A2DI - TP n°1

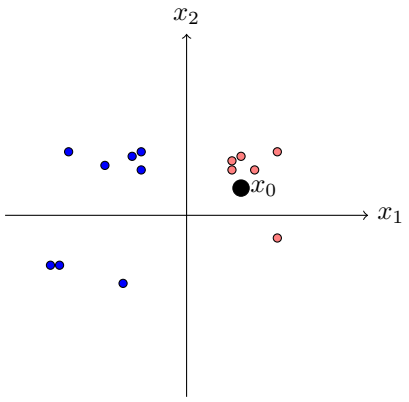
Ce premier TP sert d'échauffement avant d'aborder des techniques de ML avancée. Nous étudierons deux algorithmes déterministes dans un contexte de classification supervisée. Ces algorithmes mettront en lumière un certain nombre de concepts et de défis évoqués en cours.

Exercice n°1 : k-ppv

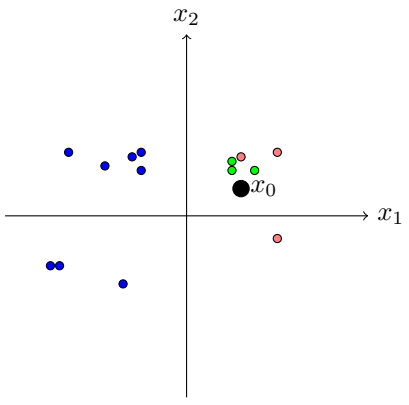
L'algorithme des k plus proches voisins est le plus simple des algorithmes de classification supervisée. Il est non-paramétrique et ne nécessite pas de phase d'apprentissage.

On rappelle si après le principe de cet algorithme :

— On part d'ensemble d'apprentissage $\mathcal{D}$ constitués de n exemples associés à une classe : $(\mathbf{x_i}, c_i)_{i=1}^{n}$.
— Les exemples $\mathbf{x}_i$ appartiennent à l'espace d'attributs $\mathbb{X}$.
— Un attribut est une dimension de $\mathbb{X}$.
— Les classes c_i appartiennent à un ensemble noté $\mathcal{C}$.
— Un nouvel exemple "non-vu" précedemment arrive : x_0. Plaçons cet exemple dans l'espace des attributs :



— Supposons qu'on choisisse l'hyper-paramètre $k = 3$. En utilisant, la distance euclidienne dans $\mathbb{X}$, on sait trouver les 3 plus proches voisins de x_0 :



— Parmi les 3 voisins, on procède à un vote à la majorité : sans ambiguité, on prédit que x_0 appartient à la classe *red*.

— On choisit souvent k impair pour éviter des *ex aequo* dans le vote.

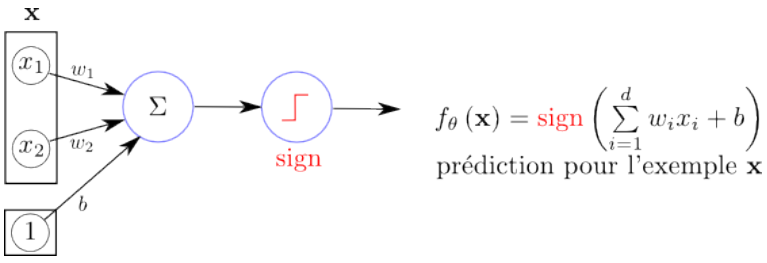
Questions :

1. Lancez **Spyder** et créez un script python pour cet exercice. Chargez le dataset IRIS à l'aide des commandes python suivantes :
```
from sklearn import datasets
data=datasets.load_iris()
```
2. Identifier n le nombre d'exemples, $\sharp\mathcal{C}$ le nombre de classes, $d = \dim(\mathbb{X})$ le nombre d'attributs et les natures des attributs.
3. Répartissez le dataset en 2 sous-parties : l'ensemble d'apprentissage $\mathcal{D}_{\text{app}}$ et l'ensemble de test $\mathcal{D}_{\text{test}}$. L'ensemble d'apprentissage contiendra 50% des données. Veillez à équilibrer les représentants de chaque classe dans les deux ensembles.
4. Créez une fonction `kppv`. Cette fonction aura en entrée un exemple `x`, l'ensemble d'apprentissage $\mathcal{D}_{\text{app}}$ et le paramètre k. Elle retourne la classe prédite pour l'exemple `x` selon l'algorithme des k-ppv.
On pourra entre autres utiliser les fonctions `numpy.argsort`, `numpy.bincount` et `numpy.argmax`.
5. Testez votre fonction en boucle sur tout les exemples contenus dans $\mathcal{D}_{\text{test}}$ et comparez classe prédite et classe réelle. En déduire le taux de bonne classification.
6. Tracez le taux de bonne classification en fonction de k (k allant de 1 à 100). Quelle valeur de k offre la meilleure généralisation ? Pour tracer des courbes en python, on utilisera le module `matplotlib`.
7. Tracez le temps moyen d'execution sur 100 appels à la fonction `kppv` en fonction de la taille de $\mathcal{D}_{\text{app}}$. On fera varier la taille de 20% du dataset à 95% par pas de 5%. Quelle limite voyez-vous à l'utilisation de l'algorithme k-ppv ?
On pourra entre autres utiliser la fonction `time.time`

