CHAPITRE 5

DES ALGORITHMES D'APPRENTISSAGE

I - Introduction

Le concept d'***intelligence artificielle*** n'est pas aisé à définir. L'idée essentielle repose sur la capacité pour une machine à accomplir des tâches usuellement dévolues aux humains et surtout la capacité à apprendre et à se perfectionner.

En général ce sont des types de problèmes –reconnaissance de formes, traitement du langage, classement et étiquetage automatique (*classification*), classification automatique (*clustering*),...– ou des types de méthodes –programmation logique, méthodes bayésiennes, apprentissage supervisé ou non supervisé,...– qui sont considérés comme relevant de l'intelligence artificielle.

L'***apprentissage automatique*** (*machine learning*) est le domaine de l'intelligence artificielle fondé sur des techniques mathématiques et statistiques permettant aux ordinateurs d'apprendre à partir de données, d'améliorer leurs performances, d'être capables de résoudre des problèmes pour lesquels ils n'ont pas été explicitement programmés.

Les principes de l'apprentissage reposent sur les étapes suivantes :

- on considère des données ;
- dans une phase d'apprentissage, la machine explore les données ;
- dans une phase de test, on compare la prédiction faite par la machine à la véritable réponse ;
- éventuellement, on ajuste son modèle...

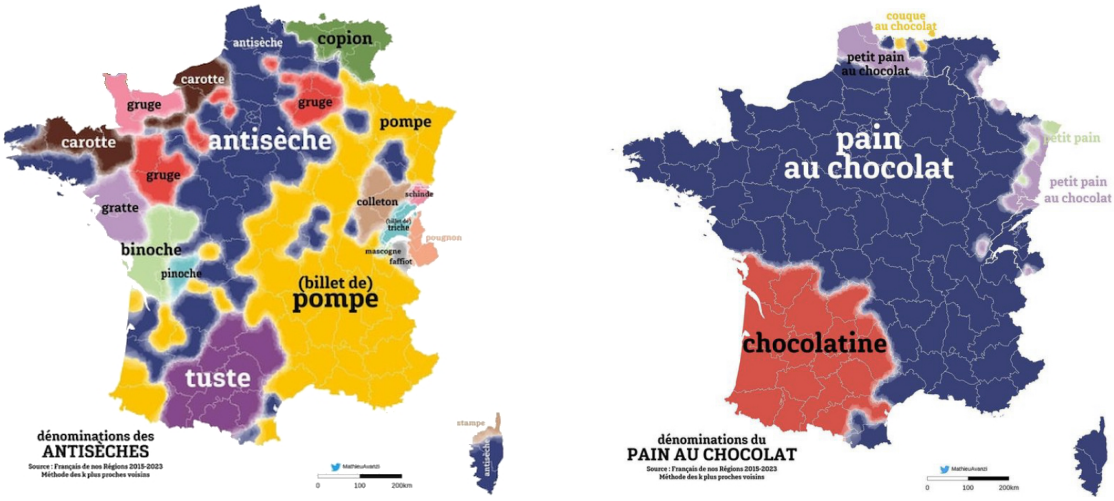
On distingue l'apprentissage supervisé de l'apprentissage non supervisé :

- pour l'apprentissage supervisé, on s'intéresse à un problème de classement :
  - on dispose de données (l'ensemble d'apprentissage) déjà classées ou étiquetées ;
  - on souhaite, pour les nouvelles données, attribuer une étiquette en se fondant sur ce que l'on sait par l'ensemble d'apprentissage.
- pour l'apprentissage non supervisé, on ne connaît pas *a priori* la structure des données et la machine cherche à identifier les paramètres, reconnaître la structure et effectuer elle-même la classification.

Précisons enfin que l'on a plusieurs type de problèmes :

- des problèmes de classement : il s'agit de prédire une valeur qualitative.
  Par exemple, un joueur joue à Chifoumi devant la caméra et l'ordinateur identifie s'il a fait pierre, ciseaux ou feuille.
- des problèmes de régression : il s'agit de déterminer une valeur quantitative, un lien entre des variables, un seuil intéressant pour un paramètre,...

