

CHAPITRE	
2	MACHINE LEARNING

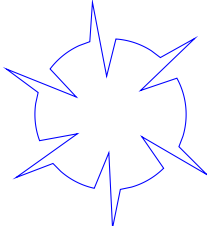


Table des matières

1 Comprendre le Machine Learning :	1
2 Construire une IA :	2
3 Machine Learning en Python.....	6
3.1 La régression linéaire	6
3.2 Les voisins de mes voisins	9
4 En conclusion	11

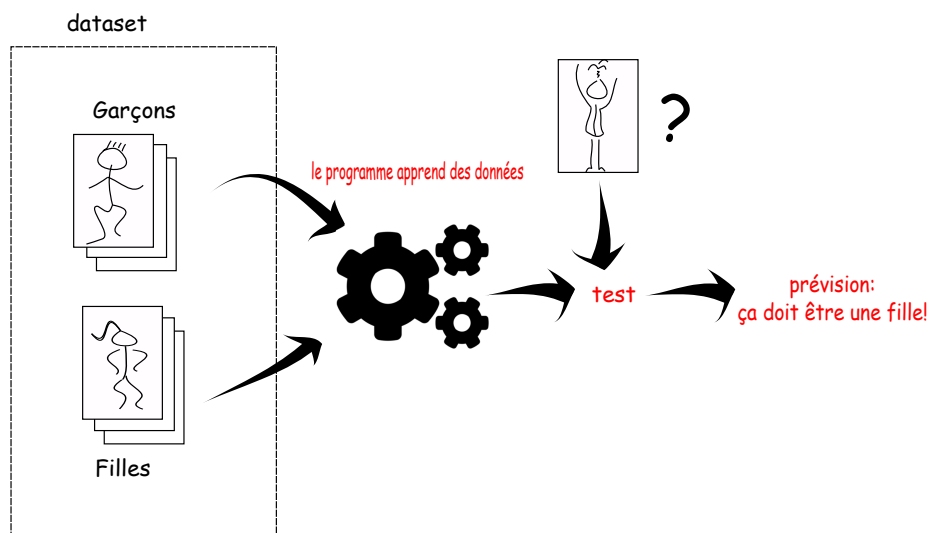
1 Comprendre le Machine Learning :

Dans le Machine Learning, on parle d'**apprentissage supervisé** lorsque la machine apprend à partir de données étiquetées. D'un jeu de données connues(Dataset), le programme définit un modèle puis l'utilise pour prédire un résultat sur une nouvelle entrée.



Le Machine Learning (ML) comme le Deep Learning sont des domaines de l'intelligence artificielle

L'apprentissage profond repose sur l'utilisation de **réseaux de neurones** et permet de répondre à des questions qu'il n'aurait pas forcément étudié sur des exemples préalables.



Le programme ne sert qu'à la tâche pour laquelle il est conçu : fille ou garçon. Si vous lui proposez la photo d'une chaise, il arrivera à la même conclusion, farfelue dans ce cas.

Exercice n°1

- ❶ Allez sur le site <https://fr.vittascience.com/ia/images.php>.
- ❷ Créer les catégories chat et chien en chargeant les jeux de données (dataset).
- ❸ Entraînez le modèle.
- ❹ Récupérez la photo d'un chat ou d'un chien et soumettez-là à l'IA.
- ❺ Vous pouvez recommencer l'opération avec d'autres photos, d'animaux ou non.

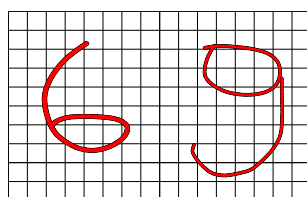
La reconnaissance des visages et plus généralement des formes dans les images est au coeur des intérêts applicatifs des IA. Mais comment un programme peut-il me reconnaître ? Comment un radar peut-il extraire les composantes d'une plaque d'immatriculation ?



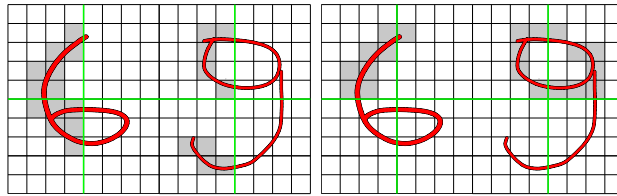
Comment un programme peut-il apprendre d'une forme ?

