ALGORITHMES D'APPRENTISSAGE

B. Landelle

Table des matières

I Algorithme des k plus proches voisins **2**
1 Présentation 2
2 Classification par apprentissage supervisé 3
3 Reconnaissance de caractères 5

II Algorithme des k-moyennes **8**
1 Présentation 8
2 Classification par apprentissage non supervisé 8
3 Application à la compression d'image 10

III Le perceptron **11**
1 Présentation 11
2 Implémentation d'un perceptron 14
3 Perceptron multi-classes 15
4 Implémentation d'un perceptron multi-classes 18

I Algorithme des k plus proches voisins

1 Présentation

L'algorithme des k *plus proches voisins* (*knn* pour k *nearest neighbors*) avec k un entier non nul est un algorithme d'apprentissage *supervisé*, à savoir un algorithme qui apprend automatiquement à partir d'une base d'apprentissage.

L'algorithme de k plus proches voisins est présenté dans ce cours en tant qu'algorithme de *classification*, c'est-à-dire pour classer une observation parmi des catégories existantes. On peut aussi utiliser cet algorithme à des fins de régression (prédiction d'une variable quantitative) mais cet aspect ne sera pas abordé ici.

On peut considérer la situation exposée sur la figure 1 où sont représentés des points de $\mathbb{R}^2$ appartenant à trois catégories distinctes. La proximité, au sens de la distance euclidienne dans $\mathbb{R}^2$, permet d'identifier les trois amas de points correspondant à chacune des catégories.

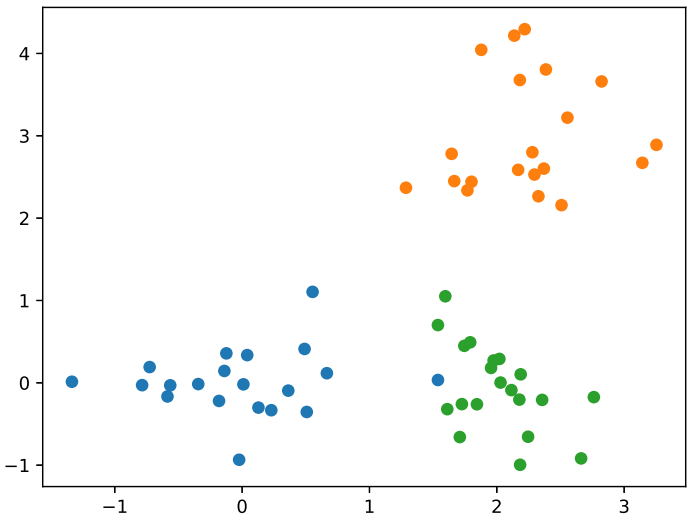

FIGURE 1 – Nuage de points issus de trois catégories distinctes

L'objectif de l'algorithme des k plus proches voisins est, étant donné un nouveau point $\mathrm{X} = (x, y) \in \mathbb{R}^2$, d'identifier les k éléments les plus proches dans la base d'apprentissage et d'en déduire la catégorie à laquelle appartient X.

Définition 1. *Soient k, d et C des entiers non nuls et $\mathscr{A} = (\mathrm{X}_i, c_i)_{1 \leqslant i \leqslant n}$ une base d'apprentissage avec $\mathrm{X}_i \in \mathbb{R}^d$ et $c_i \in [\![1 ; \mathrm{C}]\!]$ la classe (ou catégorie) du point X_i pour tout $i \in [\![1 ; n]\!]$. L'algorithme des k* plus proches voisins *associe à un nouveau point $\mathrm{X} \in \mathbb{R}^d$ une classe majoritaire des k éléments de la famille $(\mathrm{X}_i)_{1 \leqslant i \leqslant n}$ les plus proches de X au sens de la métrique euclidienne dans $\mathbb{R}^d$.*

Le choix de k est un problème à part entière. Des techniques statistiques dites de *validation croisée* sont parfois utilisées pour effectuer ce choix mais ces considérations dépassent le cadre de ce cours. Dans la littérature spécialisée, il est recommandé en général de faire le choix d'un k entier impair et parfois, on trouve aussi la règle heuristique $k \simeq \sqrt{n}$.

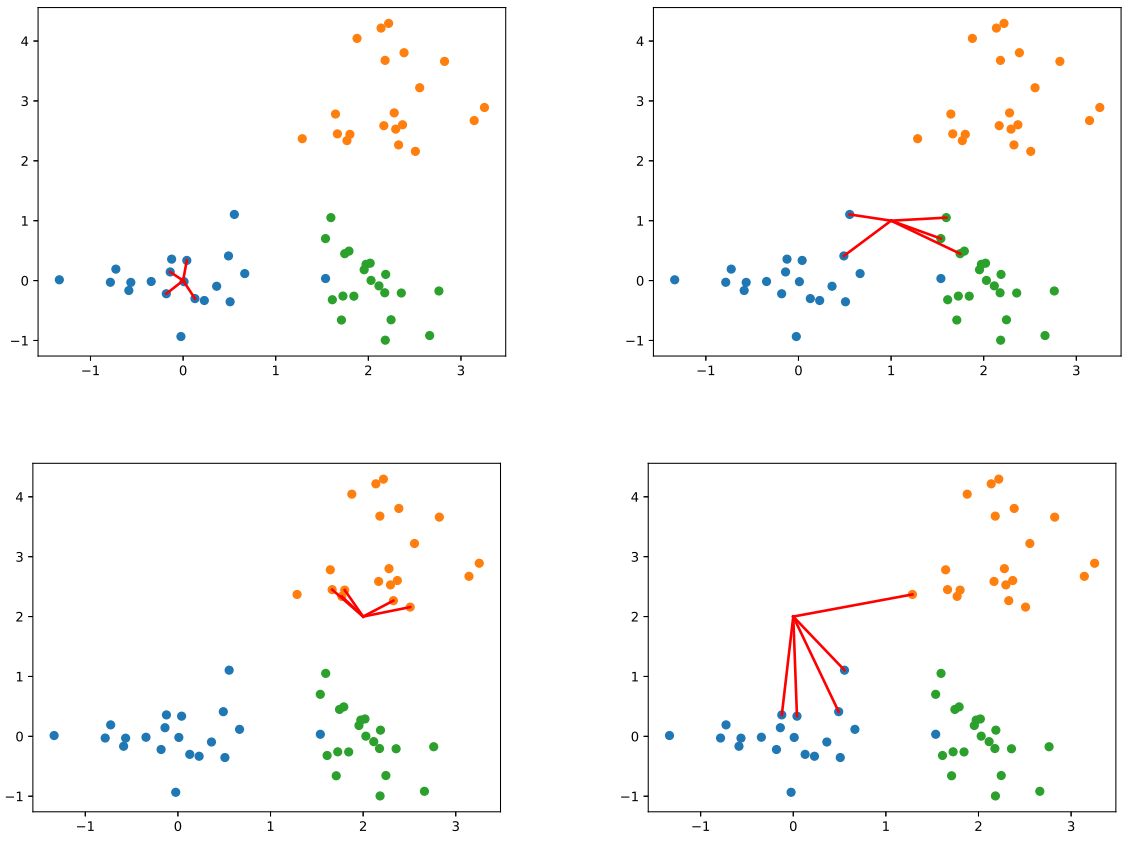

FIGURE 2 – Observation des 5 plus proches voisins pour différents points

2 Classification par apprentissage supervisé

On reprend les notations de la définition 1 et on note $\|\cdot\|$ la norme euclidienne sur $\mathbb{R}^d$.

