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* :

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

$(x, y) \in S$ 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** :
  - l'ensemble $X_S \subset X \subset \mathbb{R}^D$ pour lequel on connaît la classe de chaque objet
  - $i \in [\![0, D-1]\!]$ la coordonnée selon laquelle on trie
- **Sortie finale** : un arbre D-dimensionnel décrivant les positions relatives de chaque objet de X_S

```
1  Construit(i, X_S) :
2      n = |X_S|
3      si n == 0 :
4          renvoyer Vide
5      sinon :
6          X_S = tri de X_S dans l'ordre croissant de
7                   la i-ème coordonnée
8          val = X_S[⌊n/2⌋]
9          X_g, X_d = séparer strictement X_S en l'indice ⌊n/2⌋
10         g = Construit(d, i+1 mod D, X_g)
11         d = Construit(d, i+1 mod D, X_d)
12         renvoyer Noeud(val, g, d)
```

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

- **Entrée** :
  - un arbre D-dimensionnel décrivant les positions relatives de chaque objet de $S \subset X \times Y$, avec $X \subset \mathbb{R}^D$
  - $i \in [\![0,\, D-1]\!]$ la coordonnée courante
  - un objet x de classe inconnue
- **Sortie** : une pile contenant le chemin de la racine (en queue) vers une feuille de $\mathcal{A}$ modélisant le plus proche voisin de x (en tête).

```
1  explore(A, i, x) :
2      si A est Vide :
3          renvoyer []
4      sinon :
5          on identifie A = Noeud(val, g, d)
6          x_i = i-ème coordonnée de x
7          v_i = i-ème coordonnée de val
8          si x_i <= v_i :
9              renvoyer A :: explore(g, i+1 mod D, x)
10         sinon :
11             renvoyer A :: explore(d, i+1 mod D, x)
```

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

On munit $X \subset \mathbb{R}^D$ d'une distance $\mathrm{d} : X^2 \to \mathbb{R}_+$

- **Entrée initiale** :
  - une pile contenant le chemin de la racine (en queue) vers une feuille de $\mathcal{A}$ modélisant le plus proche voisin de x (en tête)
  - $i \in [\![0,\, D-1]\!]$ la coordonnée courante
  - 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

```
1  file = file de priorité max vide
2  kPPV(l, i, x, k) :
3      Noeud(val, g, d), new_l = depiler l
4      x_i = i-ème coordonnée de x
5      v_i = i-ème coordonnée de val
6      si file[0].prio < |x_i − v_i|
7          si |file| < k :
8              ajouter x àla file avec la priorité d(x, val)
9          sinon :
10             si file[0].prio > d(x, val) :
11                 supprimer l'élément de priorité maximale de file
12                 ajouter x àla file avec la priorité d(x, val)
13
14     sinon : // risque de voisins plus proches,
15             // par lesquels on ne passerait pas : on va regarder
16         chemin = explore(Noeud(val, g, d), i mod D, x)
17         kPPV(chemin, |chemin| + i mod D, x, k)
18
19     si |file| < k : // besoin d'encore des voisins ?
20         kPPV(new_l, i − 1 mod D, x, k) // on remonte l'arbre
21
22     renvoyer la classe majoritaire parmi les éléments de file
```

Définition 5.6 - *entropie de Shannon*

Soit $S = \{(x_i, y_i),\, i \in [\![1, n]\!]\} \subset X \times Y$ un ensemble d'apprentissage. Pour $y \in Y$, on définit $C_y = \{x \in X, (x, y) \in S\}$ *l'ensemble des objets de classe* y.
L'*entropie de Shannon* de S est :

$$H(S) = -\sum_{\substack{y \in Y \\ C_y \neq \varnothing}} \frac{|C_y|}{n} \ln\left(\frac{|C_y|}{n}\right)$$

L'entropie de Shannon mesure le désordre dans la distribution des classes de S.

