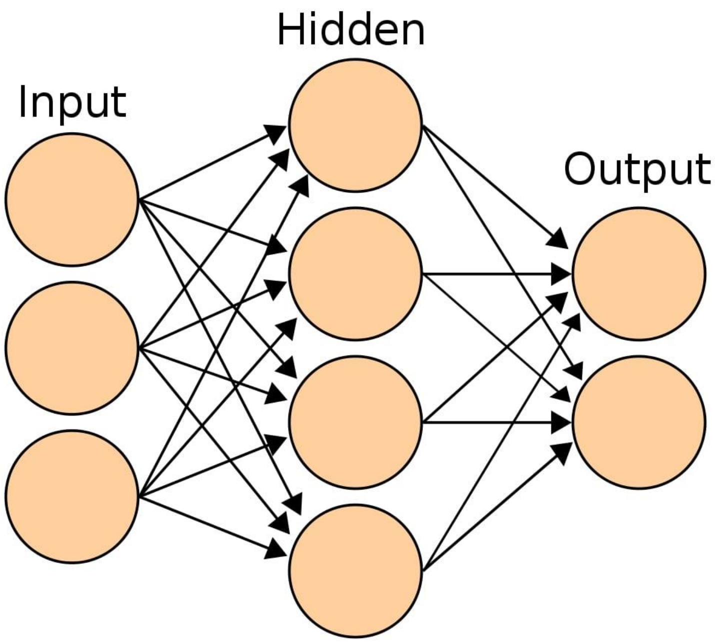
Fouille de Données

Data Mining

Classification - Partie 4

La classification avec les RNA

- **Série TP 5 – MLP Neural Nets for classification with Scikit Learn**



https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

La classification avec les RNA

- **Série TP 5 – MLP Neural Nets with Scikit Learn**

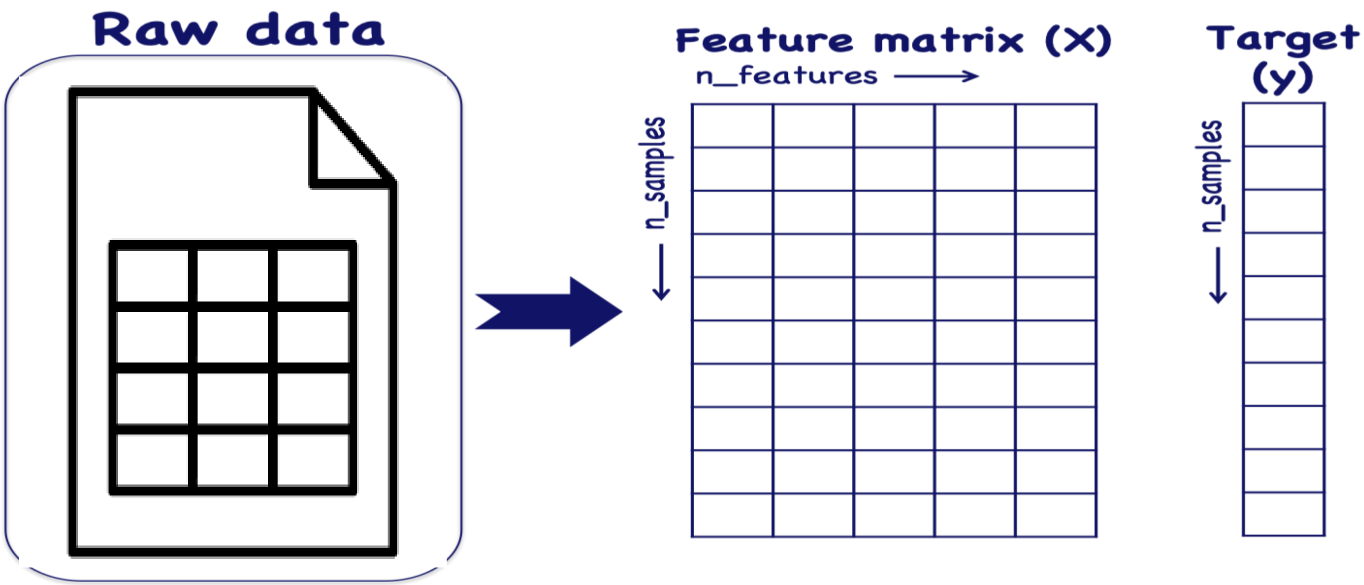
`BernoulliRBM`	Bernoulli Restricted Boltzmann Machine (RBM).
`MLPClassifier`	Multi-layer Perceptron classifier.
`MLPRegressor`	Multi-layer Perceptron regressor.

https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.html

La classification avec les RNA

- **sklearn.neural_network.MLPClassifier** implements Multi-layer Perceptron classifier algorithm.

- Takes as **input** two arrays: an **array X** of shape (n_samples, n_features) holding the **training samples**, and an **array Y** of integer values, shape (n_samples,), holding the **class labels** for the training samples.



La classification avec les RNA

- MLPClassifier - Multi-layer Perceptron classifier.
- This model optimizes the log-loss function using LBFGS or stochastic gradient descent.

MLPClassifier

```
class sklearn.neural_network.MLPClassifier(hidden_layer_sizes=(100,),
activation='relu', *, solver='adam', alpha=0.0001, batch_size='auto',
learning_rate='constant', learning_rate_init=0.001, power_t=0.5,
max_iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False,
warm_start=False, momentum=0.9, nesterovs_momentum=True,
early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999,
epsilon=1e-08, n_iter_no_change=10, max_fun=15000)
```

[source]

La classification avec les RNA

- **Main MLPClassifier Hyperparameters**

Hyperparameter	Type / Example	Meaning
`hidden_layer_sizes`	tuple, e.g. `(100,)` or `(50, 25)`	Defines the **number and size of hidden layers**. Each number = neurons per layer.
`activation`	str, e.g. `'relu'`, `'tanh'`, `'logistic'`, `'identity'`	The **activation function** applied to neurons in hidden layers.

La classification avec les RNA

- **Main MLPClassifier Hyperparameters**

Hyperparameter	Type / Example	Meaning
`solver`	str, e.g. `'adam'`, `'sgd'`, `'lbfgs'`	Optimization algorithm used to update weights.
`learning_rate_init`	float, e.g. `0.001`	Initial **step size** for weight updates.
`max_iter`	int, e.g. `300`	Maximum number of **training iterations (epochs)**.

La classification avec les RNA

- **MLPClassifier algorithm and solvers**

- All MLPClassifier solvers ultimately use **backpropagation**, but they differ in how they perform the optimization step (i.e., how they update the weights using gradients).

- Backpropagation — it's the algorithm for computing gradients of the **loss** with respect to the weights. Every training iteration in MLPClassifier uses backpropagation to calculate those gradients.

- The **solver** is the algorithm that uses those gradients (from backprop) to **update** the weights.

- MLPClassifier provides **three solvers**, each implementing a different optimization strategy.

- Backpropagation → compute weights gradients; Solver → update weights.

