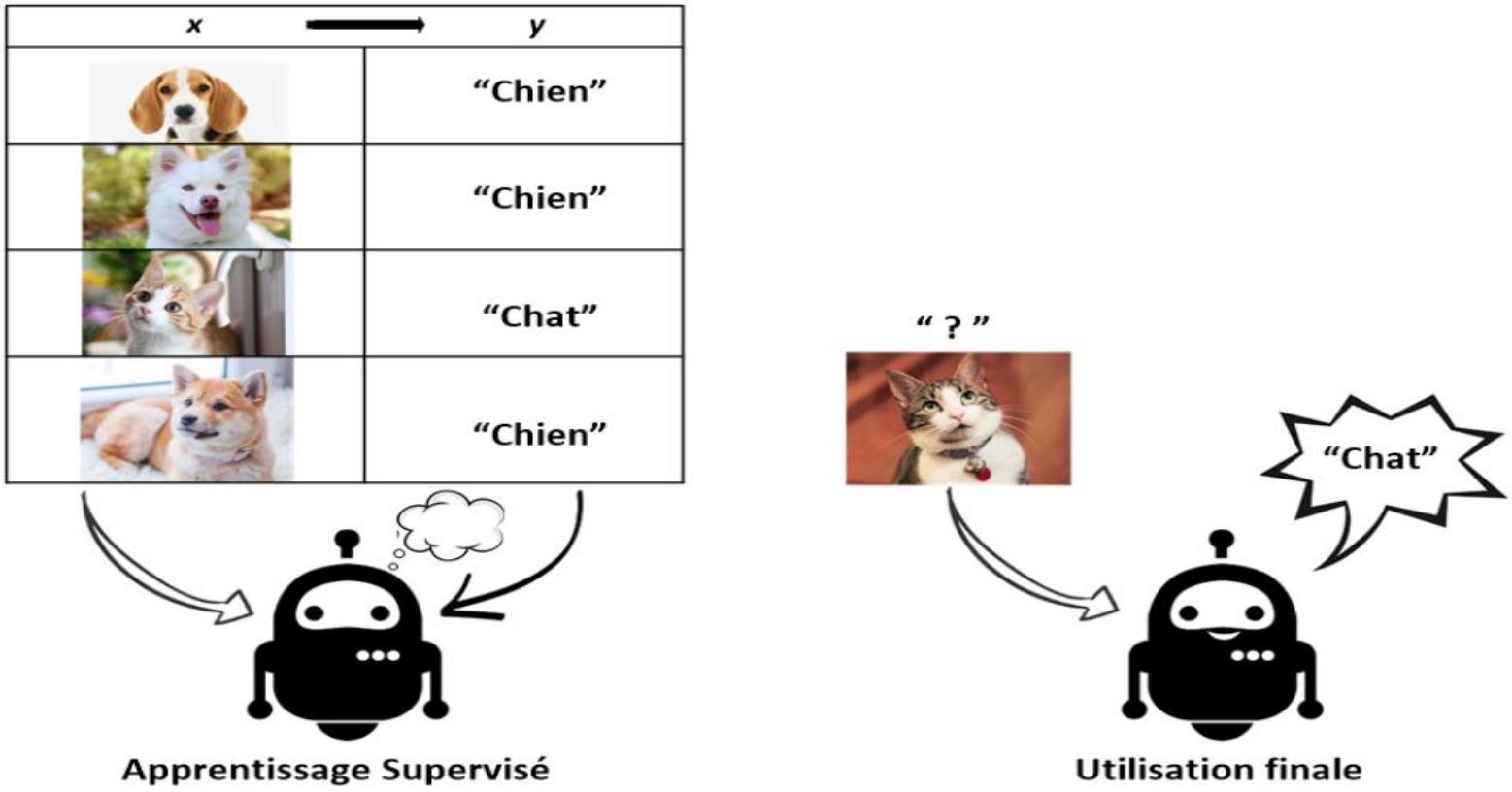
2022/2023

K-nearest neighbors kNN

Réalisé par: Naïma Daghfous

Apprentissage Supervisé

- Apprentissage automatique = apprendre un modèle formel à partir de données observées



