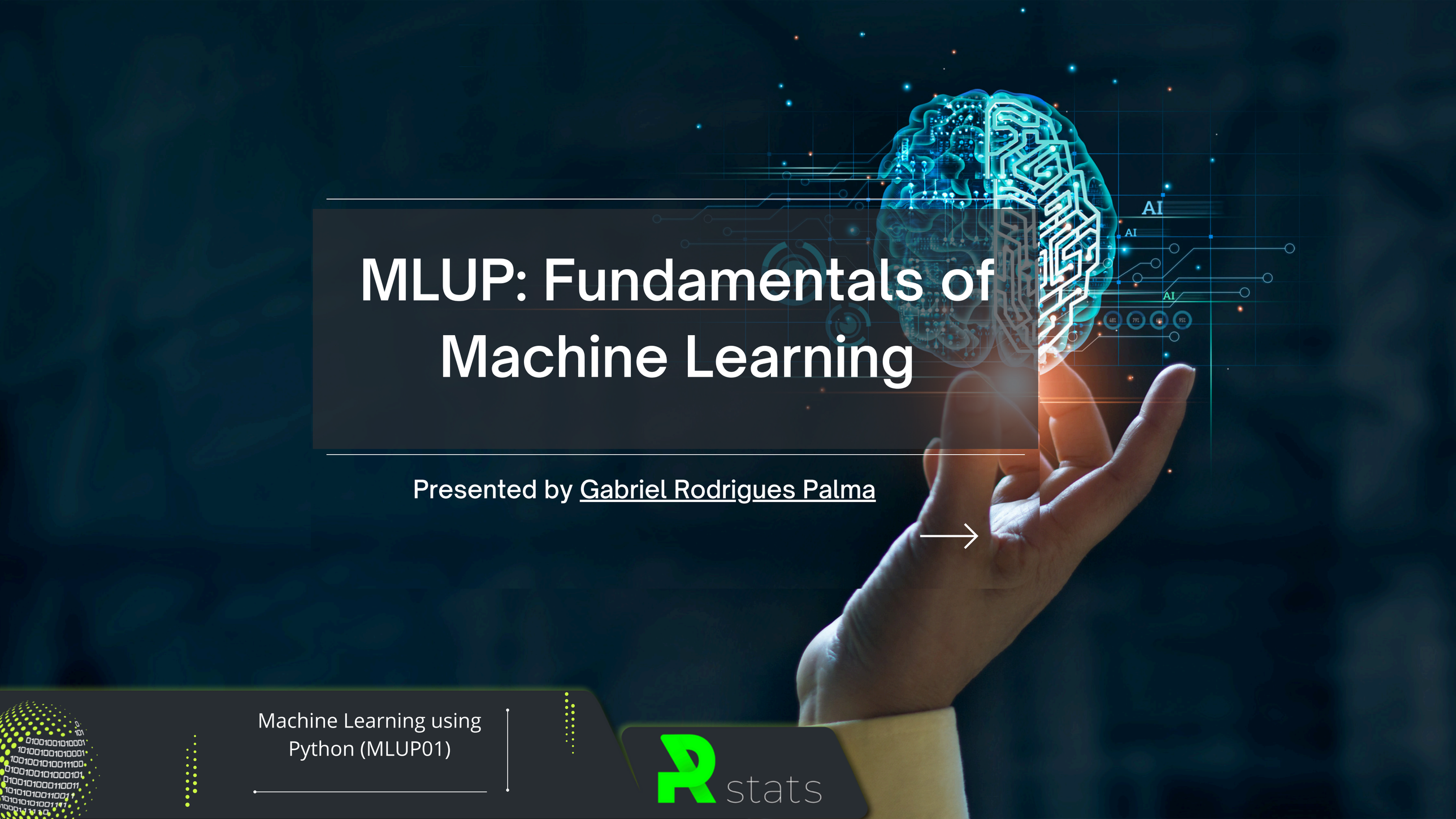
MLUP: Fundamentals of Machine Learning

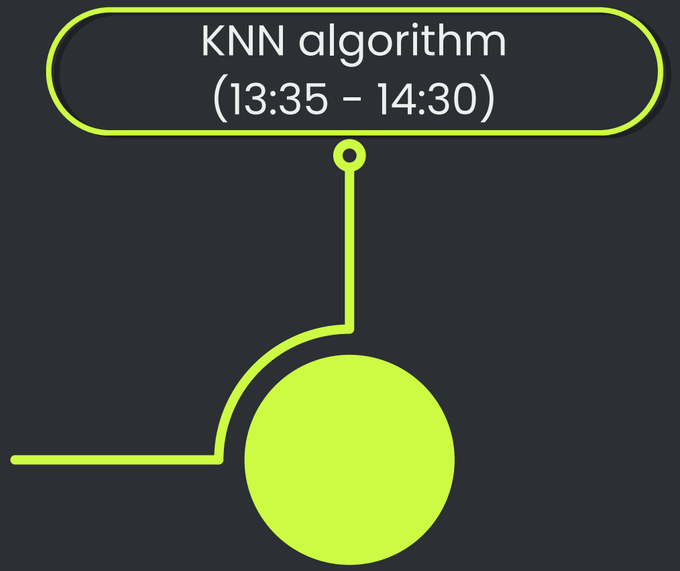
Presented by Gabriel Rodrigues Palma



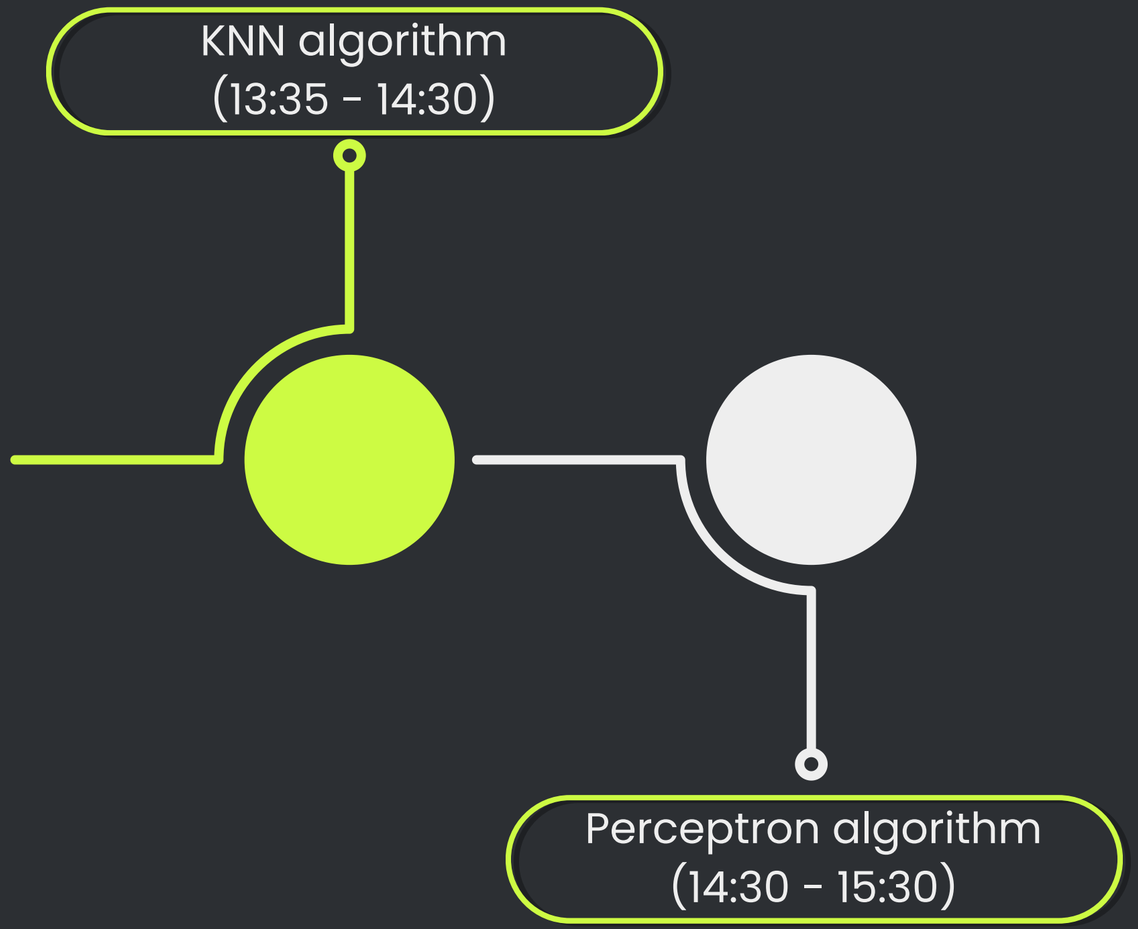
Machine Learning using Python (MLUP01)

stats

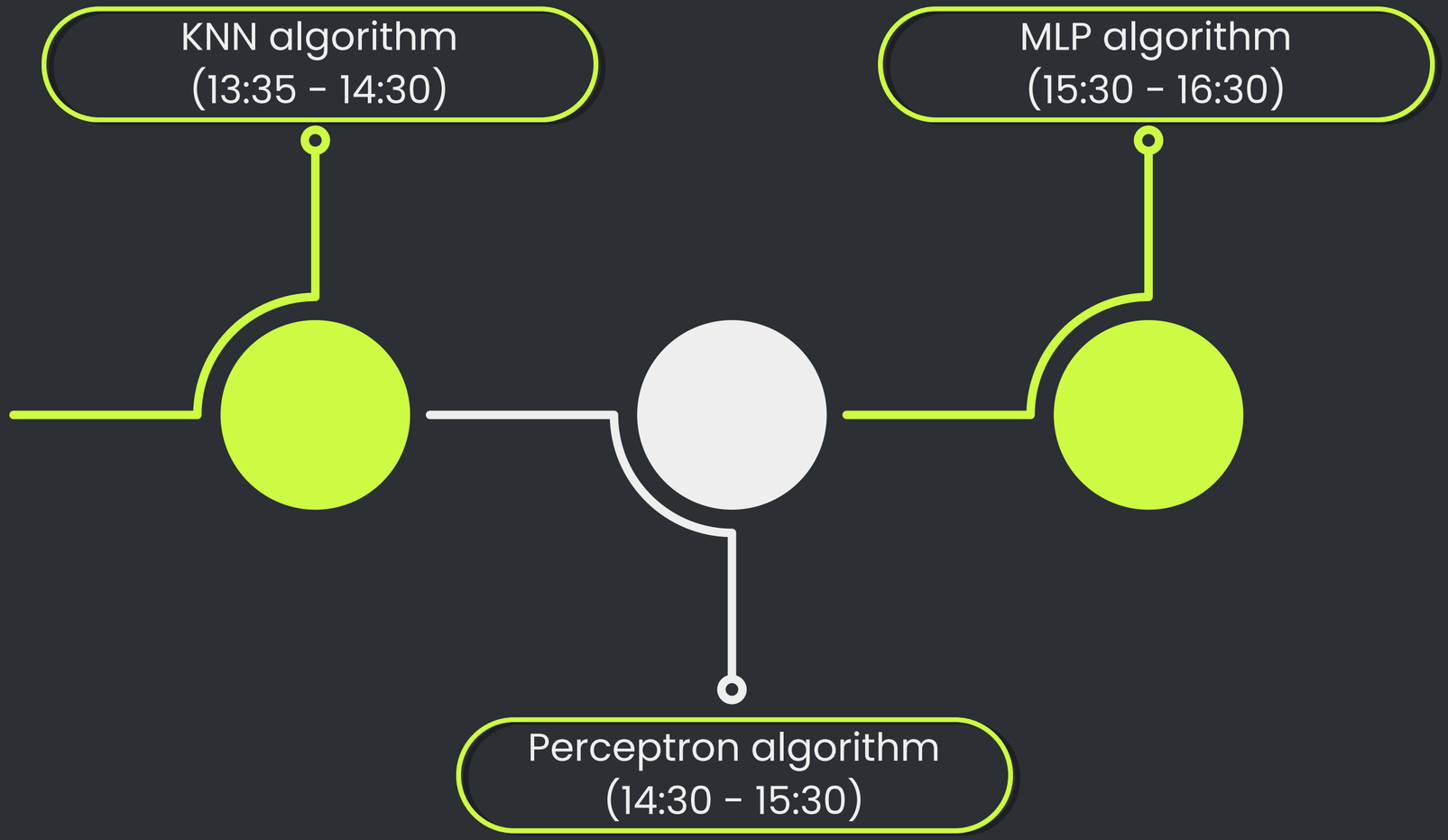
Day 4 (13:30 - 17:30)



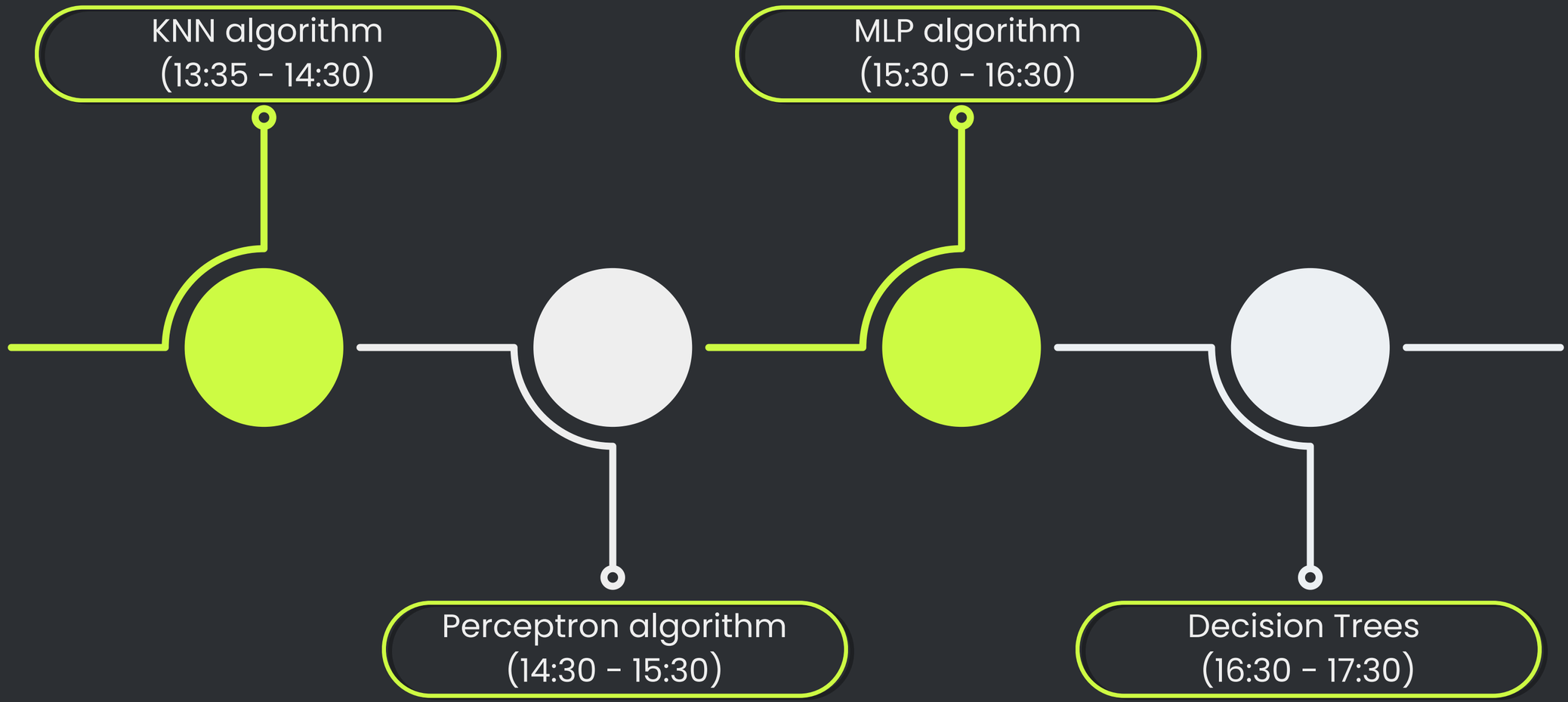
Day 4 (13:30 - 17:30)