Exercice n°2 : Perceptron

L'algorithme du perceptron est l'ancêtre des réseaux de neurones que nous étudierons plus en détail dans un autre chapitre. C'est un algorithme paramétrique cherchant à trouver dans l'espace d'attributs une séparation linéaire entre les classes (à supposer qu'une telle séparation existe).

La fonction de prédiction $f_{\boldsymbol{\theta}} : \mathbb{X} \to \mathcal{C}$ du perceptron s'obtient selon le schéma suivant :



La fonction sign est définie comme suit :

$$\begin{aligned} \text{sign} : \mathbb{R} &\longrightarrow \{-1; +1\}, \\ x &\longrightarrow \begin{cases} +1 & \text{si } x \geq 0 \\ -1 & \text{si } x < 0 \end{cases}. \end{aligned} \tag{1}$$

Imaginons qu'une banque cherche à déterminer s'il est rentable de vous attribuer une carte bancaire. C'est un problème de classification binaire et on choisit arbitrairement la convention suivante :
— autoriser la carte : classe +1,
— refuser la carte : classe -1.

Les attributs pour un client donné sont typiquement sont niveau moyen de découvert, son solde moyen d'épargne, son âge, son salaire, etc. Les attributs du client forme le vecteur $\mathbf{x}$.

En combinant ces différents attributs on obtient un score $s = \sum_{i=1}^{d} w_i x_i$. Si le score s est au dessus du seuil $(-b)$ alors on a $f_{\boldsymbol{\theta}}(\mathbf{x}) = +1$ et on autorise la carte (et sinon l'inverse). Contraimemnt au k-ppv, le perceptron nécessite un algorithme d'apprentissage afin de déterminer de bonnes valeurs pour les w_i et b. Nous exposons ci-après son principe.

Introduisons le vecteur $\boldsymbol{\theta}$ de dimension $d+1$ et regroupant tous les paramètres du modèle :

$$\boldsymbol{\theta} = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \\ b \end{pmatrix}. \tag{2}$$

Introduisons également le vecteur $\mathbf{x}^+$ correspondant au concaténement de $\mathbf{x}$ avec 1 :

$$\mathbf{x}^+ = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \\ 1 \end{pmatrix}. \tag{3}$$

On remarque alors qu'en utilisant le produit scalaire usuel, on obtient une écriture simplifiée de la fonction de prédiction :

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = \text{sign}\left(\boldsymbol{\theta}^T.\mathbf{x}^+\right). \tag{4}$$

La classe prédite pour l'exemple $\mathbf{x}$ est donc :

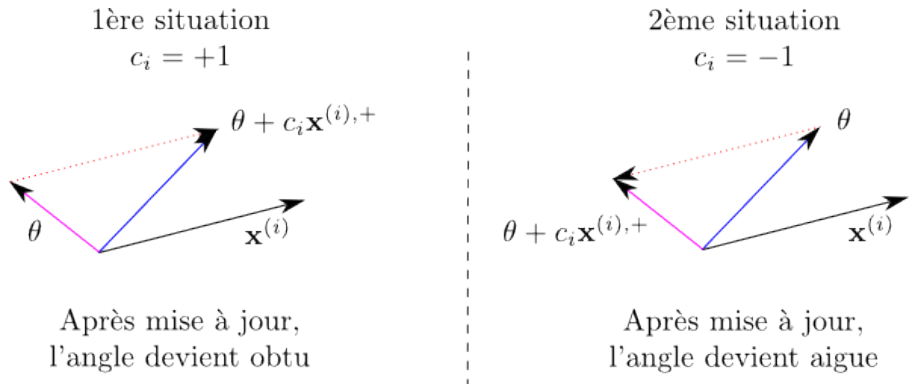
— $+1$ si l'**angle** formé par les vecteurs $\mathbf{x}^+$ et $\boldsymbol{\theta}$ est **aigüe**,

— -1 si l'**angle** formé par les vecteurs $\mathbf{x}^+$ et $\boldsymbol{\theta}$ est **obtu**.