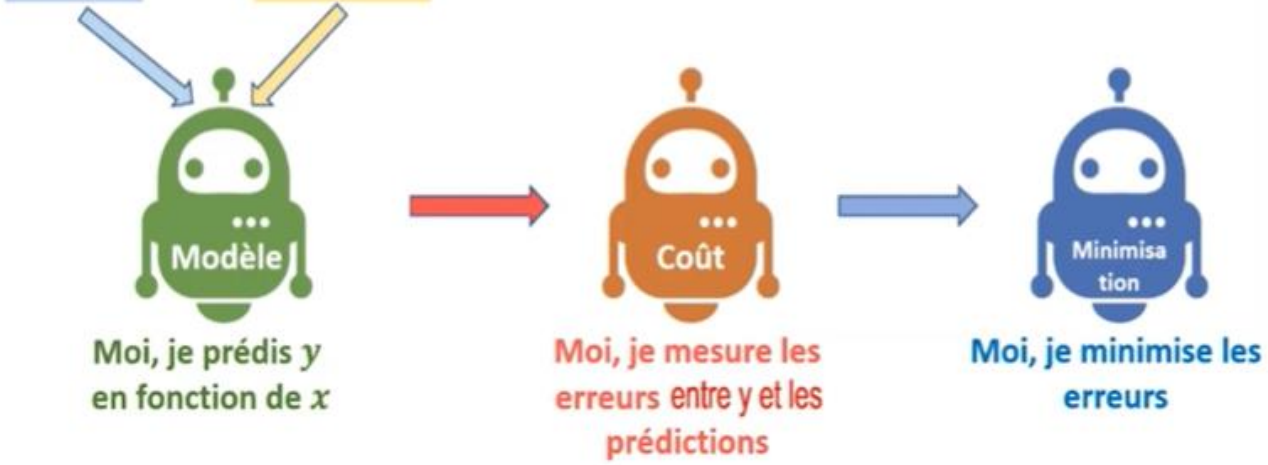
La machine apprend à partir de milliers d'exemples x, y

Rappel : Schéma de l'Apprentissage Supervisé

Dataset (x, y)

y	x_1	x_2	x_3	...	x_n
$y^{(1)}$	$x_1^{(1)}$	$x_2^{(1)}$	$x_3^{(1)}$	...	$x_n^{(1)}$
$y^{(2)}$	$x_1^{(2)}$	$x_2^{(2)}$	$x_3^{(2)}$	...	$x_n^{(2)}$
$y^{(3)}$	$x_1^{(3)}$	$x_2^{(3)}$	$x_3^{(3)}$	...	$x_n^{(3)}$
...	...	...	...	...	...
$y^{(m)}$	$x_1^{(m)}$	$x_2^{(m)}$	$x_3^{(m)}$	...	$x_n^{(m)}$