2 Construire une IA :

Pour répondre en partie à la problématique précédente, nous allons construire notre propre programme de reconnaissance, notre propre IA : celle qui distingue un 6 d'un 9 !



De façon totalement arbitraire, comptons les carreaux (dans une image ce serait un pixel) qui contiennent une information à gauche d'une ligne de référence puis au dessus d'une ligne de référence :

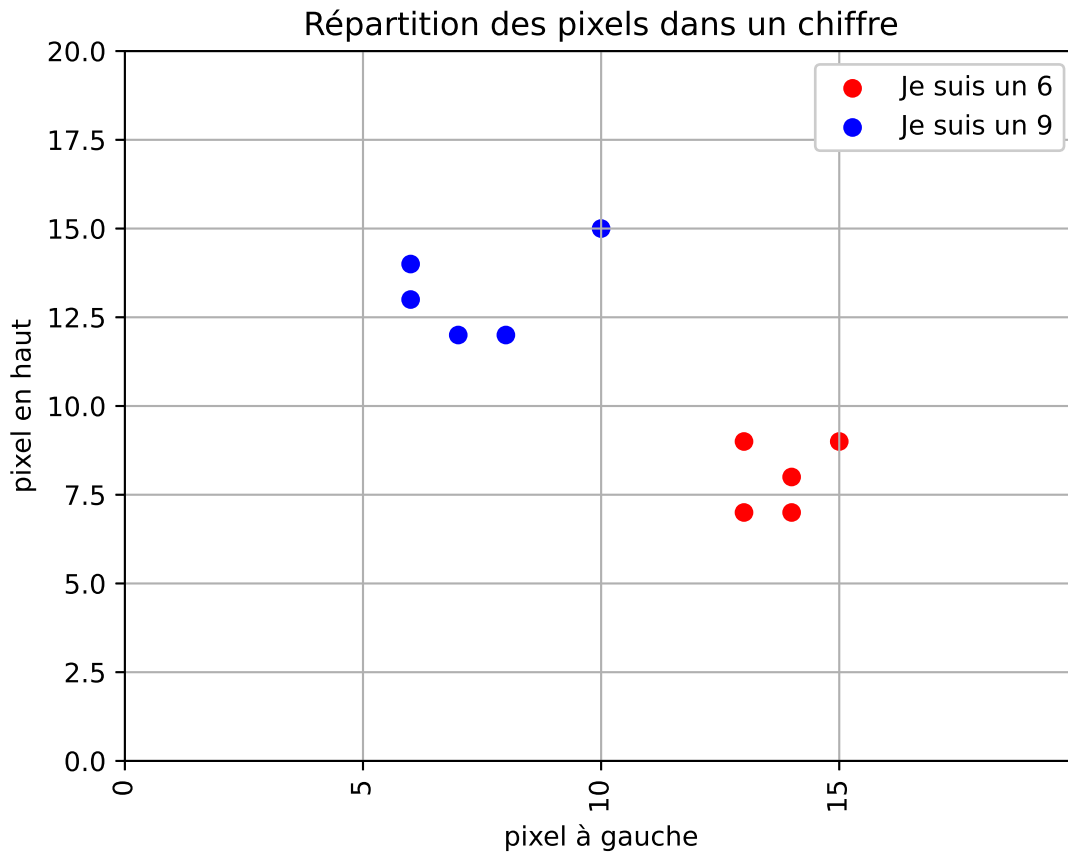


Le six est alors associé au couple (13,8) et le 9 au couple (9,12) (dans la réalité il s'agirait de dizaines de milliers de pixels ...)

Recommençons l'expérience manuscrite pour construire un dataset plus complet ! On obtient les résultats suivants :