Day 4 (13:30 - 17:30)



Day 4 (13:30 - 17:30)



KNN algorithm

Machine Learning using Python (MLUP01)

stats

KNN algorithm

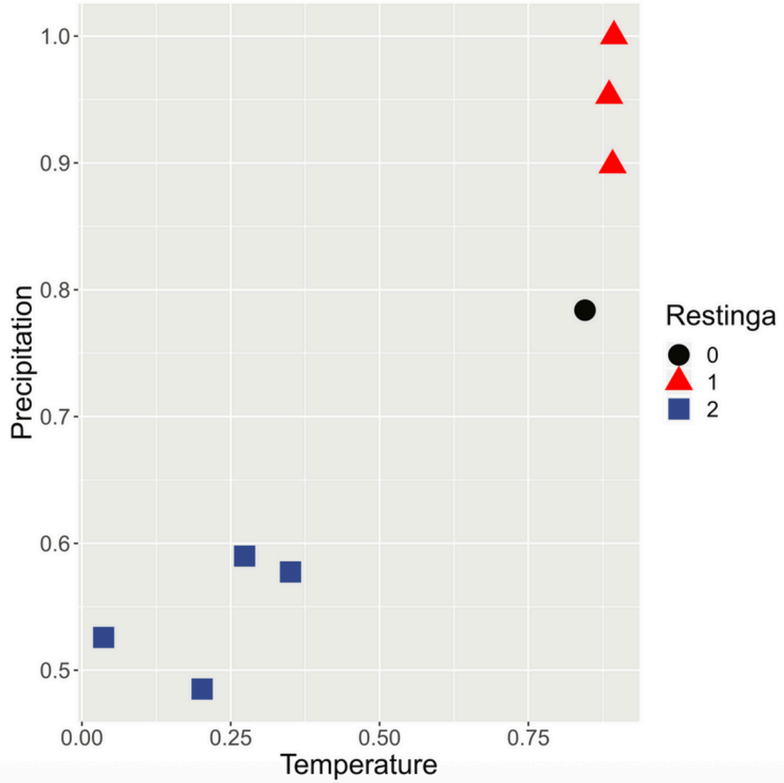
Machine Learning using Python (MLUP01)

stats

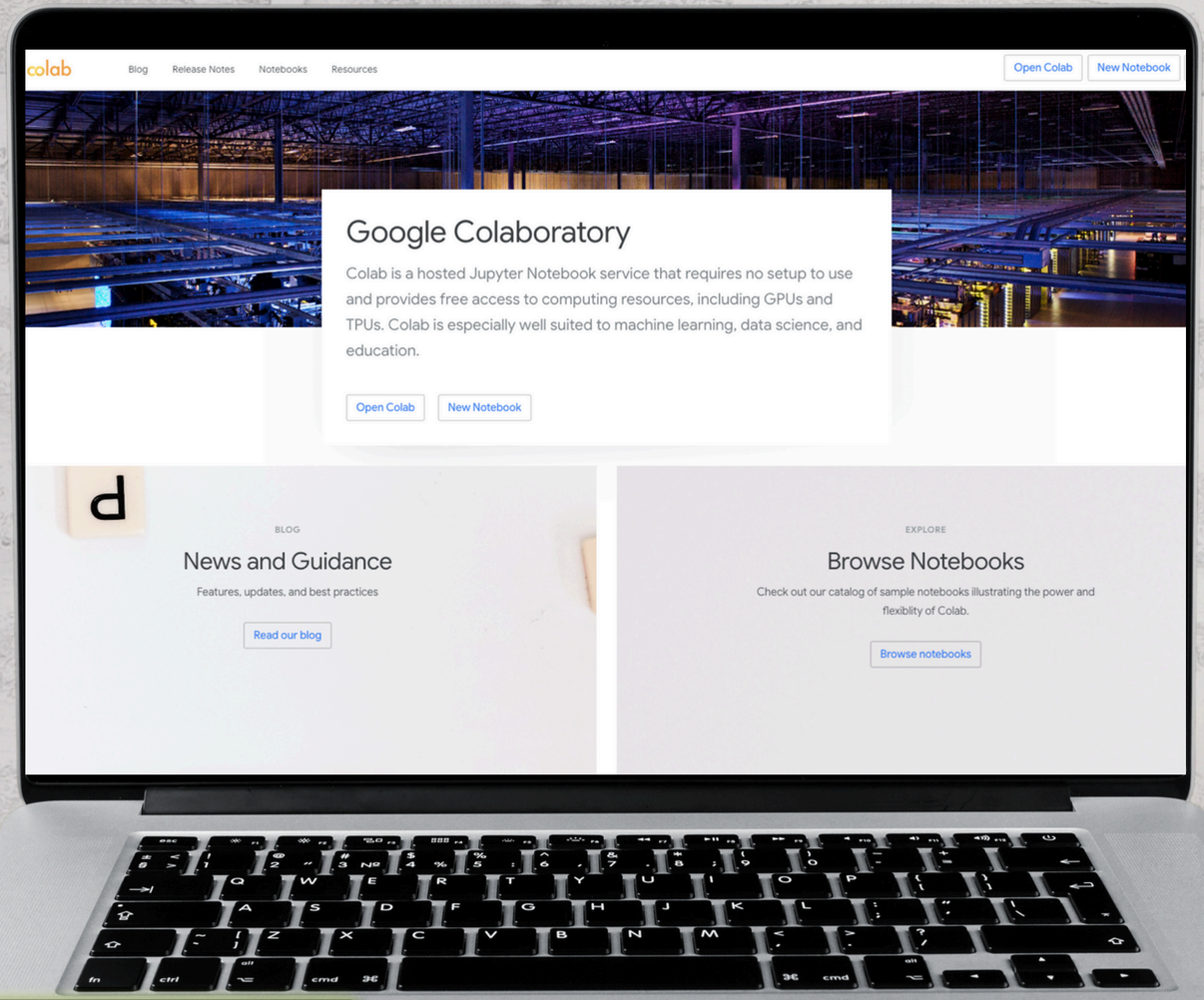
KNN algorithm

Temp. Range	Humidity	Precipitation	Atm	Wind	FAPAR	Ludwigia leptocarpa	Restinga
0.72	0.61	0.59	0.98	0.19	0.67	1	1
0.92	0.58	0.52	0.99	0.17	0.37	0	0
0.86	0.82	0.62	1.00	0.12	0.80	0	1
0.82	0.87	0.63	1.00	0.17	1.00	0	0

KNN algorithm



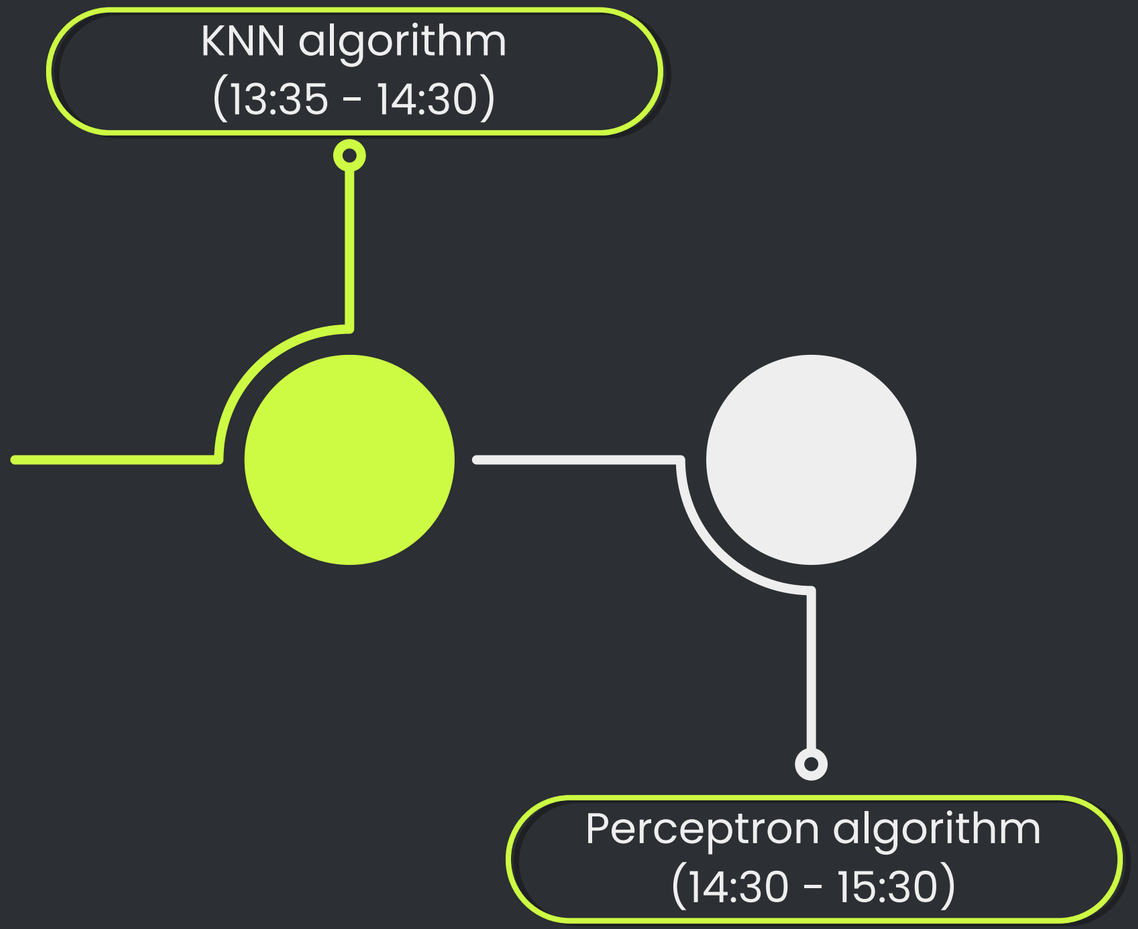
KNN algorithm



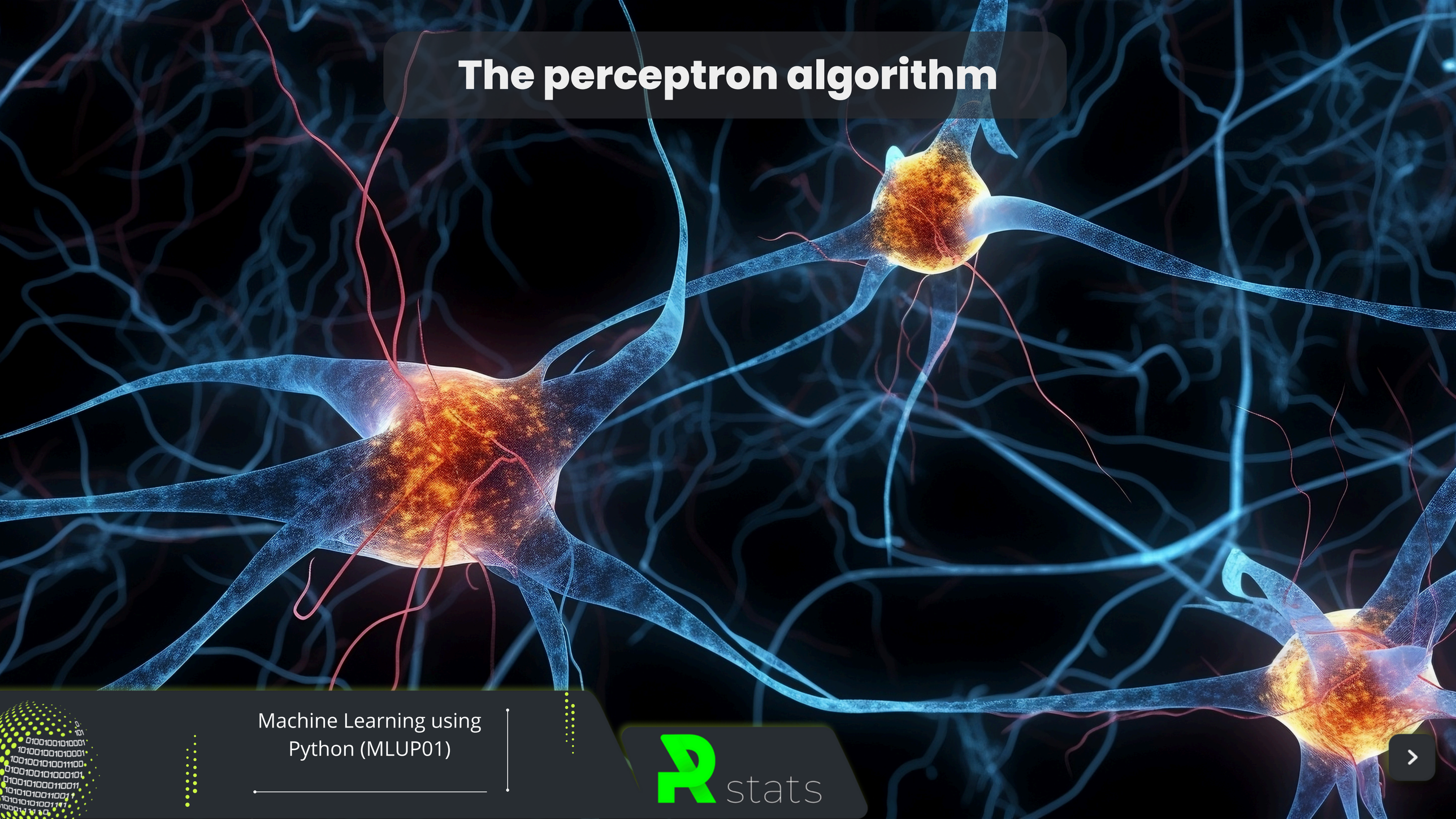
Machine Learning using Python (MLUP01)

stats

Day 4 (13:30 - 17:30)



