
classification
problem
perceptron

About Us

AYA QUBBAJ 11924192
LEEN HODALI 11924198

We present our application designed to address binary and multi-class classification challenges. Classification involves employing machine learning techniques to categorize data points into two or multiple distinct categories. Our application offers effective solutions for this problem domain.

Dual-Class Classifier :

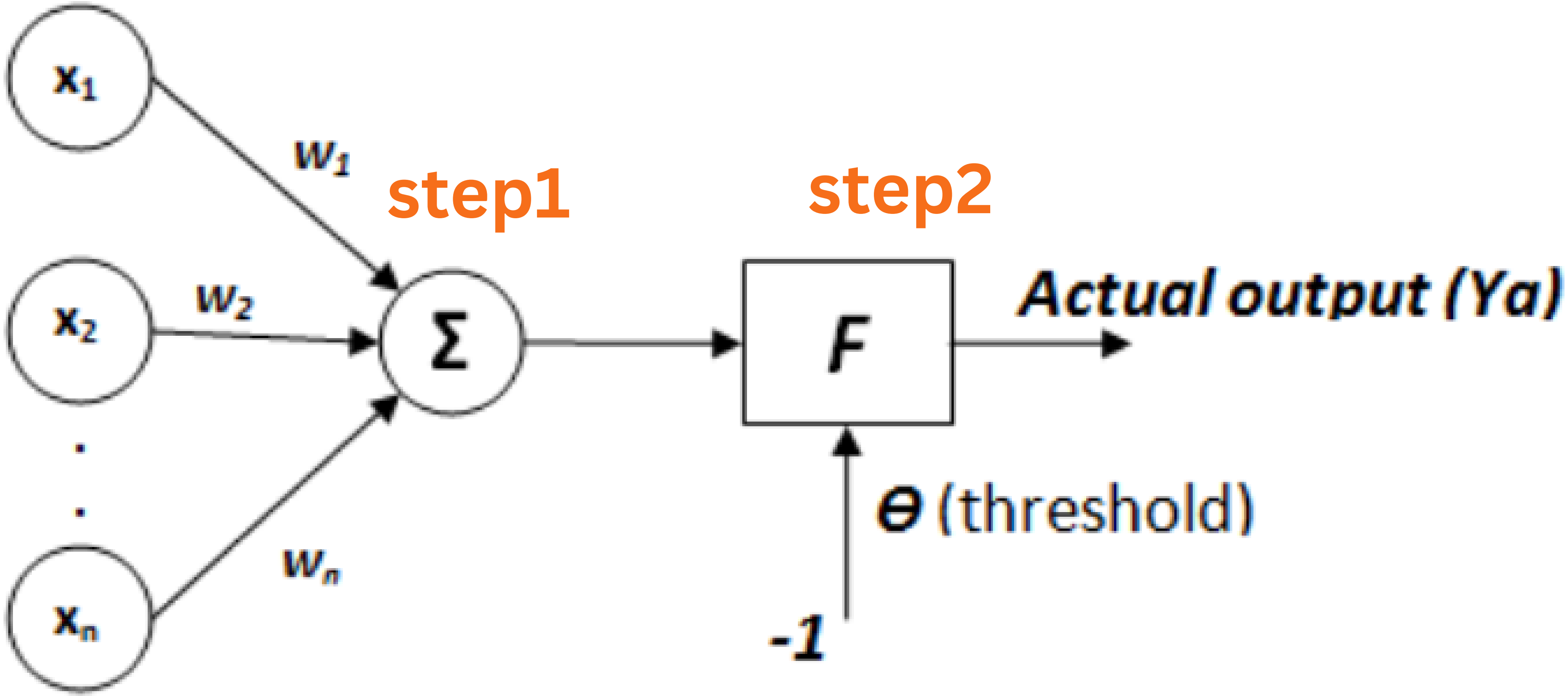
binary

multi-class

Perceptron

• It's a single node neural network that can take different inputs
but produce only one output
• Perceptron is usually used to classify the data into two parts.
Therefore, it is also known as a Linear Binary Classifier.