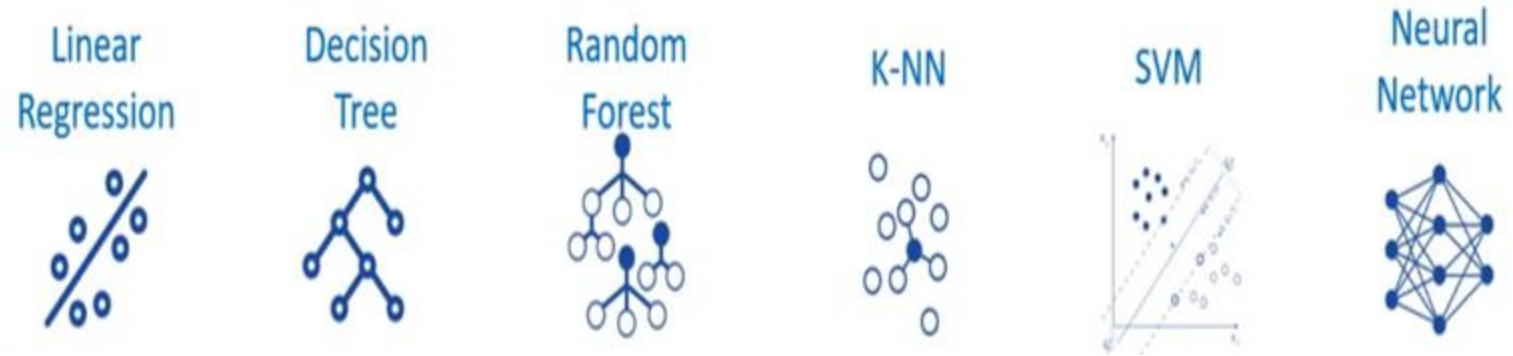
target *features*



1. Dataset
   y : Target
   x : features
2. Modèle
   paramètres
3. Fonction Coût
4. Algorithme de minimisation

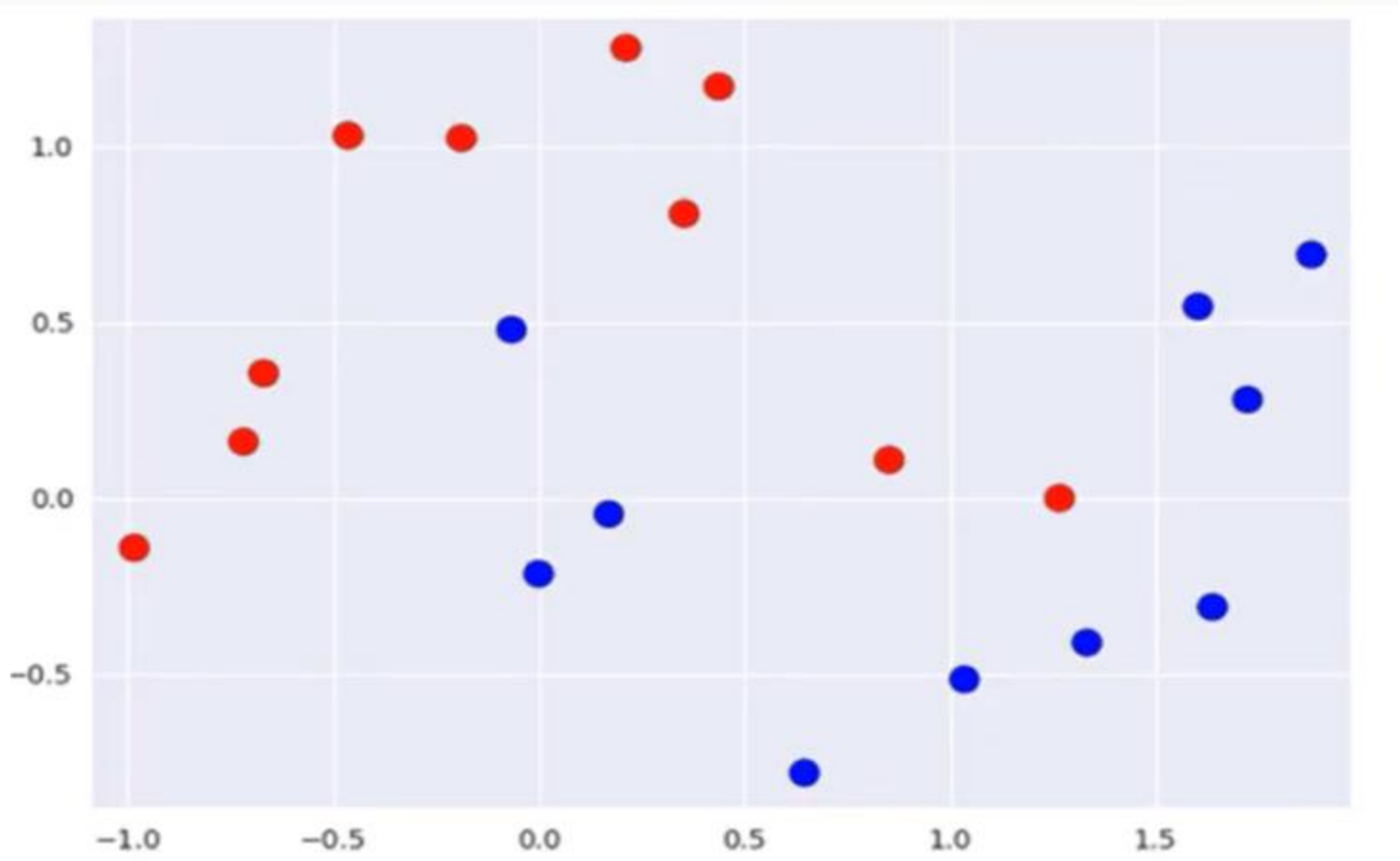
KNN

- Le Machine Learning consiste à créer un **modèle** à l'aide de données pour permettre à un ordinateur d'apprendre à effectuer une tâche spécifique.



