

Chapitre 9

Apprentissage supervisé / non supervisé

Ce chapitre décrit quelques idées simples et efficaces sous-jacentes à l'intelligence artificielle et plus spécifiquement à l'apprentissage automatique (machine-learning). Dans ce domaine :

- L'apprentissage supervisé consiste à prendre une décision sur des données prises individuellement, en s'appuyant sur de bonnes décisions fournies dans un catalogue de données semblables (la base d'apprentissage). Par exemple reconnaître un monument, la tour eiffel, sur une image, distinguer des caractères, chiffres ou lettres, etc.
- L'apprentissage non supervisé consiste à rechercher des structures sous-jacentes, en l'absence de connaissances à priori. Par exemple calculer des similarités entre deux textes, pour détecter du plagiat, détecter des similarité de couleurs de pixels pour la compression d'image avec perte, etc...

Donné un ensemble E , une **partition** de E est une famille $(A_i)_i$ de parties de E telles que :

$$\bigcup A_i = E \quad \text{et} \quad i \neq j \implies A_i \cap A_j = \emptyset$$

- Classifier E consiste à déterminer une partition de E . Bien sur elle doit être pertinente, les éléments d'une même partie étant semblables/analogues selon un certain critère prédéfini, par une distance entre des éléments.
- Classifier/étiqueter des éléments de E consiste alors, donnée une partition de E à déterminer quelles parties A_i les contient ; A_i est la classe ou l'étiquetage.
- Dans l'apprentissage supervisé, on souhaite classer des éléments de E donnés en

les comparant à une base d'apprentissage d'éléments de même type, dont on connaît l'étiquetage.

- Dans l'apprentissage non supervisé, on souhaite classer des éléments de E donnés, en se donnant le nombre de classes, mais sans base d'apprentissage connue à priori.

Dans les deux cas la similarité entre des élément sera défini par une distance, c'est à dire :

$$d : E \times E \longrightarrow \mathbb{R}_+$$

vérifiant :

- $d(x = y) = 0 \iff x = y$ (Séparation)
- $d(x, y) = d(y, x)$ (Symétrie)
- $d(x, y) \leq d(x, z) + d(z, y)$ (Inégalité triangulaire)

Deux éléments seront d'autant plus semblables que leur distance est faible.

Pour l'apprentissage supervisé nous allons voir l'algorithme des *k plus proches voisins* (*k-nearest neighbors*).

Pour l'apprentissage non supervisé nous allons voir l'algorithme des *k-moyennes* (*k-means*).

1 Apprentissage supervisé

1.1 Exemple concret

Pour la reconnaissance automatique du code postal saisi sur l'adresse d'une enveloppe postale, il s'agit de classer les 5 chiffres manuscrits saisis dans les cases appropriés dans les 10 classes, de 0 à 9, de la numérotation.

5	0	4	1	9	2	1	3	1	4
3	5	3	6	1	7	2	8	6	9
4	0	9	1	1	2	4	3	2	7
3	8	6	9	0	5	6	0	7	6
1	8	7	9	3	9	8	5	9	3
3	0	7	4	9	8	0	9	4	1
4	4	6	0	4	5	6	1	0	0
1	7	1	6	3	0	2	1	1	7
9	0	2	6	7	8	3	9	0	4
6	7	4	6	8	0	7	8	3	1

7

On dispose par exemple d'une base d'apprentissage de 100 chiffres manuscrits, dont on connaît la valeur cardinale 0,1,...,9 (tableau ici à gauche¹). Donnée un caractère manuscrit (ici à droite ce qui s'apparente à un 7) on détermine sa valeur en le comparant à la base.

Dans l'algorithme des k plus proches voisins on cherche les k caractères du catalogue les plus proches du caractère manuscrit. Sa valeur est alors la valeur majoritaire parmi ces k caractères.

Si par exemple $k = 10$, et que les k plus proches voisins correspondent à cinq 7, trois 1 et deux 9, le chiffre manuscrit est étiqueté par 7.