How does the neuron determine its output?

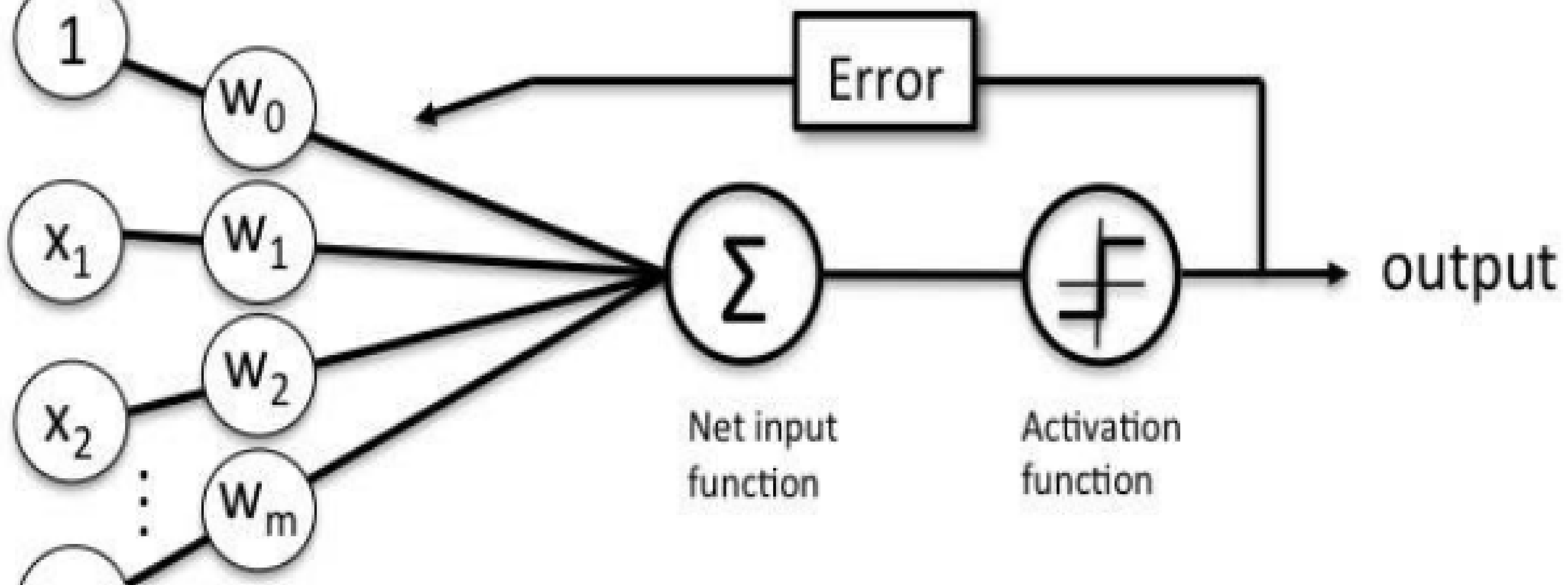


$$x_1 w_1 + x_2 w_2 + \ldots + x_n w_n$$

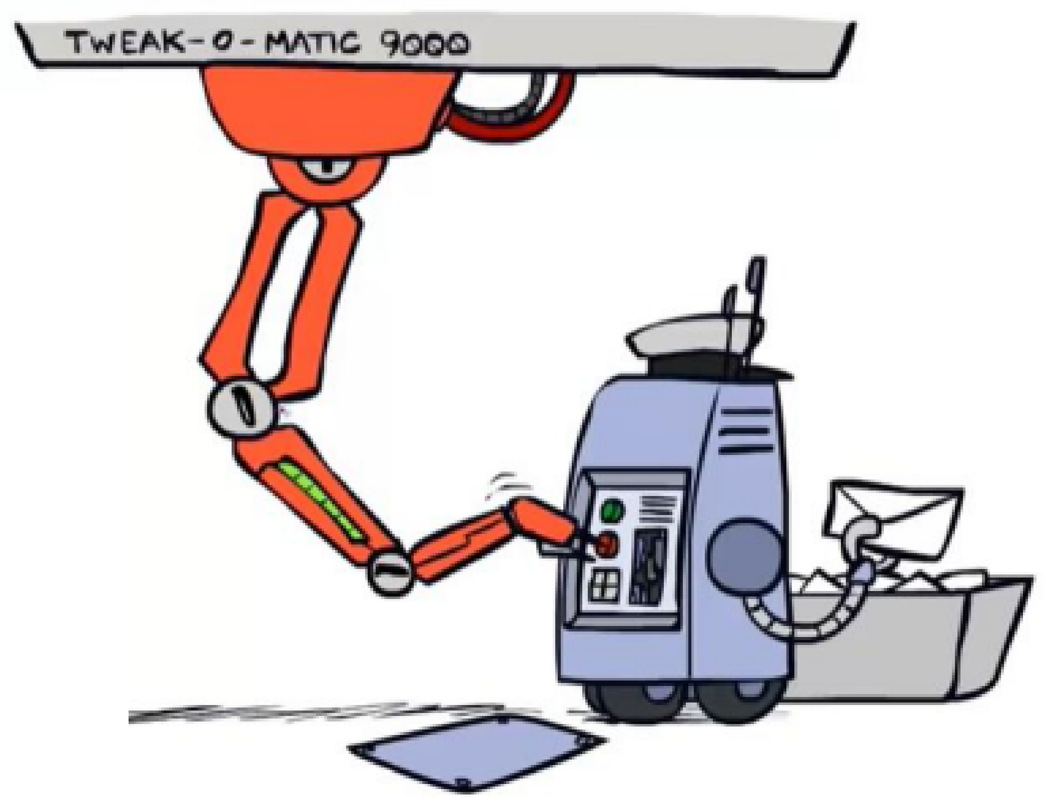
$$Ya = F\left(\sum_{i=1}^{n} x_i w_i - \theta\right)$$

How does a perceptron learn

❖ This is done by making small adjustments in the weights to reduce the difference between the actual and desired outputs of the perceptron



Perceptron rule.



Step 1: Initialization

- Set initial weights $w_1, w_2, \ldots, w_n$ and threshold to random numbers in the range [-0.5, 0.5]