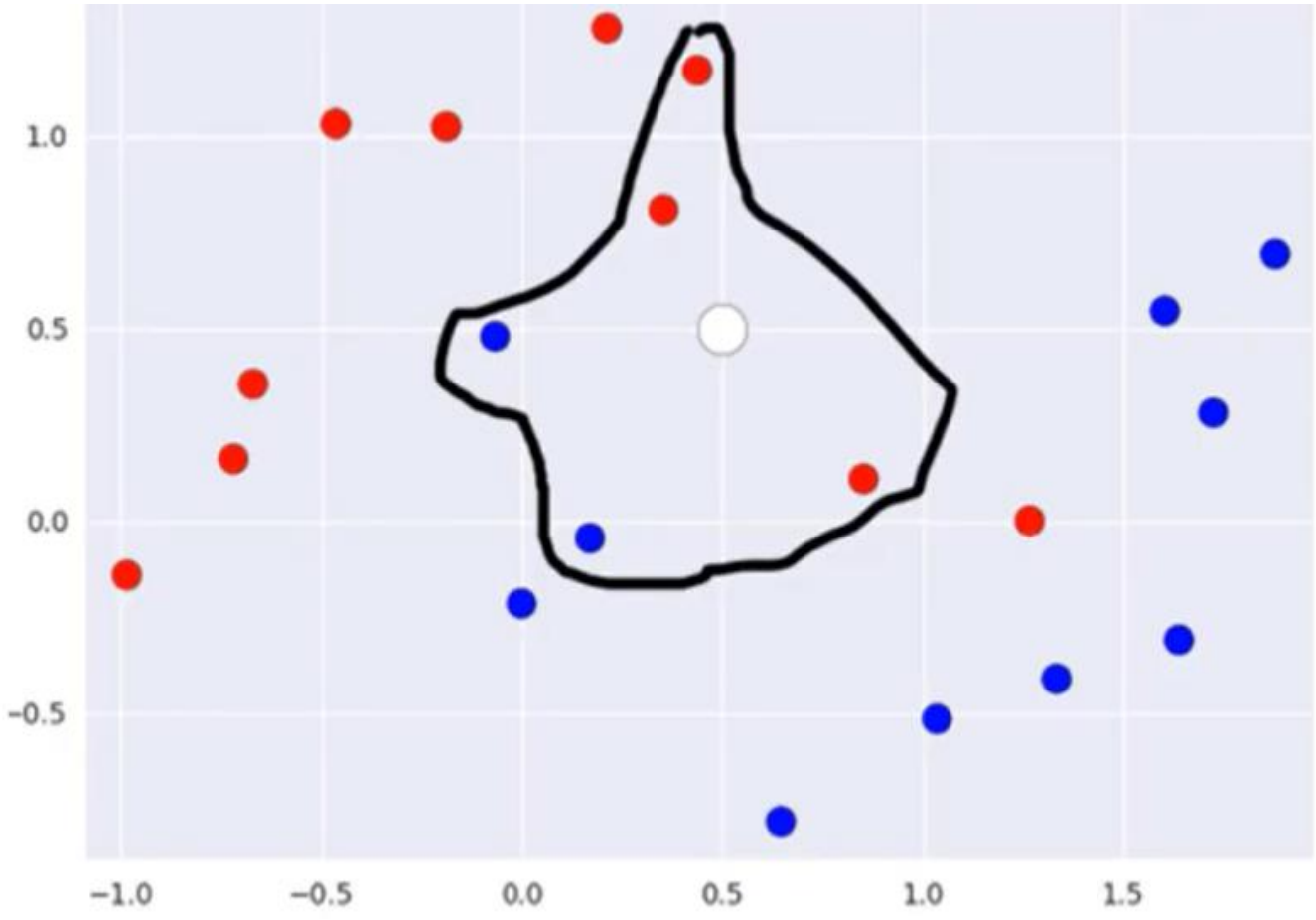
- L'algorithme des **K plus proches voisins ou K-nearest neighbors (kNN)** est un algorithme de Machine Learning qui appartient à la classe des algorithmes d'apprentissage supervisé simple et facile à mettre en œuvre qui peut être utilisé pour résoudre les problèmes de classification et de régression.