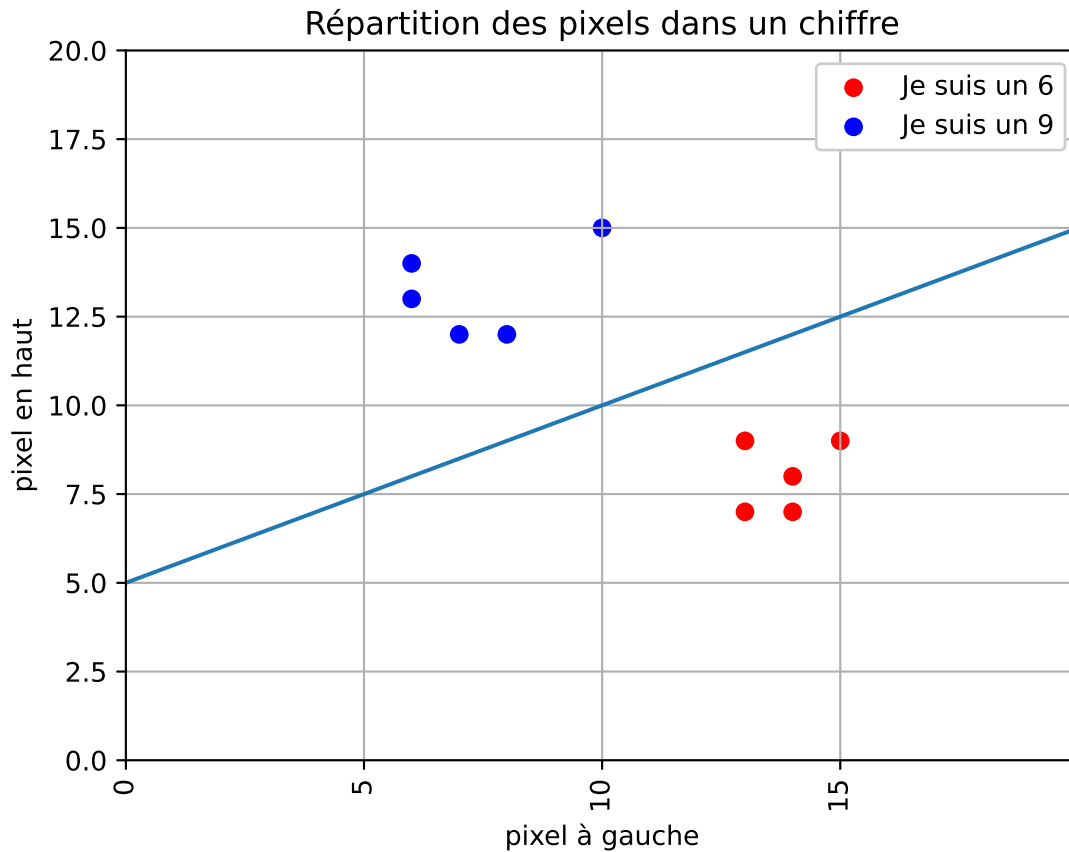
Pour le six	(14,8)	(15,9)	(13,9)	(14,7)	(13,7)
Pour le neuf	(6,14)	(6,13)	(7, 12)	(10,15)	(8, 12)

Et avec un graphique, c'est encore mieux !



Le graphique fait clairement apparaître deux catégories. Une IA va alors chercher à formaliser cette séparation. Par exemple en décidant de construire une frontière entre les deux groupes :

- ➔ tous ceux qui seront au dessus seront considérés comme un 9
- ➔ et tous ceux qui seront en dessous considérés comme un 6 !



Mais comment construire cette frontière ?

Si l'on considère que la frontière est une droite alors rien de plus simple : il suffit de s'appuyer sur les programmes de mathématiques :



Dans un repère du plan, toute droite admet une équation cartésienne de la forme $ax + by + c = 0$ et réciproquement.

Si les termes génériques en mathématiques sont x et y pour désigner les deux coordonnées alors il s'agirait dans notre cas, de déterminer trois coefficients a, b et c tels que :

$$ag + bh + c = 0$$

où (g, h) désigne le couple gauche/haut étudié ci-dessus.

Par exemple, si on considère la droite tracée ci-dessus son équation serait alors :