L’algorithme Backpropagation: Pseudo-code

Same for all optimizers / solvers

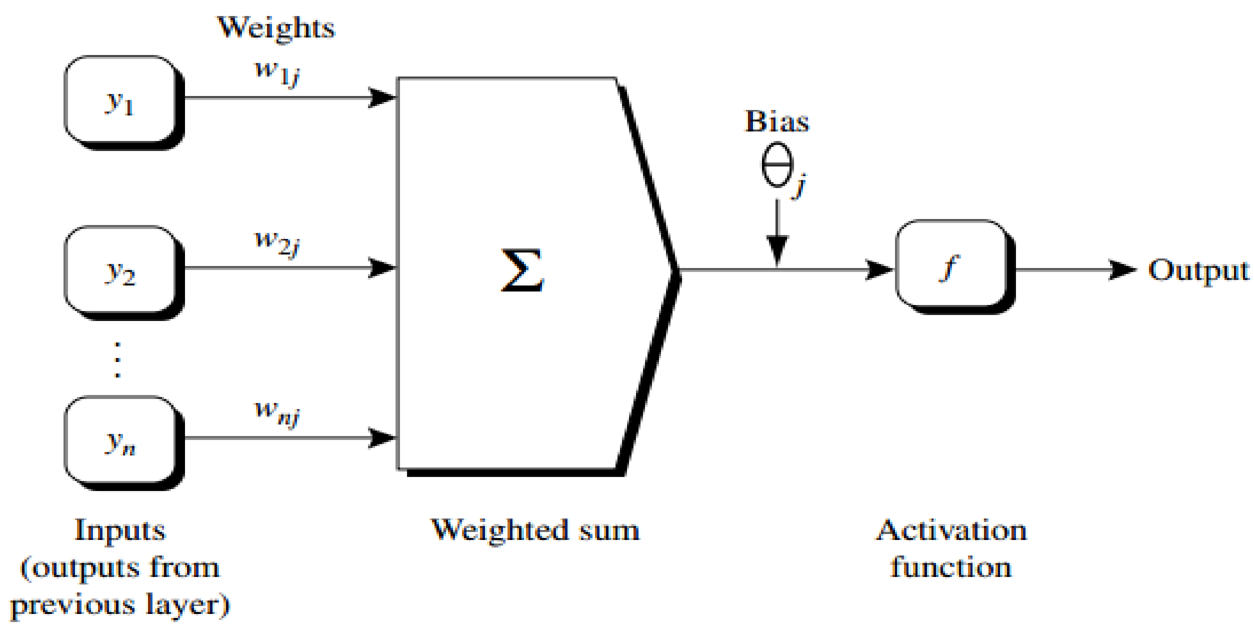
// Propagate the inputs forward:

for each hidden or output layer unit j {

$I_j = \sum_i w_{ij} O_i + \theta_j$; //compute the net input of unit j with respect to the previous layer, i

$O_j = \frac{1}{1+e^{-I_j}}$; } // compute the output of each unit j

Sigmoid activation function



L'algorithme Backpropagation: Pseudo-code

Same for all optimizers / solvers

// Backpropagate the errors:

for each unit j in the output layer

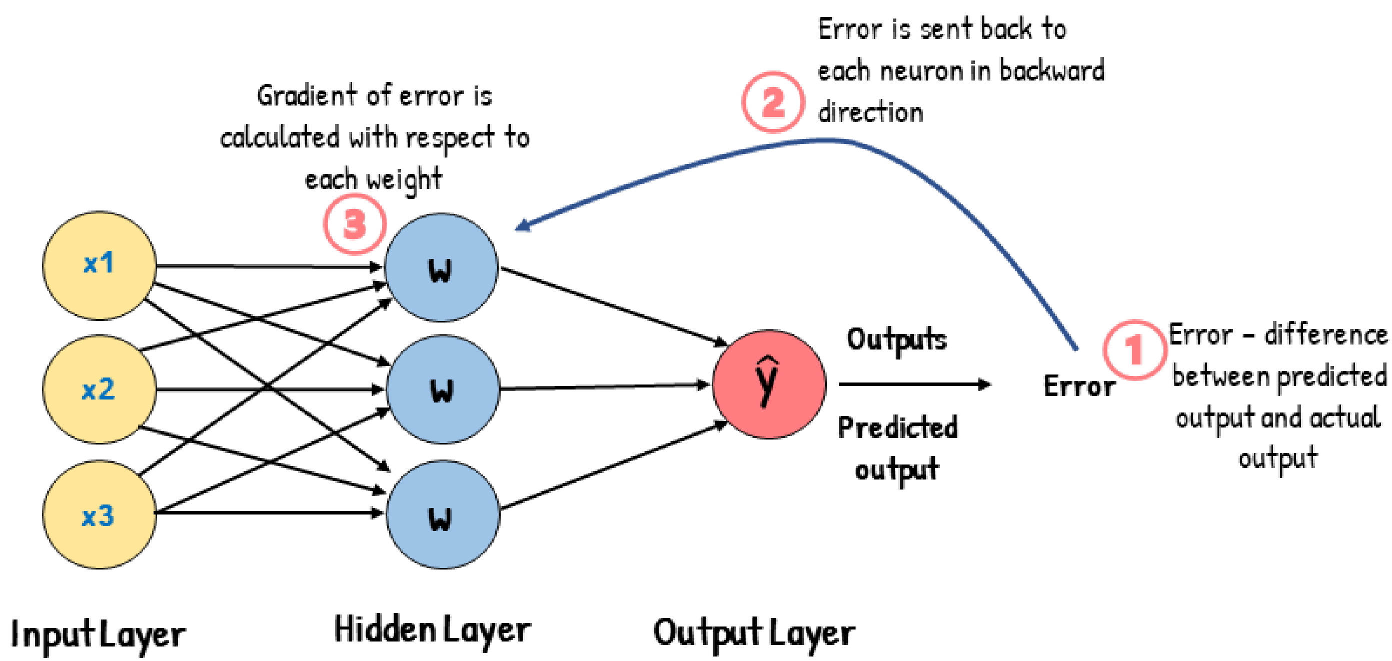
$Err_j = O_j(1 - O_j)(T_j - O_j)$; // compute the error

T: Target value

for each unit j in the hidden layers, from the last to the first hidden layer

$Err_j = O_j(1 - O_j)\sum_k Err_k w_{jk}$; // compute the error with respect to the next higher layer, k

Sigmoid activation function

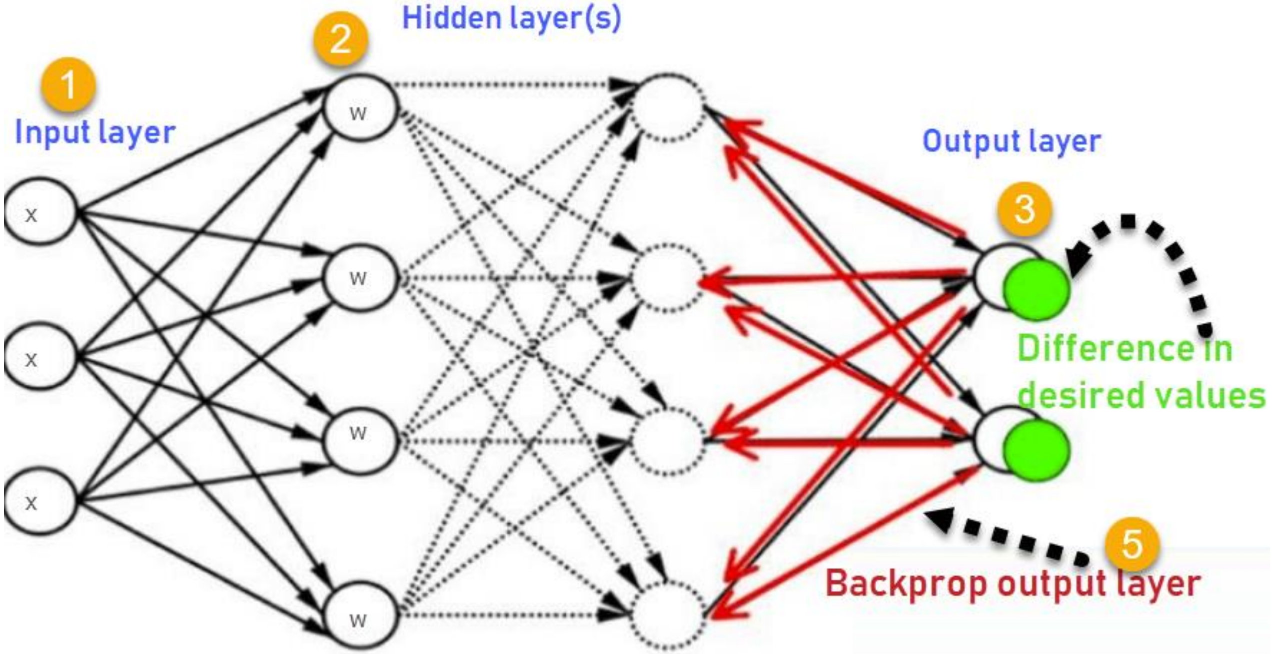


L'algorithme Backpropagation: Pseudo-code

Different for optimizers / solvers

for each weight w_{ij} in *network* {
$\Delta w_{ij} = (l) Err_j O_i$; // weight increment
$w_{ij} = w_{ij} + \Delta w_{ij}$; } // weight update
for each bias θ_j in *network* {
$\Delta \theta_j = (l) Err_j$; // bias increment
$\theta_j = \theta_j + \Delta \theta_j$; } // bias update
} }