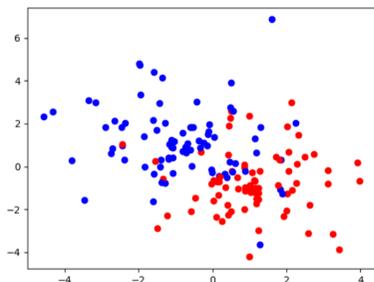
Le principe est simple, et ne dépend que du choix d'une distance entre caractères ; chaque caractère est représenté par un tableau de $n \times n$ pixels, au format bitmap, chaque pixel prenant la valeur 0 (noir) ou 1 (blanc). On peut par exemple considérer la distance :

$$d(C, C') = \sum_{1 \leq i, j \leq n} |C[i, j] - C'[i, j]|$$

qui compte le nombre de pixels différents. Mais on peut imaginer d'autres distances, dont il s'agira de tester la fiabilité ensuite.

1.2 Algorithme des k plus proches voisins

Afin d'implémenter l'algorithme des k plus proches voisins, simplifions le problème : E sera l'ensemble des points d'un domaine du plan, la distance sera la distance euclidienne, il y aura deux classes d'objets, les points bleus et les points rouges². On a représenté dans le carré $[-4, 4]^2$ du plan, la base d'apprentissage, un ensemble de 80 points bleus et de 80 points rouges ; ils ont été obtenus grâce à deux lois gaussiennes centrées en $(1, -1)$ et $(-1, 1)$ et de même rayon.



On souhaite partitionner le carré en deux zones, bleus ou rouges selon l'algorithme des k plus proches voisins. Selon toute probabilité, le partitionnement optimal devrait être proche de celui délimité par la première bissectrice $y = x$, en bleu au dessus et rouge au-dessous.

La base d'apprentissage sera constituée d'une liste B de couples (pos , coul) où :

- pos est un couple (x, y) donnant les coordonnées du point dans le repère du plan,
- coul est la chaîne 'bleu' ou 'rouge' selon la couleur du point.

```
>>> B
[[ (0.0067, 1.2345), 'bleu' ], ... , ((1.1235, 2.6765), 'rouge' )]
```

On commence par définir la fonction distance (ici la distance euclidienne) :

```
def d(p1, p2):
    """Distance euclidienne entre deux points p1, p2 du plan"""
    return ((p1[0]-p2[0])**2 + (p1[1]-p2[1])**2)**.5
```

L'algorithme des k plus proches voisins est implémenté dans la fonction qui suit ; elle prend en argument la base B , un point p du plan, donné par un couple de coordonnées (x, y) , l'entier k , et renvoie la chaîne 'bleu' ou 'rouge' :

```
def kPlusProchesVoisins(B, p, k):
    # tri de B selon les distances à p croissantes
    V = sorted(B, key = lambda x : d(x[0], p))[:k]
    # restriction aux k plus proches :
    V = V[:k]
    bleu, rouge = 0, 0
    for voisin in V:
        if voisin[1] == 'bleu':
            bleu += 1
        elif voisin[1] == 'rouge':
            rouge += 1
    if bleu >= rouge:
        return 'bleu'
    else:
        return 'rouge'
```

Ici, en présence de deux classes, afin d'éviter les cas d'égalités on pourra choisir k impair.

1. Les images sont issues d'une base mise à disposition par Yann le Cun à l'adresse : <http://yann.lecun.com/exdb/mnist/>

2. L'exemple présenté ici est issu de l'ouvrage "Apprentissage statistique" de G.Dreyfus, J.-M. Martinez, M. Samuelides, M.B. Gordon, F.Badra., S.Thiria aux éditions Eyrolles, ISBN 978-2-212-12229-9

• Algorithme des k -plus proches voisins :

