Définition 5.1 - *apprentissage*

L'*apprentissage* en informatique permet l'approche de différents problèmes :

- la *classification*, par exemple déterminer un objet sur une image, ou un son sur un flux audio ;
- la *régression*, par exemple prévoir la valeur du cours de la bourse.

Définition 5.2 - *apprentissage supervisé*

L'*apprentissage supervisé* consiste à entraîner un modèle ou un algorithme à l'aide d'un *ensemble d'apprentissage* :

$$S = \{(x_i,\, y_i),\, i \in [\![1,\, n]\!]\} \subset X \times Y \quad \text{avec} \begin{cases} X & \text{l'ensemble des \textit{objets} manipulés par le modèle ou l'algorithme} \\ Y & \text{l'ensemble des \textit{classes} ou \textit{valeurs} associées aux objets de } X \end{cases}$$

$(x,\, y) \in S$ signifie que *l'objet x est dans la classe y* ou *a pour valeur y*.
On cherche pour un objet inconnu x à déterminer une classe ou une valeur y convenable en s'appuyant sur l'ensemble d'apprentissage S.

Définition 5.3 - *fonction de prédiction*

À tout modèle ou algorithme d'apprentissage supervisé on peut associer une *fonction $f : X \to Y$ dite de prédiction* qui à un objet associe la *classe estimée raisonnable par le modèle ou l'algorithme*.

Définition 5.4 - *fonction de perte*

À tout modèle ou algorithme d'apprentissage supervisé on peut associer une fonction $L : Y^2 \to \mathbb{R}_+$ qui à une prédiction associe une *valeur mesurant son inexactitude*. On a :

$$\forall y \in Y,\, L(y,\, y) = 0$$

Définition 5.5 - *fonction de risque empirique*

Pour tout modèle ou algorithme d'apprentissage supervisé on peut associer à toute fonction f de prédiction une espérance appelée *risque R* par :

$$\begin{aligned} R(f) &= \mathbb{E}_{X,Y}\Big(L\big(Y,\, f(X)\big)\Big) \\ &= \sum_{(x,y)\in X\times Y} L\big(y, f(x)\big)\mathbb{P}_{X,Y}(x,y) \end{aligned}$$

En pratique, on n'a jamais accès à $\mathbb{P}_{X,Y}$. On définit alors le *risque empirique comme étant la moyenne des pertes sur l'ensemble d'apprentissage*. Lui est calculable :

$$R_{\text{emp}}(f) = \frac{1}{|S|}\sum_{(x,y)\in S} L(y, f(x))$$

Dès lors, un algorithme d'apprentissage supervisé mettra en œuvre des algorithmes d'optimisation afin de trouver une fonction f qui minimise le risque empirique.

Implémentation - *algorithme de classification des k plus proches voisins - classique*

- **Entrée** :
  - un ensemble d'apprentissage $S \subset X \times Y$ indexé sur $[\![0,\, n]\!]$
  - une distance $\mathrm{d} : X^2 \to \mathbb{R}_+$
  - un objet x de classe inconnue
  - k le nombre de voisins à considérer
- **Sortie** : la classe y du voisin majoritairement présent parmi les k plus proches de x

```
file = file de priorité max vide
pour tout i ∈ ⟦0, n⟧ :
    si |file| < k :
        ajouter x_i àfile avec la priorité d(x, x_i)
    sinon :
        si d(x, x_i) < file[0] :
            supprimer le maximum de file
            ajouter x_i àfile avec la priorité d(x, x_i)
renvoyer la classe majoritaire parmi les éléments de file
```

Implémentation - *construction d'un arbre d-dimensionnel*

Cet algorithme constitue un prétraîtement de l'ensemble d'apprentissage, pour "faciliter" la recherche des k plus proches voisins en aval. Attention, si d est trop grand, on fait face au fléau de la dimension qui rend la méthode trop peu efficace.

- **Entrée initiale** :
  - l'ensemble $X_S \subset X$ pour lequel on connaît la classe de chaque objet
  - d la dimension de X_S
- **Sortie** : un arbre d-dimensionnel décrivant les positions relatives de chaque objet de X_S

```
Construit(i, X_courant) :
    n = |X_courant|
    si n == 0 :
        renvoyer Vide
    sinon :
        X_courant = tri de X_courant dans l'ordre croissant de
                la i-ème coordonnée
        val = X_courant[⌊n/2⌋]
        X_g, X_d = séparer strictement X_courant en deux listes,
                en l'indice ⌊n/2⌋
        g = Construit(d, i+1 mod d, X_g)
        d = Construit(d, i+1 mod d, X_d)
        renvoyer Noeud(val, g, d)
```

Implémentation - *algorithme de classification des k plus proches voisins - avec arbre d-dimensionnel (1/2)*

- **Entrée initiale** :
  - un arbre d-dimensionnel décrivant les positions relatives de chaque objet de X_S
  - un objet x de classe inconnue
  - k le nombre de voisins à considérer
- **Sortie** : une pile contenant le chemin de la racine vers une feuille de $\mathcal{A}$ modélisant le plus proche voisin de x.

```
explore(𝒜, i, x) :
    si 𝒜 est Vide :
        renvoyer []
    sinon :
        on identifie 𝒜 = Noeud(val, g, d)
        x_i = i-ème coordonnée de x
        v_i = i-ème coordonnée de val
        si x_i <= v_i :
            renvoyer 𝒜 :: explore(g, i+1 mod d, x)
        sinon :
            renvoyer 𝒜 :: explore(d, i+1 mod d, x)
```

Implémentation - *algorithme de classification des k plus proches voisins - avec arbre d-dimensionnel (2/2)*

- **Entrée initiale** :
  - une distance d : $X^2 \to \mathbb{R}_+$
  - `l` la pile du chemin parcouru vers le plus proche de x, contenant des sommets de $\mathcal{A}$ un arbre d-dimensionnel
  - un objet x de classe inconnue
  - k le nombre de voisins à considérer
- **Sortie** : la classe y du voisin majoritairement présent parmi les k plus proches de x

```
file = file de priorité max vide
kPPV(l, i, x, k) :
    Noeud(val, g, d), q = depiler l
    x_i = i-ème coordonnée de x
    v_i = i-ème coordonnée de val
    si file[0].prio < d(x_i E_i, v_i E_i) // distance unidimensionnelle :
            // dépend de d mais souvent égal à |x_i - v_i|
        si besoin, ajouter/remplacer x àfile avec la priorité d(x,val)
    sinon :
        chemin = explore(Noeud(val, g, d), i mod d, x)
        kPPV(chemin, |file| + i mod d, x, k)
    renvoyer la classe majoritaire parmi les éléments de file
```