Sigmoid activation function



La classification avec les RNA

1. Import necessary modules
2. Load & explore the dataset : iris dataset
3. Split the DataFrame into features (X) and target/class (y)
4. Create training and test sets
5. Scaling and normalizing the data features
6. Train the model
7. Predict and Evaluate : Accuracy & Confusion matrix

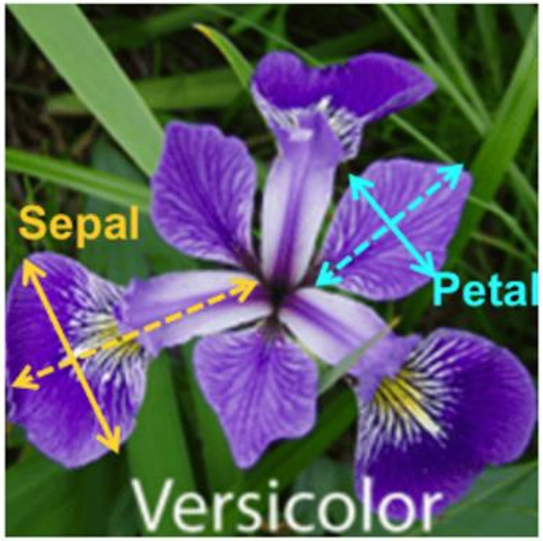
La classification avec les RNA

Iris Dataset - Multi-Class Classification using MLPClassifier

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

from sklearn.neural_network import MLPClassifier

from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

Loading and exploring dataset

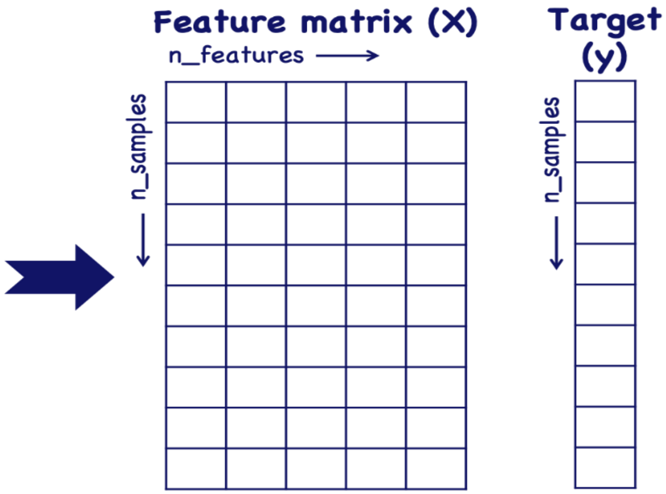
```
df = pd.read_csv('Datasets/iris.csv')
```



sepal.length	sepal.width	petal.length	petal.width	variety
4.6	3.2	1.4	0.2	Setosa
5.2	4.1	1.5	0.1	Setosa
6.9	3.1	4.9	1.5	Versicolor
5.5	2.5	4.0	1.3	Versicolor
5.5	4.2	1.4	0.2	Setosa
4.6	3.1	1.5	0.2	Setosa
6.1	2.6	5.6	1.4	Virginica
5.6	3.0	4.5	1.5	Versicolor
6.4	2.8	5.6	2.2	Virginica
5.7	2.6	3.5	1.0	Versicolor

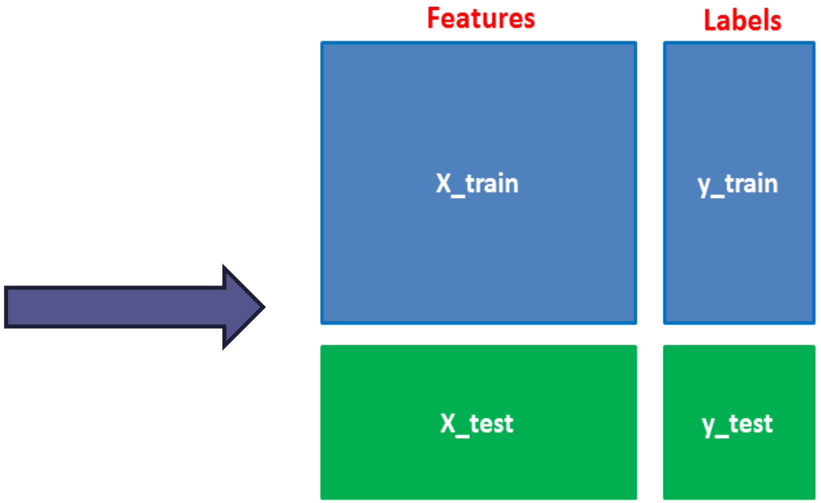
La classification avec les RNA

sepal.length	sepal.width	petal.length	petal.width	variety
5.1	3.5	1.4	0.2	Setosa
4.9	3.0	1.4	0.2	Setosa
4.7	3.2	1.3	0.2	Setosa
4.6	3.1	1.5	0.2	Setosa
5.0	3.6	1.4	0.2	Setosa



La classification avec les RNA

sepal.length	sepal.width	petal.length	petal.width	variety
5.1	3.5	1.4	0.2	Setosa
4.9	3.0	1.4	0.2	Setosa
4.7	3.2	1.3	0.2	Setosa
4.6	3.1	1.5	0.2	Setosa
5.0	3.6	1.4	0.2	Setosa



La classification avec les RNA

- **Split the DataFrame into features (X) and target/class (y)**
- **Create training and test sets**

```
X = df[['sepal.length', 'sepal.width', 'petal.length', 'petal.width']]
y = df['variety']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
X_train.shape # 4 inputs/cols
```

```
(120, 4)
```

```
X_test.shape
```

```
(30, 4)
```