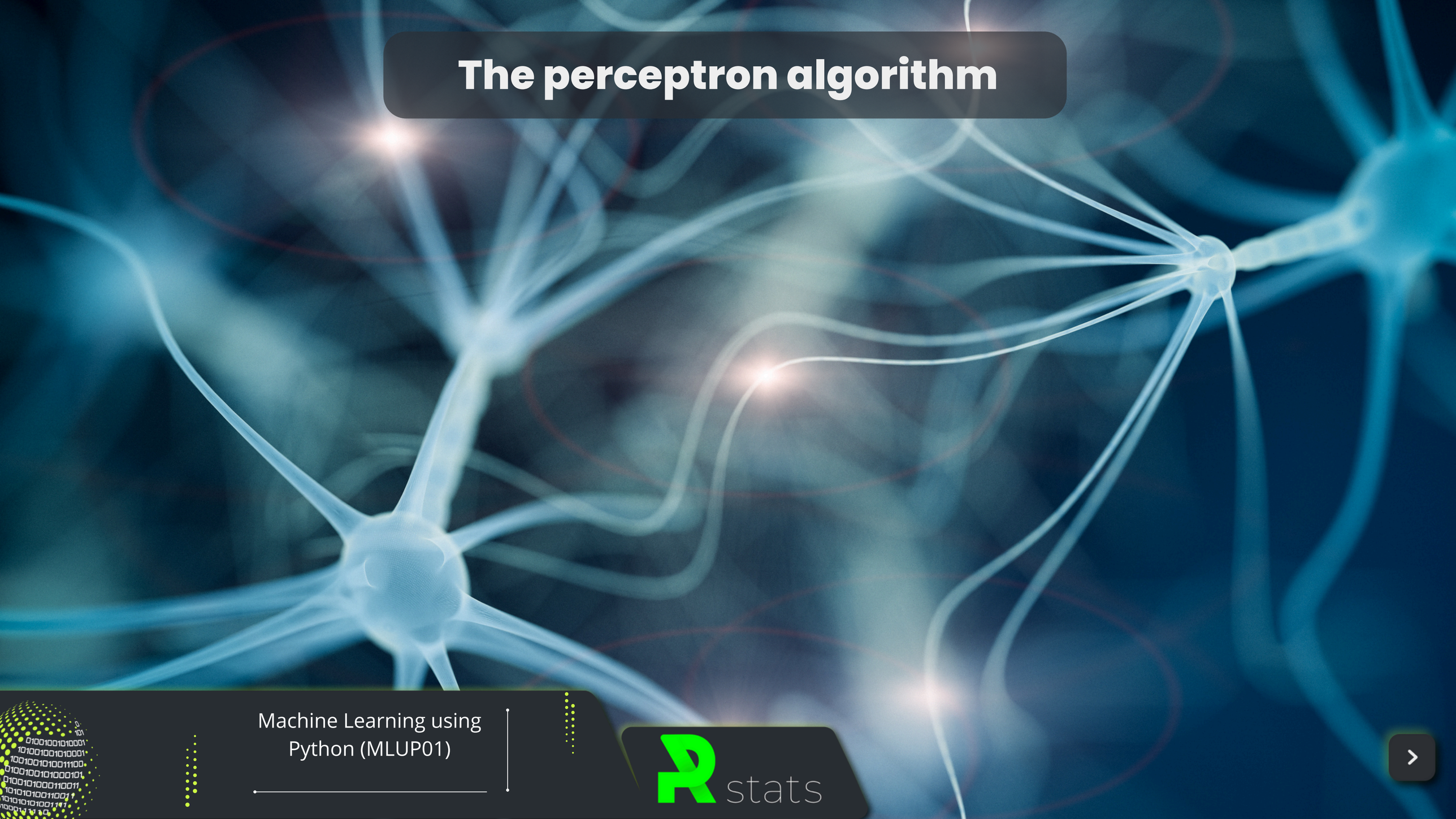
The perceptron algorithm



Machine Learning using Python (MLUP01)

stats

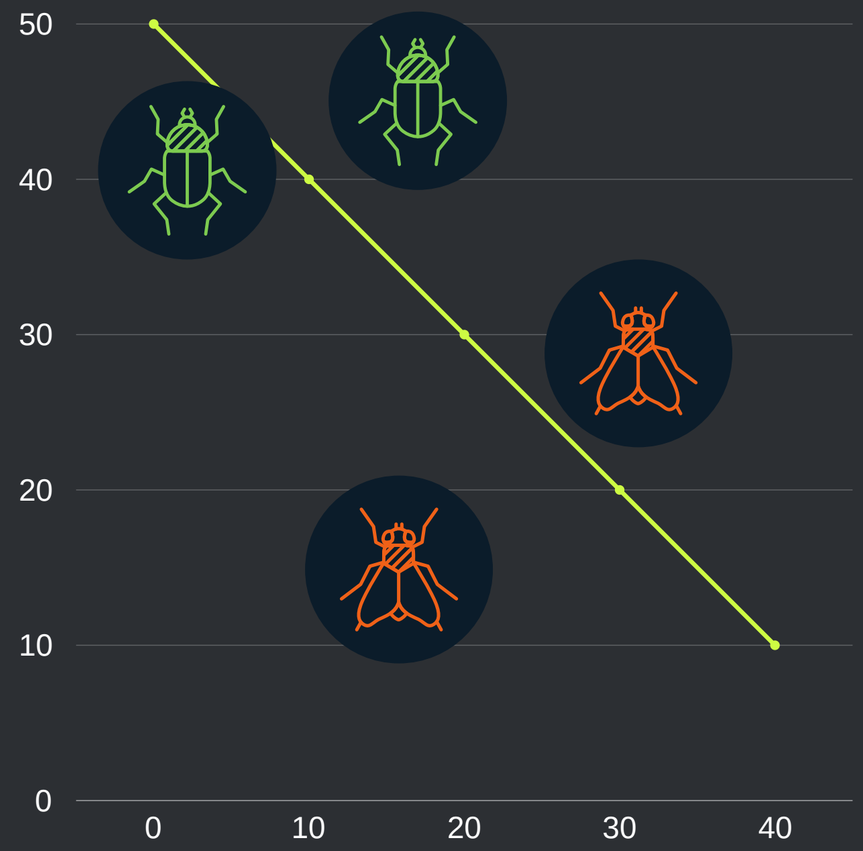
The perceptron algorithm



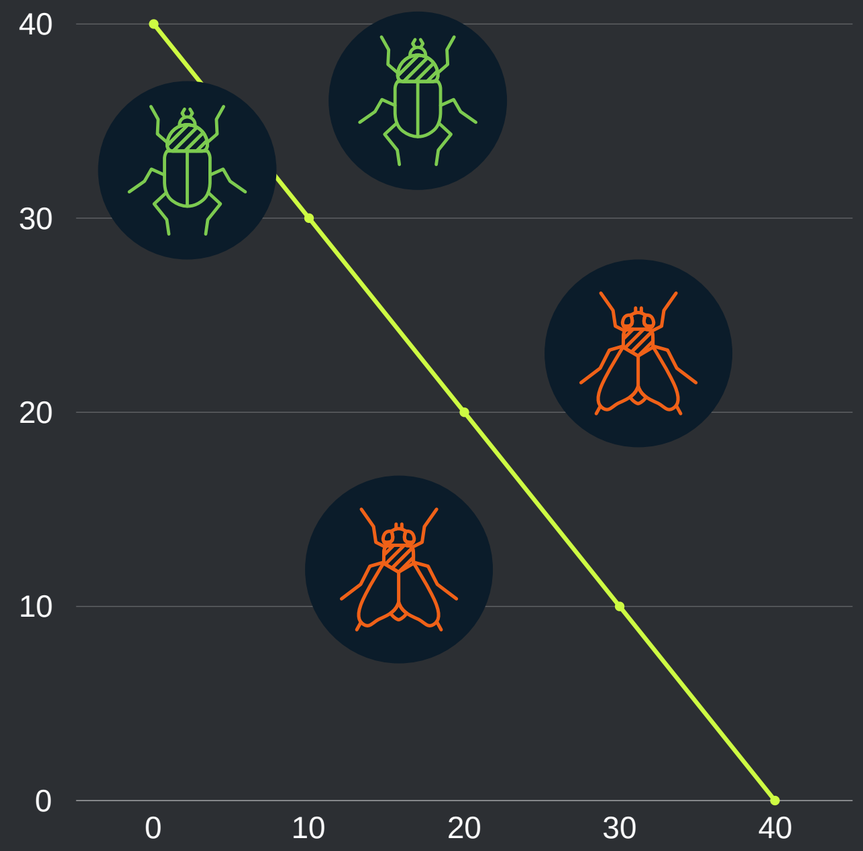
Machine Learning using Python (MLUP01)