Data Set



KNN

K=5
5-NN



KNN: Principe

-1-
Sélectionner le nombre K de voisins

-2-
Calculer la distance

$$\sum_{i=1}^{n} |x_i - y_i|$$

Euclidienne

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

Manhattan

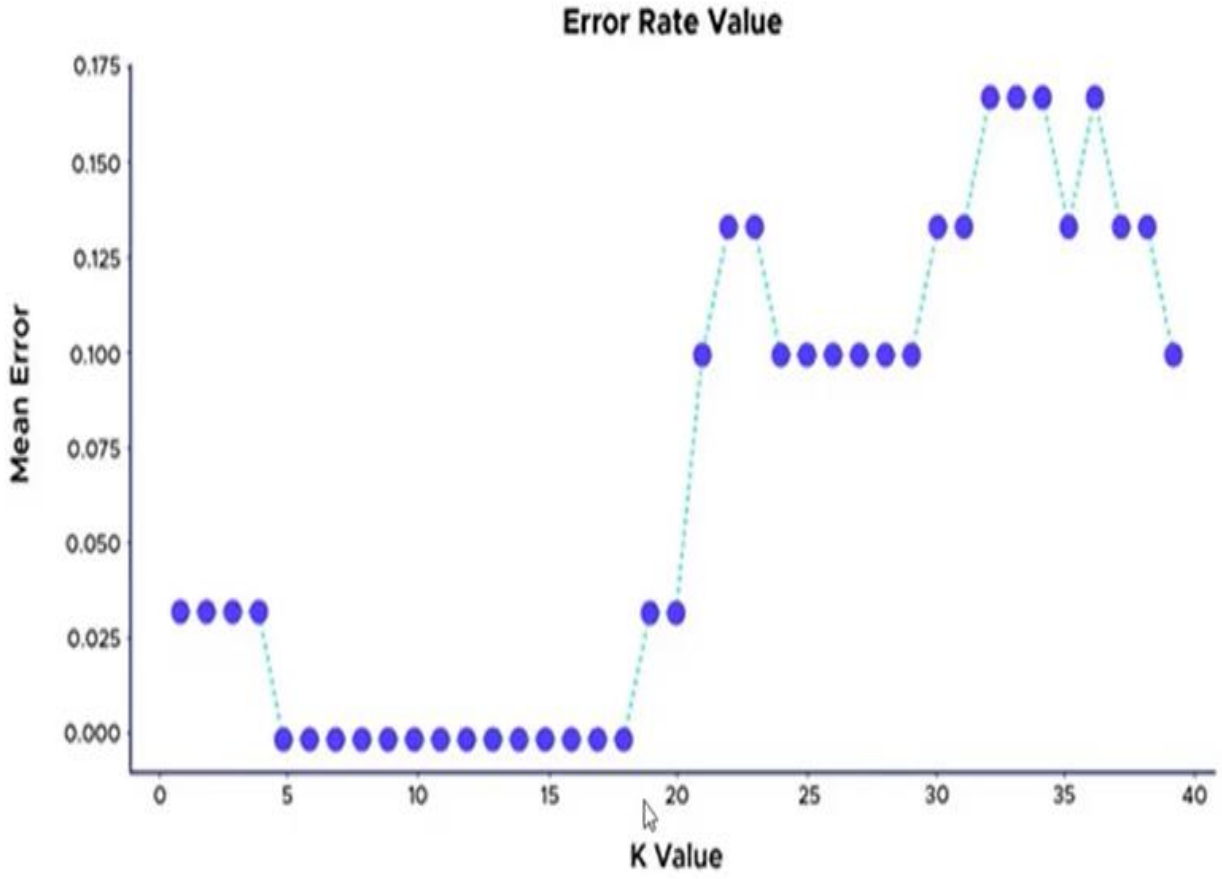
-3-
Prendre les K voisins les plus proches selon la distance calculée.

-4-
Parmi ces K voisins, compter le nombre de points appartenant à chaque catégorie.

-5-
Attribuer le nouveau point à la catégorie la plus présente parmi ces K voisins.