$$0.5g - h + 5 = 0$$

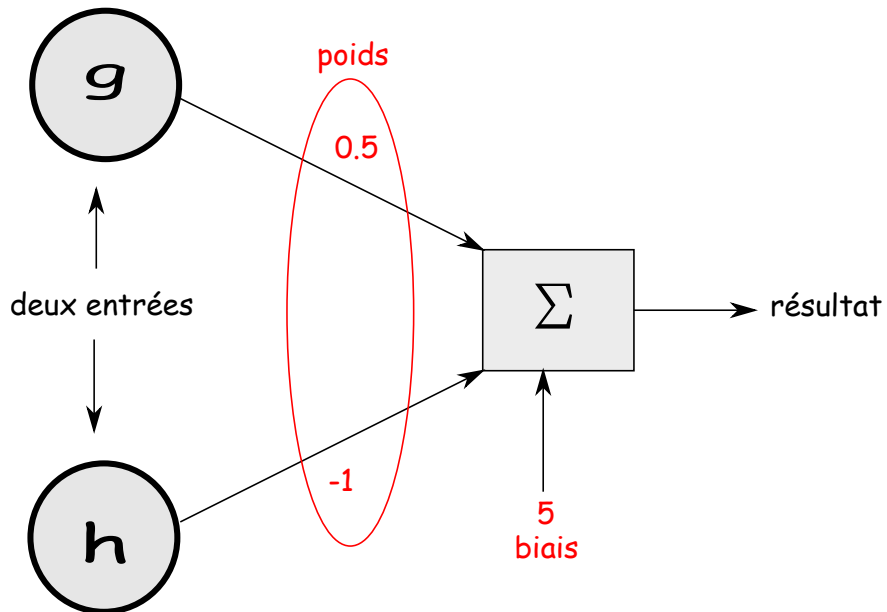
de telle façon que pour toutes coordonnées (g, h) « au-dessus » de cette droite, on aurait

$$0.5g - h + 5 < 0$$

et inversement pour toutes coordonnées (g, h) « au-dessous » de cette droite, on aurait

$$0.5g - h + 5 > 0$$

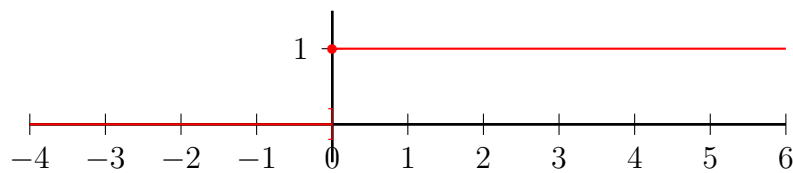
On peut illustrer la situation par un arbre pondéré(son véritable nom sera donné plus tard...) :



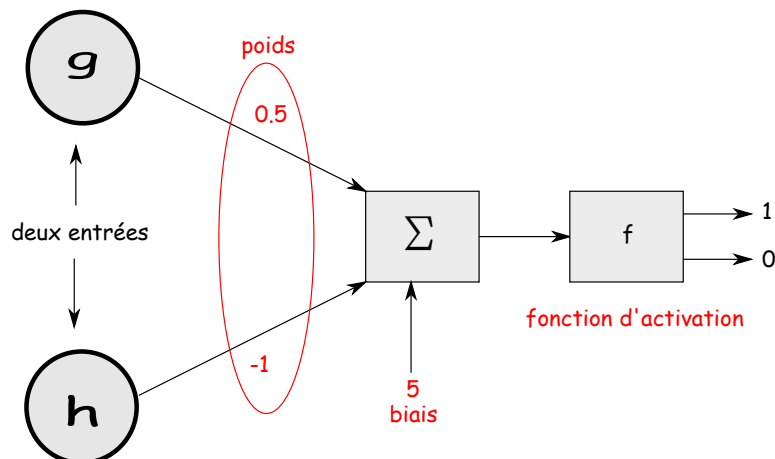
Cet « arbre » prend donc en entrées deux valeurs g et h obtenues à partir de la lecture d'un chiffre, dont on a attribué des poids, des coefficients, et fourni en sortie un résultat :

- ➔ si le résultat est positif alors il s'agirait d'un 6
- ➔ si le résultat est négatif alors il s'agirait d'un 9.

Nous allons donc ajouter une fonction f qui retourne 1 si le résultat est positif et 0 sinon par exemple, la fonction de Heaviside dont la courbe est donnée ci-dessous :



La fonction qui assure ce rôle est appelée **fonction d'activation**.Ce qui donne :





Félicitations! Vous venez de concevoir votre premier **neurone artificiel**!

Il s'agit plus précisément d'un perceptron dont le principe de fonctionnement reprend celui du neurone humain.



Le principe de Machine Learning est de choisir des poids qui minimisent les erreurs commises par ce neurone.