stats

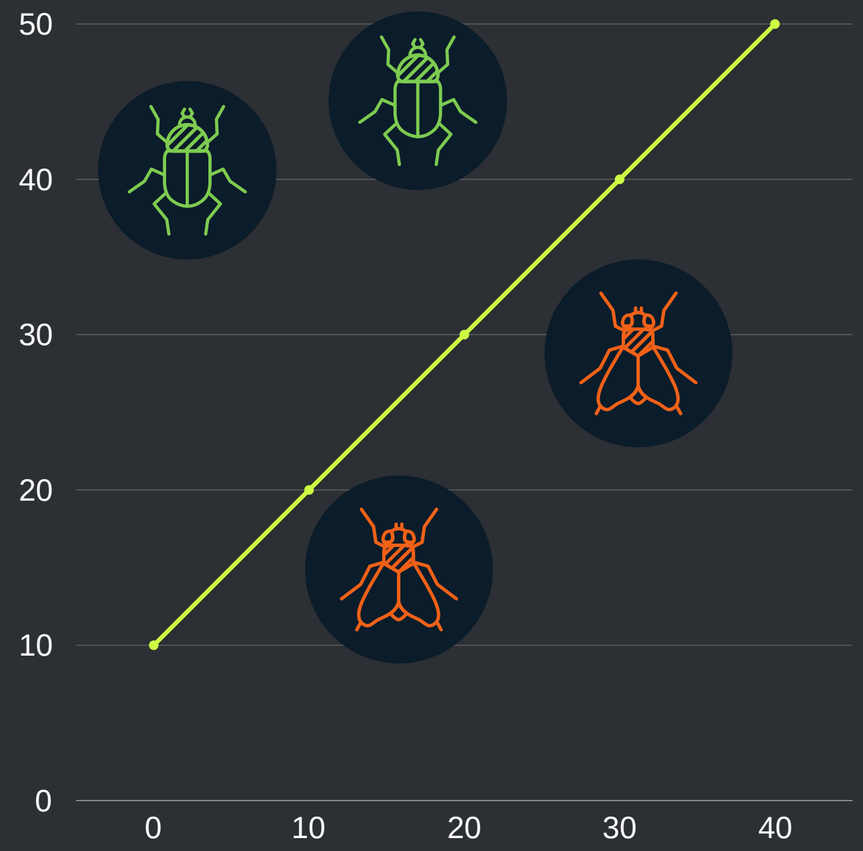
The perceptron algorithm



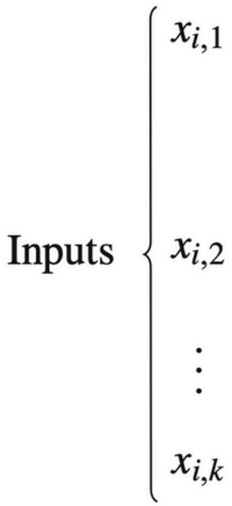
The perceptron algorithm



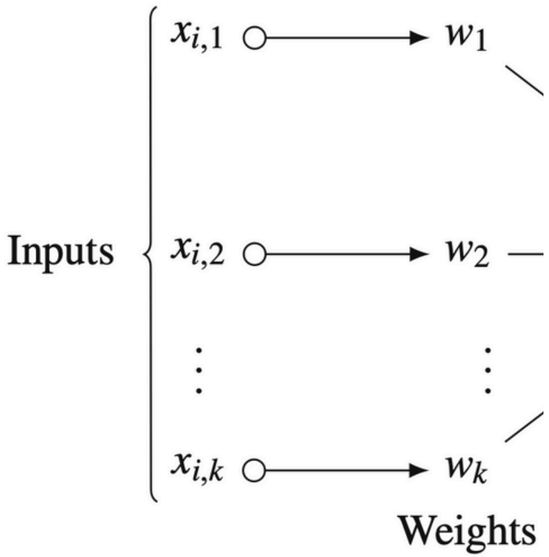
The perceptron algorithm



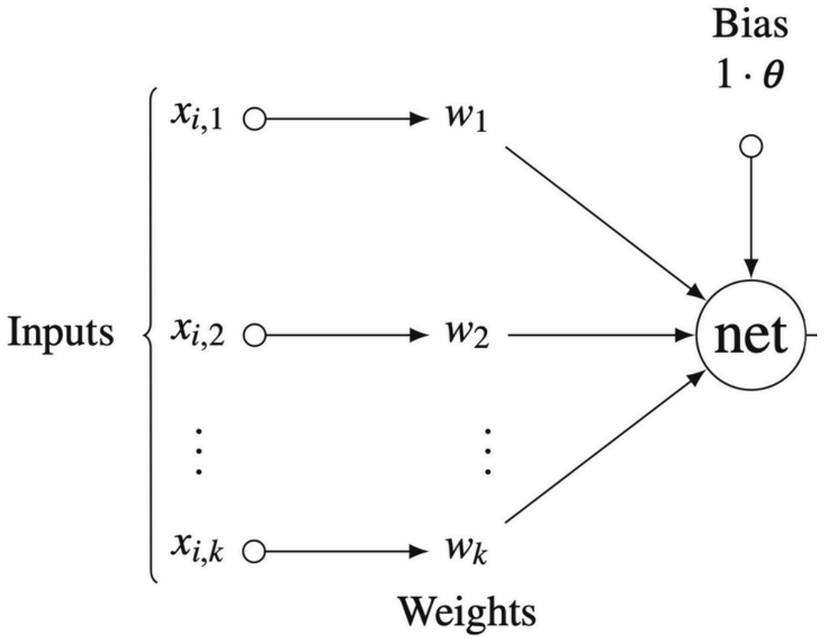
The perceptron algorithm



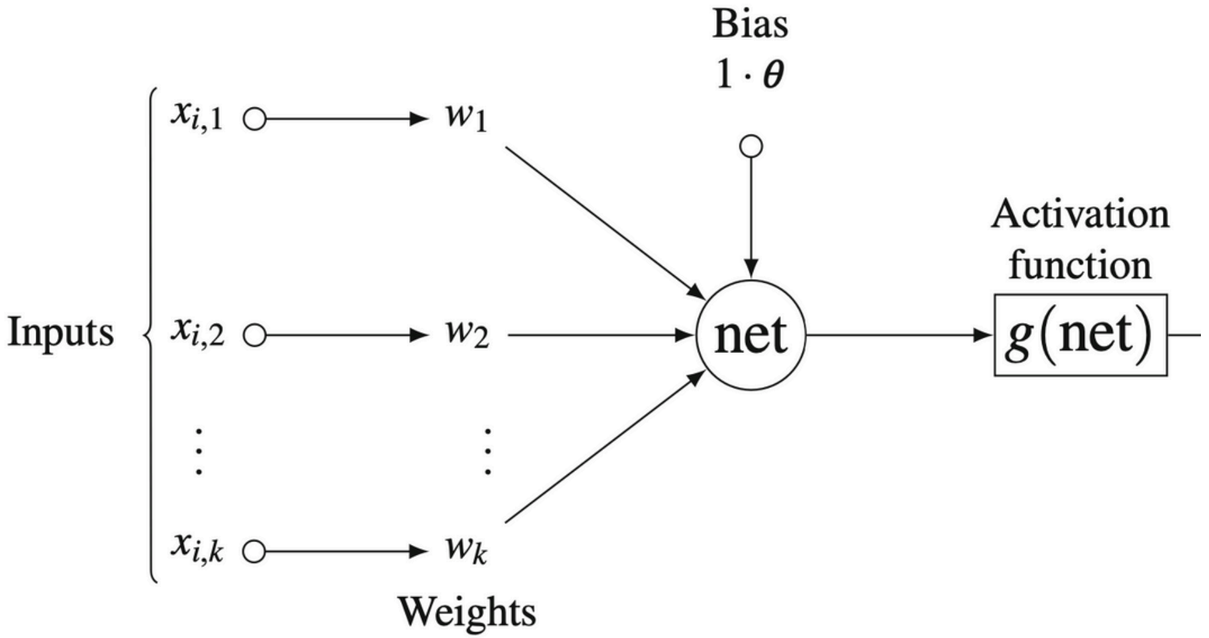
The perceptron algorithm



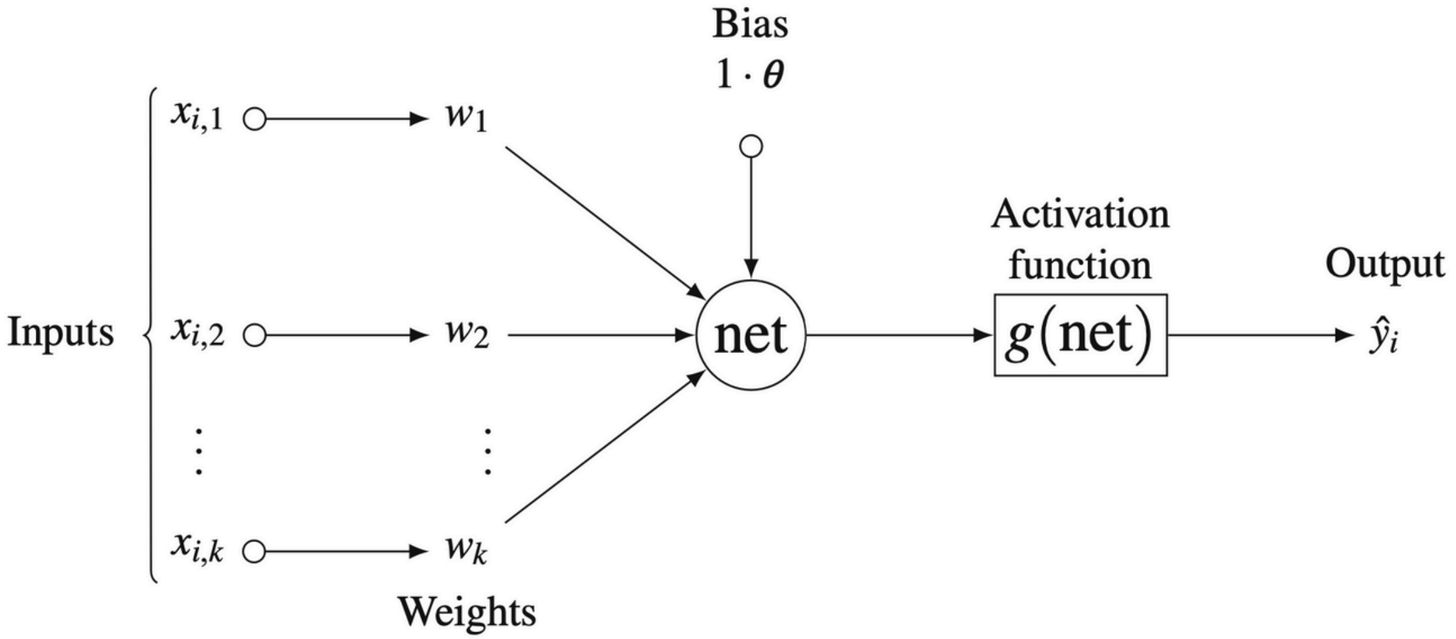
The perceptron algorithm



The perceptron algorithm



The perceptron algorithm

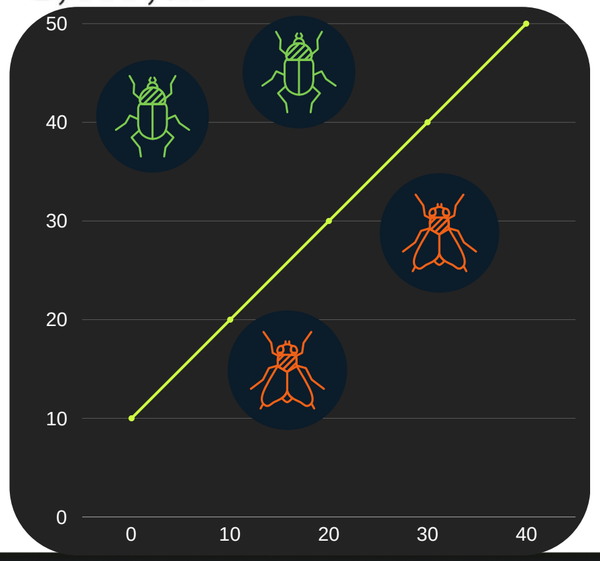


The perceptron algorithm

The Perceptron algorithm was proposed by Rosenblatt [11] in 1957. It considers an artificial neuron or unit[4] that receives an input example x_i to produce an output class y_i, as illustrated in Fig. 1.29. Such input example $x_i \in \mathbb{R}^k$, i.e. it contains a list of indexed variables $j = 1, \ldots, k$.

The perceptron algorithm

The Perceptron algorithm was proposed by Rosenblatt [11] in 1957. It considers an artificial neuron or unit[4] that receives an input example x_i to produce an output class y_i, as illustrated in Fig. 1.29. Such input example $x_i \in \mathbb{R}^k$, i.e. it contains a list of indexed variables $j = 1, \ldots, k$.

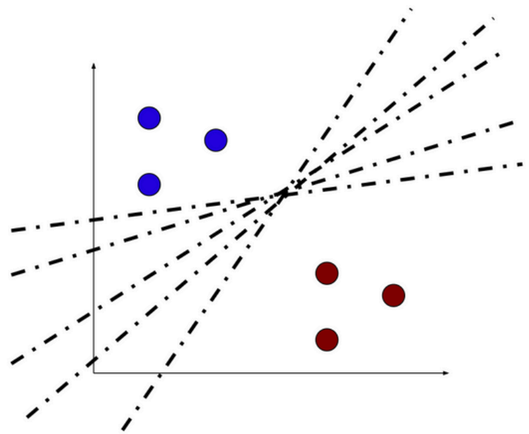


The perceptron algorithm

The neuron multiplies every input value $x_{i,j}$ by a weight w_j as well as constant 1 by the bias term[5] denoted by θ. As a next step, this algorithm computes a term $\text{net} = \sum_{j=1}^{k} x_{i,j} w_j + \theta$.

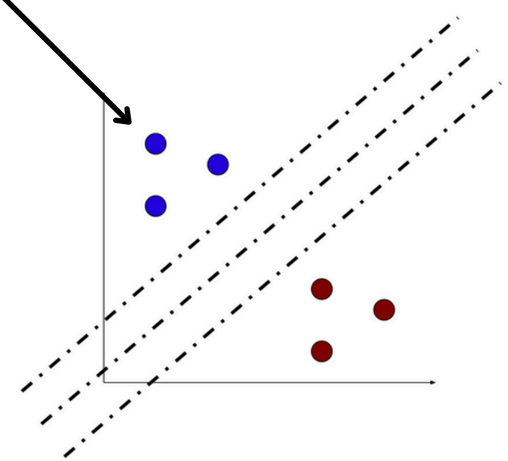
The perceptron algorithm

The neuron multiplies every input value $x_{i,j}$ by a weight w_j as well as constant 1 by the bias term[5] denoted by θ. As a next step, this algorithm computes a term $\text{net} = \sum_{j=1}^{k} x_{i,j} w_j + \theta$.



The perceptron algorithm

The neuron multiplies every input value $x_{i,j}$ by a weight w_j as well as constant 1 by the bias term[5] denoted by θ. As a next step, this algorithm computes a term $\text{net} = \sum_{j=1}^{k} x_{i,j} w_j + \theta$.



The perceptron algorithm

Next, the Perceptron applies a heaviside (or step) function $g(\text{net})$ (Eq. (1.10)), in which ϵ is a user-defined parameter, producing the output class $\hat{y}_i$ for example x_i.

$$g(\text{net}) = \begin{cases} 1, & \text{iff net} > \epsilon \\ 0, & \text{otherwise} \end{cases} \tag{1.10}$$

The perceptron algorithm

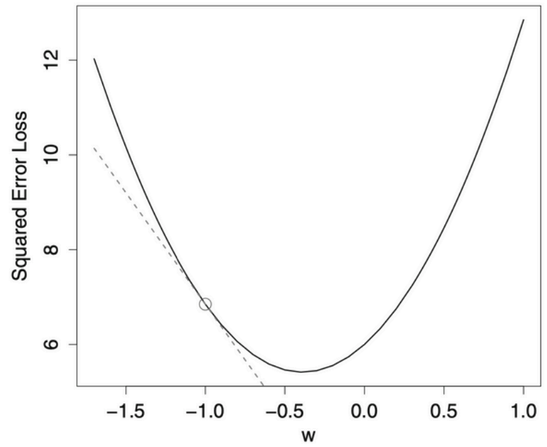
Perceptron while finding a good linear approximation function to solve this problem. The Perceptron aims to converge to the minimum of function E^2.

$$E^2 = \sum_{i=1}^{n} \ell_{\text{squared}}(x_i, y_i, g(x_i)) = \sum_{i=1}^{n} (y_i - g(x_i))^2 \tag{1.12}$$

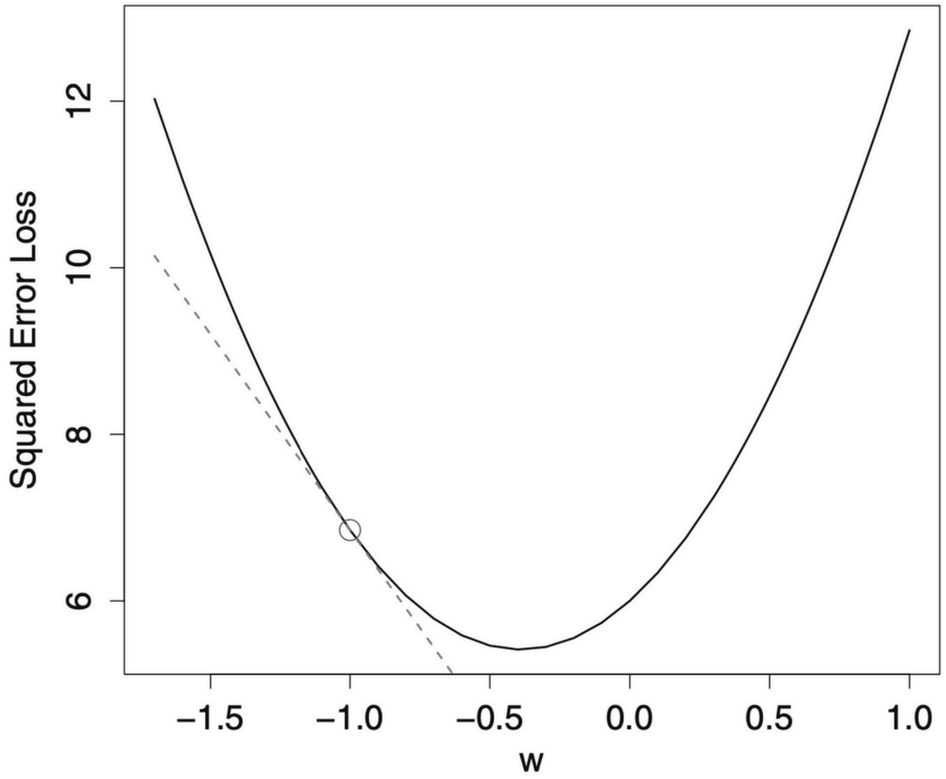
The perceptron algorithm

Perceptron while finding a good linear approximation function to solve this problem. The Perceptron aims to converge to the minimum of function E^2.