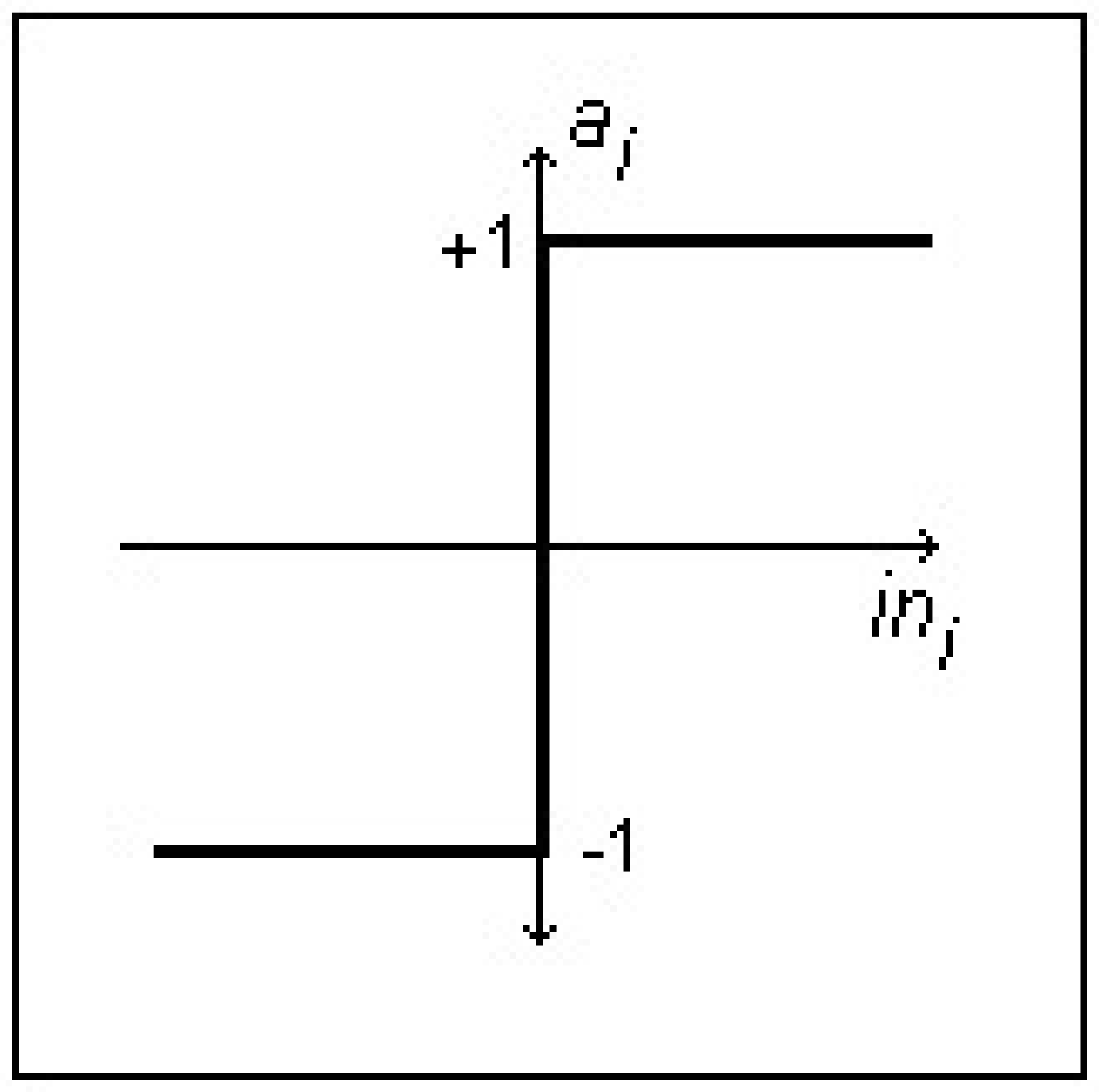
Step 2: Activation

- Activate the perceptron by applying inputs $x_1(p)$, $x_2(p)$, $x_3(p)$, ...$x_n(p)$, and desired output $y_d(p)$ .Calculate the actual output at iteration $p = 1$

$$Y(p) = step\left[\sum_{i=1}^{n} x_i(p) w_i(p) - \theta\right]$$

- where n is the number of the perceptron inputs, and step is a step activation function.

activation function
sign function



- **Step 3: Weight training**

  error $e = Y_{expected} - Y_{actual}$

  - Update the weights of the perceptron

$$w_i(p+1) = w_i(p) + \Delta w_i(p),$$

where Δw is the weight correction at iteration p. The weight correction is computed by the delta rule:

$$\Delta w_i(p) = \alpha \times x_i(p) \times e(p)$$

α is the *learning rate* (between 0 and 1)

- **Step 4: Iteration**
  - Increase iteration p by one, go back to Step 2 and repeat the process until convergence.