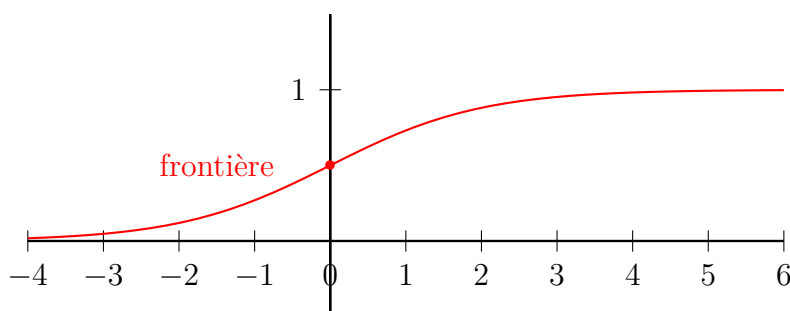
On parle d'erreurs quand le neurone affirme que c'est un six alors que c'est autre chose. Rappelons que dans la première phase de l'apprentissage supervisé, le neurone s'entraîne sur des jeux de données étiquetées : il sait donc si il se trompe ou non.

Les méthodes mathématiques mises en jeu ne sont pas du niveau d'un élève de terminale (descente du gradient...) mais ce sont grossièrement des méthodes de dérivation et d'optimisation, comme celles vues au lycée (minimum d'une fonction sur un intervalle...).

Nous avons choisi une fonction d'activation simple qui retourne 1 (probabilité de l'événement certain) ou 0 (probabilité de l'événement impossible) et donc un neurone qui se prononce de façon brutale : c'est un six ou pas!

Dans la réalité des fonctions plus souples comme la **sigmoïde**(ci-dessous) sont utilisées pour associer au couple (g, h) d'entrée une probabilité comprise entre 0 et 1.

En effet, plus un point est éloigné de la frontière, plus on est certain de le catégoriser. Mais un point proche de la frontière peut laisser des doutes...



Si un chiffre produit un couple (g, h) sur la frontière alors le neurone donne comme résultat 0 en entrée de la fonction d'activation qui retourne alors 0.5 dans ce dernier cas. Logique, non? Dans la réalité, la reconnaissance des chiffres manuscrits est bien plus compliquée que cela. La classification se fait par un réseau de neurones particulier appelé réseau de convolution (CNN) ou par un SVM (voir plus loin).

Le paragraphe suivant propose l'utilisation de bibliothèques Python réalisant des tâches de classification.

3 Machine Learning en Python

3.1 La régression linéaire

La statistique livre aussi des algorithmes d'apprentissage : la **régression linéaire** est une technique mathématique qui permet d'exprimer une quantité Y en fonction d'une variable X sous certaines conditions. Nous allons en décrire le principe par du code Python.

Exercice n°2 Installation des bibliothèques

Installer les bibliothèques nécessaires puis importez-les.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn.datasets import make_regression
4 from sklearn.linear_model import SGDRegressor
```

Nous allons maintenant créer des données virtuelles.

Exercice n°3

- ➊ Ajouter les lignes de codes suivantes au fichier précédent.
- ➋ Exécutez plusieurs fois pour observer que les données changent.
- ➌ Documentez la fonction `make_regression`.

```
1 np.random.seed(0)
2 x, y = make_regression(n_samples = 100, n_features = 1, noise = 10)
3 plt.scatter(x, y)
4 plt.show()
```

Nous allons maintenant construire un modèle théorique : celui qui va permettre d'exprimer y en fonction de x à l'aide d'une fonction affine.



La régression linéaire est une méthode qui permet de déterminer les coefficients d'une fonction affine donnant y en fonction de x avec quelques erreurs d'approximations.

Notre choix porte sur `SGDRegressor`, choix classique, même si plusieurs modèles plus fins existent.

Exercice n°4 Définition du modèle et entraînement

Insérer les codes suivants.

```
1 modele = SGDRegressor(max_iter = 100, eta0 = 0.0001)
2 modele.fit(x,y)
3 print('Coeff R =', model.score(x, y))
4 plt.plot(x, model.predict(x), c = 'red', lw = 3)
```

Le modèle étant construit, on peut l'utiliser sur une entrée quelconque `choix` et visualiser ce que prévoit alors le modèle.

Exercice n°5 Utilisation du modèle

Ajouter les lignes suivantes en choisissant une valeur de `choix` autour de 0.