II - Un exemple d'apprentissage supervisé : l'algorithme des k plus proches voisins

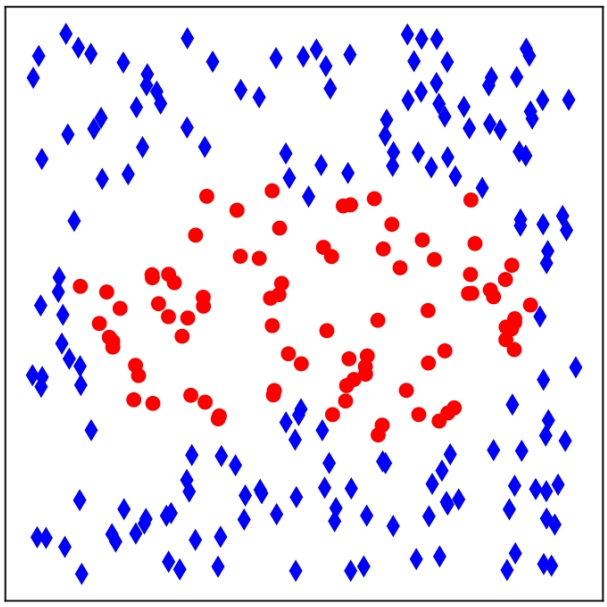


Le problème est le suivant. On souhaite classer ou étiqueter des données appartenant à un ensemble muni d'une distance (ou d'une mesure de similarité). Les étiquettes sont déterminées et on dispose d'un ensemble d'apprentissage dans lequel chaque donnée est étiquetée ou classée.
L'idée de l'algorithme des k plus proches voisins (*k nearest neighbors, kNN*) pour prédire l'étiquette d'un élément x consiste à :

- calculer les k éléments de l'ensemble d'apprentissage les plus proches de x ;
- attribuer à x l'étiquette la plus fréquente dans ce voisinage.

Exemple

On considère un ensemble des points de coordonnées $(x, y) \in [0,1]^2$ appartenant à l'une ou l'autre de deux catégories représentées par des losanges bleus ou des disques rouges. La figure suivante propose un ensemble de 250 données d'entraînement.



Une fois ces données acquises, nous souhaitons qu'à partir d'un point de coordonnées (x, y), la machine lui attribue une catégorie (bleu ou rouge).

On suppose que les données d'apprentissage sont stockées dans un tableau `donnees` sous la forme `(x,y,c)` où `x` et `y` désignent les coordonnées du point et `c` désigne `"b"` ou `"r"` pour indiquer la couleur.

On doit d'abord définir une fonction de distance entre deux données :

```
def distance(a, b):
    """ entrée : a, b sont sous la forme (x, y, c)
        sortie : distance (au carré) entre les deux points correspondant à a et b
    """
    return (a[0]-b[0])**2 + (a[1]-b[1])**2
```

On doit disposer d'une fonction de tri sous la forme :

```
def tri(x, y, T):
    """ entrée : x, y (float) coordonnées du point considéré,
                 T tableau des données
        sortie : tableau ayant le même contenu que T mais trié
                    par distance croissante vis-à-vis de (x,y)
    """
```

On peut alors programmer la fonction principale :

```
def plusProchesVoisins(x, y, k):
    """ entrée : x, y (float) coordonnées du points considéré,
                 k (int) nombre de voisins considérés
        sortie : étiquette de (x,y) égale à "b" ou "r"
    """
    t = tri(x, y, T)[:k]
    compteur = 0
    for point in t:
        if point[2] = "r":
            compteur += 1
    if 2 * compteur > k:
        return "r"
    else:
        return "b"
```

On a intérêt à choisir k impair, pourquoi ?

On teste systématiquement un tel algorithme d'apprentissage. On sélectionne une partie des données dont l'étiquette est connue et on compare leurs étiquettes avec ce qui est prédit par l'algorithme.

Le ***matrice de confusion*** permet d'évaluer la qualité du système de classification. En ligne, on indique les différentes classes et les colonnes correspondent aux classes estimées.

Exemple

Reprenons l'exemple précédent. La matrice de confusion est de la forme :

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

où :

- a est le nombre de points rouges identifiés comme rouges ;
- b est le nombre de points rouges identifiés comme bleus ;
- c est le nombre de points bleus identifiés comme rouges ;
- d est le nombre de points bleus identifiés comme bleus.