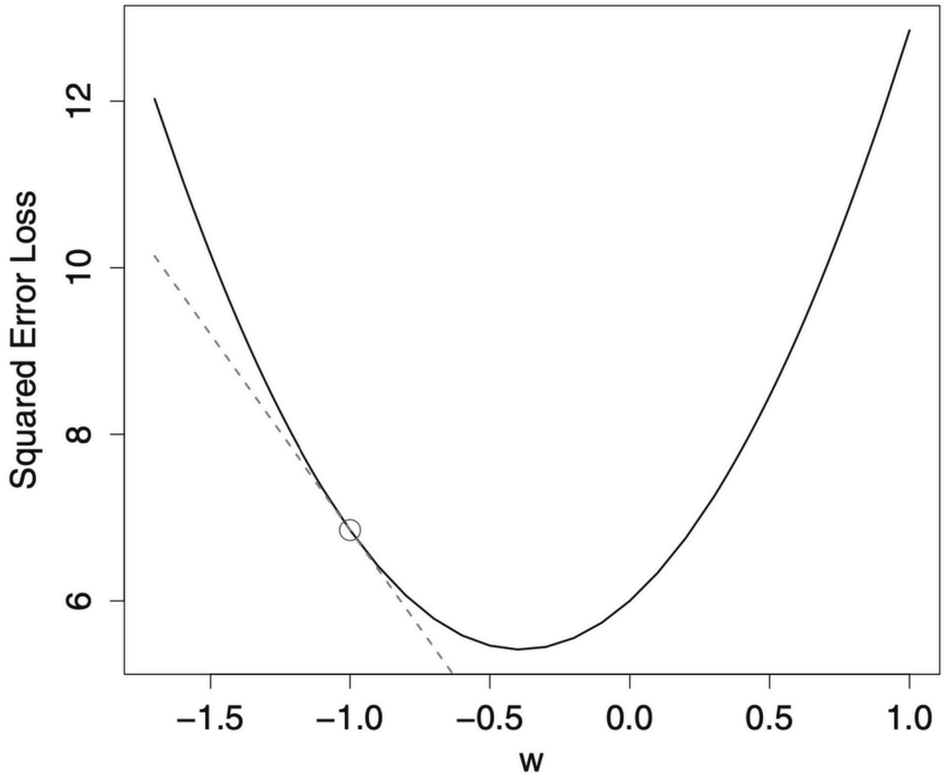
$$E^2 = \sum_{i=1}^{n} \ell_{\text{squared}}(x_i, y_i, g(x_i)) = \sum_{i=1}^{n} (y_i - g(x_i))^2 \tag{1.12}$$



The perceptron algorithm



The perceptron algorithm



$$w_1(t+1) = w_1(t) - \eta \frac{\partial E^2}{\partial w_1}$$

The perceptron algorithm

$$E^2 = \sum_{i=1}^{n} (y_i - f(x_i))^2 =$$

The perceptron algorithm

$$E^2 = \sum_{i=1}^{n} (y_i - f(x_i))^2 = \sum_{i=1}^{n} \left(y_i - \left[\sum_{j=1}^{k} x_{i,j} w_j + \theta \right] \right)^2,$$

The perceptron algorithm

$$E^2 = \sum_{i=1}^{n} (y_i - f(x_i))^2 = \sum_{i=1}^{n} \left(y_i - \left[\sum_{j=1}^{k} x_{i,j} w_j + \theta \right] \right)^2,$$

so, $\frac{\partial E^2}{\partial w_1}$ and $\frac{\partial E^2}{\partial \theta}$ are found via the chain rule for derivatives as follows:

The perceptron algorithm

$$E^2 = \sum_{i=1}^{n} (y_i - f(x_i))^2 = \sum_{i=1}^{n} \left(y_i - \left[\sum_{j=1}^{k} x_{i,j} w_j + \theta \right] \right)^2,$$

so, $\frac{\partial E^2}{\partial w_1}$ and $\frac{\partial E^2}{\partial \theta}$ are found via the chain rule for derivatives as follows:

$$\frac{\partial E^2}{\partial w_1} = 2 \sum_{i=1}^{n} (y_i - f(x_i)) \frac{\partial [y_i - f(x_i)]}{\partial w_1},$$

The perceptron algorithm

$$E^2 = \sum_{i=1}^{n} (y_i - f(x_i))^2 = \sum_{i=1}^{n} \left(y_i - \left[\sum_{j=1}^{k} x_{i,j} w_j + \theta \right] \right)^2,$$

so, $\frac{\partial E^2}{\partial w_1}$ and $\frac{\partial E^2}{\partial \theta}$ are found via the chain rule for derivatives as follows:

$$\frac{\partial E^2}{\partial w_1} = 2 \sum_{i=1}^{n} (y_i - f(x_i)) \frac{\partial [y_i - f(x_i)]}{\partial w_1},$$

and:

$$\frac{\partial E^2}{\partial \theta} = 2 \sum_{i=1}^{n} (y_i - f(x_i)) \frac{\partial [y_i - f(x_i)]}{\partial \theta}.$$

The perceptron algorithm

Recall y_i is the expected output class for example i (i.e., a label defined by some specialist), thus when differentiating it in terms of either w_1 or θ, y_i is disconsidered. Differentiating $\frac{\partial - f(x_i)}{\partial w_1}$ and $\frac{\partial - f(x_i)}{\partial \theta}$:

The perceptron algorithm

Recall y_i is the expected output class for example i (i.e., a label defined by some specialist), thus when differentiating it in terms of either w_1 or θ, y_i is disconsidered. Differentiating $\frac{\partial - f(x_i)}{\partial w_1}$ and $\frac{\partial - f(x_i)}{\partial \theta}$:

$$\frac{\partial - f(x_i)}{\partial w_1} = -\frac{\partial \left[x_{i,1} w_1 + \theta\right]}{\partial w_1} = -x_{i,1},$$

The perceptron algorithm

Recall y_i is the expected output class for example i (i.e., a label defined by some specialist), thus when differentiating it in terms of either w_1 or θ, y_i is disconsidered. Differentiating $\frac{\partial - f(x_i)}{\partial w_1}$ and $\frac{\partial - f(x_i)}{\partial \theta}$:

$$\frac{\partial - f(x_i)}{\partial w_1} = -\frac{\partial \left[x_{i,1} w_1 + \theta\right]}{\partial w_1} = -x_{i,1},$$

and:

$$\frac{\partial - f(x_i)}{\partial \theta} = -\frac{\partial \left[x_{i,1} w_1 + \theta\right]}{\partial \theta} = -1.$$

The perceptron algorithm

The Gradient Descent method is formulated as follows:

$$w_1(t+1) = w_1(t) - \eta \frac{\partial E^2}{\partial w_1}$$

The perceptron algorithm

The Gradient Descent method is formulated as follows:

$$w_1(t+1) = w_1(t) - \eta \frac{\partial E^2}{\partial w_1}$$

$$= w_1(t) - \eta \, 2 \sum_{i=1}^{n} (y_i - f(x_i))(-x_{i,1}),$$

The perceptron algorithm

The Gradient Descent method is formulated as follows:

$$
\begin{aligned}
w_1(t+1) &= w_1(t) - \eta \frac{\partial E^2}{\partial w_1} \\
&= w_1(t) - \eta\, 2 \sum_{i=1}^{n} (y_i - f(x_i))(-x_{i,1}), \\
&= w_1(t) - \eta\, 2 \sum (y_i - [x_{i,1} w_1 + \theta])(-x_{i,1}),
\end{aligned}
$$

The perceptron algorithm

The Gradient Descent method is formulated as follows:

$$\theta(t+1) = \theta(t) - \eta \frac{\partial E^2}{\partial \theta}$$

The perceptron algorithm

The Gradient Descent method is formulated as follows:

$$\theta(t+1) = \theta(t) - \eta \frac{\partial E^2}{\partial \theta}$$

$$= \theta(t) - \eta \, 2 \sum_{i=1}^{n} (y_i - f(x_i))(-1),$$

The perceptron algorithm

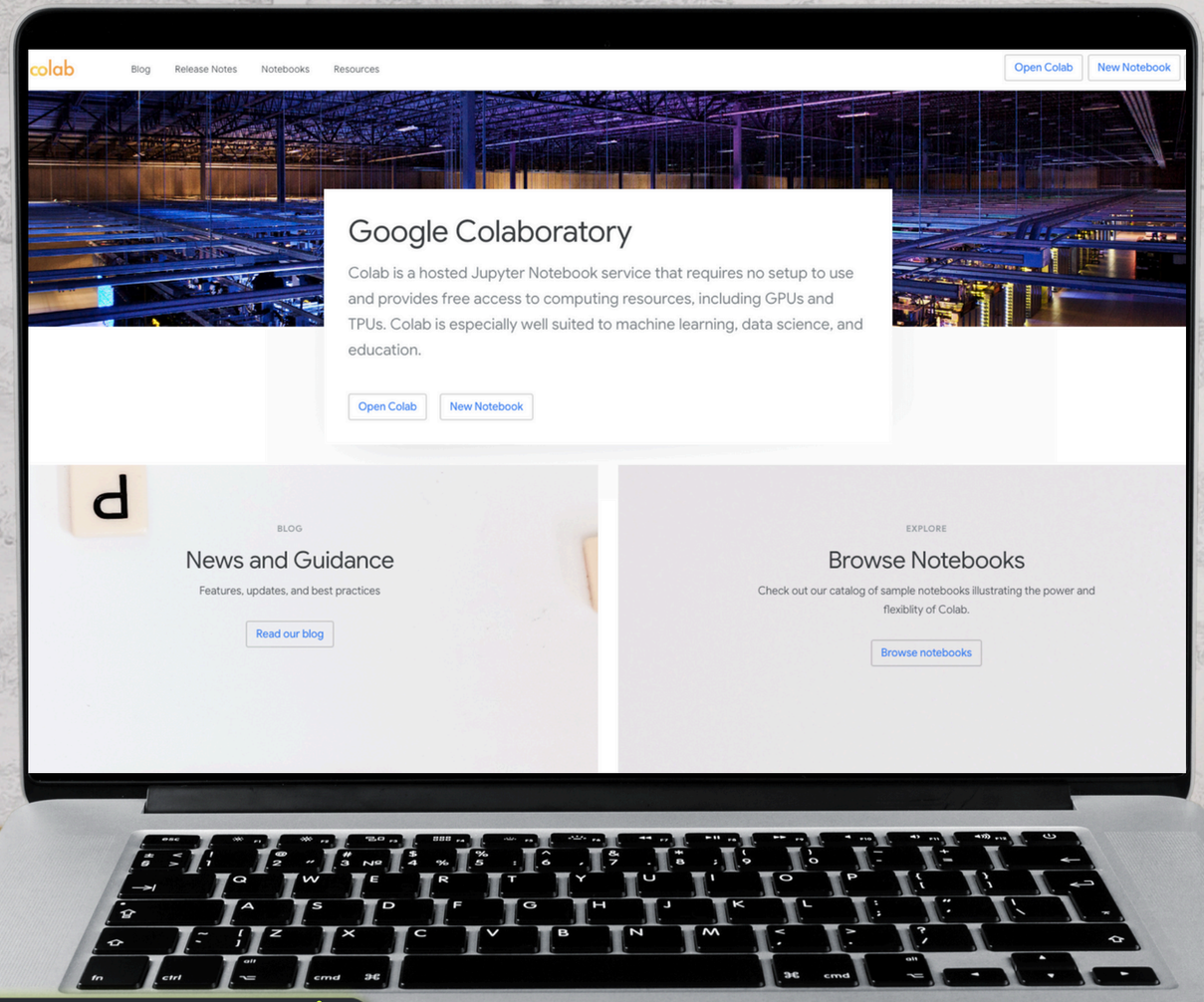
The Gradient Descent method is formulated as follows:

$$\theta(t+1) = \theta(t) - \eta \frac{\partial E^2}{\partial \theta}$$

$$= \theta(t) - \eta\, 2 \sum_{i=1}^{n} (y_i - f(x_i))(-1),$$

$$= \theta(t) - \eta\, 2 \sum_{i=1}^{n} (y_i - \left[x_{i,1} w_1 + \theta\right])(-1).$$