La classification avec les RNA

- **Feature Scaling - Data Scaling**

Ex : Problem

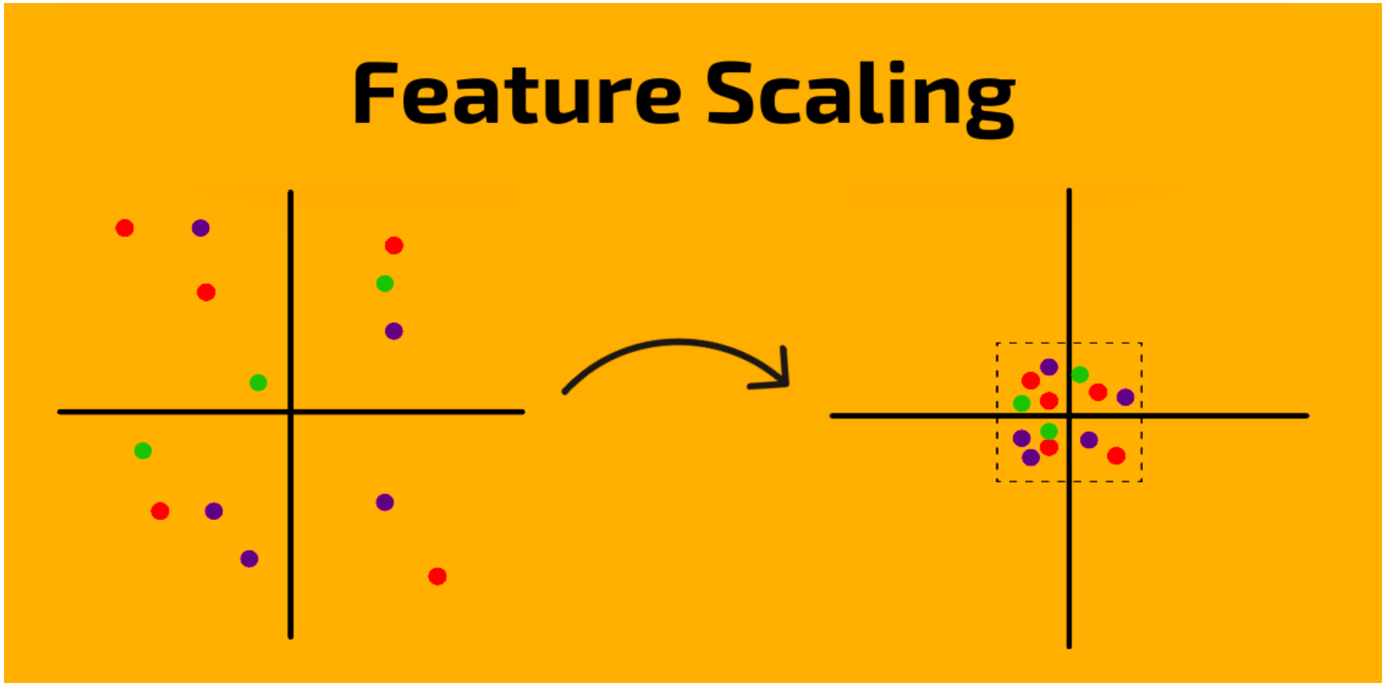
```
df.head()
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

La classification avec les RNA

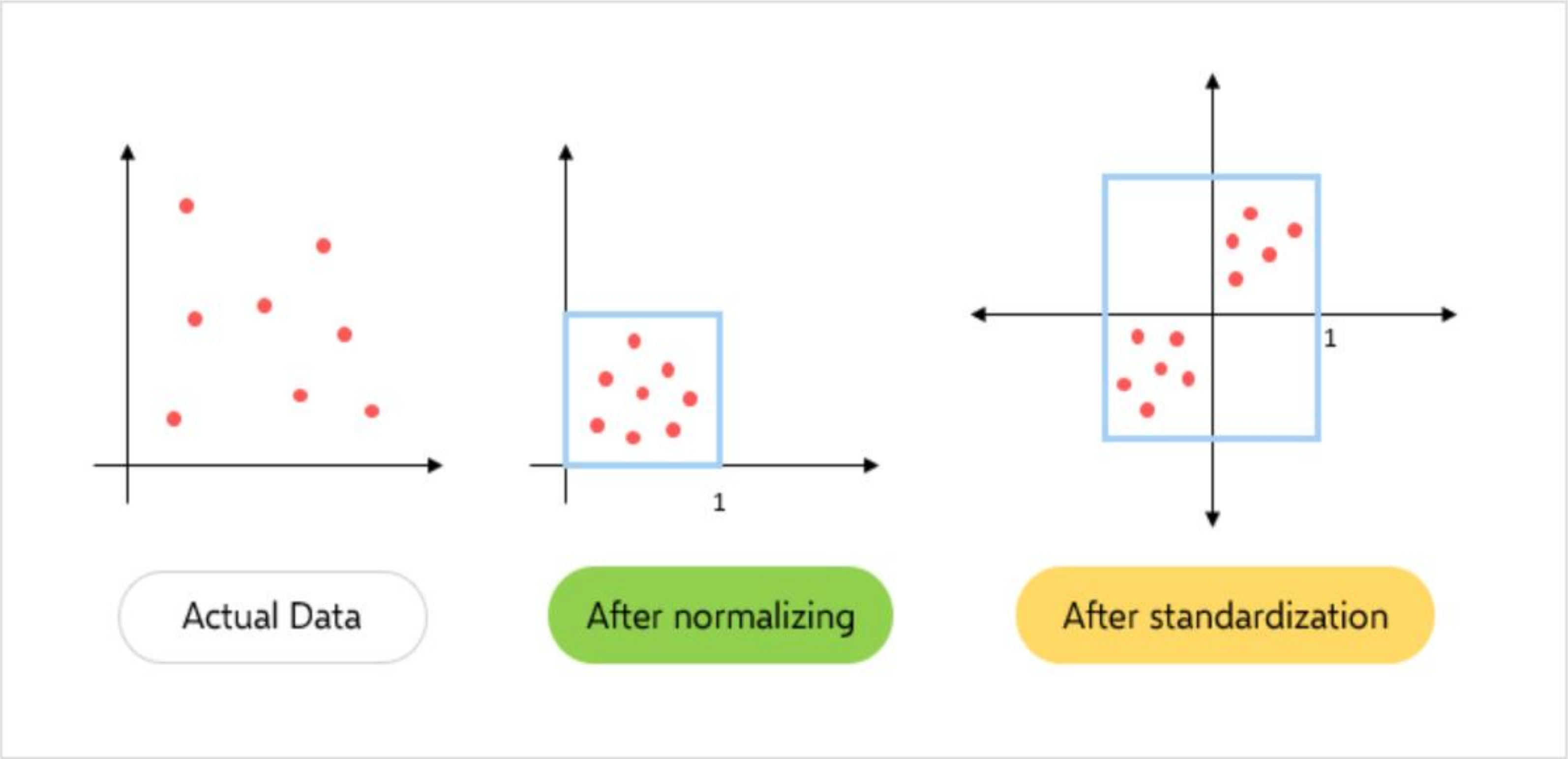
- **Feature scaling** is a crucial step in the feature **transformation** process that ensures all features are on a **similar scale**.

- It is the process that **normalizes the range** of input columns and makes it useful for further visualization and machine learning model training.



La classification avec les RNA

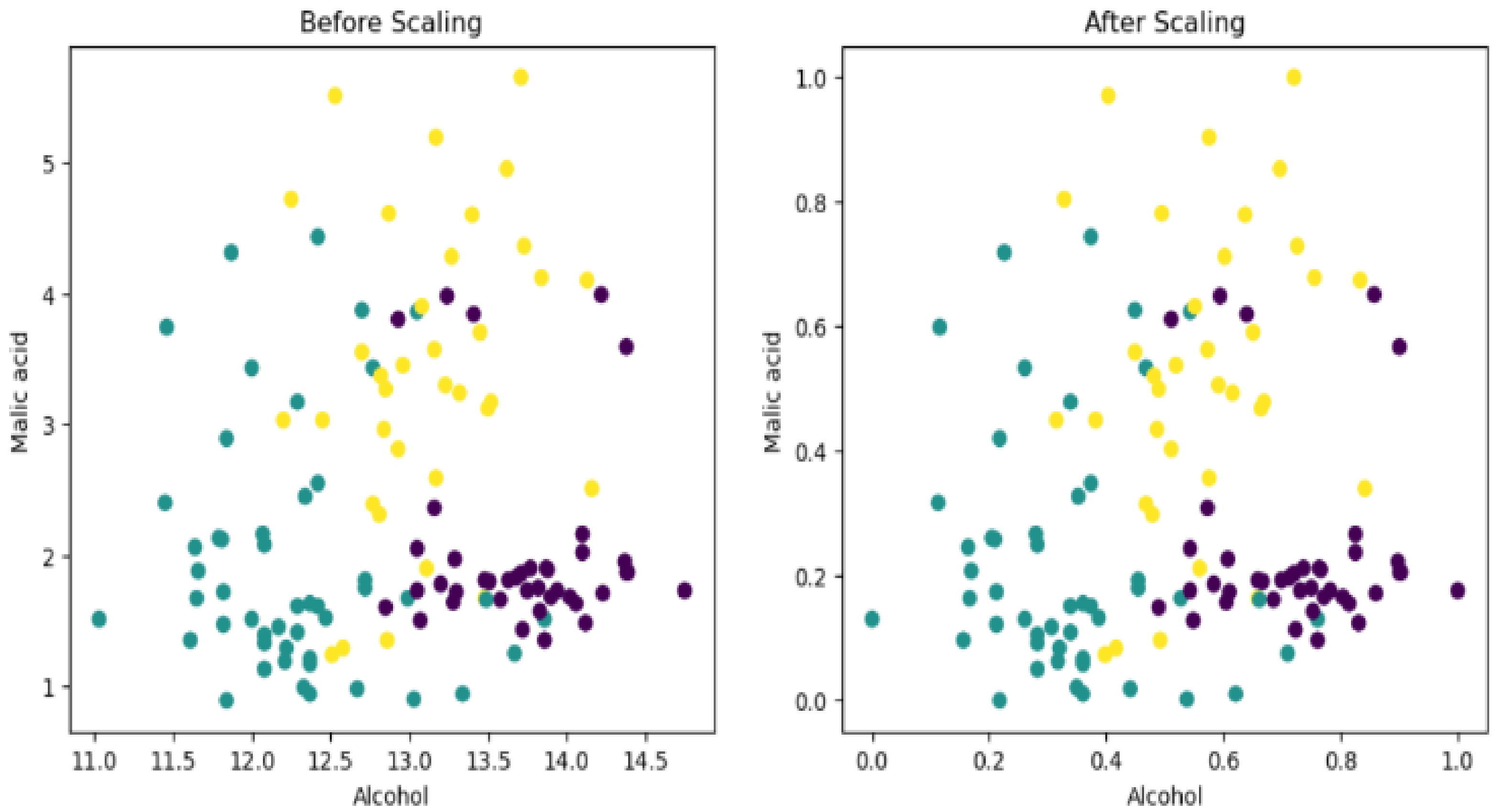
- **Feature Scaling**



A visual representation of feature scaling techniques – Source: someka.net

La classification avec les RNA

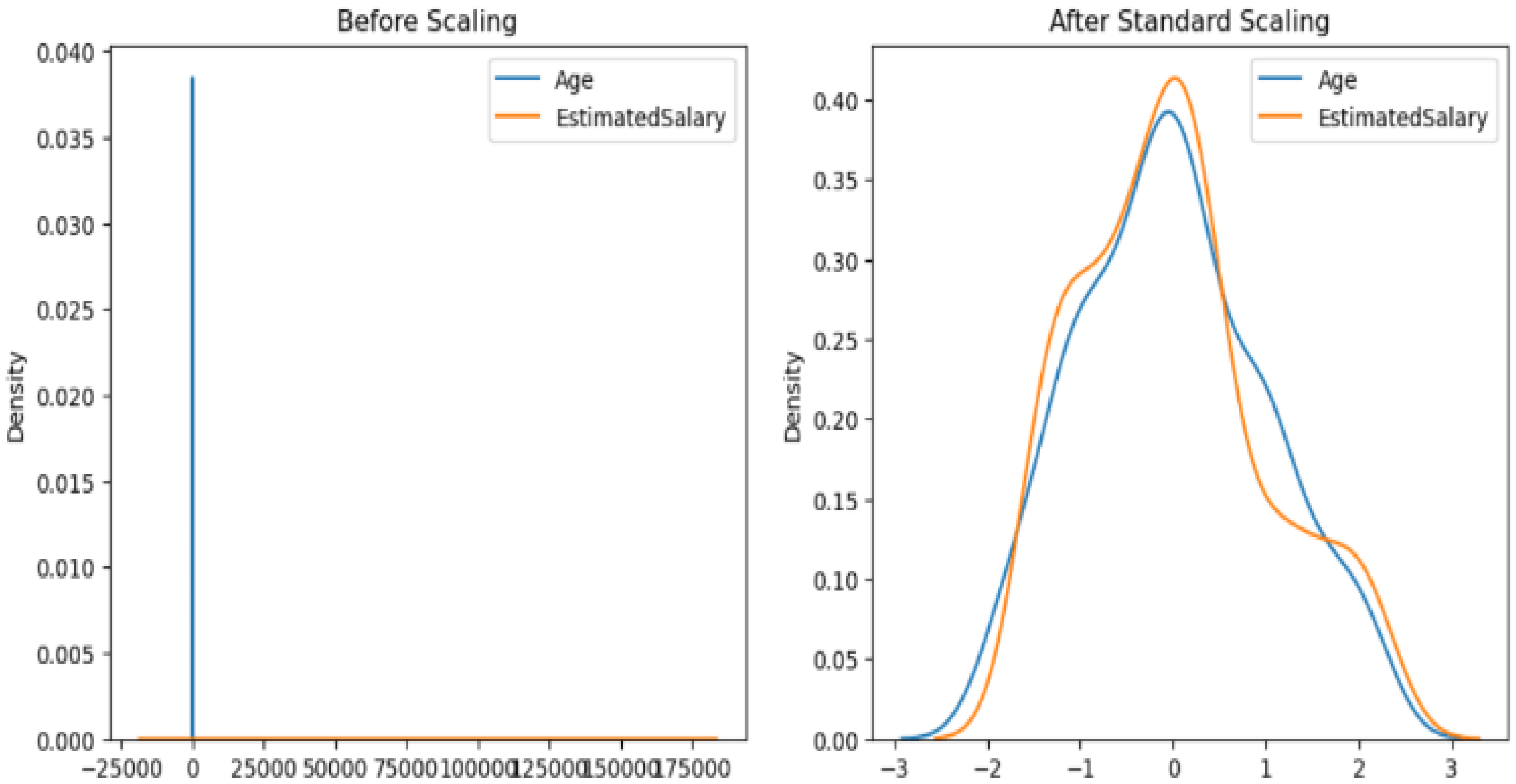
- **Feature Scaling**



https://datasciencedojo.com/blog/feature-scaling/

La classification avec les RNA

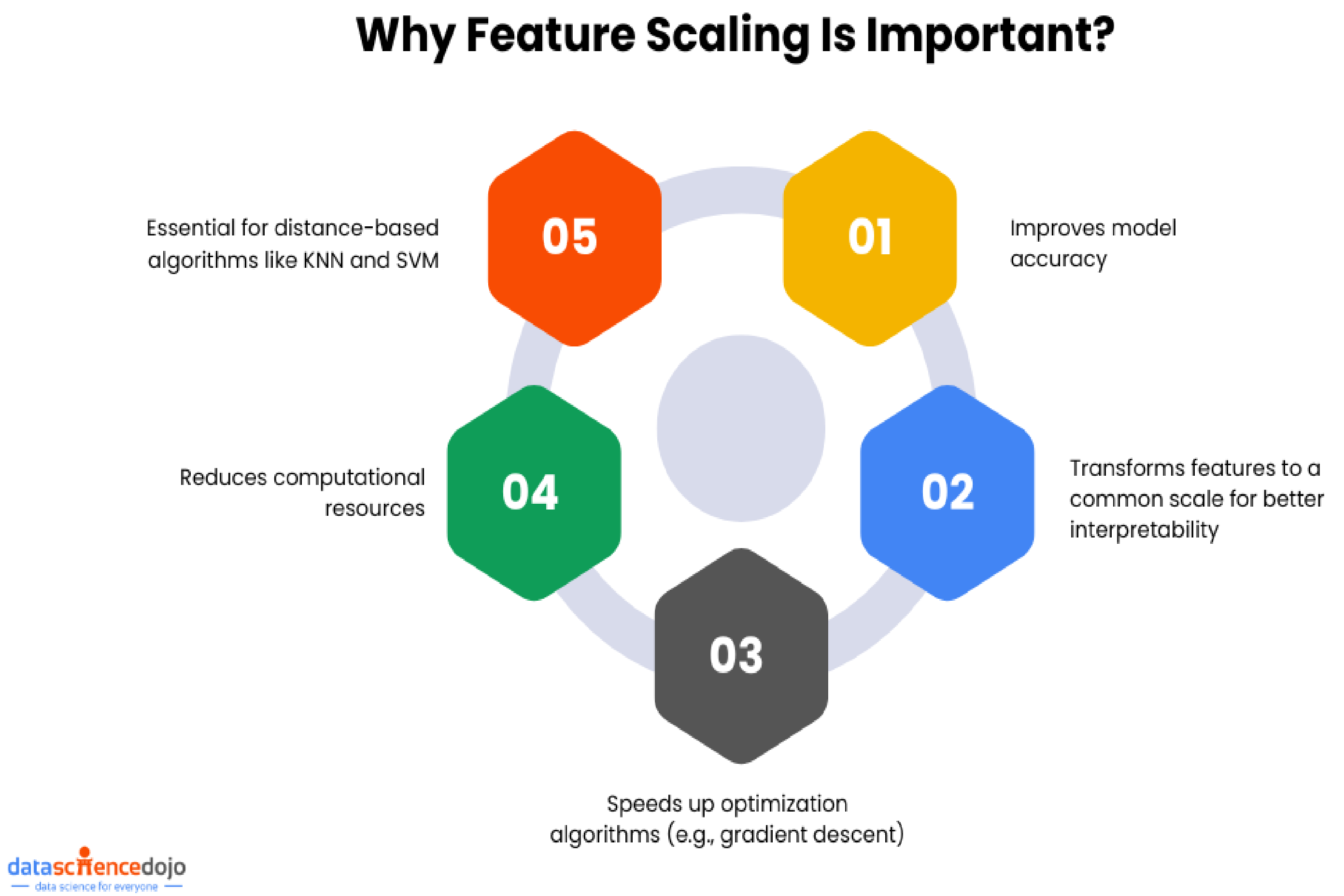
- **Feature Scaling**



https://datasciencedojo.com/blog/feature-scaling/