L'algorithme de classification supervisée des k plus proches voisins consiste en les étapes suivantes :

- Saisie d'un nouveau point $\mathrm{X} \in \mathbb{R}^d$;
- Calculer $d_i = \|\mathrm{X} - \mathrm{X}_i\|$ pour tout $i \in [\![1\,;\, n]\!]$;
- Trier par ordre croissant la liste $[d_1, \ldots, d_n]$ en $\left[d_{\sigma(1)}, d_{\sigma(2)}, \ldots, d_{\sigma(n)}\right]$ avec σ permutation de $[\![1\,;\, n]\!]$ telle que $d_{\sigma(1)} \leqslant d_{\sigma(2)} \leqslant \ldots \leqslant d_{\sigma(n)}$;
- Choix d'une classe majoritaire dans la liste $\left[c_{\sigma(1)}, \ldots, c_{\sigma(k)}\right]$.

Pour l'implémentation, on importe les modules :

```
import numpy as np, numpy.linalg as alg
```

Les points de $\mathbb{R}^d$ sont codés par des tableaux de type `ndarray` et on utilise la fonction `alg.norm` pour le calcul de la norme euclidienne sur $\mathbb{R}^d$.

La fonction `majo(L)` d'argument `L` une liste non vide renvoie un élément majoritaire de celle-ci (élément pas nécessairement unique). La fonction `sorted(L,key=lambda u:u[1])` d'argument `L` une liste de couples renvoie la liste triée par ordre croissant selon le deuxième argument du couple. On implémente enfin l'algorithme des k plus proches voisins :

```
def knn(k,X,A):
    """knn(k:int,pt:list,pop:list)->int
    k : argument k dans l'algorithme knn
    X : nouveau point
    A : base d'apprentissage,
        liste de sous-listes où A[c] est la classe c de A
        chaque sous-liste A[c] contient les points de R^d
    Renvoie la classe de X selon algorithme knn"""
    res=[]
    C=len(A)
    for c in range(C):
        for X_i in A[c]:
            res.append([c,alg.norm(X-X_i)])
    res=sorted(res,key=lambda u:u[1])
    return majo([x[0] for x in res[:k]])
```

On conserve les notations de la définition 1.

Définition 2. *Soit* $\mathscr{T} = (\mathrm{Y}_\ell, e_\ell)_{1\leqslant \ell \leqslant p}$ *une base de test avec* $\mathrm{Y}_\ell \in \mathbb{R}^d$ *et* $e_\ell \in [\![1\,;\mathrm{C}]\!]$ *la classe du point* Y_ℓ *pour tout* $\ell \in [\![1\,; p]\!]$. *La* matrice de confusion *est une matrice de* $\mathrm{M} = \left(m_{i,j}\right)_{1\leqslant i,j\leqslant \mathrm{C}} \in \mathscr{M}_{\mathrm{C}}(\mathbb{R})$ *où* $m_{i,j}$ *désigne le nombre d'éléments de* $\mathscr{T}$ *de classe* i *dont la classe estimée par l'algorithme des* k *plus proches voisins est* j, *c'est-à-dire*

$$\forall (i,j) \in [\![1\,;\mathrm{C}]\!]^2 \qquad m_{i,j} = \sum_{\ell=1}^{p} \mathbb{1}_{\{[i,j]\}}\left([e_\ell, \mathbf{knn}(k, \mathrm{Y}_\ell, \mathscr{A})]\right)$$

Simulation :

On considère comme base d'apprentissage la famille de couples dont la représentation est donnée en figure 1 puis on génère de nouveaux points pour une base de test.

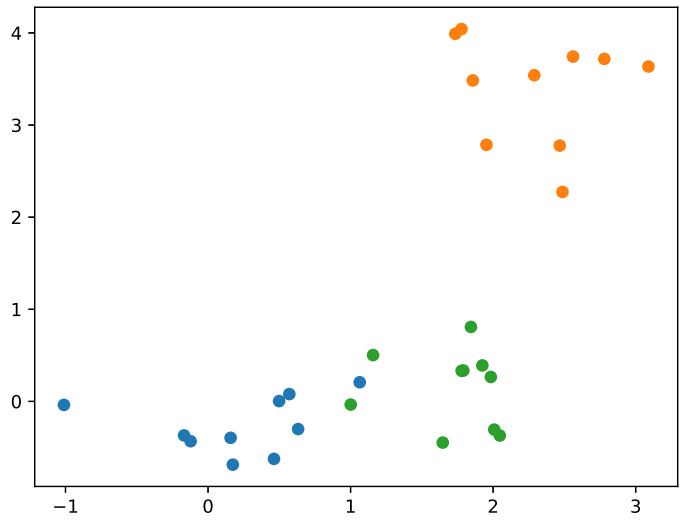

FIGURE 3 – Nuage de points de la base de test

On obtient comme matrice de confusion :

```
[[10  0  0]
 [ 0 10  0]
 [ 1  0  9]]
```

Si la matrice de confusion est diagonale, cela signifie que la classification a été parfaitement réussie. Dans le cas contraire, les termes non nuls hors de la diagonale sont les comptages des erreurs de classification.

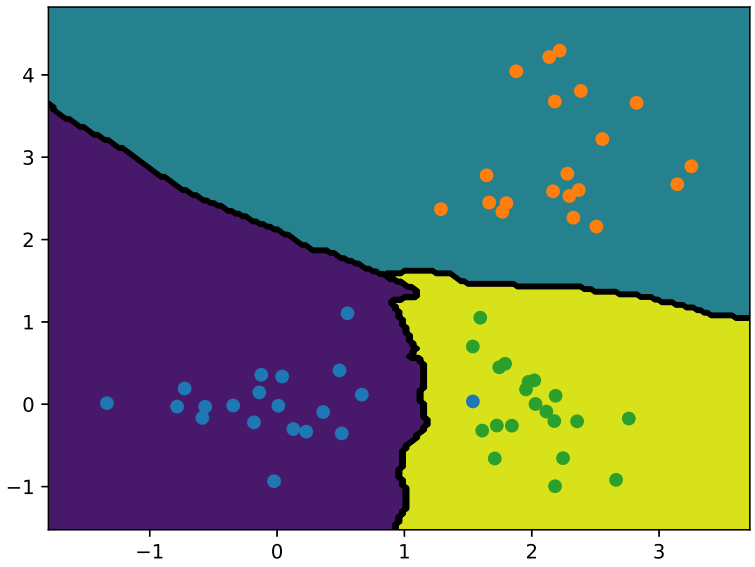

FIGURE 4 – Zones délimitées par les 5 plus proches voisins

3 Reconnaissance de caractères

On peut utiliser l'algorithme des k plus proches voisins pour effectuer par exemple de la reconnaissance de caractères. On considère une base d'apprentissage $\mathscr{A} = (\mathrm{X}_i, c_i)_{1 \leqslant i \leqslant n}$ où X_i désigne l'image d'un chiffre et c_i désigne le chiffre lui-même dans $[\![0\,;\, 9]\!]$ pour tout $i \in [\![1\,;\, n]\!]$. Ainsi, pour chaque image de cette base d'apprentissage, le chiffre représenté sur l'image correspond à sa catégorie.