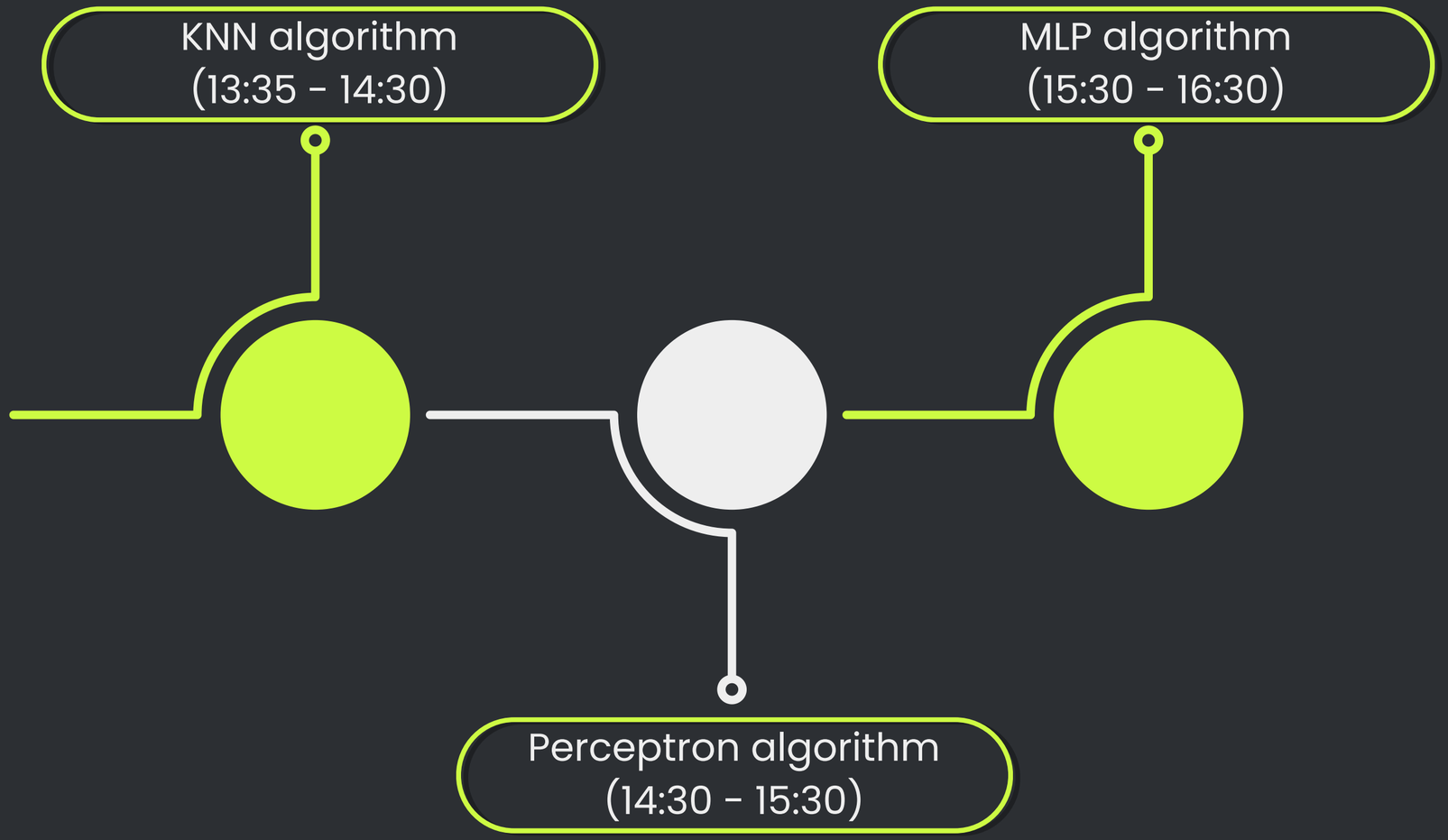
The perceptron algorithm



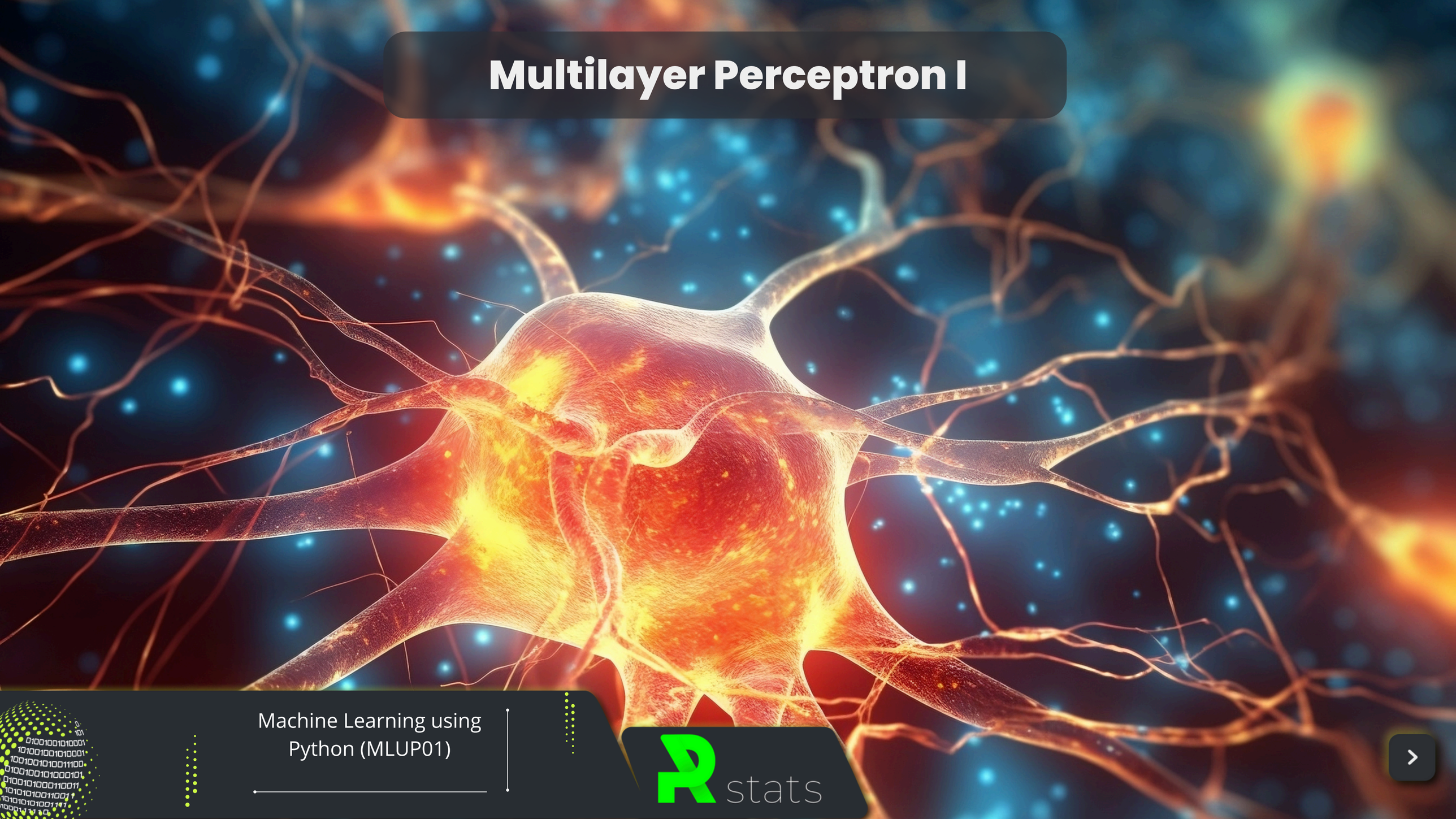
Machine Learning using Python (MLUP01)

stats

Day 4 (13:30 - 17:30)



Multilayer Perceptron I



Machine Learning using Python (MLUP01)

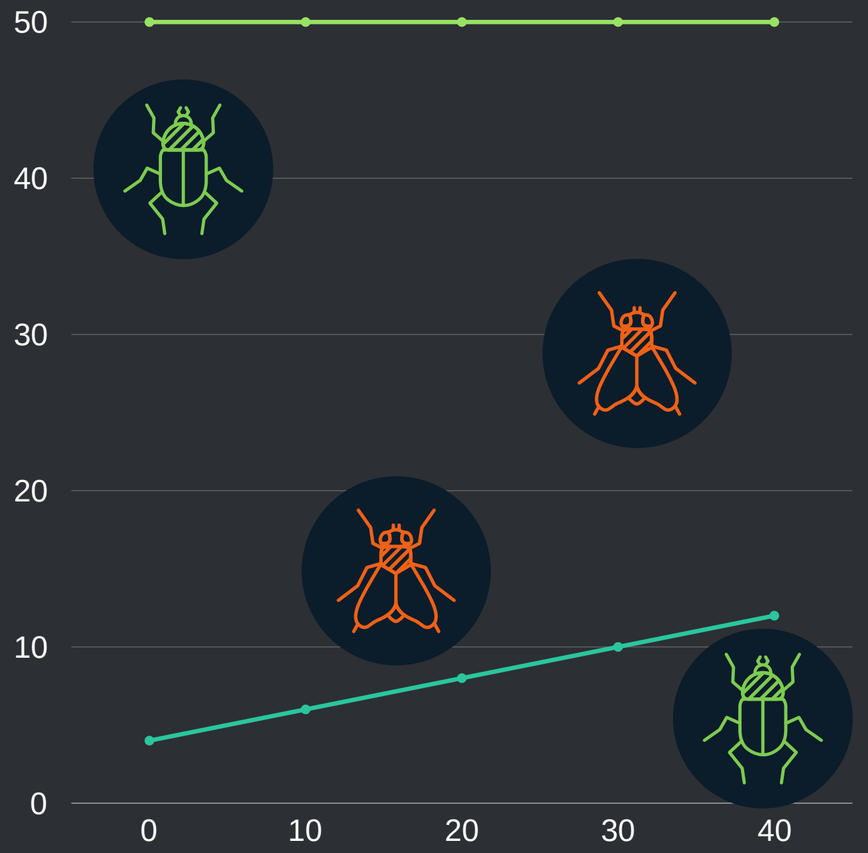
stats

Multilayer Perceptron I

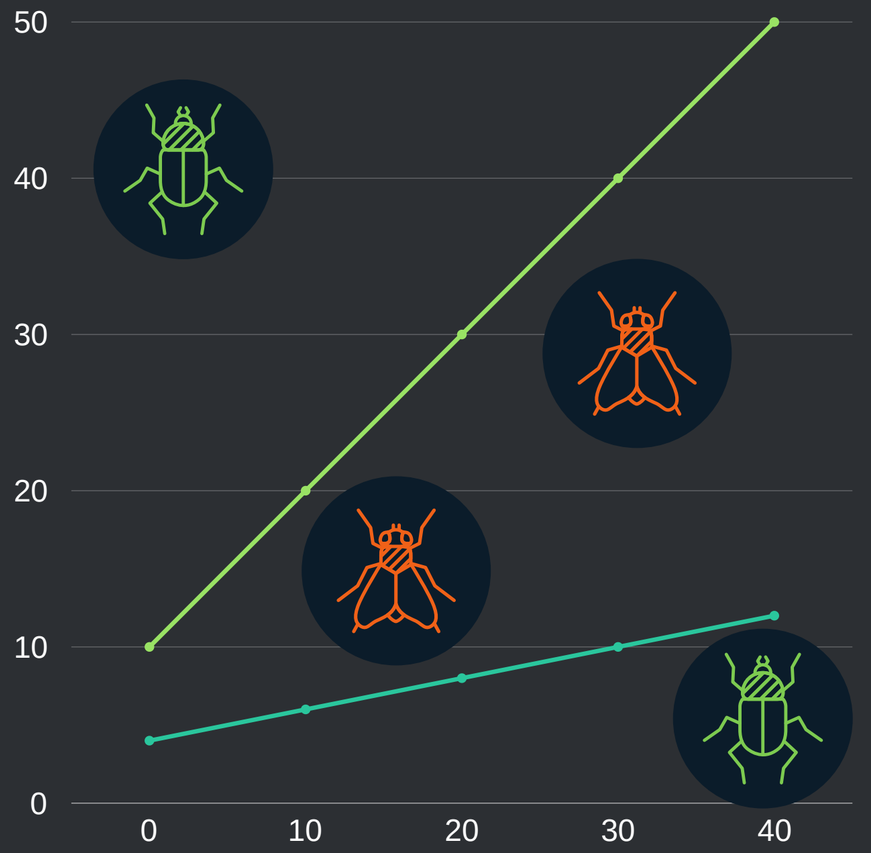
Machine Learning using Python (MLUP01)