Par exemple, la matrice :

$$\begin{pmatrix} 27 & 4 \\ 2 & 66 \end{pmatrix}$$

signifie que, sur 100 données, il y a 27 points rouges identifiés comme rouges mais 4 identifiés comme bleus, etc.

Dans l'idéal, la matrice serait diagonale !

En comparant les matrices de confusion pour différentes valeurs de k, on peut alors choisir le k semblant le plus pertinent.

Exemple

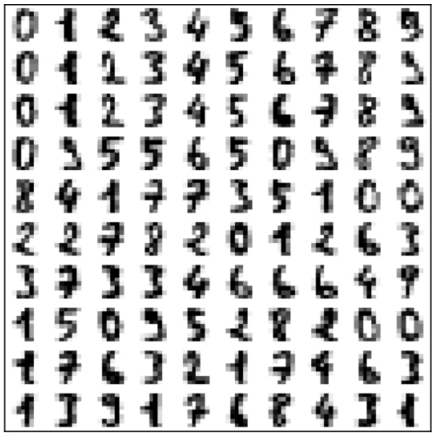
Toujours avec le même exemple, imaginons que l'on obtienne :

k	1	3	7	17	31
matrice	$\begin{pmatrix}27 & 4\\ 2 & 66\end{pmatrix}$	$\begin{pmatrix}30 & 1\\ 3 & 66\end{pmatrix}$	$\begin{pmatrix}28 & 3\\ 4 & 65\end{pmatrix}$	$\begin{pmatrix}27 & 4\\ 4 & 65\end{pmatrix}$	$\begin{pmatrix}28 & 3\\ 10 & 59\end{pmatrix}$

Il semble pertinent dans ce cas de choisir $k = 3$.

Exercice

Nous disposons de 1797 images de 8×8 pixels en niveau de gris représentant des chiffres de 0 à 9, par exemple :



Plus précisément, ces données nous sont fournies par l'intermédiaire de deux tableaux :

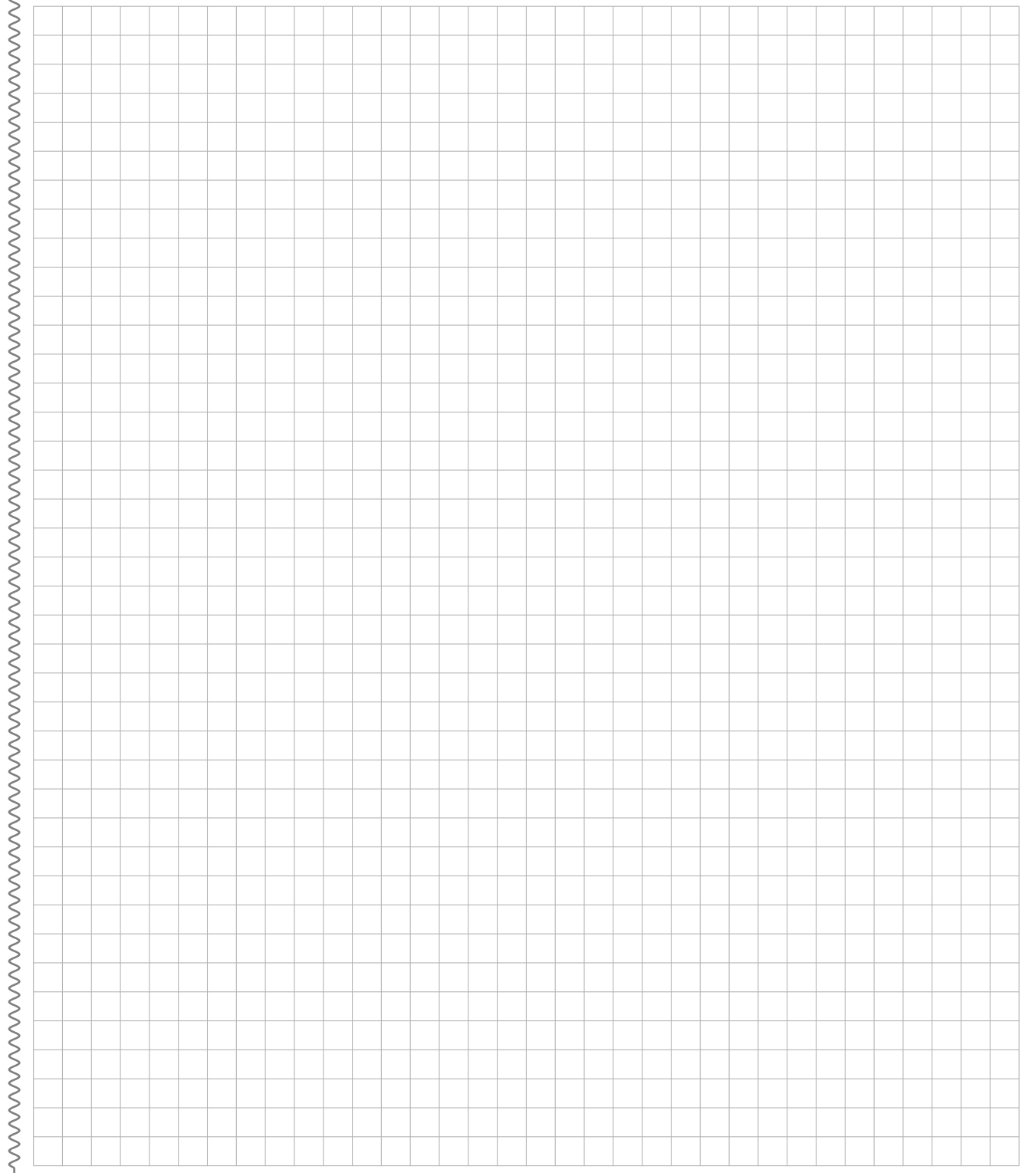
- X est une matrice de taille 1797×64 où chaque X[k] est un vecteur de $\mathbb{R}^{64}$ qui représente l'image d'un chiffre ;
- Y est un vecteur de taille 1797 où chaque Y[k] est le label de X[k], autrement dit l'entier entre 0 et 9 qui est représenté par X[k].