On utilise comme base d'apprentissage la base de donnée MNIST [1] accessible en ligne.

1. Base de données de chiffres écrits à la main `http://yann.lecun.com/exdb/mnist/`

FIGURE 5 – Base d'apprentissage MNIST

On importe les bibliothèques :

```
import idx2numpy
import matplotlib.pyplot as plt
```

puis on exécute le code :

```
imagefile = 'train-images.idx3-ubyte'
imagearray = idx2numpy.convert_from_file(imagefile)
labelfile=  'train-labels.idx1-ubyte'
labelarray = idx2numpy.convert_from_file(labelfile)
```

La variable `imagearray` est un tableau contenant 60 000 images de taille 28×28 et la variable `labelarray` est un tableau contenant les chiffres représentés sur les images de `imagearray`. Ainsi, pour `i` un entier dans $[\![0\,;\,59\,999]\!]$, l'appel `imagearray[i]` renvoie un tableau codant une image de taille 28×28 et `labelarray[i]` renvoie le chiffre représenté sur l'image. Par exemple, pour afficher l'image d'indice 4 dans base MNIST et le chiffre qu'elle représente, on saisit :

```
print("Image MNIST indice 4 :")
print(labelarray[4])
plt.imshow(imagearray[4], cmap=plt.cm.binary)
plt.show()
```

On obtient :

```
Image MNIST indice 4 :
9
```

et on observe :

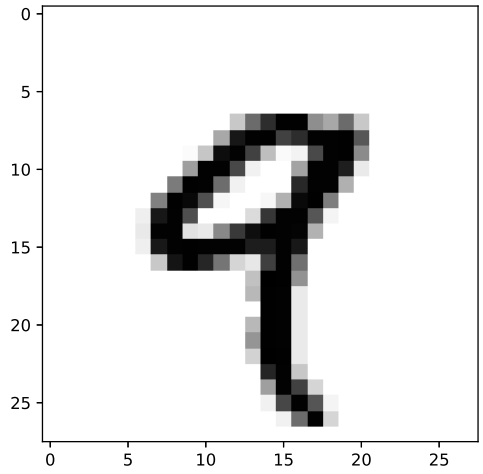

FIGURE 6 – Image d'indice 4 extraite de la base MNIST

En faisant croître la taille de la base d'apprentissage, on observe l'amélioration du taux de reconnaissance de caractères. Ainsi, pour $k = 11$, en testant les 100 dernières images de la base MNIST avec une base d'apprentissage constituée des n premières images de cette même base pour n variant entre 50 et 5000, on obtient l'évolution du taux de reconnaissance illustrée par la figure suivante :

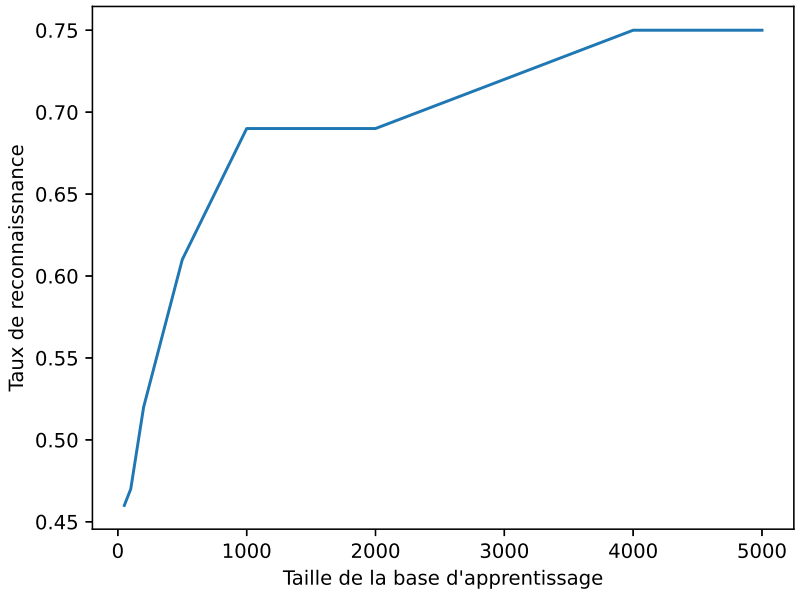

FIGURE 7 – Taux de reconnaissance en fonction de la taille d'apprentissage

La croissance du taux de reconnaissance avec la taille de la base d'apprentissage est conforme à l'intuition d'un apprentissage. Il s'agit d'une première approche de reconnaissance de caractères qui, bien que naïve, montre déjà une certaine efficacité. En particulier, le choix de la métrique euclidienne est un choix arbitraire dont rien ne dit qu'il soit le plus adapté.

D'autres stratégies existent, notamment celles qui s'appuient sur des *réseaux de neurones artificiels* et présentent des performances exceptionnelles.

II Algorithme des k-moyennes

Comme pour les k plus proches voisins, l'espace $\mathbb{R}^d$ avec d entier non nul est muni de la norme euclidienne notée $\|\cdot\|$.

1 Présentation

L'algorithme des *k-moyennes* (*k-mean*) avec k un entier non nul est un algorithme d'apprentissage *non supervisé*, à savoir un algorithme qui apprend automatiquement, directement à partir des données fournies en entrée, à les partitionner en k classes.

Définition 3. *L'algorithme des* k-moyennes *est une heuristique cherchant à partitionner un ensemble fini* S *de points de* $\mathbb{R}^d$ *en* k *sous-ensembles* $\mathrm{S}_1, \mathrm{S}_2, \ldots, \mathrm{S}_k$ *appelés* clusters *tels que la quantité*

$$\mathrm{V}(\mathrm{S}_1, \mathrm{S}_2, \ldots, \mathrm{S}_k) = \sum_{i=1}^{k} \sum_{x \in \mathrm{S}_i} \|x - \bar{x_i}\|^2 \quad \textit{avec} \quad \bar{x_i} = \frac{1}{|\mathrm{S}_i|} \sum_{x \in \mathrm{S}_i} x$$

soit minimale. Cette quantité $\mathrm{V}(\mathrm{S}_1, \ldots, \mathrm{S}_k)$ *est appelée* variance intra-classe *et les isobarycentres* $\bar{x_i}$ *de chaque cluster sont appelés* centroïdes.

Cet algorithme d'optimisation est une *heuristique*, à savoir une méthode qui fournit efficacement une réponse que l'on espère raisonnable mais sans garantie qu'elle soit une solution optimale. On peut démontrer la terminaison de l'algorithme des k-moyennes. Ainsi, l'algorithme converge vers une partition limite qui est un minimum local de la fonction V, pas nécessairement un minimum global.

Comme pour les k plus proches voisins, le choix de k est ici encore un problème à part entière abordé dans la littérature spécialisée sur la base de critères statistiques.

2 Classification par apprentissage non supervisé

On reprend les notations de la définition 3.