```

1 choix = np.array([[1]])
2 print((choix, model.predict(choix)))

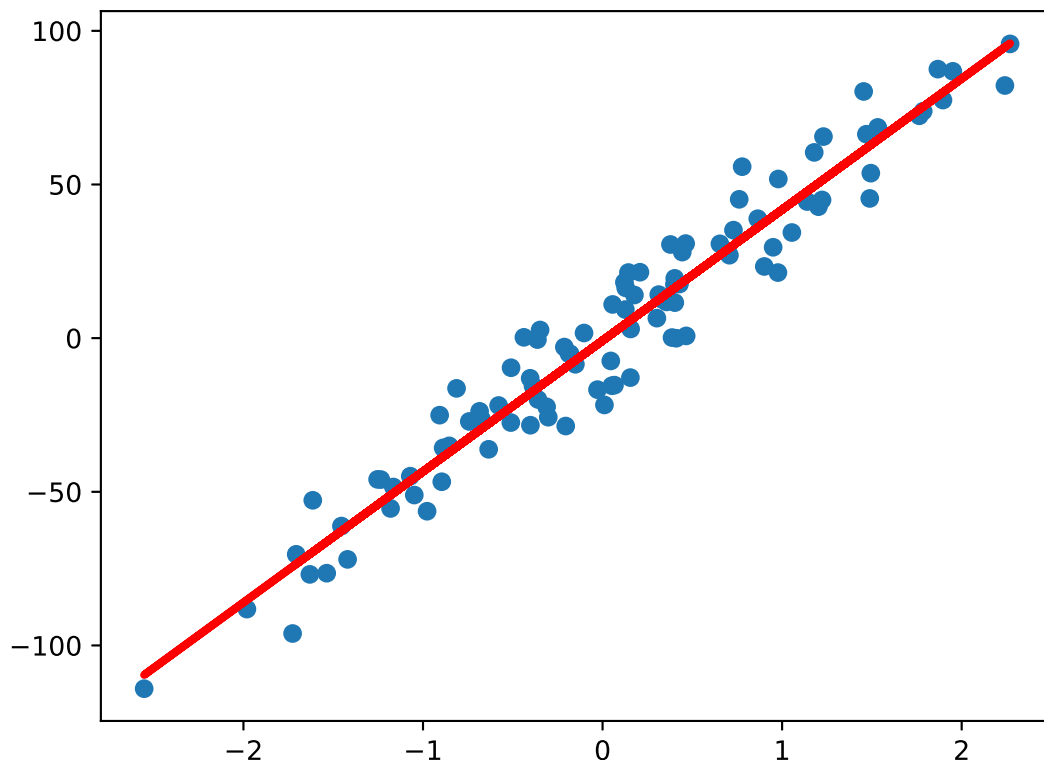
```

et pour visualiser la régression, vous pouvez ajouter ces dernières lignes de codes :

```

1 print('Coeff R =', model.score(x, y))
2 print(model.coef_, model.intercept_)
3 plt.plot(x, model.predict(x), c = 'red', lw = 3)

```



La régression linéaire est **pertinente** lorsqu'il s'agit d'ajuster un nuage de points presque alignés.

La fonction **coût** est la fonction qui comptabilise la différence entre les couples empiriques (x, y) et les couples théoriques $(x, \text{model.predict}(x))$.

Pour comparer les données théoriques des données réelles, on peut ajouter ces lignes de code :

```

1 from random import randint
2 alea = randint(0, 99)
3 choix_alea_dans_x = x[alea]
4 correspondance_dans_y = y[alea]
5 valeur_predit_x = model1.predict([choix_alea_dans_x])
6 print(f"La valeur théorique pour {choix_alea_dans_x} est {valeur_predit_x}
  → pour une valeur {correspondance_dans_y} dans les données")

```


Pour être complet, cette méthode cherche les coefficients a et b d'une fonction affine qui minimisent la fonction **coût**.

3.2 Les voisins de mes voisins

Je propose ici une autre méthode de classification des apprentissages supervisés : l'algorithme KNN pour k-Nearest-Neighbors, ou plus simplement l'algorithme des k plus proches voisins.

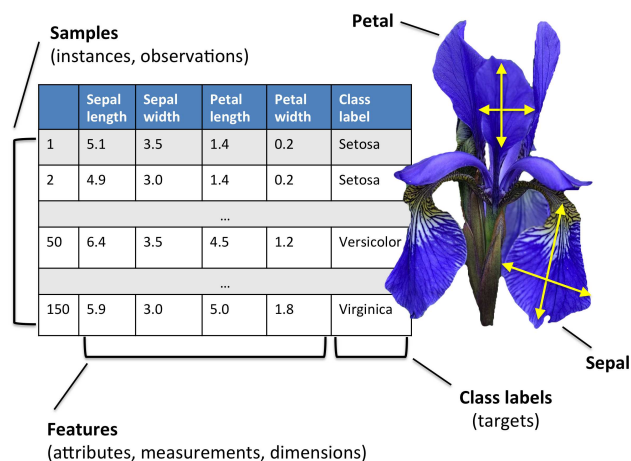
Contrairement aux réseaux de neurones, cet algorithme ne nécessite pas de phase d'entraînement : il faut juste stocker le jeu de données d'apprentissage.

Exercice n°6 Chargement du jeu de données

Dans un nouveau fichier python, insérer les lignes de codes suivantes.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from sklearn import datasets
4 from sklearn.neighbors import KNeighborsClassifier
```