stats

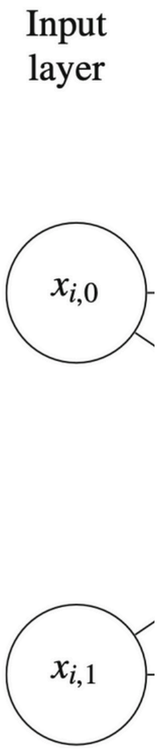
Multilayer Perceptron I



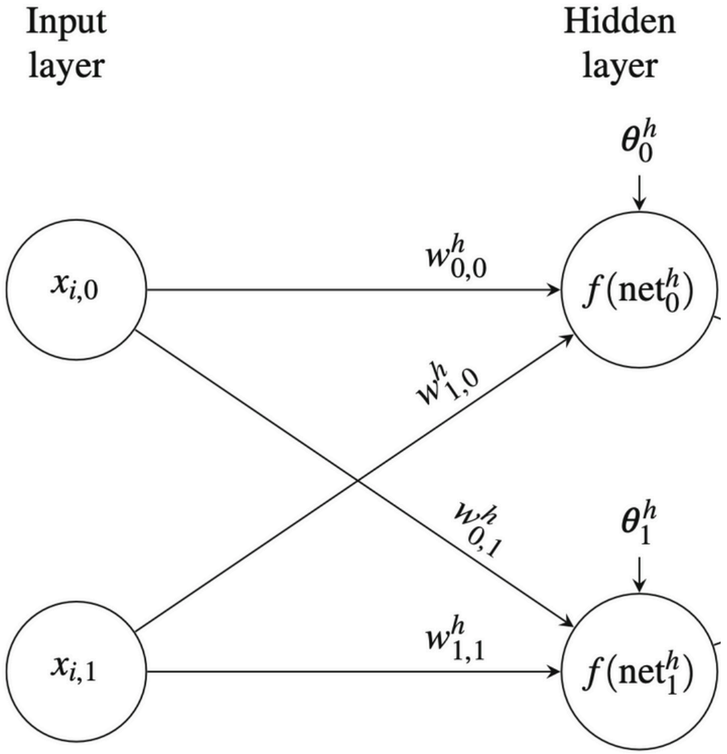
Multilayer Perceptron I



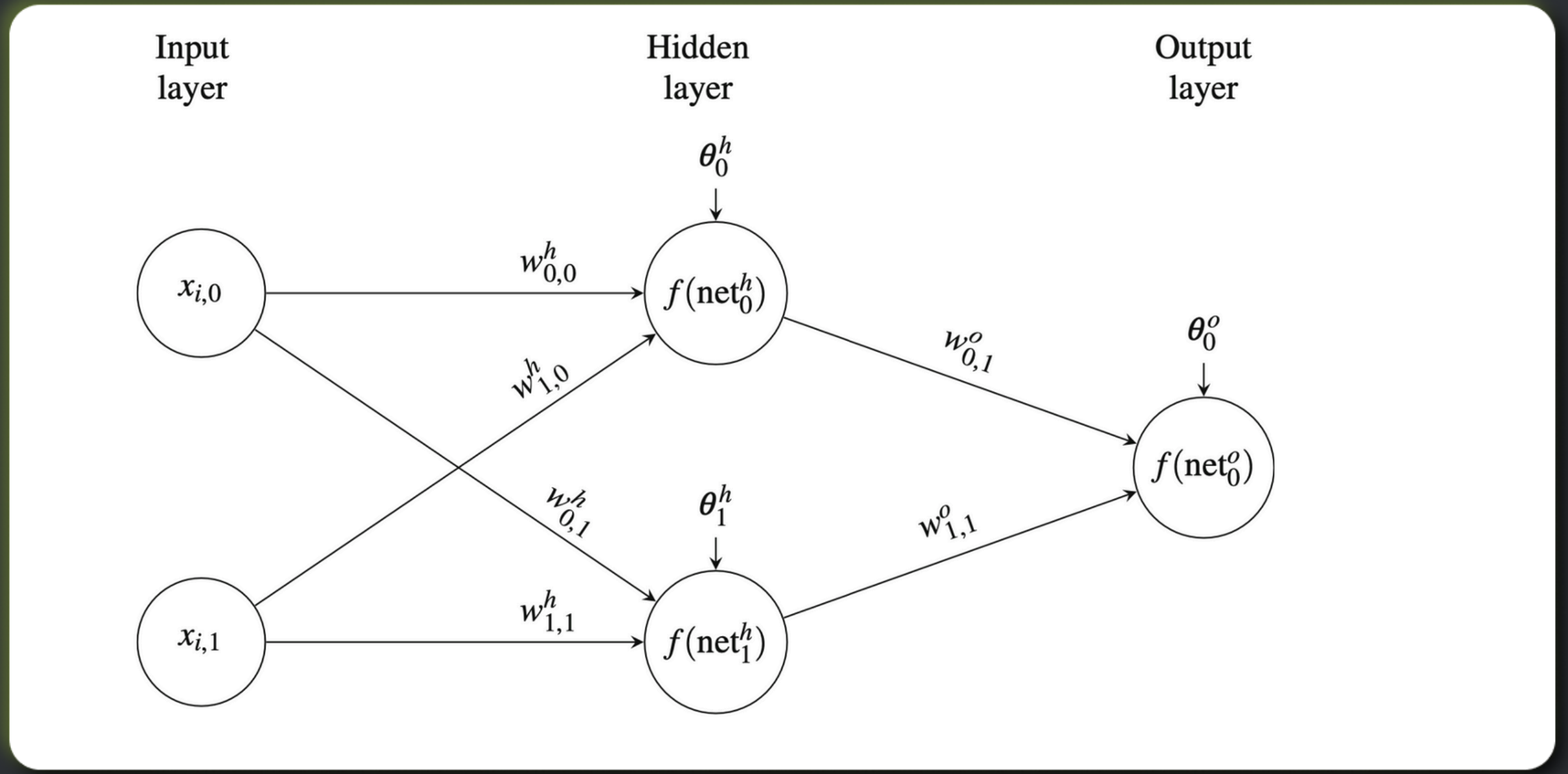
Multilayer Perceptron I



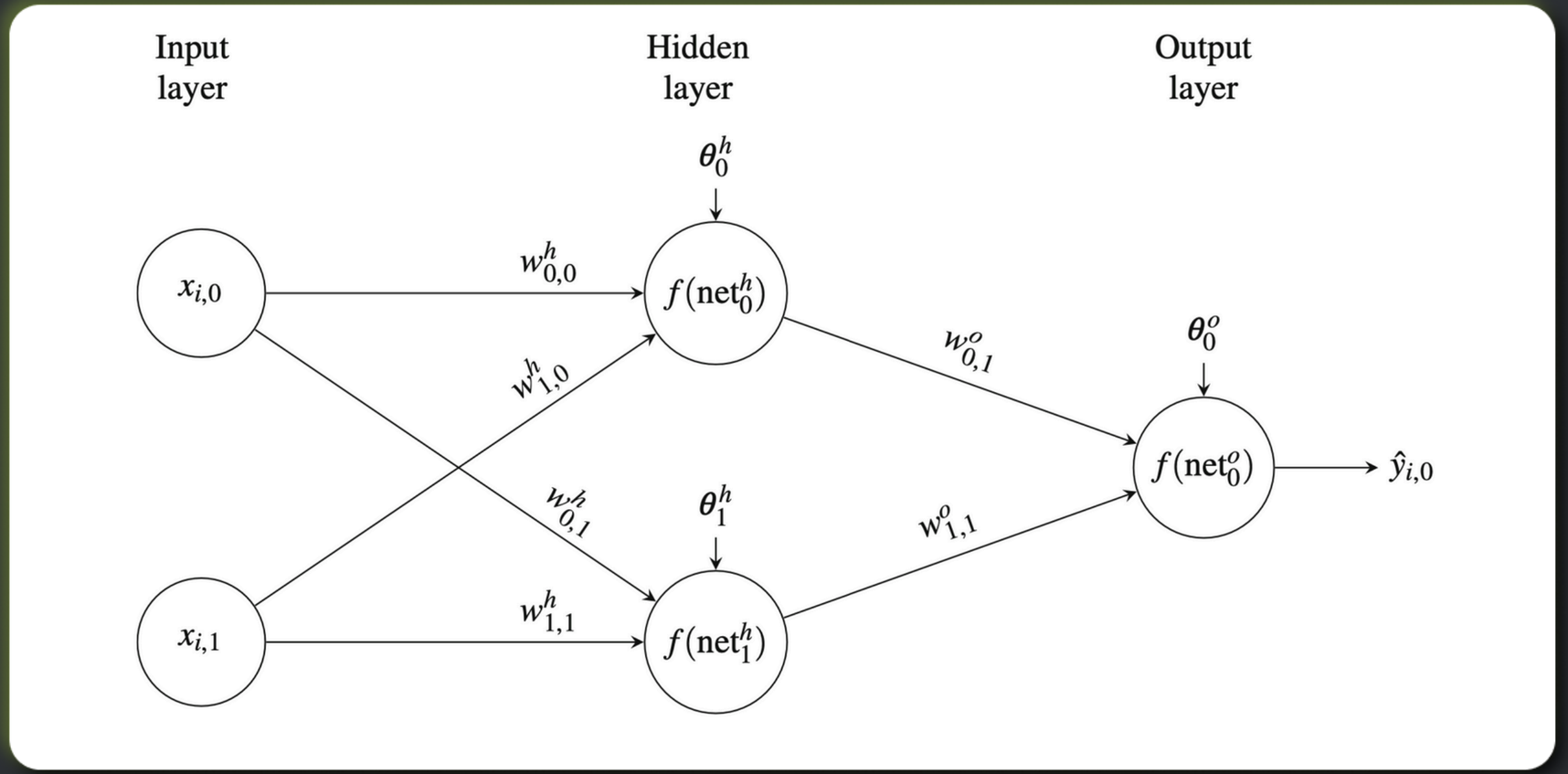
Multilayer Perceptron I



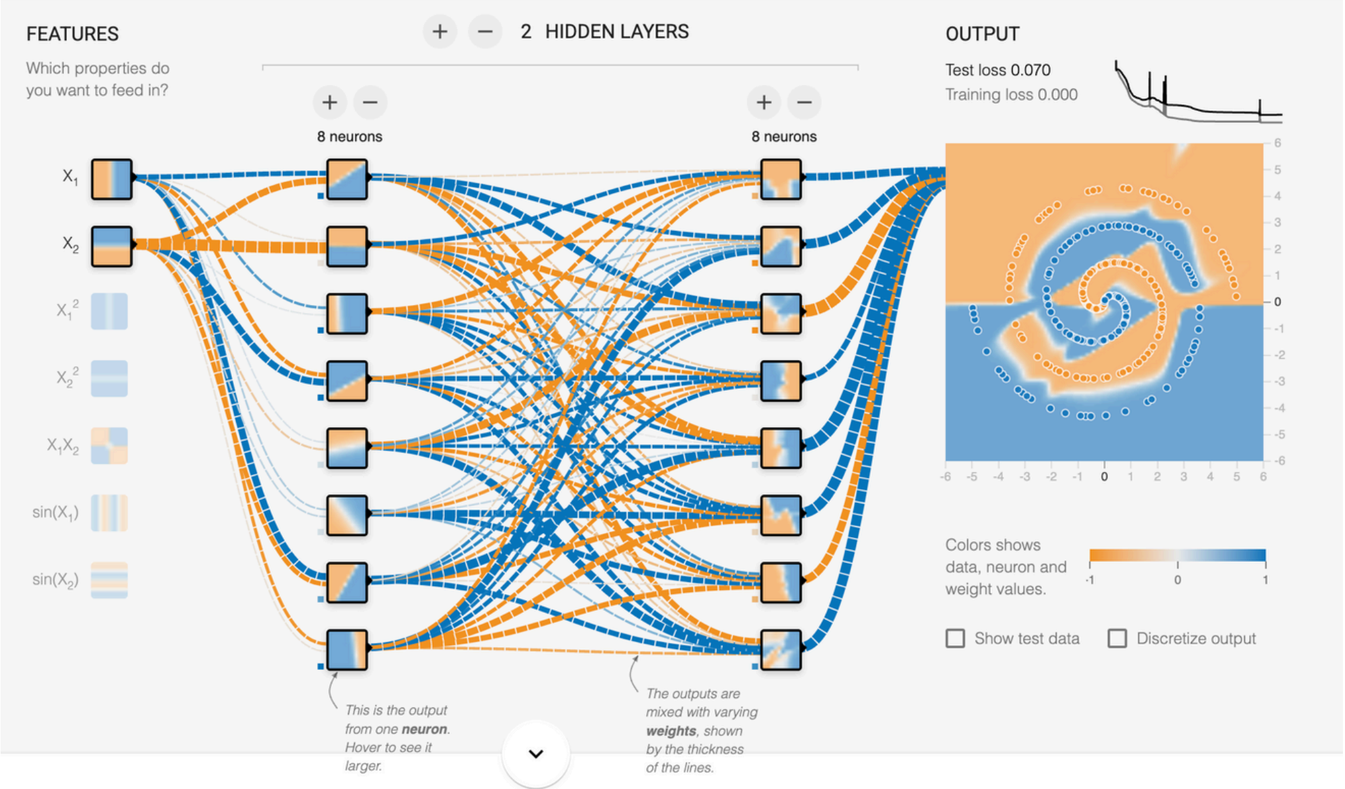
Multilayer Perceptron I



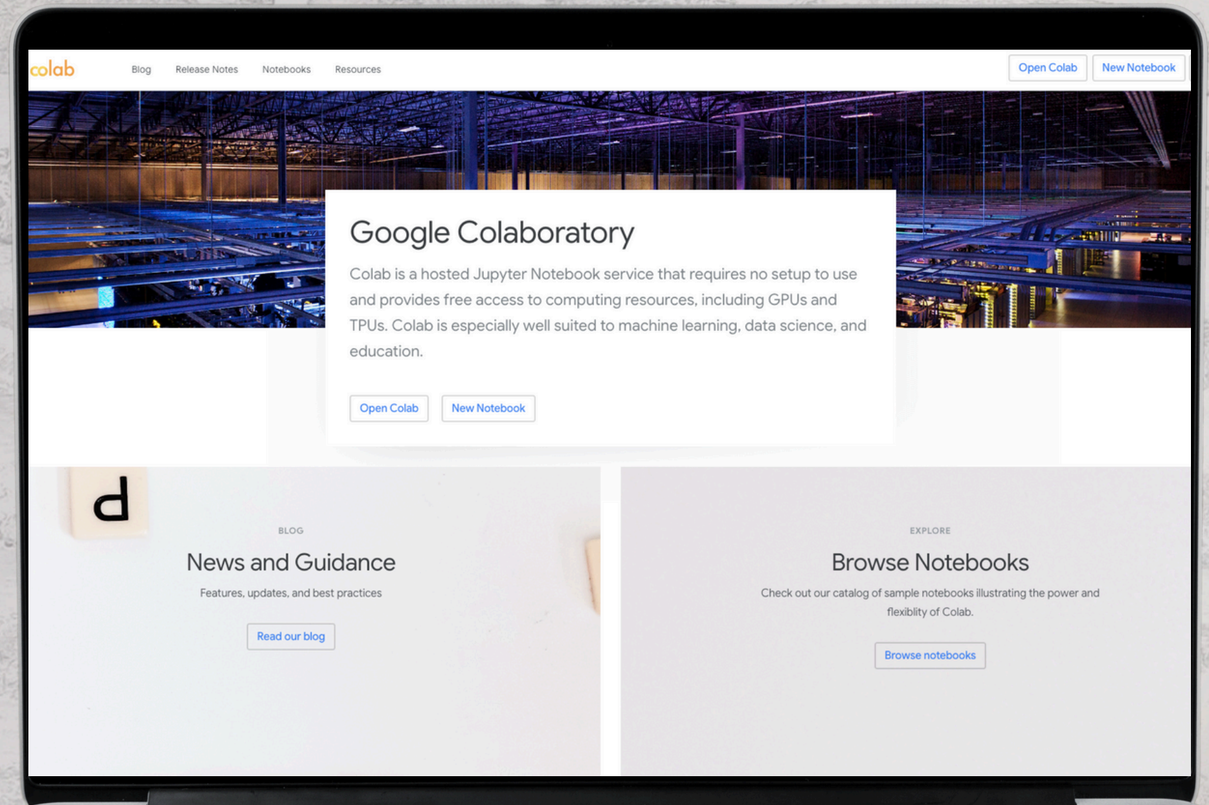
Multilayer Perceptron I



Multilayer Perceptron I



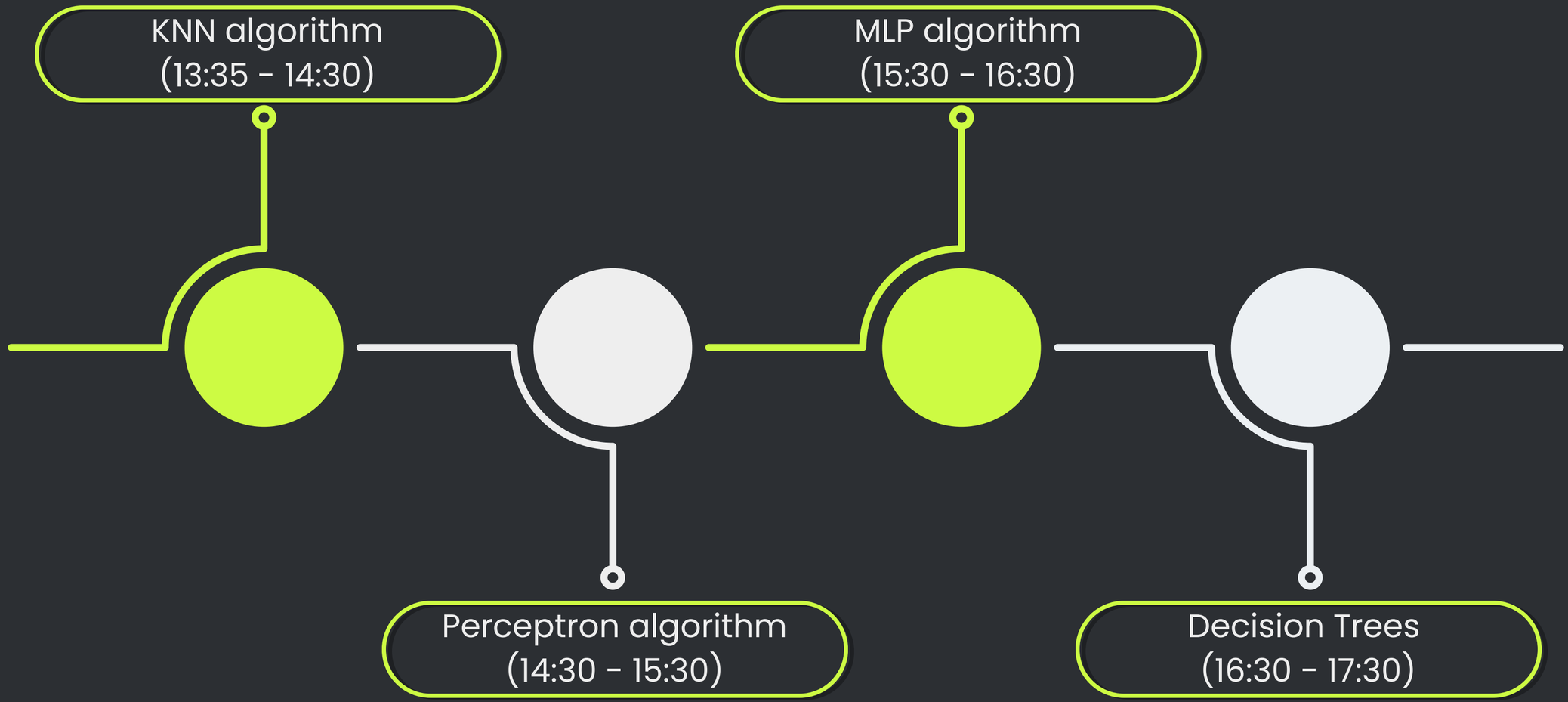
Multilayer Perceptron I



Machine Learning using Python (MLUP01)

stats

Day 4 (13:30 - 17:30)

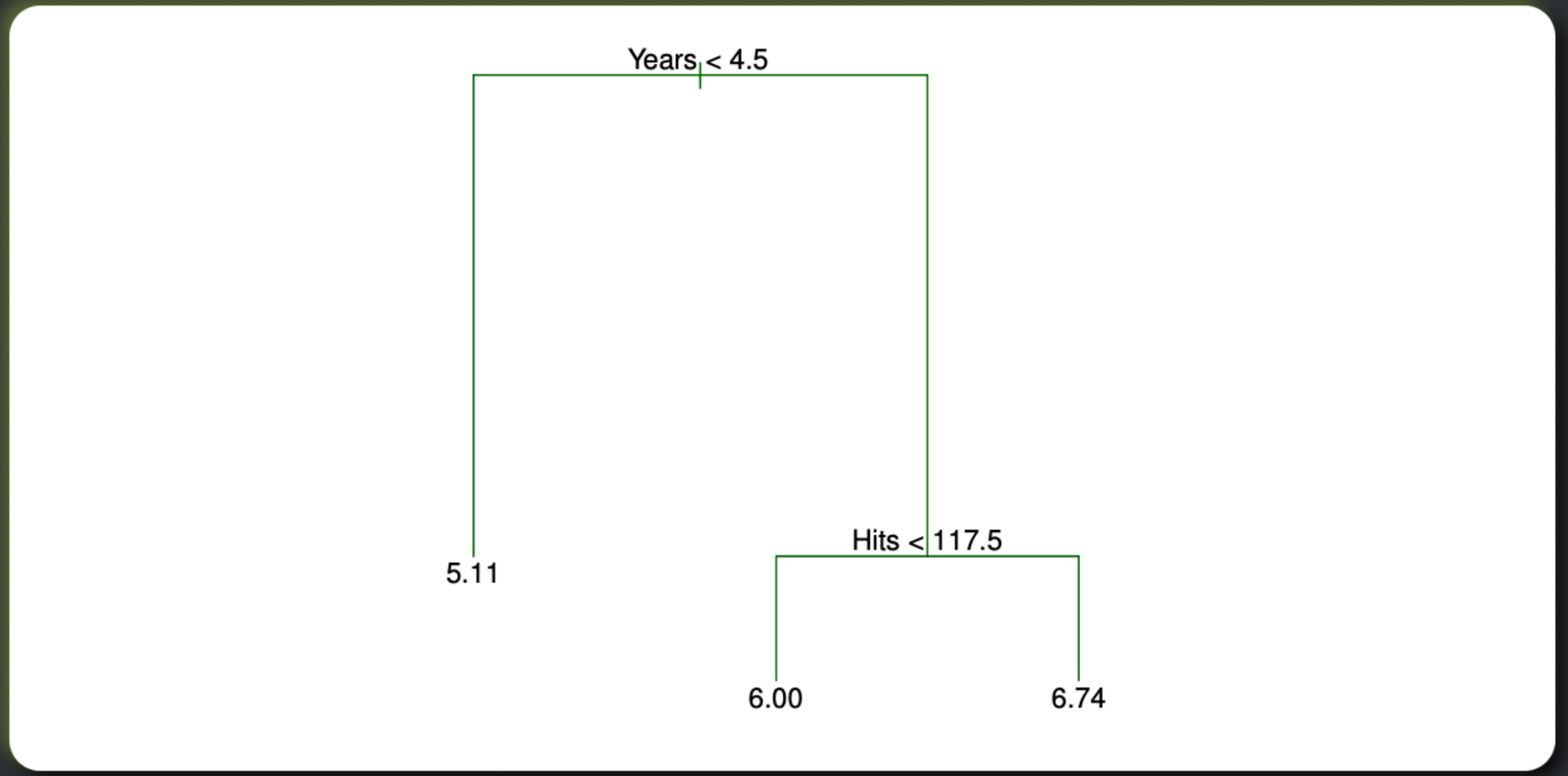


Decision Trees

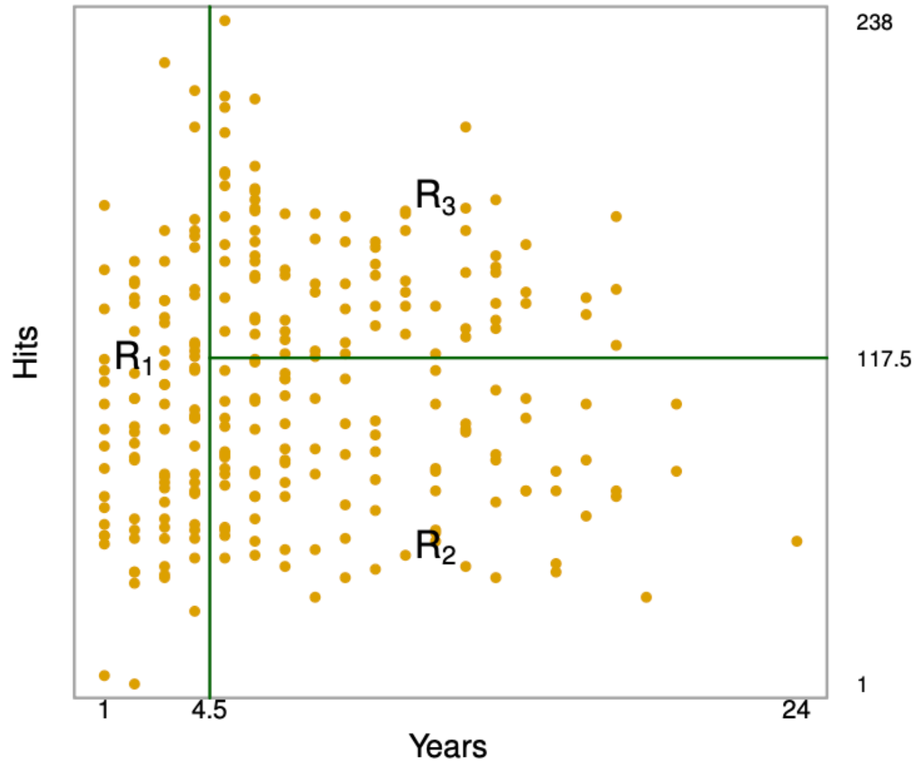
Machine Learning using Python (MLUP01)

stats