Les états successifs des clusters et de leurs centroïdes dans l'algorithme des k-moyennes sont notés :

$$\forall \ell = 0, 1, 2 \ldots \qquad \mathrm{S}_1^{(\ell)}, \mathrm{S}_2^{(\ell)}, \ldots, \mathrm{S}_k^{(\ell)} \quad \text{et} \quad \bar{x_1}^{(\ell)}, \bar{x_2}^{(\ell)}, \ldots, \bar{x_k}^{(\ell)}$$

L'algorithme de classification non supervisé des k-moyennes consiste en les étapes suivantes :

- Initialisation (état $\ell = 0$) : chaque point de S est placé au hasard dans $\mathrm{S}_1^{(0)}, \mathrm{S}_2^{(0)}, \ldots, \mathrm{S}_k^{(0)}$, la variable $\mathrm{V}^{(0)}$ reçoît $+\infty$ et une variable booléenne de test de décroissance reçoit `True` ;
- Tant que le test de décroissance est vrai (on passe de l'état ℓ à $\ell + 1$) :
  - On calcule les centroïdes à savoir les $\bar{x}_i{}^{(\ell)} = \dfrac{1}{\left|\mathrm{S}_i^{(\ell)}\right|} \displaystyle\sum_{x \in \mathrm{S}_i^{(\ell)}} x$ pour $i \in [\![1\,;\, k]\!]$;
  - On calcule une nouvelle partition $\mathrm{S}_1^{(\ell+1)}, \mathrm{S}_2^{(\ell+1)}, \ldots, \mathrm{S}_k^{(\ell+1)}$ telle que chaque point $x \in \mathrm{S}$ est affecté dans $\mathrm{S}_i^{(\ell+1)}$ avec i indice pour lequel $\|x - \bar{x}_i{}^{(\ell)}\|$ est minimale ;
  - On calcule la nouvelle valeur de $\mathrm{V}^{(\ell+1)}$ variance intra-classe et la variable booléenne de test de décroissance reçoit la valeur de $\mathrm{V}^{(\ell+1)} < \mathrm{V}^{(\ell)}$.

Pour l'implémentation, on importe les modules :

```
import numpy as np, numpy.random as rd, numpy.linalg as alg
```

La variable `pop` est une liste de points au format `ndarray` et on utilise la fonction `alg.norm` pour le calcul de la norme euclidienne sur $\mathbb{R}^d$.

```
# Initialisation
decrease=True
V=float('inf')

# Partition initiale
S=[[] for i in range(k)]
for pt in pop:
    tirage=rd.randint(0,k)
    S[tirage].append(pt)
centr=[0]*k

# Tant que la variance intra-classe décroît
while decrease:
    # Calcul des centroïdes
    for i in range(k):
        centr[i]=sum(S[i])/len(S[i])
    # Mise à jour de la partition
    S=[[] for i in range(k)]
    V_aux=0
    for pt in pop:
        dmin=float('inf')
        for i in range(k):
            d=alg.norm(pt-centr[i])
            if d<dmin:
                dmin,ind=d,i
        S[ind].append(pt)
        V_aux+=dmin**2
    if V_aux<V:
        V=V_aux
    else:
        decrease=False
```

Comme avec les k plus proches voisins, on considère la situation d'une liste de points de $\mathbb{R}^2$. On souhaite donc répartir ces points en 3 classes de sorte qu'au sein d'une même classe, les points soient proches les uns des autres.

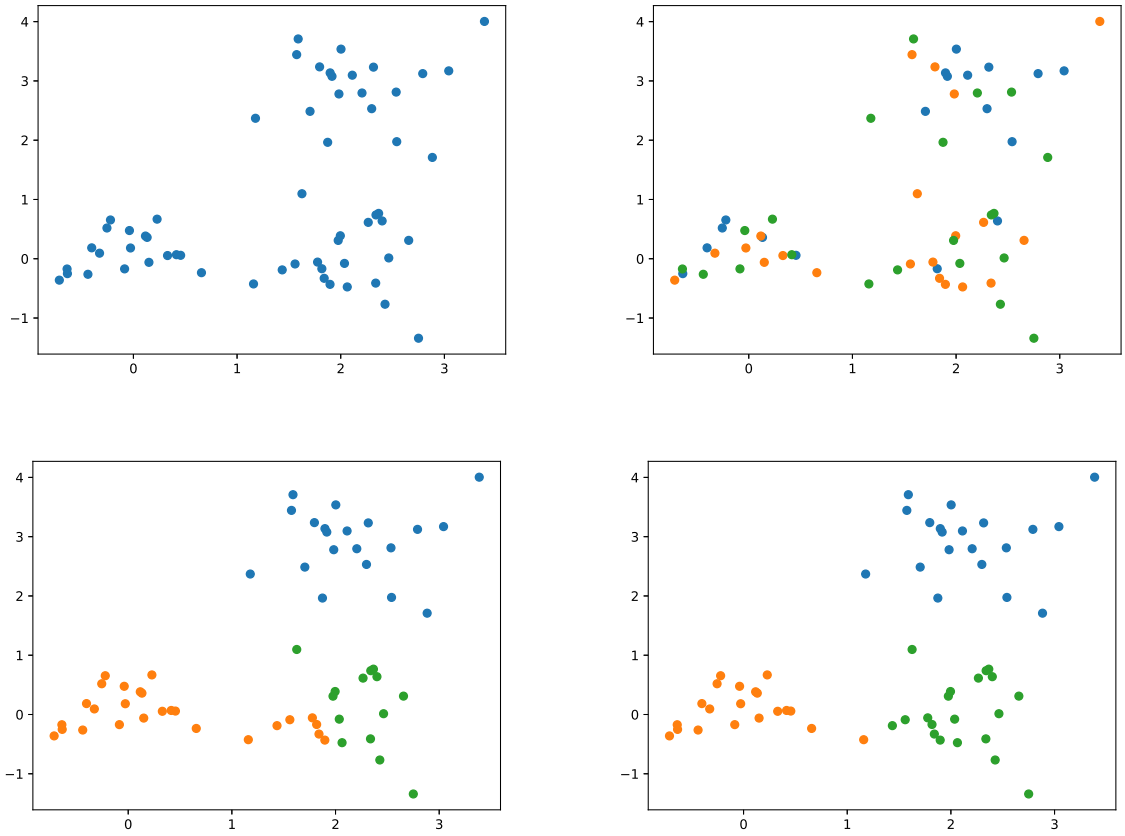

FIGURE 8 – Étapes de la classification par k-moyennes avec $k = 3$

La première figure comporte l'ensemble des points mélangés, sans classification. On observe ensuite la partition initiale qui correspond à une affectation aléatoire des points dans 3 classes. Puis, en seulement deux itérations, l'algorithme des k-moyennes converge vers une partition des points tout à fait raisonnable.

3 Application à la compression d'image

L'algorithme des k-moyennes fournit également, pour chaque cluster S_i un représentant moyen qu'est le centroïde $\bar{x}_i$.

On peut donc envisager l'utilisation de cet algorithme pour compresser une image. En effet, on peut importer une image en couleur sous forme tableau `ndarray` de taille $L \times C \times 3$ avec L le nombre de lignes, C et le nombre de colonnes et les 3 composantes de la dernière dimension codent l'intensité des couleurs rouge, vert et bleue dont la combinaison permet le codage de la plupart des couleurs perçus par l'œil humain. Il suffit ensuite de choisir un nombre k de couleurs et d'appliquer l'algorithme des k-moyennes sur le tableau. Les centroïdes fournissent les candidats pour constituer la palette des couleurs retenues pour la compression.