Cet ensemble de données se compose de quatre champs, à savoir la longueur du sépale, la largeur du sépale, la longueur du pétale et la largeur du pétale. Il contient également une super classe qui contient trois fleurs différentes, Iris setosa, Iris versicolor et Iris virginica.



On peut alors observer que les données sur les Iris se composent en quatre catégories. Par la suite on ne gardera que les deux premières pour simplifier tout cela.

Exercice n°7 Algorithme KNN

- 1 Recopier le code suivant
- 2 Avec $k = 3$, dans quelle catégorie se trouve les fleurs à classer ?
- 3 Et avec $k = 4$?

④ Effectuer une boucle pour tester le modèle sur différentes valeurs de k .

```
1  #--- Chargement des données ---#
2  iris = datasets.load_iris()

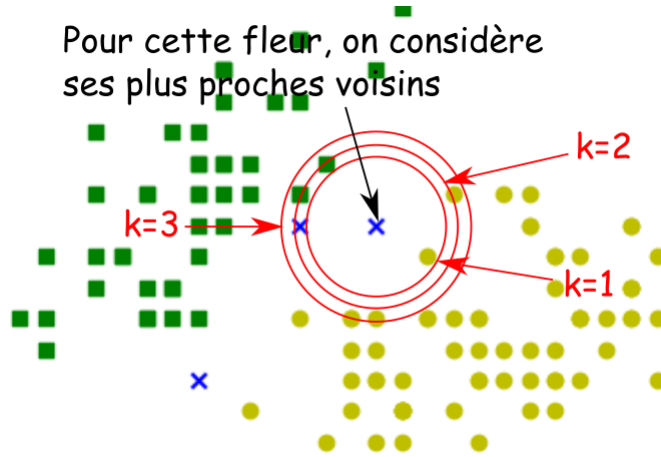
3  # choix de deux variables de type np.array
4  X = iris.data[:, :2] # Pour ne garder que les deux premières données
5  y = (iris.target != 0) * 1 # Pour n'avoir que deux classes(0 ou 1)

6  #--- visualisation des données ---#
7  plt.figure(figsize = (10, 6))
8  plt.scatter(X[y == 0][:, 0], X[y == 0][:, 1], marker = 's',color = 'g', label
9  ↪ = '0')
10 plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1], color = 'y', label = '1')

11 #Fleur à classifier
12 Iries_To_Predict = [
13     [5.4, 3.3],
14     [5.5, 2.5],
15     [7, 3],
16     [3,2],
17     [5,2.8],
18     [5.7, 3.3]
19 ]
20 # Transformation des données en listes
21 X_p = []
22 Y_p = []
23 for fleur in Iries_To_Predict:
24     X_p.append(fleur[0])
25     Y_p.append(fleur[1])
26 plt.scatter(X_p , Y_p, marker = 'x',color = 'b', label = 'test')
27 plt.legend()
28 #--- Algorithme KNN ---#
29 k = 3 # choix du nombre de voisins
30 model = KNeighborsClassifier(n_neighbors = k) # construction d'un objet de
31 ↪ KNN
32 model.fit(X, y) # Entraînement du modèle
33 print(model.predict(Iries_To_Predict))# Resultats du modèle
34 #--- Pour montrer le graphique ---#
35 plt.show()
```

L'algorithme des KNN consiste à compter les k voisins les plus près de la cible à classifier. Ce sont ces voisins qui lui transmettent leur catégorie.

Pour cette fleur, on considère ses plus proches voisins



Pour $k = 1$, le plus proche voisin est un rond : la fleur est considérée comme tel.
 Pour $k = 3$, les trois plus proches voisins d'une même catégorie sont des ronds : la fleur est considérée comme tel.
 Pour $k = 2$, ce sont les deux carrés en haut à gauche qui sont les plus près même si cela ne se joue pas à grand chose...

