Sesión 2

En esta segunda sesión aplicaremos el algoritmo Perceptrón a algunas tareas de clasificación. Se proporciona una sencilla implementación del algoritmo Perceptrón y de su aplicación al conjunto de datos iris. El objetivo principal de esta sesión es extender el ejemplo dado a otras tareas tratando de minimizar el error de test.

Perceptrón aplicado a iris

Lectura del corpus: comprobamos también que las matrices de datos y etiquetas tienen las filas y columnas que toca

In [1]:
```
import numpy as np; from sklearn.datasets import load_iris
iris = load_iris(); X = iris.data.astype(np.float16);
y = iris.target.astype(np.uint).reshape(-1, 1);
print(X.shape, y.shape, "\n", np.hstack([X, y])[:5, :])
```

```
(150, 4) (150, 1)
 [[5.1015625  3.5        1.40039062 0.19995117 0.        ]
 [4.8984375  3.         1.40039062 0.19995117 0.        ]
 [4.69921875 3.19921875 1.29980469 0.19995117 0.        ]
 [4.6015625  3.09960938 1.5        0.19995117 0.        ]
 [5.         3.59960938 1.40039062 0.19995117 0.        ]]
```

Partición del corpus: Creamos un split de iris con un 20% de datos para test y el resto para entrenamiento (training), barajando previamente los datos de acuerdo con una semilla dada para la generación de números aleatorios. Aquí, como en todo código que incluya aleatoriedad (que requiera generar números aleatorios), conviene fijar dicha semilla para poder reproducir experimentos con exactitud.

In [2]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=True, random_state=23)
print(X_train.shape, X_test.shape)
```

```
(120, 4) (30, 4)
```

Implementación de Perceptrón: devuelve pesos en notación homogénea, $\mathbf{W} \in \mathbb{R}^{(1+D)\times C}$; también el número de errores e iteraciones ejecutadas

In [3]:
```
def perceptron(X, y, b=0.1, a=1.0, K=200):
    N, D = X.shape; Y = np.unique(y); C = Y.size; W = np.zeros((1+D, C))
    for k in range(1, K+1):
        E = 0
        for n in range(N):
            xn = np.array([1, *X[n, :]])
            cn = np.squeeze(np.where(Y==y[n]))
            gn = W[:,cn].T @ xn; err = False
            for c in np.arange(C):
                if c != cn and W[:,c].T @ xn + b >= gn:
                    W[:, c] = W[:, c] - a*xn; err = True
            if err:
                W[:, cn] = W[:, cn] + a*xn; E = E + 1
        if E == 0:
            break;
    return W, E, k
```

Aprendizaje de un clasificador (lineal) con Perceptrón: Perceptrón minimiza el número de errores de entrenamiento (con margen)

$$\mathbf{W}^* = \underset{\mathbf{W}=(\boldsymbol{w}_1,\ldots,\boldsymbol{w}_C)}{\operatorname{argmin}} \sum_n \mathbb{I}\left(\max_{c\neq y_n} \boldsymbol{w}_c^t \boldsymbol{x}_n + b \;>\; \boldsymbol{w}_{y_n}^t \boldsymbol{x}_n\right)$$

In [4]:
```
W, E, k = perceptron(X_train, y_train)
print("Número de iteraciones ejecutadas: ", k)
print("Número de errores de entrenamiento: ", E)
print("Vectores de pesos de las clases (en columnas y en notación homogénea):\n", W);
```

```
Número de iteraciones ejecutadas:  200
Número de errores de entrenamiento:  2
Vectores de pesos de las clases (en columnas y en notación homogénea):
 [[  10.           85.         -142.        ]
 [ -49.421875    -68.19140625 -176.47265625]
 [  50.171875     -1.72460938 -181.06445312]
 [-189.91210938  -87.70507812   68.69726562]
 [ -86.40258789 -137.78149414  157.88415527]]
```

Cálculo de la tasa de error en test:

In [5]:
```
X_testh = np.hstack([np.ones((len(X_test), 1)), X_test])
y_test_pred  = np.argmax(X_testh @ W, axis=1).reshape(-1, 1)
err_test = np.count_nonzero(y_test_pred != y_test) / len(X_test)
print(f"Tasa de error en test: {err_test:.1%}")
```

```
Tasa de error en test: 16.7%
```

Ajuste del margen: experimento para aprender un valor de b

In [6]:
```
for b in (.0, .01, .1, 10, 100):
    W, E, k = perceptron(X_train, y_train, b=b, K=1000)
    print(b, E, k)
```

```
0.0 3 1000
0.01 5 1000
0.1 3 1000
10 6 1000
100 6 1000
```

Interpretación de resultados: los datos de entrenamiento no parecen linealmente separables; no está claro que un margen mayor que cero pueda mejorar resultados, sobre todo porque solo tenemos 30 muestras de test; con margen nulo ya hemos visto que se obtiene un error (en test) del 16.7%