K: Hyper paramètre

Choisir le K pour lequel la classification sera la meilleure

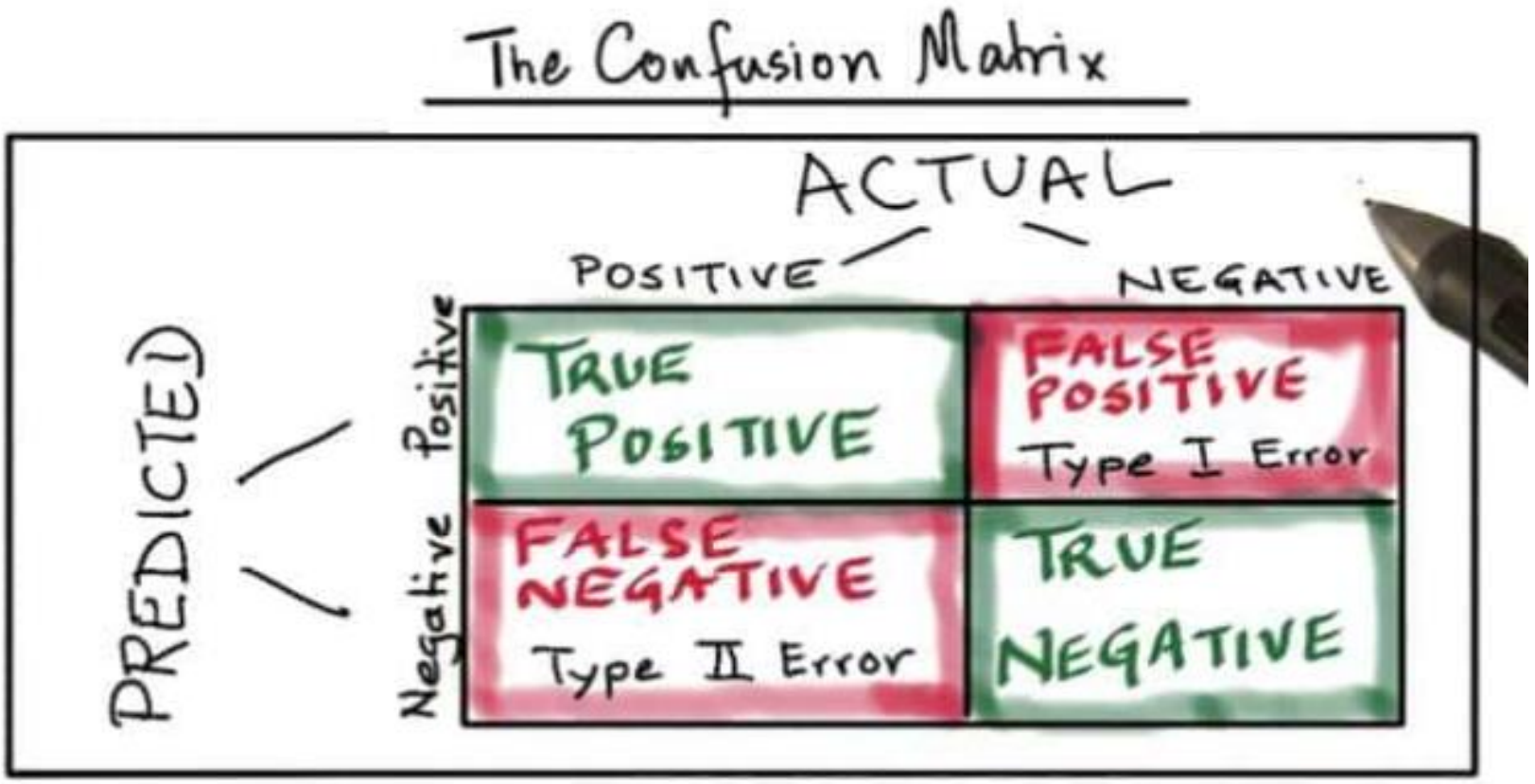


Matrice de confusion

- Afin de **mesurer les performances d'un modèle de Machine Learning**, on utilise généralement la **Confusion Matrix** ou matrice de confusion

- Une Confusion Matrix est un **résumé des résultats de prédictions sur un problème de classification**. Les prédictions correctes et incorrectes sont mises en lumière et réparties par classe. Les résultats sont ainsi comparés avec les valeurs réelles.

Matrice de confusion



Matrice de confusion

- Il faut **bien comprendre les quatre terminologies principales** : TP, TN, FP et FN. Voici la définition précise de chacun de ces termes :

♣**TP (True Positives)** : les cas où la prédiction est positive, et où la valeur réelle est effectivement positive. Exemple : le Test COVID19 est positif et la personne est bel et bien malade.