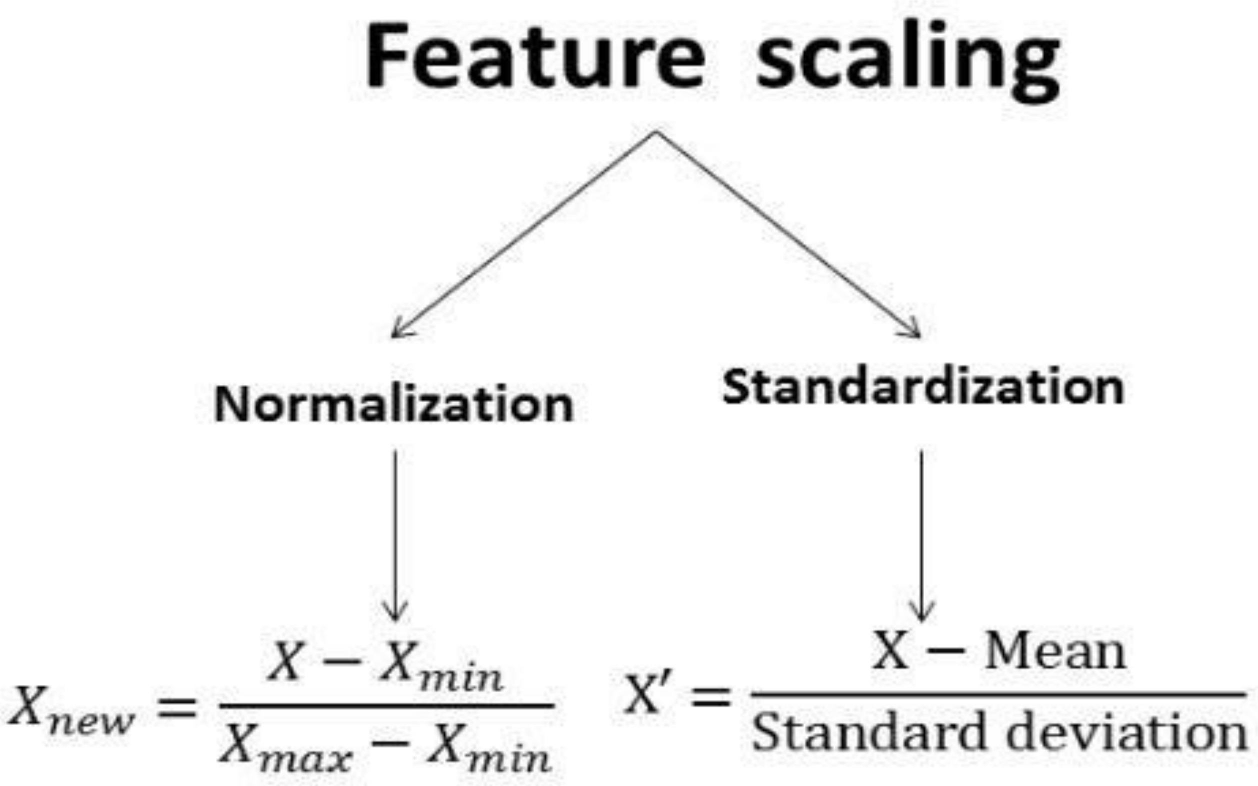
La classification avec les RNA

- **Feature Scaling - Data Scaling**



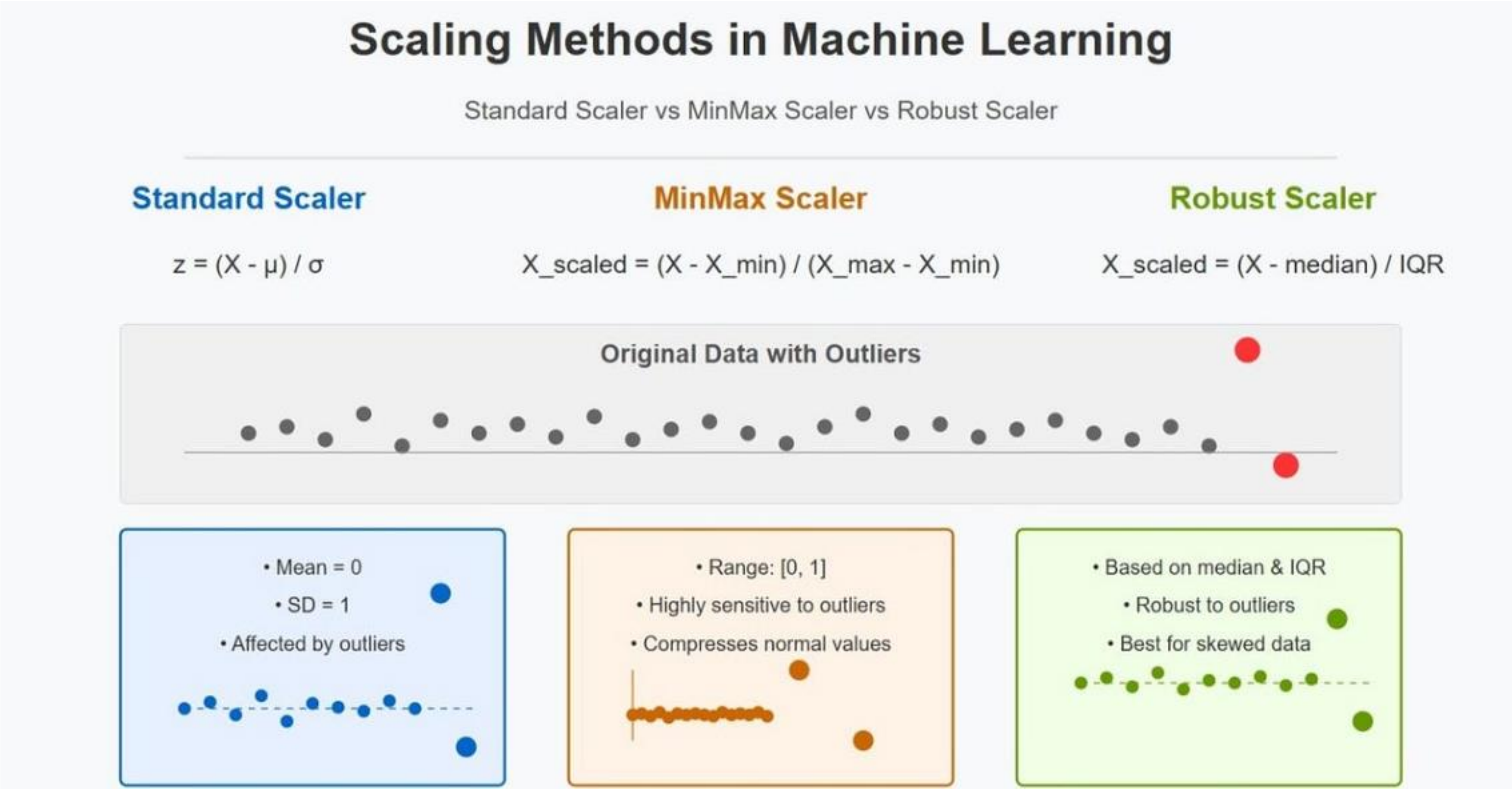
La classification avec les RNA

- **Feature Scaling - Strategies**



La classification avec les RNA

- **Feature Scaling - Strategies in Scikit Learn**



https://ashutoshtripathi.com/2021/06/12/what-is-feature-scaling-in-machine-learning-normalization-vs-standardization/

MinMaxScaler - Normalization

$$x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

#	Emp	Age	Salary
1	Emp1	44	73000
2	Emp2	27	47000
3	Emp3	30	53000
4	Emp4	38	62000
5	Emp5	40	57000
6	Emp6	35	53000
7	Emp7	48	78000

Normalization →

Age	Normalized Age	Salary	Normalized Salary
44	0.80952381	73000	0.838709677
27	0	47000	0
30	0.142857143	53000	0.193548387
38	0.523809524	62000	0.483870968
40	0.619047619	57000	0.322580645
35	0.380952381	53000	0.193548387
48	1	78000	1

Range 0-1 **Range 0-1**

How to calculate Normalized value?
X = 35, min = 27, max = 48 for column Age.
Xnorm(for 35) = $\frac{35-27}{48-27}$ = 0.3809