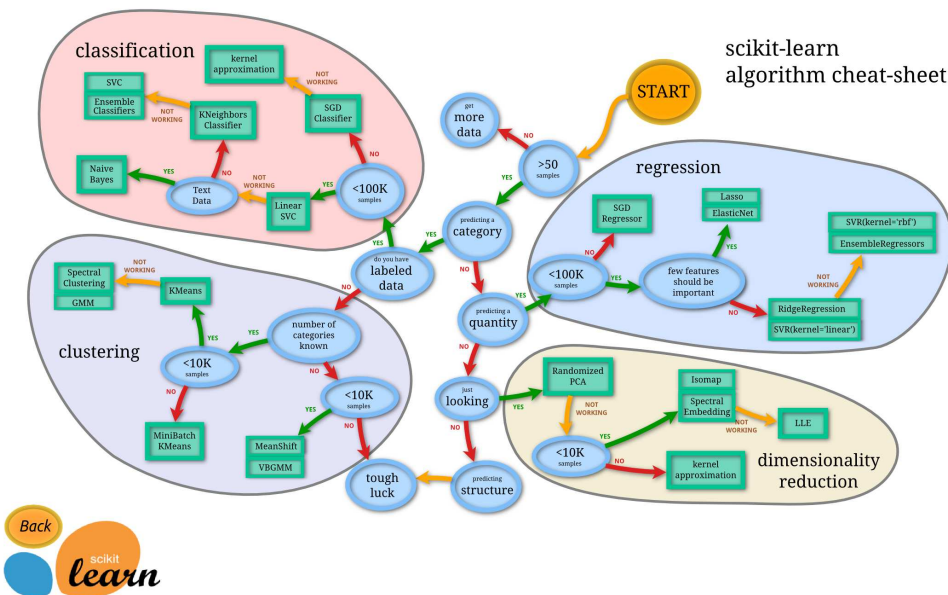
4 En conclusion

le Machine Learning en Python est associé à un package développé par l'INRIA : Scikit-Learn.



Pour le Deep Learning, c'est le package TensorFlow qui est une référence.

Vous trouverez sur le site <https://scikit-learn.org/stable/>, toutes sources d'inspiration pour vos Machine Learning. Attention tout de même, ce n'est pas toujours facile de comprendre quels sont les modèles mathématiques utilisés. Patience!



N'oubliez pas les trois phases du Machine Learning : entraînement(`fit..`), évaluation(`score..`) et utilisation(`predict`) et ceci quel que soit le modèle choisi.

Pour finir ce modeste travail, la version `sklearn` des classements des chiffres manuscrits est donnée ci-après. On retrouve les trois étapes du Machine Learning évoquées ci-dessus. Le modèle choisi est un SVC(Support Vector Classification).

```
1 import matplotlib.pyplot as plt
2
3 # Import datasets, classifiers and performance metrics
4 from sklearn import datasets, metrics, svm
5 from sklearn.model_selection import train_test_split
6
7 digits = datasets.load_digits()
8
9 _, axes = plt.subplots(nrows = 1, ncols = 4, figsize = (10, 3))
10 for ax, image, label in zip(axes, digits.images, digits.target):
11     ax.set_axis_off()
12     ax.imshow(image, cmap=plt.cm.gray_r, interpolation = "nearest")
13     ax.set_title("Training: %i" % label)
14 plt.show()
15
16 # flatten the images
17 n_samples = len(digits.images)
18 data = digits.images.reshape((n_samples, -1))
19
20 # Create a classifier: a support vector classifier
21 clf = svm.SVC(gamma = 0.001)
22
23 # Split data into 50% train and 50% test subsets
24 X_train, X_test, y_train, y_test = train_test_split(
25     data, digits.target, test_size=0.5, shuffle=False
26 )
27
28 # Learn the digits on the train subset
29 clf.fit(X_train, y_train)
30
31 # Predict the value of the digit on the test subset
32 predicted = clf.predict(X_test)
33
34 _, axes = plt.subplots(nrows = 1, ncols = 4, figsize = (10, 3))
35 for ax, image, prediction in zip(axes, X_test, predicted):
36     ax.set_axis_off()
37     image = image.reshape(8, 8)
38     ax.imshow(image, cmap=plt.cm.gray_r, interpolation="nearest")
39     ax.set_title(f"Prediction: {prediction}")
40 plt.show()
```