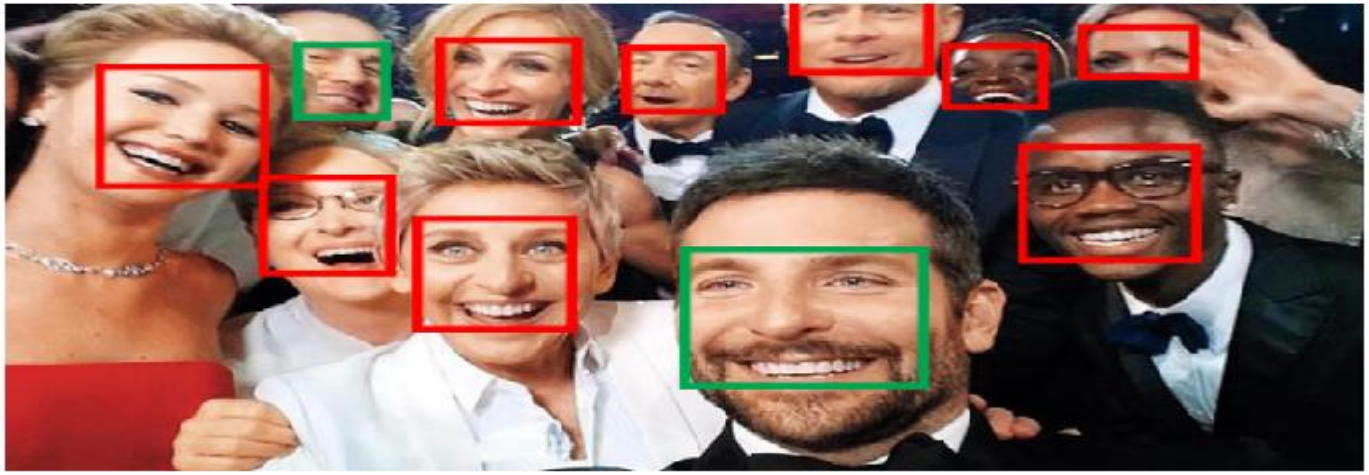
♣**TN (True Negatives)** : les cas où la prédiction est négative, et où la valeur réelle est effectivement négative. Exemple : : le Test COVID19 est négatif et la personne n'est effectivement pas malade.

Matrice de confusion

- **FP (False Positive)** : les cas où la prédiction est positive, mais où la valeur réelle est négative. Exemple : : le Test COVID19 est positif et la personne n' est pas malade.

- **FN (False Negative)** : les cas où la prédiction est négative, mais où la valeur réelle est positive. Exemple : : le Test COVID19 est négatif et la personne est malade.

Matrice de confusion: Exemple



Résultat d'une détection de Bradley Cooper

Nous avons donc y = [0,0,0,0,0,0,0,0,0,0,0,1] et y pred = [0,0,0,0,0,0,0,0,0,0,1,1]

		Predicted class: 0	Predicted class: 1
True class	0	9 true negative (tn)	1 false positive (fp)
	1	0 false negative (fn)	1 true positive (tp)

Matrice de confusion pour la détection de Bradley Cooper

Accuracy

- L'accuracy permet de connaître la proportion de bonnes prédictions par rapport à toutes les prédictions. L'opération est simplement :

Nombre de bonnes prédictions / Nombre total de prédictions

$$\text{Accuracy} = \frac{tn + tp}{tn + fp + fn + tp}$$

- Dans l'exemple de Bradley Cooper :

$$\text{Accuracy} = \frac{10}{11} = 0.9$$

Précision (precision)

- La précision correspond au nombre d'exemples de **correctement** attribués à la classe i par rapport au nombre total d' exemples **prédits** comme appartenant à la classe i (total **predicted** positive).

$$\text{Precision} = \frac{\mathbf{tp}}{\mathbf{tp + fp}}$$

- La précision permet de mesurer **le coût des faux positifs,** c'est-à-dire ceux détectés par erreur. Si l'on cherche à limiter les faux positifs, c'est **cet** indicateur que l'on va chercher à minimiser.