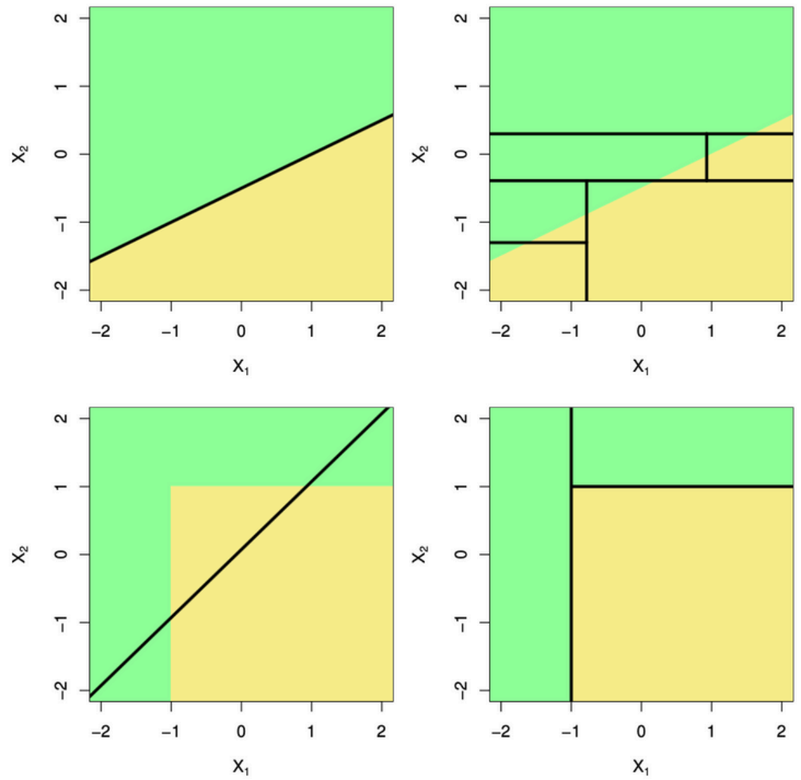
Decision Trees



Decision Trees



Decision Trees



Decision Trees

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.

2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.

3. Use K-fold cross-validation to choose α. That is, divide the training observations into K folds. For each $k = 1, \ldots, K$:

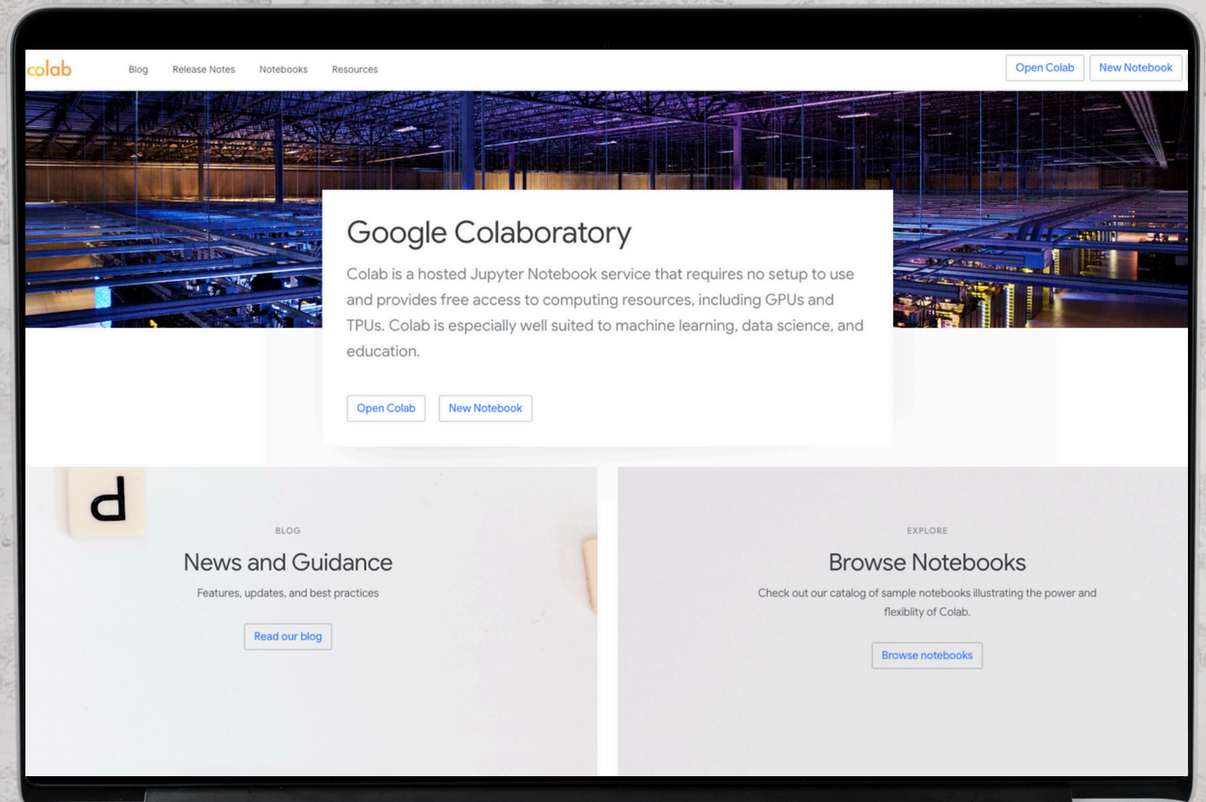
   (a) Repeat Steps 1 and 2 on all but the kth fold of the training data.

   (b) Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α.

   Average the results for each value of α, and pick α to minimize the average error.

4. Return the subtree from Step 2 that corresponds to the chosen value of α.

Decision Trees



Machine Learning using Python (MLUP01)

stats