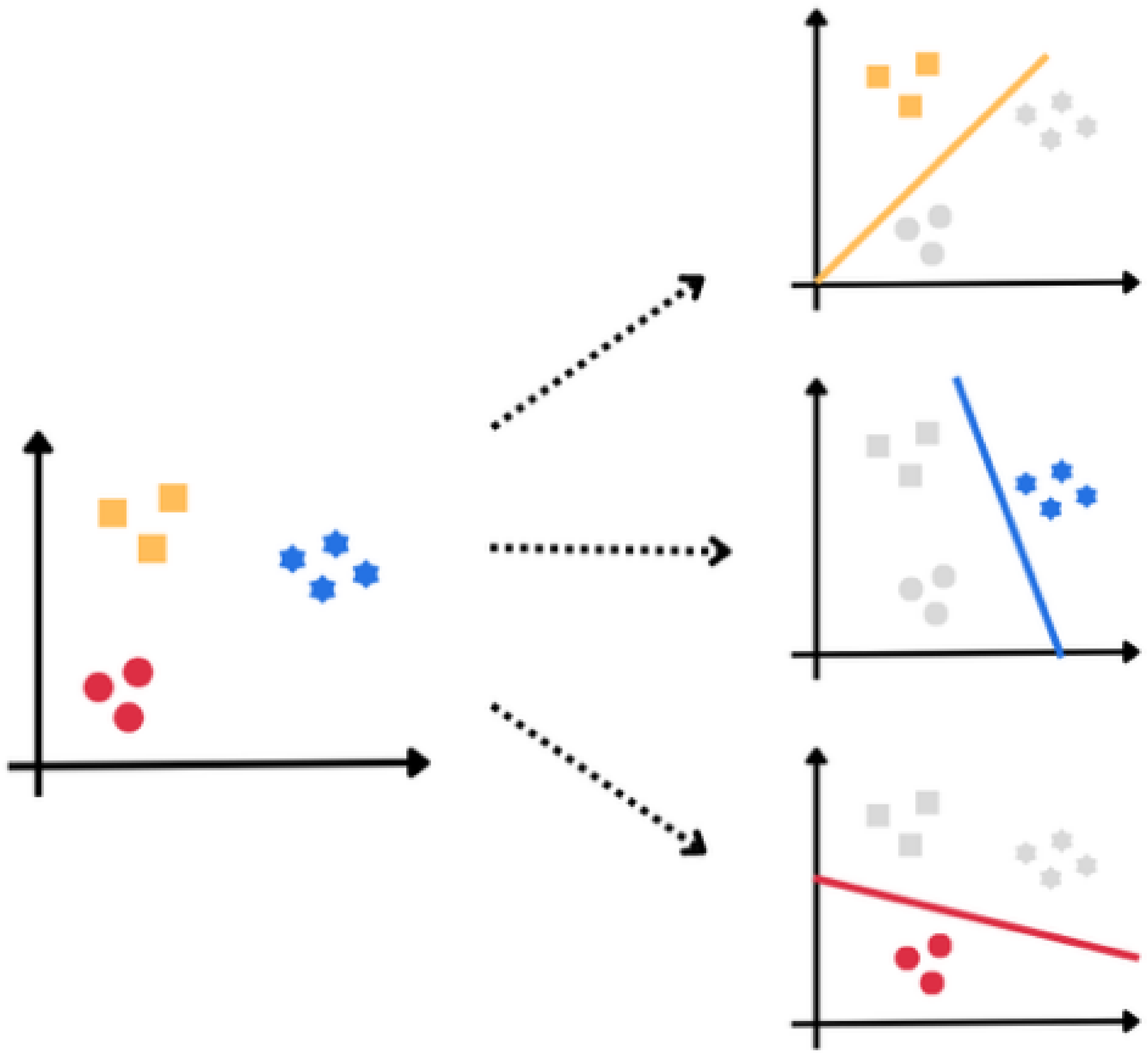
multi-class classification

One-against-all

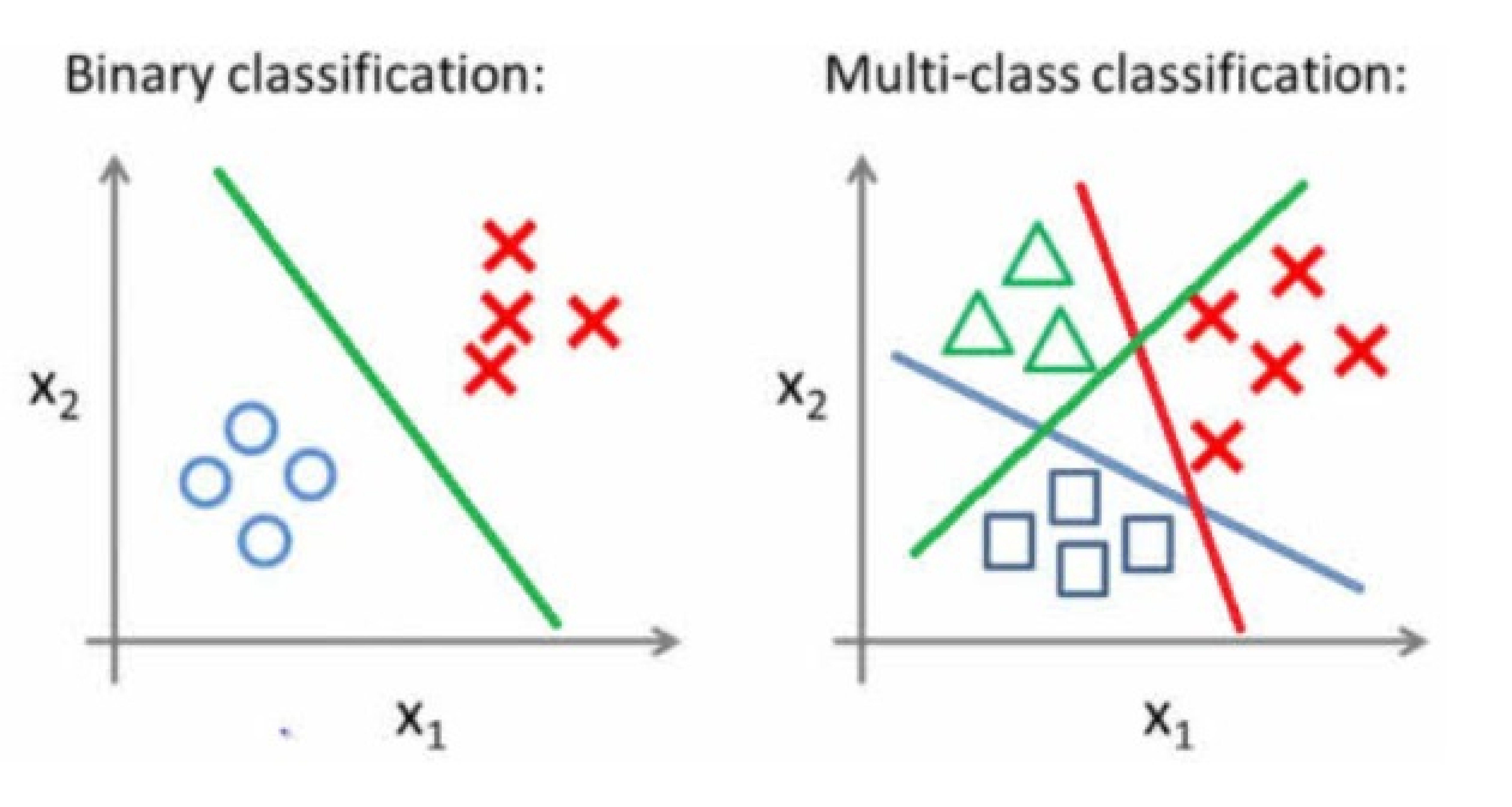
The one-aginst-all (also known as 1-v-r or one-versus-rest) is the probably earliest implementation for multi-class SVM classification.
In this approach, an SVM is constructed for each class by discriminating that class against the remaining (M − 1) classes. The number of SVMs used in this approach is M. A test pattern x is classified by using the winner-takes-all decision strategy, i.e., the class with th maximum value of the discriminant function f(x) is assigned to it. All the N training examples are used in constructing an SVM for a class. The SVM for class k is constructed using the set of training examples and their desired outputs, (xi , yi).. The desired output yi for a training example xi is defined as follows:

$$y_i = \begin{cases} +1 & \text{if } c_i = k \\ -1 & \text{if } c_i \neq k \end{cases}$$

The examples with the desired output yi = +1 are called positive examples and the examples with the desired output yi = −1 are called negative examples. An optimal hyperplane is constructed to separate N/M positive examples from N(M − 1)/M negative examples. The one against-all algorithm was implemented in MSVM.m with option ’1vr’ which extends a binary SVMs implementation SVM.m.



When we have a set of classes, each time one is taken and considered positive while the rest are considered negative, and a line is drawn on this basis, and we repeat the process for all classes


Binary classification:
x2
x1
Multi-class classification:
x2
x1


SEND
HELP