Q1– Nous allons partager ces données en deux parties sensiblement égales : la première consacrée à l'apprentissage, la seconde au test. Dans la partie consacrée à l'apprentissage, on va choisir 90 représentants de chacune de nos dix classes, il restera donc 897 données pour nous servir de test.
Écrire des instruction Python permettant d'obtenir deux listes `donnees` et `test` contenant les indices (entre 0 et 1796) correspondant respectivement à une donnée d'apprentissage ou à une donnée de test.

Q2– Écrire une fonction `distance(U, V)` donnant la distance euclidienne entre les deux vecteurs de $\mathbb{R}^{64}$ représentés par les tableaux `U` et `V`.

Q3– Écrire une fonction de tri d'argument un vecteur `U` et renvoyant une liste avec le même contenu que `donnees` mais triée par ordre croissant de distance entre `U` et `X[i]`.

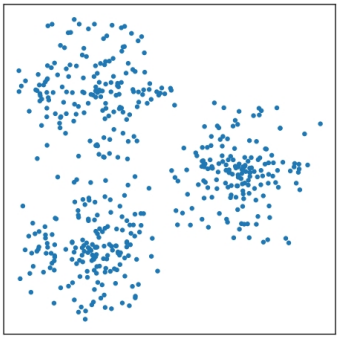
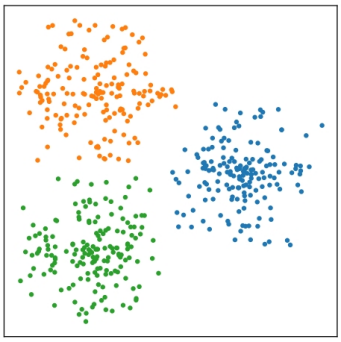
Q4– Écrire une fonction `PlusProcheVoisin(i, k)` donnant l'étiquette la plus fréquente parmi les k plus proches voisins de X[i].



III - Un exemple d'apprentissage non supervisé : l'algorithme des k moyennes

L'idée est désormais d'obtenir une classification sans information préalable sur une partition éventuelle des données. On souhaite que la machine effectue elle-même les regroupements de données «similaires». L'humain se contente de donner le nombre de classes dans la partition.