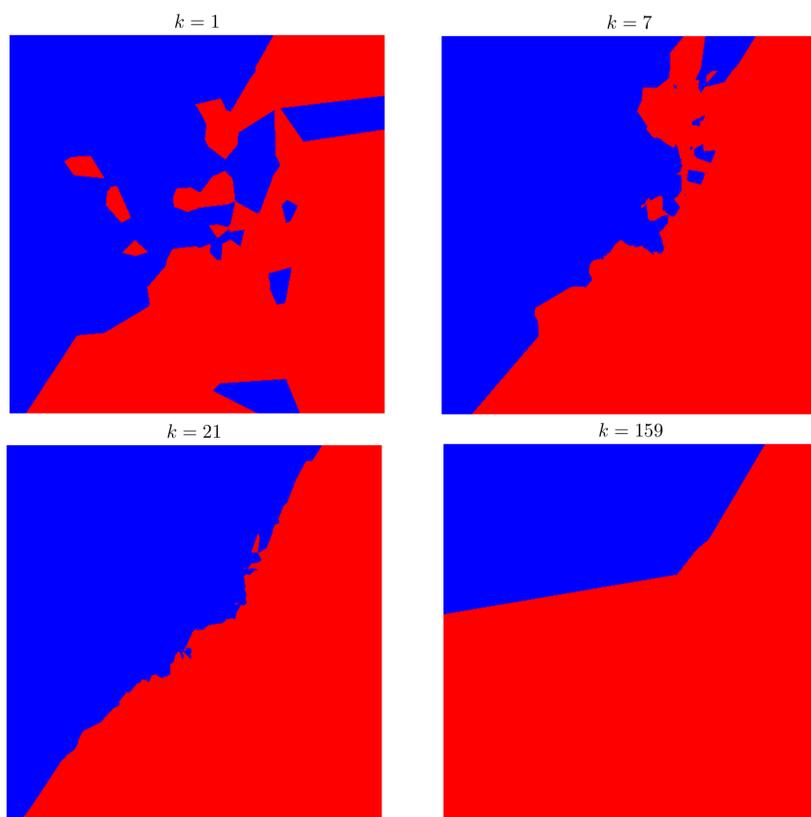
Donnés :

- Un ensemble E , une distance d sur E , et des classes de E , C_1, \dots, C_n .
- Une base B de n points de E dont on connaît la classe C_i .
- Un entier strictement positif $k \leq n$.
- Un point p de E .

On renvoie une estimation de la classe du point p :

- En déterminant les k voisins de p dans B les plus proches pour d .
- En renvoyant la classe majoritaire parmi ces k voisins.

On a appliqué cet algorithme à tous les points du carré $[-2, 2]^2$ et tracé les partitions bleus/rouges obtenues pour différentes valeurs de k , 1, 7, 21, 159 (bien sûr, vu qu'il y a 160 points dans la base d'apprentissage, nécessairement $k \leq 160$). On obtient :



- Pour $k = 1$, tous les points de la base ont conservé leur couleur : c'est normal puis-

qu'ils sont leur 1-plus proche voisin. La couleur d'un point est celle du point de la base le plus proche.

- Plus k augmente plus la frontière devient régulière et le résultat proche du résultat optimal.
- Mais pour k trop grand la solution n'est pas très bonne ; les points sont majoritairement rouges à cause de la plus grande dispersion de certains points bleus (dont celui tout en haut à droite) les points rouges proches devenant alors plus souvent majoritaires.
- Nous qui connaissons la solution, constatons que c'est pour la valeur $k = 21$ que le résultat est le plus probant.

Déterminer quelle valeur de k considérer est alors un problème compliqué.

1.3 Test ; matrice de confusion

Afin de tester la pertinence de l'algorithme, que ce soit pour le choix de la distance d ou de l'entier k , on peut procéder à des tests, en se donnant outre la base d'apprentissage une autre base, de test, constituée de points dont on connaît la classe à priori. On applique l'algorithme à tous les points de cette base test, et on constitue la matrice de confusion, définie par :

- C'est une matrice carrée (n, n) où n est le nombre de classe.
- L'élément ligne i colonne j est le nombre d'éléments de la base test dans la classe C_i que l'algorithme aura détecté dans la classe C_j .

Ainsi le test est d'autant plus probant que la matrice de confusion est proche d'être diagonale ; une matrice diagonale représentant un résultat parfait.