Admettons qu'il existe un exemple $\mathbf{x}^{(i)} \in \mathcal{D}_{\text{app}}$ qui soit mal classifié par $f_{\boldsymbol{\theta}}$. Il faut alors corriger cette fonction en mettant à jour les paramètre selon la règle suivante :

$$\boldsymbol{\theta} \longleftarrow \boldsymbol{\theta} + c_i \times \mathbf{x}^{(i),+}. \tag{5}$$

La figure ci-après explique pourquoi cette règle fonctionne.

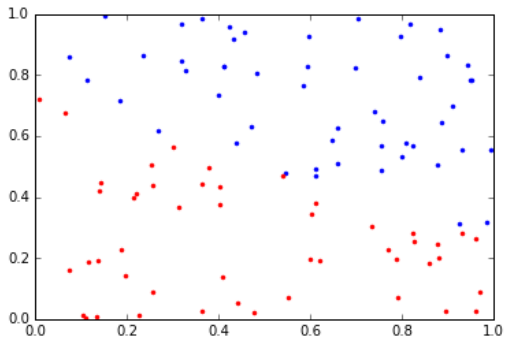


Enfin, on peut montrer qu'en choisissant n'importe quel exemple malclassifié et en répétant cette règle de mise à jour, on converge vers un vecteur $\boldsymbol{\theta}$ permettant de classifier correctement tous les exemples d'apprentissage.

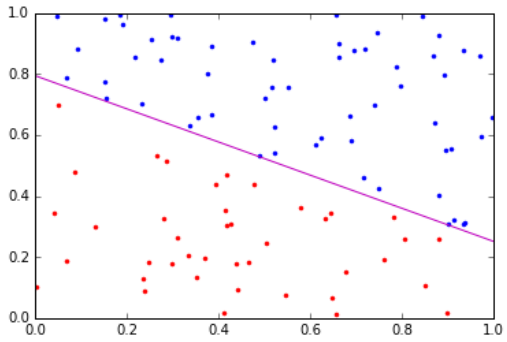
Questions :

1. Nous allons travailler avec des données synthétiques dans cet exercice. Tirez 100 points uniformément dans le carré $[0;1] \times [0;1]$. Chaque point est un exemple d'apprentissage et on a donc $d = 2$.

2. Pour chaque point, attribuez lui la classe +1 si $-\frac{1}{2}x_1 + 0.75 \leq x_2$. Les autres exemples appartiendront à la classe -1. Il n'y aura pas forcément le même nombre d'exemples par classe. Affichez ce dataset avec `matplotlib`. Voici un exemple de visualisation souhaitée :



3. Comme au 1er exercice, découpez cet ensemble en $\mathcal{D}_{\text{app}}$ et $\mathcal{D}_{\text{test}}$ (80% / 20 %).
4. Créez une fonction `ptrain` qui prend en entrée $\mathcal{D}_{\text{app}}$ et fournit $\boldsymbol{\theta}$ après convergence. On initialisera $\boldsymbol{\theta}$ aléatoirement (fonction `numpy.random.random`).
5. Superposez à votre visualisation du dataset un tracé de la droite d'équation $x_2 = -\frac{w_1 x_1 + b}{w_2}$ où les paramètres w_1, w_2 et b ont été obtenus après appel à la fonction `ptrain`. Voici un exemple de visualisation souhaitée :



6. Créez une fonction `ptest` qui prend en entrée un exemple $\mathbf{x}$ et $\boldsymbol{\theta}$ et fournit une prédiction selon la formule $f_{\boldsymbol{\theta}}(\mathbf{x})$.
7. Prédisez la classe de chaque exemple de test et calculez le taux de bonne classification correspondant.
8. Renouvelez ces étapes en faisant varier d de 2 à 20. Tracez le taux de bonne classification en fonction de d. Pour un tracé plus parlant, il faudra afficher le taux moyen sur 100 réalisations de l'expérience. Cette courbe illustre t-elle le phénomène de la malédiction de la dimension ?