FIGURE 9 – Compression d'image par k-moyennes avec $k = 8$, Château Sarran, Antony

FIGURE 10 – Palette des 8 couleurs retenues pour la compression

III Le perceptron

Dans ce qui suit, les lettres d et n désignent des entiers non nuls. L'espace $\mathbb{R}^{d+1}$ est muni de son produit scalaire canonique.

1 Présentation

Le *perceptron* est le premier modèle de neurone artificiel inventé en 1957 par Franck Rosenblatt [2]. Il a été conçu pour imiter le fonctionnement d'un réseau de neurones biologiques.

Le perceptron à d entrées (dans la littérature spécialisée, on lit parfois d « neurones » en entrée, formulation qui peut sembler abusive) auxquelles on ajoute un biais égal à 1 possède une unique couche, un seul neurone, qui effectue le calcul de la combinaison linéaire

$$\langle \mathrm{W}, \mathrm{X} \rangle = w_0 + \sum_{i=1}^{d} w_i x_i$$

avec $\mathrm{X} = (1, x_1, \ldots, x_d)$ et $\mathrm{W} = (w_0, \ldots, w_d)$ où les w_i désignent des poids réels.

2. Franck Rosenblatt, 1928-1971, psychologue américain.

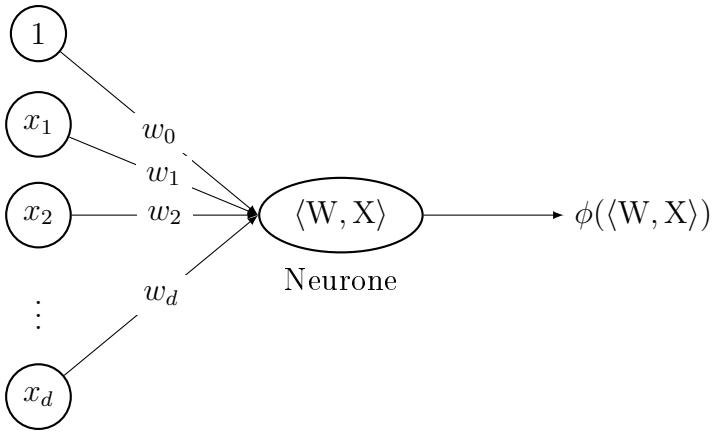

FIGURE 11 – Perceptron à d entrées

Le perceptron est présenté dans ce cours en tant qu'algorithme de de classification. La fonction ϕ considérée ici est la fonction signe, à savoir

$$\forall t \in \mathbb{R} \qquad \phi(t) = \begin{cases} 1 & \text{si } t > 0 \\ 0 & \text{si } t = 0 \\ -1 & \text{si } t < 0 \end{cases}$$

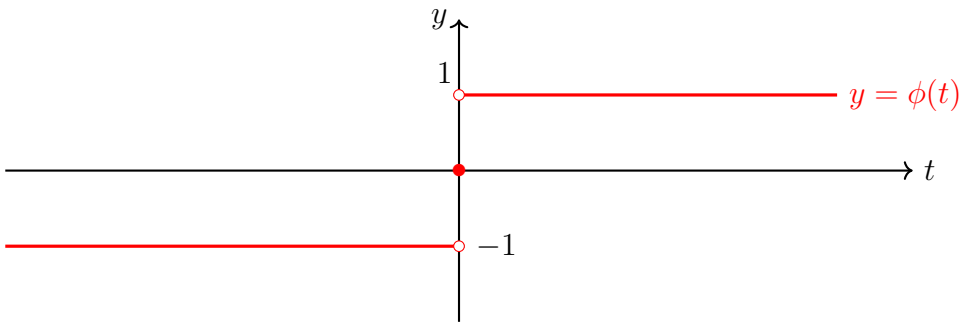

FIGURE 12 – Fonction signe

Le perceptron peut aussi être utilisé pour effectuer de la régression mais cet aspect ne sera pas développé ici.

On considère la situation décrite par la figure 13. Des points de $\mathbb{R}^2$ issus de deux catégories distinctes sont représentés.

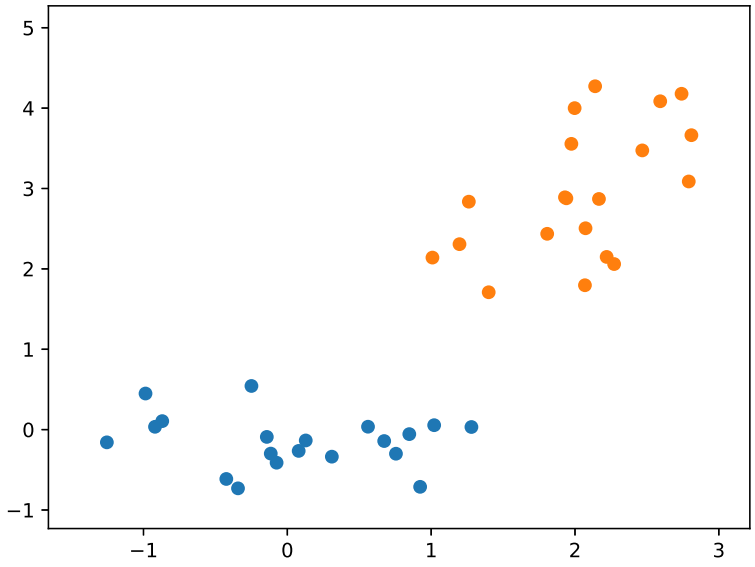

FIGURE 13 – Nuage de points issus de deux catégories distinctes

On peut aisément imaginer tracer un trait qui sépare ces deux catégories ce qui revient précisément à déterminer une droite d'équation $w_0 + w_1 x + w_2 y = 0$ avec w_0, w_1 et w_2 des réels. Les demi-plans $w_0 + w_1 x + w_2 y > 0$ et $w_0 + w_1 x + w_2 y < 0$ désignent alors les deux classes identifiées. Le biais en entrée est indispensable : c'est le degré de liberté qui permet de ne pas considérer que des droites passant par l'origine. L'enjeu de la classification est donc la détermination du vecteur $\mathrm{W} = (w_0, w_1, w_2)$ de $\mathbb{R}^3$.