Par exemple, dans notre implémentation des deux points rouges/bleus, la matrice de confusion est une matrice 2×2

$$\begin{pmatrix} \text{VP} & \text{FN} \\ \text{FP} & \text{VN} \end{pmatrix}$$

constituée de :

- VP : nombre de vrais positifs : les points rouges reconnus comme tels,
- VN : nombre de vrais négatifs : les points bleus reconnus comme tels,
- FN : nombre de faux négatifs : les points rouges reconnus comme bleus,
- FP : nombre de faux positifs ; les points bleus reconnus comme rouges.

Les résultats obtenus sur une base test de 100 points obtenus à l'aide des mêmes distributions gaussiennes donne pour matrice de confusions M_k , selon les valeurs de k :

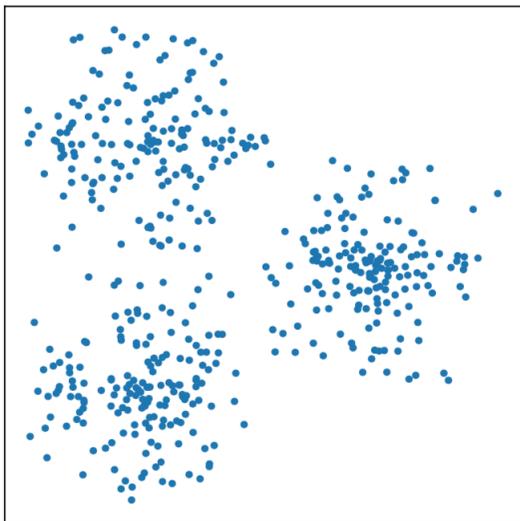
$$M_1 = \begin{pmatrix} 38 & 12 \\ 10 & 40 \end{pmatrix} \quad M_7 = \begin{pmatrix} 42 & 8 \\ 11 & 39 \end{pmatrix} \quad M_{21} = \begin{pmatrix} 43 & 7 \\ 8 & 42 \end{pmatrix} \quad M_{159} = \begin{pmatrix} 47 & 3 \\ 20 & 30 \end{pmatrix}$$

C'est bien pour $k = 21$ que la matrice de confusion est la plus proche d'une matrice diagonale (on pourrait prendre pour critère : $\sum_{i \neq j} |M[i, j]|$).

2 Apprentissage non supervisé

2.1 Partitionnement de données

Un problème plus complexe consiste à déterminer une structure sous-jacente à une ensemble de données sans s'aider d'une base d'apprentissage. Il s'agira pour l'ordinateur, de regrouper les données en des classes, selon leur similarité définie par une distance dans l'ensemble E des données possibles. Par exemple dans le plan muni de la distance euclidienne, l'oeil humain tend à discerner trois parties dans le nuage de points suivant :



Comment partitionner les points du nuage en 3 parties? Soit P l'ensemble de ces points, $k = 3$, il s'agit de déterminer une partition C_1, C_2, C_3 de P qui minimise :

$$\sum_{i=1}^k \sum_{p \in C_i} \|p - m_i\|$$

où m_i désigne l'isobarycentre de C_i .

Une résolution naïve est impossible en pratique, puisque le nombre de partitions en 3 parties est exponentiel égal à $3^{\#P}$; même rien que le nombre de partitions en trois

parties de tailles égales d'un ensemble de cardinal $\#P = 3n$ donnerait :

$$\binom{3n}{n} \times \binom{2n}{n} = \frac{(3n)!(2n)!}{(n!)^2(2n)!n!} = \frac{(3n)!}{(n!)^3} \sim \frac{\left(\frac{3n}{e}\right)^{3n} \sqrt{6\pi n}}{\left(\frac{n}{e}\right)^{3n} 2\pi n \sqrt{2\pi n}} \sim \frac{\sqrt{3}}{2\pi} \times \frac{27^n}{n} = \frac{3\sqrt{3}}{2\pi} \times \frac{3^{\#P}}{\#P}$$

2.2 Algorithme des k -moyennes

Le calcul d'une classification optimale serait trop couteux. L'algorithme des k -moyennes propose un algorithme glouton : il retournera une bonne solution mais rien n'assure de son caractère optimal; on risque de trouver un minimum local qui n'est pas global.