https://ashutoshtripathi.com/2021/06/12/what-is-feature-scaling-in-machine-learning-normalization-vs-standardization/

StandardScaler - Standardization

$$x_{\text{stand}} = \frac{x - \text{mean}(x)}{\text{standard deviation }(x)}$$

#	Emp	Age	Salary
1	Emp1	44	73000
2	Emp2	27	47000
3	Emp3	30	53000
4	Emp4	38	62000
5	Emp5	40	57000
6	Emp6	35	53000
7	Emp7	48	78000
		Mean = 37.42857 Std. Dev. = 6.883876	Mean = 60428.5714 Std. Dev = 10499.7570

Standardization

How to calculate Standardized value?
X = 35, mean = 37.42, Std. Dev. = 6.88 for column Age.
$X_{std}(\text{for } 35) = \frac{35-37.42}{6.88} = -0.3527$

Age	Standardized Age	Salary	Standardized Salary
44	0.954611636	73000	1.197306616
27	-1.514927162	47000	-1.278941158
30	-1.079126198	53000	-0.707499364
38	0.083009708	62000	0.149663327
40	0.373543684	57000	-0.326538168
35	-0.352791257	53000	-0.707499364
48	1.535679589	78000	1.673508111
	Mean = 0 **Std. dev. = 1**		**Mean = 0** **Std. dev. = 1**

La classification avec les RNA

- **Scaling Iris Data with Standard Scaler**