♣Dans l'exemple de Bradley Cooper : $\text{Precision} = \frac{1}{2} = 0{,}5$

Rappel (recall)

- Le rappel correspond au nombre d'exemples **correctement** attribués à la classe i par rapport au nombre total d'exemples **appartenant** à la classe i (total true positive).

$$\text{Recall} = \frac{tp}{tp + fn}$$

- Le rappel permet d'estimer combien d'exemples réellement positifs le modèle a **reconnu** par rapport au nombre de fois où il **aurait dû** reconnaitre

♣Dans l'exemple de Bradley Cooper : $\text{Recall} = \frac{1}{1} = 1$

Score

- Le Score combine subtilement la précision et le rappel. Il est intéressant et plus intéressant que l'accuracy car le nombre de vrais négatifs (tn) n'est pas pris en compte

$$\text{Score} = 2\frac{\text{precision} * \text{recall}}{(\text{precision} + \text{recall})} = 2\frac{(tp/(tp+fp)) * (tp/(tp+fn))}{tp/(tp+fp) + tp/(tp+fn)}$$

- Dans l'exemple de Bradley Cooper :

$$\text{Score} = \frac{1}{1,5} = 0,7$$

KNN: Pratique

KNN: Pratique

```
# Importer les librairies
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_digits
from sklearn.metrics import confusion_matrix
```

```
#Charger le dataset digits
digits = load_digits()
X, y = digits.data, digits.target
```

```
# Diviser le dataset en jeu d'apprentissage et je de test
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2)
```

```
#Créer le modèle knn
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train, y_train)
```

```
#Le taux de précision
score = knn.score(X_test, y_test)
print('Score: %f' % score)
```

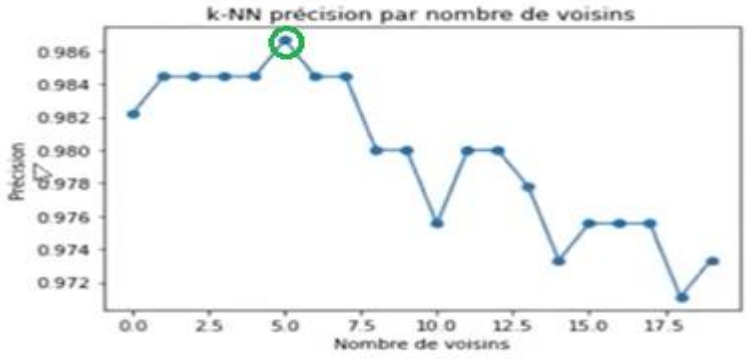
KNN: Pratique

```
#Tester k de 1 à 20
neighbors = np.arange(1,21)
train_accuracy =np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

for i,k in enumerate(neighbors):

    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    train_accuracy[i] = knn.score(X_train, y_train)
    test_accuracy[i] = knn.score(X_test, y_test)
```

```
print(test_accuracy)
```

```
[0.98222222 0.98444444 0.98444444 0.98444444 0.98444444 0.98666667
 0.98444444 0.98444444 0.98       0.98       0.97555556 0.98
 0.98       0.97777778 0.97333333 0.97555556 0.97555556 0.97555556
 0.97111111 0.97333333]
```

```
#Visualisation taux de précision (Test)
plt.plot(test_accuracy, 'o-')
plt.title('k-NN précision par nombre de voisins')
plt.xlabel('Nombre de voisins')
plt.ylabel('Précision')
plt.show()
```



KNN: Pratique

Pour k= 6

```
y_pred=knn.predict(X_test)
```

```
#Matrice de confusion
cm = confusion_matrix(y_test, y_pred)
cm
```

```
array([[47,  0,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0, 48,  0,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0, 27,  0,  0,  0,  0,  0,  0,  0],
       [ 0,  0,  0, 40,  0,  1,  0,  0,  0,  0],
       [ 0,  0,  0,  0, 55,  0,  0,  0,  0,  0],
       [ 0,  0,  0,  0,  0, 49,  0,  0,  0,  2],
       [ 0,  0,  0,  0,  0,  0, 47,  0,  0,  0],
       [ 0,  0,  0,  0,  0,  0,  0, 58,  0,  0],
       [ 0,  2,  0,  0,  0,  0,  0,  0, 39,  0],
       [ 0,  0,  0,  0,  0,  0,  0,  0,  1, 34]], dtype=int64)
```

```
np.bincount(y_test)
```

```
array([47, 48, 27, 41, 55, 51, 47, 58, 41, 35], dtype=int64)
```