versicolor")
plt.plot(X[y==2, 0], X[y==2, 1], 'o', label="Iris Virginica")
plt.grid()
plt.legend()
plt.xlabel('petal length', fontsize=14)
plt.ylabel('petal width', fontsize=14)
plt.title("Iris dataset", fontsize=14)
plt.show()
```



Defino la clase a clasificar

In [70]:
```
y= (iris.target ==0).astype(np.int64)

plt.plot(X[y==1, 0], X[y==1, 1], 's', label="Iris Setosa")
plt.plot(X[y==0, 0], X[y==0, 1], 'x', label="No Iris Setosa")
plt.grid()
plt.legend()
plt.xlabel('petal length', fontsize=14)
plt.ylabel('petal width', fontsize=14)
plt.title("Iris dataset", fontsize=14)
plt.show()
```



Defino la clase del Perceptron

In [83]:
```
class Perceptron():
  def __init__(self, size):
    self.w = np.random.randn(size)
```

```
    self.ws = []

  def __call__(self, w, x):
    return np.dot(x, w) > 0

  def fit(self, x, y, epochs, lr):
    x = np.c_[np.ones(len(x)), x]
    for epoch in range(epochs):
        # Batch Gradient Descent
        y_hat = self(self.w, x)
        # función de pérdida (MSE)
        l = 0.5*(y_hat - y)**2
        # derivadas
        dldh = (y_hat - y)
        dhdw = x
        dldw = np.dot(dldh, dhdw)
        # actualizar pesos
        self.w = self.w - lr*dldw
        # guardar pesos para animación
        self.ws.append(self.w.copy())
```

Normalizamos

In [84]:
```
print(X.mean(axis=0), X.std(axis=0))

X_mean, X_std = X.mean(axis=0), X.std(axis=0)
X_norm = (X - X_mean) / X_std

print(X_norm.mean(axis=0), X_norm.std(axis=0))
```

```
[3.758      1.19933333] [1.75940407 0.75969263]
[-4.26325641e-16 -4.73695157e-16] [1. 1.]
```

Entrenamos el perceptron

In [85]:
```
np.random.seed(42)

perceptron = Perceptron(3)
epochs, lr = 20, 0.1
perceptron.fit(X_norm, y, epochs, lr)
```

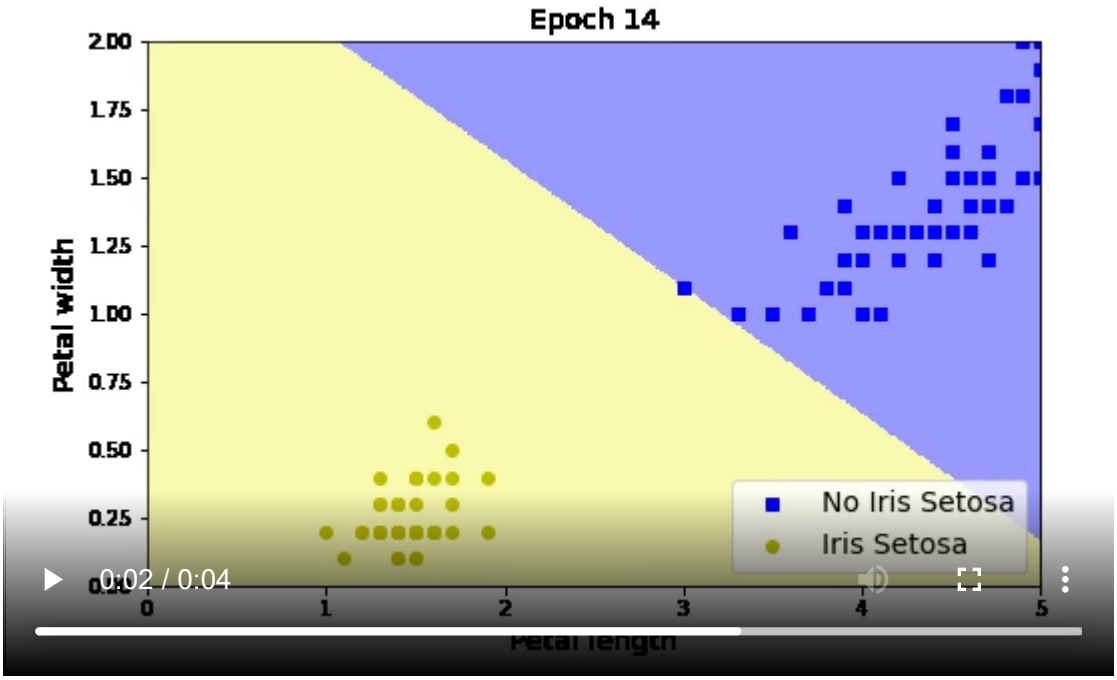
Graficamos

In [87]:
```
fig = plt.figure(figsize=(8, 5))
ax = fig.add_subplot(111, autoscale_on=False)
def plot(i, axes = [0, 5, 0, 2], label="Iris Setosa"):
    ax.clear()
    w = perceptron.ws[i]
    tit = ax.set_title(f"Epoch {i+1}", fontsize=14)
    x0, x1 = np.meshgrid(
            np.linspace(axes[0], axes[1], 500).reshape(-1, 1),
            np.linspace(axes[2], axes[3], 200).reshape(-1, 1),
```

```
        )
    X_new = (np.c_[x0.ravel(), x1.ravel()] - X_mean)/X_std
    X_new = np.c_[np.ones(len(X_new)), X_new]
    y_predict = perceptron(w, X_new)
    zz = y_predict.reshape(x0.shape)
    ax.plot(X[y==0, 0], X[y==0, 1], "bs", label=f"No {label}")
    ax.plot(X[y==1, 0], X[y==1, 1], "yo", label=label)
    ax.contourf(x0, x1, zz, cmap=custom_cmap)
    ax.set_xlabel("Petal length", fontsize=14)
    ax.set_ylabel("Petal width", fontsize=14)
    ax.legend(loc="lower right", fontsize=14)
    ax.axis(axes)
    return ax

anim = animation.FuncAnimation(fig, plot, frames=epochs, interval=200)
plt.close()
anim
```

Out[87]:



Revisamos

In [75]:
```
# últimos pesos encontrados

w = perceptron.ws[-1]
w
```

Out[75]:
```
array([-13.70328585, -24.26938962, -23.16387156])
```

In [76]:
```
x_new = [-13, -24, -23]
y_pred = perceptron(w, x_new)
y_pred # Iris Setosa
```

Out[76]:
```
True
```

In [77]:
```
x_new = [1, 4, 0.5]
y_pred = perceptron(w, x_new)
y_pred # No Iris Setosa
```

Out[77]:
```
False
```

In [ ]:
```

```