```
scaler = StandardScaler()

scaler.fit(X_train)

X_train_scaled = scaler.transform(X_train)

X_test_scaled = scaler.transform(X_test)
```

La classification avec les RNA

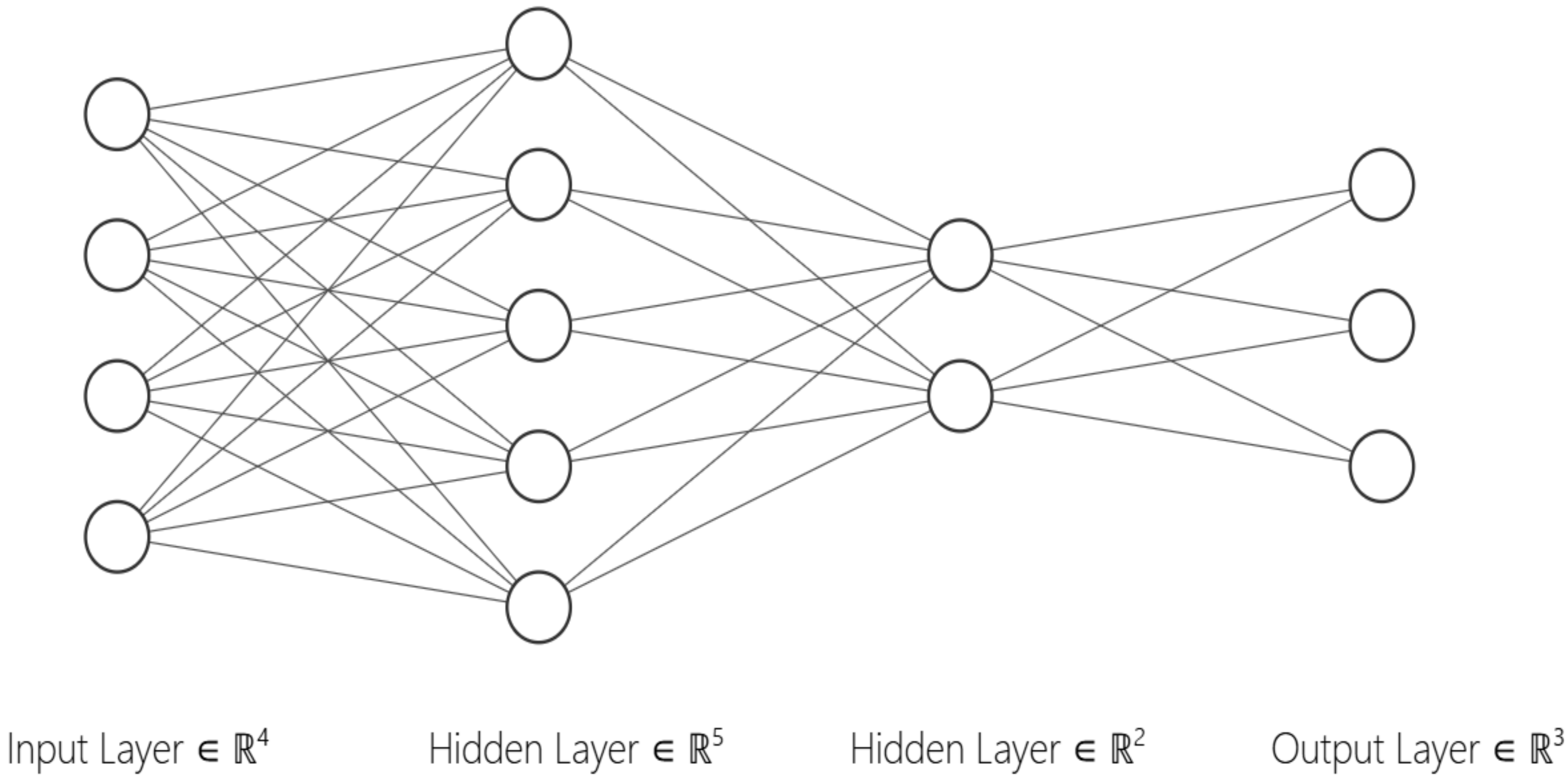
- **Training the MLP Classifier model**

```
clf = MLPClassifier(hidden_layer_sizes=(5, 2),
                    max_iter=150,
                    activation='logistic',
                    learning_rate_init=0.9)
```

```
clf.fit(X_train_scaled, y_train)
```

La classification avec les RNA

X_test.shape[1] **hidden_layer_sizes=(5, 2)** **clf.classes_**



clf.n_layers_ = 4

La classification avec les RNA

