Session 2

In this second session we will apply the Perceptron algorithm to some classification tasks. A simple implementation of the Perceptron algorithm and its application is provided. The main purpose of this session is to extend the example given to other tasks, trying to minimize test error.

Perceptron applied to the Iris dataset

Reading the dataset: we also check that the data matrix and labels have the right number of rows and columns

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.        ]]
```

Dataset partition: We create a split of the Iris dataset with 20% of data for test and the rest for training, previously shuffling the data according to a given seed provided by a random number generator. Here, as in all code that includes randomness (which requires generating random numbers), it is convenient to fix said seed to be able to reproduce experiments with accuracy.

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)
```

Perceptron implementation: returns weights in homogeneous notation, $\mathbf{W} \in \mathbb{R}^{(1+D)\times C}$; also the number of errors and iterations executed

In [4]:
```
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
```

Learning a (linear) classifier with Perceptron: Perceptron minimizes the number of training errors (with margin b)

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

In [5]:
```
W, E, k = perceptron(X_train, y_train)
print("Number of iterations executed: ", k)
print("Number of training errors: ", E)
print("Weight vectors of the classes (in columns and with homogeneous notation):\n", W);
```

```
Number of iterations executed:  200
Number of training errors:  2
Weight vectors of the classes (in columns and with homogeneous notation):
 [[  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]]
```

Calculation of test error rate:

```
In [6]: 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"Error rate on test: {err_test:.1%}")
```

```
Error rate on test: 16.7%
```

Margin adjustment: experiment to learn a value of b

```
In [7]: 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
```

Interpretation of results: the training data does not appear to be linearly separable; it is not clear that a margin greater than zero can improve results, especially since we only have 30 test samples; with a margin $b = 0.1$ we have already seen that an error (in test) of 16.7% is obtained