Par exemple, dans le dessin de gauche, on observe trois groupements de points et l'on souhaite que la machine identifie seule ces trois groupements comme sur le dessin de droite (la différence est liée aux couleurs !) :

Plus précisément :

- on dispose de m données $d_0, d_1, \dots, d_{m-1}$ (des textes, des images, des statistiques, etc.) ;
- chaque donnée est représentée par un vecteur de $\mathbb{R}^N$;
- on dispose d'une fonction de type «distance» permettant de quantifier l'écart entre les données.

La fonction que l'on cherche à définir `kmoyennes(X, k)` a pour arguments l'ensemble `X` des données et un entier `k` $\geqslant 1$, et renvoie un couple `(P,C)` formé d'une liste (ou d'un dictionnaire, ou d'un ensemble) `P` définissant une partition en `k` parties de l'ensemble des données et de la liste `C` des centres de gravité de ces parties (ou seulement `P`).

L'algorithme est le suivant :

- on choisit aléatoirement (ou non !) k données pour initialiser la liste `C` ;
- on définit une liste `P` de k listes vides ;
- pour chaque donnée `d[j]`, on calcule les k distances entre `d[j]` et `C[0]`,...,`C[k-1]` et on place `d[j]` dans la classe correspondant à l'indice `i` réalisant ce minimum (*i.e.* on place `j` dans la liste `P[i]`) ;
- on dispose alors d'une première partition et l'on calcule les centres de gravité de chacune des classes ce qui permet de mettre à jour la liste `C` ;
- on itère ce processus jusqu'à ce que `C` n'évolue plus (ou peu, ou après un nombre fixé d'itérations).

On peut montrer qu'à chaque itération, le classement s'améliore.

Exercice

Reprenons l'exemple de la reconnaissance des chiffres avec 1797 images. On souhaite appliquer l'algorithme avec $k = 10$.
Les images sont stockées dans une liste `X` où chaque `X[i]` représente un vecteur de $\mathbb{R}^{64}$.

On a besoin d'une fonction donnant le barycentre d'un ensemble `s` d'images :

```
def barycentre(s):
    G = [0 for i in range(64)]
    for x in s:
        for i in range(64):
            G[i] += x[i]
    for i in range(64):
        G[i] = G[i]/len(s)
    return G
```

Il est sans doute plus pratique de disposer du module `numpy`.
On suppose que l'on dispose de la fonction `distance` vue précédement.

Écrire une fonction `kmoyennes(X, k)` fondée sur l'algorithme présenté.

Exercice

Dans l'image de gauche ci-dessous (de 385×575 pixels), chaque pixel est un triplet (r, g, b) constitué de trois nombres flottants dans $[0, 1]$ codés sur 32 bits ; l'image est codée par un tableau numpy :

```
>>> img.shape
(385, 575, 3)
```

Q1– Rédiger des instructions donnant le nombre `N` de couleurs différentes dans cette image.

L'objectif est de réduire le nombre de couleurs à 16. On va utiliser l'algorithme avec $k = 16$ afin de regrouper les pixels en classes de couleurs (puis on affecte la valeur moyenne des couleurs de la classe correspondante).

Q2– Écrire une fonction `dist(p,q)` d'arguments deux pixels représentés par des vecteurs de $\mathbb{R}^3$ et qui renvoie leur distance euclidienne.

Q3– Écrire une fonction `initialise(img,k)` d'arguments une image et un entier et qui renvoie un tableau de k cases contenant chacune un pixel tiré au hasard.

Q4– Écrire une fonction `barycentre(img,s)` d'arguments une image et une liste `s` de coordonnées (x, y) et renvoie un pixel égal au barycentre des pixels de l'image dont les coordonnées appartiennent à `s`.

Q5– Écrire une fonction `plusProchePixel(p, l)` qui prend un pixel `p` en argument et une liste `l` de k pixels et qui renvoie l'indice qui minimise la distance entre `p` et ces pixels.

Q6– Écrire une fonction `kmoyennes(img, k)` d'arguments une image et un entier `k` et qui renvoie un tableau s de longueur `k` où chaque composante est un ensemble de coordonnées de pixels obtenu par l'algorithme des k moyennes.

Q7– Écrire une fonction `reduire(img, k)` d'arguments une image et un entier `k` et qui renvoie une nouvelle image dans laquelle seulement k couleurs sont utilisées.