```
print("Number of layers:", clf.n_layers_)     #
```

```
Number of layers: 4
```

```
print("Number of inputs:", clf.n_features_in_)
```

```
4
```

```
print("Classes:", clf.classes_)
```

```
Classes: ['Setosa' 'Versicolor' 'Virginica']
```

```
print("Number of outputs:", clf.n_outputs_)
```

```
Number of outputs: 3
```

```
print("Hidden layers:", clf.hidden_layer_sizes)
```

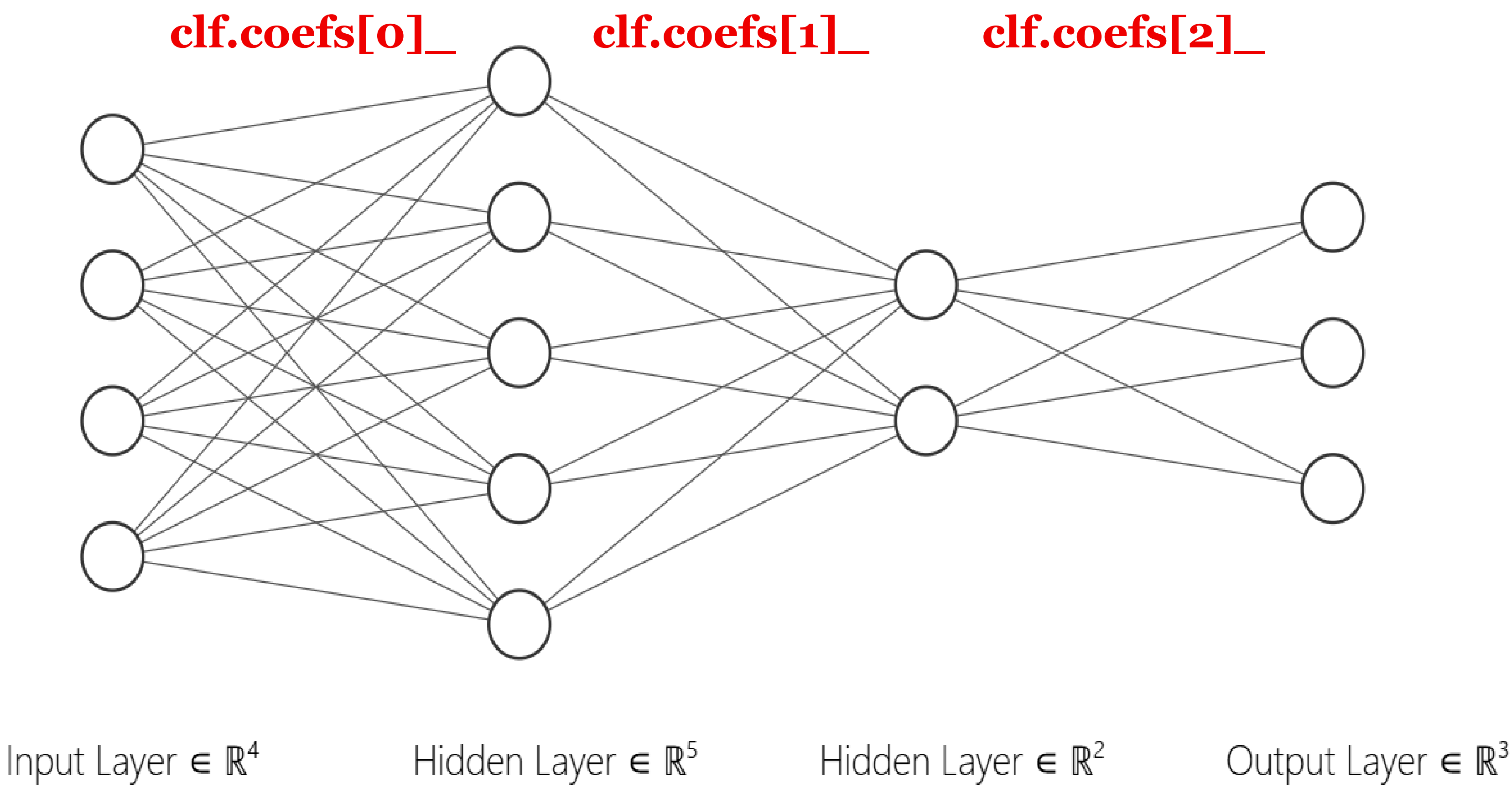
```
Hidden layers: (5, 2)
```

La classification avec les RNA

- **Printing weight values :** **clf.coefs_**

```
[array([[ 2.27934704, -4.27321999, -7.09221858, -6.72277455,  6.96607938],
        [-8.85643978,  7.4643435 ,  5.17784932,  8.54310614, -8.39953589],
        [ 6.78154801, -7.32894111, -7.77308237, -7.19026856,  8.05688015],
        [ 6.21802813, -7.18201174, -7.25137833, -7.44634327,  6.67623885]]),
 array([[-8.95873131, -7.5211549 ],
        [-5.058671  , 10.68755371],
        [-6.42668118,  4.25513765],
        [-6.40927091,  7.31061166],
        [-7.45151246, -3.54864958]]),
 array([[ 3.82125636,  3.46903772, -3.18054965],
        [ 9.07275517, -7.3438102 , -5.30585358]])]
```

La classification avec les RNA



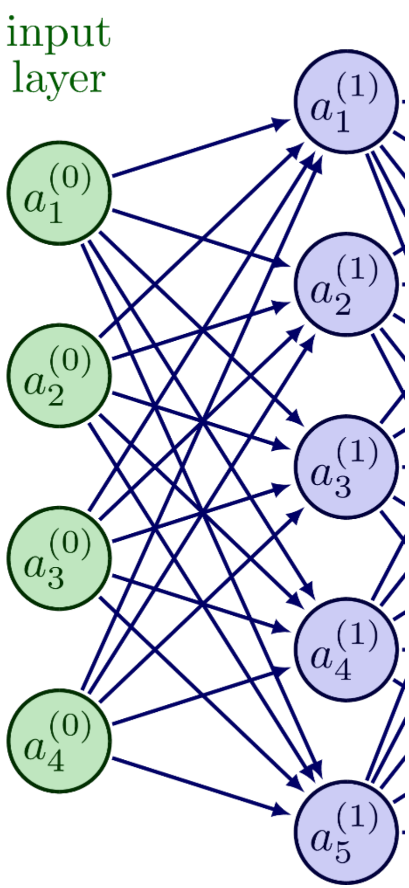
La classification avec les RNA

- **Printing weight values :** **clf.coefs[0]_**

```
[array([[ 2.27934704, -4.27321999, -7.09221858, -6.72277455,  6.96607938],
        [-8.85643978,  7.4643435 ,  5.17784932,  8.54310614, -8.39953589],
        [ 6.78154801, -7.32894111, -7.77308237, -7.19026856,  8.05688015],
        [ 6.21802813, -7.18201174, -7.25137833, -7.44634327,  6.67623885]]),
```

La classification avec les RNA

- **Printing weight values :** **clf.coefs[0]_**




```
[array([[ 2.27934704, -4.27321999, -7.09221858, -6.72277455,  6.96607938],
        [-8.85643978,  7.4643435 ,  5.17784932,  8.54310614, -8.39953589],
        [ 6.78154801, -7.32894111, -7.77308237, -7.19026856,  8.05688015],
        [ 6.21802813, -7.18201174, -7.25137833, -7.44634327,  6.67623885]]),
```

La classification avec les RNA

- **Printing bias values : clf.intercepts_**

```
clf.intercepts_   # Bias vectors per layer (hiddens & output)
```

```
[array([ 6.38271503, -9.5599952 , -5.06072358, -4.67089556,  1.39450127]),
 array([-8.64347932, -4.76406508]),
 array([-5.00661602,  2.34058155,  2.25816991])]
```

```
len(clf.intercepts_)
```

```
3
```

La classification avec les RNA

- **Model Evaluation**

```
y_preds = clf.predict(X_test_scaled)
```

```
clf.score(X_test_scaled, y_test)
```

```
0.6333333333333333
```

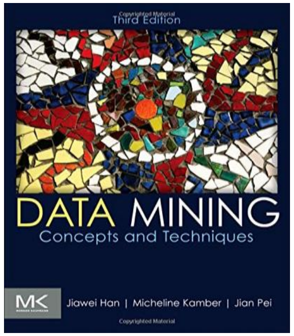
```
accuracy_score(y_test, y_preds)
```

```
0.6333333333333333
```

```
confusion_matrix(y_test, y_preds, labels=['Setosa', 'Versicolor', 'Virginica'])
```

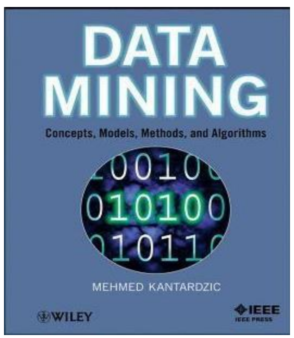
```
array([[10,  0,  0],
       [ 0,  9,  0],
       [ 0, 11,  0]], dtype=int64)
```

Ressources



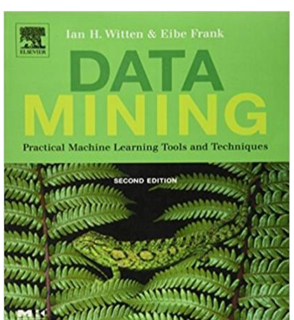
Data Mining : concepts and techniques, 3rd Edition

- ✓ Auteur : Jiawei Han, Micheline Kamber, Jian Pei
- ✓ Éditeur : Morgan Kaufmann Publishers
- ✓ Edition : Juin 2011 - 744 pages - ISBN 9780123814807



Data Mining : concepts, models, methods, and algorithms

- ✓ Auteur : Mehmed Kantardzi
- ✓ Éditeur : John Wiley & Sons
- ✓ Edition : Aout 2011 – 552 pages - ISBN : 9781118029121



Data Mining: Practical Machine Learning Tools and Techniques

- ✓ Auteur : Ian H. Witten & Eibe Frank
- ✓ Éditeur : Morgan Kaufmann Publishers
- ✓ Edition : Juin 2005 - 664 pages - ISBN : 0-12-088407-0