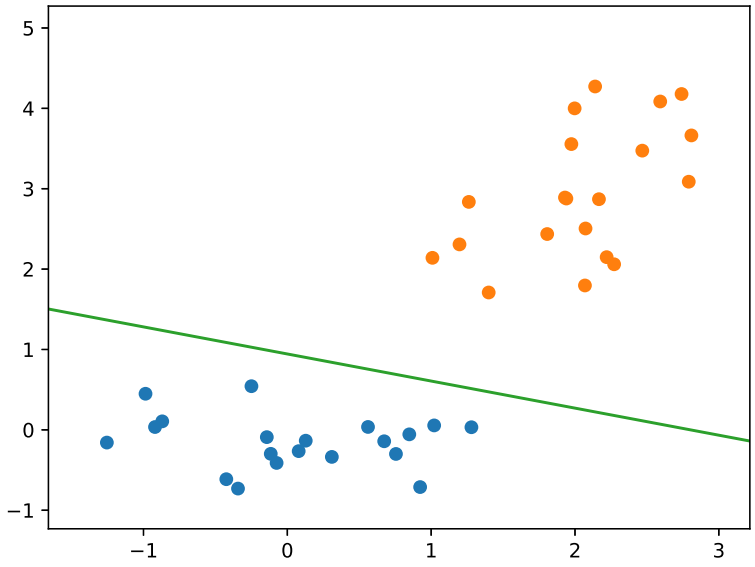

FIGURE 14 – Droite séparant les deux catégories de points

Définition 4. *Le* perceptron *à d entrées est un* classifieur linéaire*, à savoir une application définie par*

$$\forall (x_1, \dots, x_d) \in \mathbb{R}^d \qquad f(x_1, \dots, x_d) = \phi(\langle \mathrm{W}, \mathrm{X} \rangle)$$

avec $\mathrm{X} = (1, x_1, \dots, x_d)$, $\mathrm{W} = (w_0, \dots, w_d) \in \mathbb{R}^{d+1}$ *et ϕ la fonction signe.*

Remarque : Le qualificatif *linéaire* vient de la linéarité de $\mathrm{X} \in \mathbb{R}^{d+1} \mapsto \langle \mathrm{W}, \mathrm{X} \rangle$.

2 Implémentation d'un perceptron

Définition 5. *Soit* $\mathscr{S} = (\mathrm{M}_i, c_i)_{1 \leqslant i \leqslant n}$ *avec* $\mathrm{M}_i = (m_1^{(i)}, \dots, m_d^{(i)}) \in \mathbb{R}^d$ *et* $c_i \in \{-1, 1\}$ *la classe du point* M_i *pour tout* $i \in [\![1 \,;\, n]\!]$*. La famille $\mathscr{S}$ est dite* linéairement séparable *s'il existe une forme linéaire ψ non nulle sur* $\mathbb{R}^d$ *telle que*

$$\forall i \in [\![1 \,;\, n]\!] \qquad c_i \, \psi(\mathrm{M}_i) > 0$$

Remarque : La condition de séparabilité peut donc s'interpréter par

$$\forall i \in [\![1 \,;\, n]\!] \qquad \psi(\mathrm{M}_i) \begin{cases} > 0 & \text{si } c_i = 1 \\ < 0 & \text{si } c_i = -1 \end{cases}$$

autrement dit, l'hyperplan Ker ψ sépare les points de la famille $\mathscr{S}$.

Définition 6. *Soit* $\mathscr{A} = (\mathrm{X}_i, c_i)_{1 \leqslant i \leqslant n}$ *une base d'apprentissage avec* $\mathrm{X}_i = (1, x_1^{(i)}, \dots, x_d^{(i)}) \in \mathbb{R}^{d+1}$ *et* $c_i \in \{-1, 1\}$ *la classe du point* $(x_1^{(i)}, \dots, x_d^{(i)})$ *pour tout* $i \in [\![1 \,;\, n]\!]$*. On suppose que la famille $\mathscr{A}$ est linéairement séparable. L'algorithme d'apprentissage du perceptron détermine un vecteur de poids* $\mathrm{W} = (w_0, \dots, w_d) \in \mathbb{R}^{d+1}$ *tel que*

$$\forall i \in [\![1 \,;\, n]\!] \qquad c_i \, \langle \mathrm{W}, \mathrm{X}_i \rangle > 0$$

Remarque : D'après le théorème de Riesz, on peut interpréter le résultat de cet algorithme comme l'apprentissage de l'hyperplan $\mathrm{Vect}\,(\mathrm{W})^\perp$ puisque toute forme linéaire peut être vue comme produit scalaire contre un vecteur.

L'algorithme d'apprentissage du perceptron consiste en les étapes suivantes :

- Initialisation : la variable W est initialisée avec le vecteur nul de $\mathbb{R}^{d+1}$ et une variable booléenne d'apprentissage reçoit `True` ;
- Tant qu'il y a apprentissage :
  - On remet la variable d'apprentissage à `False` ;
  - Pour i variant de 1 à n :
    - Si $c_i \, \langle \mathrm{W}, \mathrm{X}_i \rangle \leqslant 0$, alors $\mathrm{W} \leftarrow \mathrm{W} + c_i \mathrm{X}_i$ et la variable d'apprentissage reçoit `True` ;

On peut démontrer (théorème de Novikoff) la correction et terminaison de l'algorithme. Intuitivement, l'étape d'apprentissage correspond à ajouter, pour un point X_i mal classifié, un contrepoids à W pour aller dans le sens d'une classification correcte.

Pour l'implémentation, on importe les modules :

```
import numpy as np, numpy.linalg as alg
```

Les points de $\mathbb{R}^d$ et les vecteurs de $\mathbb{R}^{d+1}$ sont codés par des tableaux de type `ndarray`. La base d'apprentissage est une liste de sous-listes où chaque sous-liste contient les points de $\mathbb{R}^d$ d'une même classe. La fonction `np.dot` réalise le produit scalaire canonique de $\mathbb{R}^{d+1}$.

```
def percept(A,c):
    """pecept(A:list)->ndarray
    A : base d'apprentissage, chaque classe étant une sous-liste de A
    c : classe c à séparer du reste
    Renvoie le vecteur de poids W calculé par le peceptron"""
    d=A[0][0].shape[0]
    W=np.array([0]*(d+1))
    C=len(A) # nb de classes dans A
    Learning=True
    while Learning:
        Learning=False
        for k in range(C):
            c_i=(k==c)-(k!=c)
            for pt in A[k]:
                X_i=np.append(pt,1)
                if c_i*np.dot(W,X_i)<=0:
                    Learning=True
                    W=W+c_i*X_i
    return W
```

On obtient la classification illustrée par la figure :

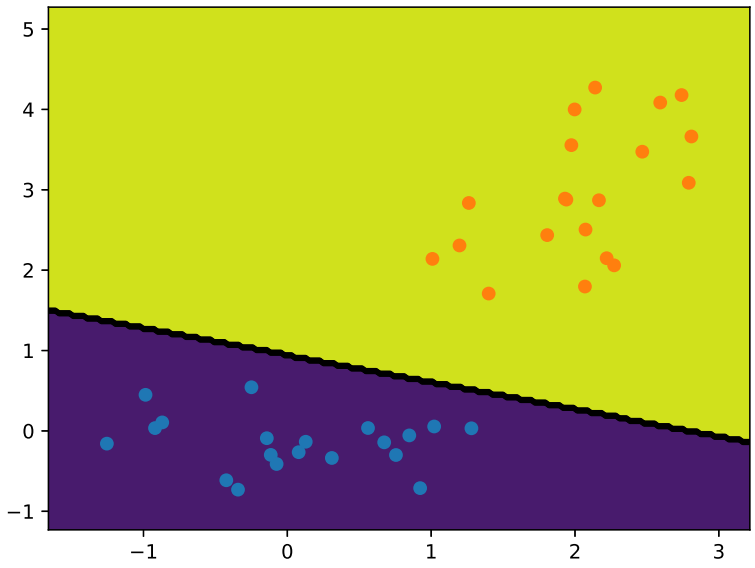

FIGURE 15 – Classification binaire par le perceptron

3 Perceptron multi-classes

Une extension naturelle à la classification binaire est la *classification multi-classes*, avec potentiellement plus que deux classes.