Algorithme des k -moyennes

Donnés :

- Un ensemble E , une distance d de E ,
- Un ensemble P de points de E ,
- Un entiers strictement positif $k < \#E$,

On renvoie une partition C_1, \dots, C_k de P en :

- Répartir aléatoirement un point de P dans classe C_1, \dots, C_k ,
- TANT QUE l'on change quelque chose :
 - Calculer les isobarycentres m_1, \dots, m_k de C_1, \dots, C_k
 - POUR chaque point x de P :
 - Calculer i tel que $\|x - m_i\|$ soit minimal
 - Déplacer x dans C_i
- Renvoyer C_1, \dots, C_k .

L'algorithme bien que de complexité exponentielle dans le pire des cas, s'avère de complexité polynomiale en moyenne; il présente en fait de très bonnes performances qui en font un algorithme très utilisé.

Implémentons l'algorithme dans notre contexte de points du plan.

```
def d(p1, p2):
    """Distance euclidienne entre deux points p1, p2 du plan"""
    return ((p1[0]-p2[0])**2 + (p1[1]-p2[1])**2)**.5
```

```
def barycentre(Ci):
    x, y = 0 0
    for e in Ci:
        x += e[0]
        y += e[1]
    return (x/len(Ci), y/ len(Ci))
```

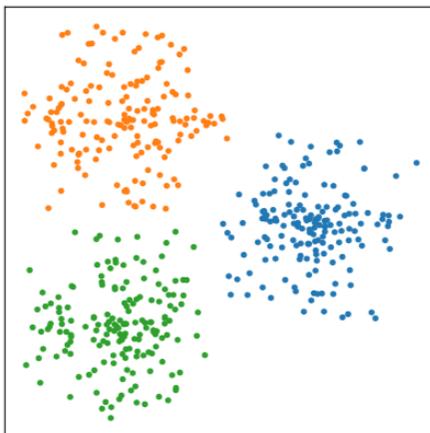
```

from random import randrange

def kMoyennes(P, k):
    # Initialisation avec k points aléatoires :
    m = [ P[randrange(len(P))] for _ in range(k) ]
    change = True
    while change:
        change = False
        C = [ [] for _ in range(k) ] # futures classes
        for x in P:
            imin, dmin = 0, float("inf") # + infini
            for i in range(k): # recherche minimum
                if d(x, m[i]) < dmin:
                    imin, dmin = i, d(x,m[i])
            C[imin].append(x) # placement de x dans sa classe
        m1 = [ barycentre(C[i]) for i in range(k) ]
        for i in range(k): # Test changement
            if m[i] != m1[i]:
                change = True
                break
        m = m1
    return C

```

Le test de changement ne se fait que sur les barycentres. Sauf cas pathologique c'est suffisant. Avec $k = 3$ on a obtenu pour notre nuage de points la partition représentée en couleur ci-dessous :



2.3 Application à la compression d'images avec perte

Une image bitmap RGB en couleur est représentée essentiellement par un tableau de ses pixels qui sont des triplets (r, g, b) des niveaux des 3 couleurs primaire, chacun évalué par un entier entre 0 et 255. Cela fait une palette de $256^3 = 2^{24}$ couleurs, chaque pixel devant alors être représenté sur 24 bits.

Un principe simple de compression avec perte, consiste alors à réduire la palette de couleurs à 2^p nuances avec $p < 24$. Le taux de compression est alors de $\frac{p}{24}$ puisqu'il ne faut plus que p bits pour stocker chaque pixel.

L'idée consiste alors à regrouper les couleurs en 2^p teintes de couleurs, à l'aide de l'algorithme des k -moyennes et avec $k = 2^p$; chaque pixel voyant sa couleur changer en l'une des 2^p couleurs la plus proche (pour la distance euclidienne de \mathbb{R}^3 par exemple).

Voici ce que l'on obtiendrait à partir de l'image originale, avec $k = 32, 16, 8$.