On considère la situation décrite par la figure 16 où sont représentés des points de $\mathbb{R}^2$ issus de trois catégories distinctes.

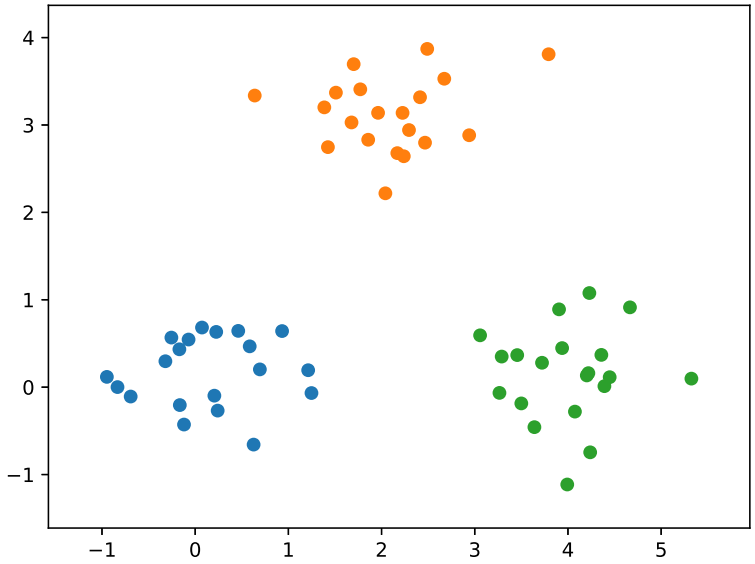

FIGURE 16 – Nuage de points issus de trois catégories distinctes

Une stratégie naïve consiste simplement à exécuter plusieurs perceptrons où chacun réalise la classification binaire d'une classe contre toutes les autres.

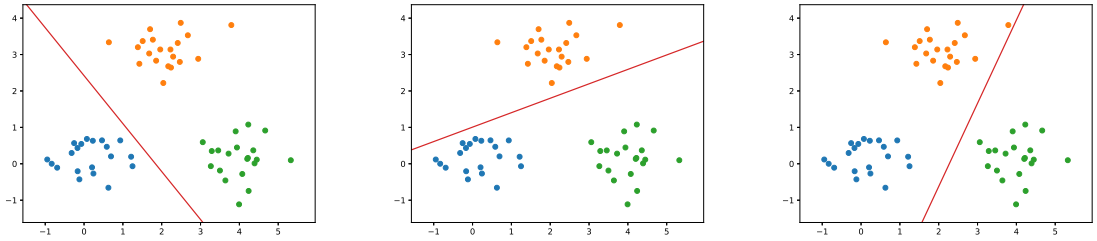

FIGURE 17 – Classifications binaires en « One vs All »

Ainsi, pour réaliser une classification en C classes numérotées de 1 à C avec C entier supérieur ou égal à 2, on utilise C perceptrons, donc C neurones, où pour $i \in [\![1\,;\,\mathrm{C}]\!]$, le perceptron d'indice i sert à classifier la classe i contre toutes les autres à savoir $[\![1\,;\,\mathrm{C}]\!] \setminus \{i\}$.

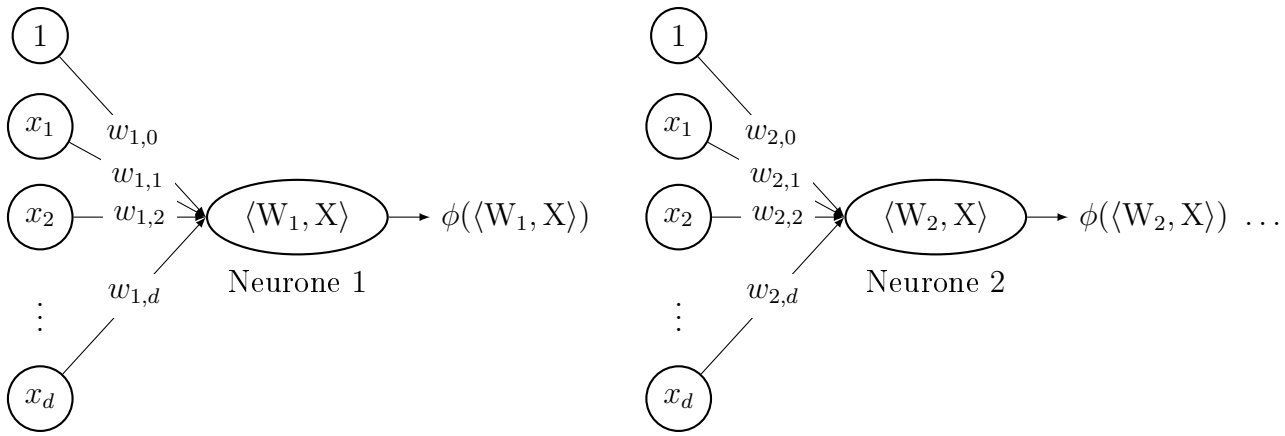

FIGURE 18 – Plusieurs perceptrons à d entrées

Une fois les classifications « One vs All » effectuées, l'espace $\mathbb{R}^d$ est partitionné selon les valeurs prises par l'application

$$\mathrm{X} \in \mathbb{R}^d \mapsto \arg \underset{k \in [\![1 \,;\, \mathrm{C}]\!]}{\mathrm{Max}} \langle \mathrm{W}_k, \mathrm{X} \rangle$$

On retient donc la classe qui maximise en un certain sens la séparation du point X dans cette classe vis-à-vis des autres.

Pour le nuage de points considéré à la figure 16, le plan $\mathbb{R}^2$ est partitionné en trois zones P_1, P_2 et P_3 telles que

$$\forall \ell \in [\![1 \,;\, 3]\!] \qquad \forall (x, y) \in \mathrm{P}_\ell \qquad \arg \underset{k \in [\![1 \,;\, 3]\!]}{\mathrm{Max}} \langle \mathrm{W}_k, (1, x, y) \rangle = \ell$$

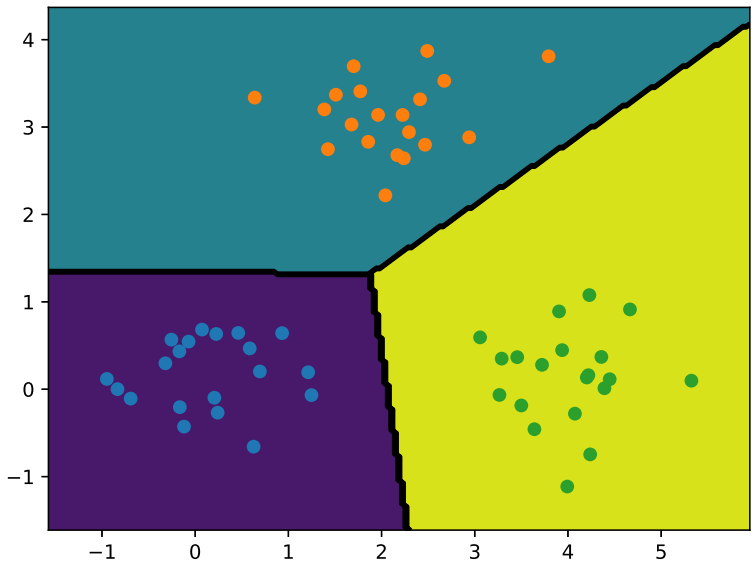

FIGURE 19 – Classification multi-classes en « One vs All »

On peut aussi imaginer un travail plus collaboratif des neurones entre eux. C'est le principe du

perceptron multi-classes qui est un réseau de neurones à une couche avec C neurones qui vont, d'une certaine façon, travailler conjointement.

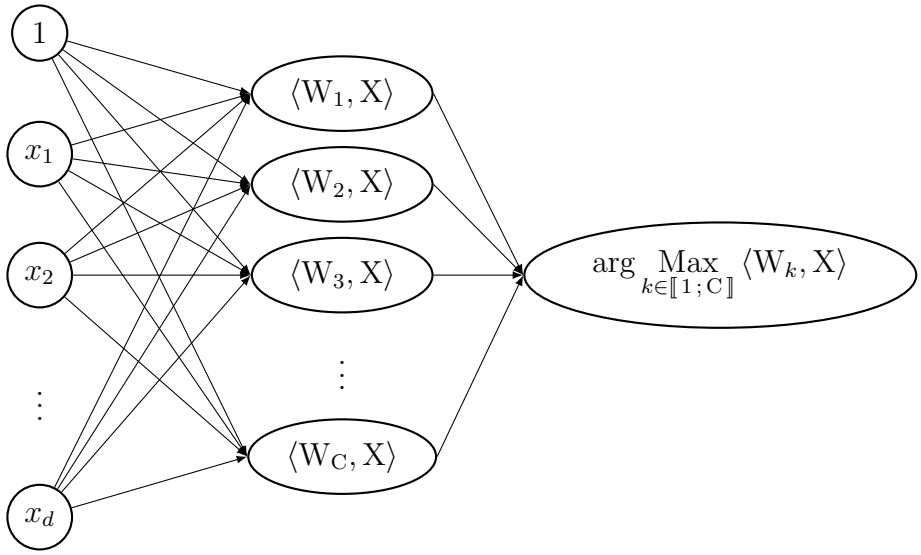

FIGURE 20 – Perceptron multi-classes

4 Implémentation d'un perceptron multi-classes

Définition 7. *Soit* $\mathscr{A} = (\mathrm{X}_i, c_i)_{1 \leqslant i \leqslant n}$ *une base d'apprentissage avec* $\mathrm{X}_i = (1, x_1^{(i)}, \ldots, x_d^{(i)}) \in \mathbb{R}^{d+1}$ *et* $c_i \in [\![1 \,;\, \mathrm{C}]\!]$ *la classe du point* $(x_1^{(i)}, \ldots, x_d^{(i)})$ *pour tout* $i \in [\![1 \,;\, n]\!]$. *L'algorithme d'apprentissage du perceptron multi-classes détermine une liste de vecteurs de poids* $[\mathrm{W}_1, \ldots, \mathrm{W}_\mathrm{C}]$ *avec* $\mathrm{W}_i = (w_{i,0}, \ldots, w_{i,d}) \in \mathbb{R}^{d+1}$ *tels que*

$$\forall i \in [\![1 \,;\, n]\!] \qquad c_i = \arg \underset{k \in [\![1 \,;\, \mathrm{C}]\!]}{\mathrm{Max}} \langle \mathrm{W}_k, \mathrm{X}_i \rangle$$

L'algorithme d'apprentissage du perceptron multi-classes consiste en les étapes suivantes :

- Initialisation : la variable tab_W est initialisée avec une liste de C vecteurs nuls de $\mathbb{R}^{d+1}$ et une variable booléenne d'apprentissage reçoit `True` ;
- Tant qu'il y a apprentissage :
  - On remet la variable d'apprentissage à `False` ;
  - Pour i variant de 1 à n :
    - On détermine $c = \arg \underset{k \in [\![1 \,;\, \mathrm{C}]\!]}{\mathrm{Max}} \langle \mathrm{tab\_W}[k], \mathrm{X}_i \rangle$;
    - Si $c \neq c_i$, c'est-à-dire si la classe prédite est différente de la vraie classe, alors :
      * On ajuste le poids pour la vraie classe $\mathrm{tab\_W}[c_i] \leftarrow \mathrm{tab\_W}[c_i] + \mathrm{X}_i$;
      * On ajuste le poids de la classe prédite $\mathrm{tab\_W}[c] \leftarrow \mathrm{tab\_W}[c] - \mathrm{X}_i$;
      * La variable d'apprentissage reçoit `True` ;

On généralise l'idée vue pour le perceptron en classification binaire : on modifie la liste des poids avec des contrepoids pour aller dans le sens d'une classification correcte.

Là aussi, on peut trouver dans la littérature spécialisée des résultats théoriques de correction et terminaison de l'algorithme.

```
def percept_mult(A):
    """pecept(A:list)->list
    A : base d'apprentissage, chaque classe étant une sous-liste de A
    Renvoie la liste des vecteurs de poids W
    calculés par le peceptron multi-classes"""
    C=len(A)
    d=A[0][0].shape[0]
    tab_W=[np.array([0]*(d+1)) for i in range(C)]
    Learning=True
    while Learning:
        Learning=False
        for c_i in range(C):
            for pt in A[c_i]:
                crit=-np.inf
                X_i=np.append(pt,1)
                # recherche indice pour max
                for k in range(C):
                    aux=np.dot(tab_W[k],X_i)
                    if aux>crit:
                       c=k
                       crit=aux
                # c : classe prédite, c_i : vraie classe
                if c!=c_i:
                    Learning=True
                    tab_W[c_i]=tab_W[c_i]+X_i
                    tab_W[c]=tab_W[c]-X_i
    return tab_W
```

On obtient la classification illustrée par la figure :

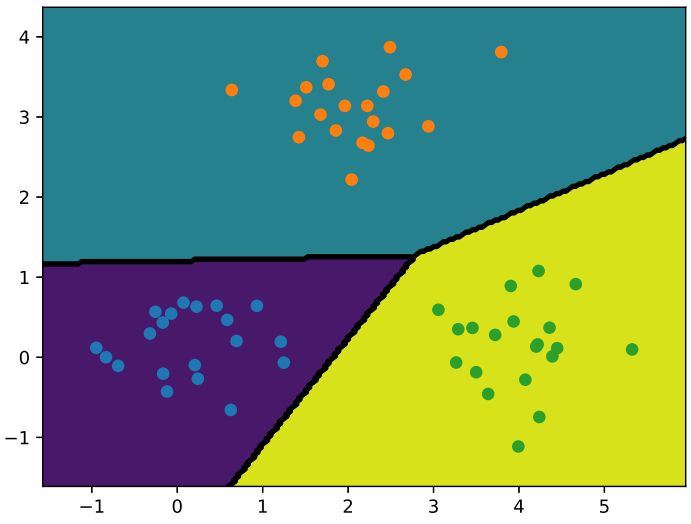

FIGURE 21 – Classification par le perceptron multi-classes

Références

[1] Base MNIST, `https://yann.lecun.com/exdb/mnist/`

[2] Chloé-Agathe Azencott, *Introduction au Machine Learning*, Dunod, 2022