Définition 5.7 - *gain d'information*

Soit $S \subset X \times Y$ un ensemble d'apprentissage, avec $X \subset A_1 \times \cdots \times A_m \times X'$. Pour $i \in [\![1, m]\!]$, On considère le critère A_i pouvant prendre les valeurs $v_1, \ldots, v_k$. On écrit alors :

$$S = \bigsqcup_{j=1}^{k} \{(x, y) \in S,\ \underbrace{x_i}_{\substack{\text{valeur du critère} \\ A_i \text{ pour } x}} = v_j\}$$

$$= \bigsqcup_{j=1}^{k} S_j \quad \text{(notation)}$$

On définit *le gain d'information du critère* A_i comme :

$$G(S, A_i) = H(S) - \sum_{j=1}^{k} \frac{|S_j|}{|S|} H(S_j)$$

Le gain d'information *renseigne sur la pertinence du critère* A_i *pour discriminer les objets de* S.

Définition 5.8 - *apprentissage non supervisé*

L'apprentissage non supervisé repose des méthodes de *clustering* ou (ou *méthode de classifcation non supervisée*). Étant donné un ensemble $z = \{x_i,\, i \in [\![1, n]\!]\}$, on en cherche une partition :

$$z = \bigsqcup_{i=1}^{k} P_i$$

Les $(P_i)_{i \in [\![1,\, k]\!]}$ sont appelés *classes* ou *clusters*. L'objectif est que ces clusters vérifient :

- petite variabilité intra-classe
- grande variabilité inter-classe

Définition 5.9 - *relation de finesse*

Soit z un ensemble d'objets, de partitions en clusters $(P_i)_{i\in[\![1,\,k]\!]}$ et $(P'_j)_{j\in[\![1,\,k']\!]}$. On dit que $(P_i)_{i\in[\![1,\,k]\!]}$ *est plus fine que* $(P'_j)_{j\in[\![1,\,k']\!]}$ lorsque *tout cluster* P'_j *est l'union de clusters* P_i.

Définition 5.10 - *variance d'une partition*

Soit z un ensemble d'objets d'un espace métrique (E, d), de partition en clusters $(P_i)_{i\in[\![1,\,k]\!]}$. *La variance de la partition* $(P_i)_{i\in[\![1,\,k]\!]}$ est :

$$V\Big((P_i)_{i\in[\![1,\,k]\!]}\Big) = \sum_{i=1}^{k} \sum_{x\in P_i} \mathrm{d}(x, g_i)$$

où, pour tout $i \in [\![1,\, k]\!]$, g_i *est le barycentre du cluster* P_i :

$$g_i = \frac{1}{|P_i|} \sum_{x\in P_i} x$$

Définition 5.11 - *hiérarchie d'un ensemble d'objets*

Soit z un ensemble d'objets. On *appelle hiérarchie de* z toute famille finie $(\pi_i)_{i\in[\![1,\,n]\!]}$ de partitions de z telle que :

1. $\forall i \in [\![1,\, n-1]\!]$, π_i est plus fine que π_{i+1}
2. π_1 est celle des singletons de z
3. $\pi_n = \{z\}$

On représente une hiérarchie à l'aide d'un *dendrogramme*.

Implémentation - *algorithme de classification hiérarchique ascencendante*

L'algorithme de CHA permet de renvoyer une partition en un nombre raisonnable de clusters d'un ensemble Z d'objets. On utilise Unir & Trouver pour modéliser la relation "appartenir au même ensemble que".

- **Entrée** :
  - Z le tableau des objets considérés
  - $\mathbb{D} : \mathcal{P}(Z)^2 \to \mathbb{R}_+$ un écart sur Z
- **Sortie** : une partition de H en un nombre raisonnable de clusters

```
1  CHA(Z,𝔻):
2      H = tableau de taille |Z| rempli de ∅
3      P = structure Unir & Trouver des singletons de Z
4      H[0] = copier P profondément
5      ecarts = tableau de taille |Z|−1 rempli de −1
6
7      pour p allant de 1 à |Z|−1 :
8          x,y = classes de P de distance minimale
9          unir x y
10         H[p] = copier P profondément
11         ecarts[p-1] = 𝔻(P[i].val, P[j].val)
12     i = indice de l'élément maximal de ecarts
13     renvoyer H[i]
```

Définition 5.12 - *écart par lien minimal*

On suppose que (Z, d) est muni d'une structure d'espace métrique. L'écart suivant est appelé *écart par lien minimal* :

$$\begin{aligned} \mathbb{D} : \mathcal{P}(Z)^2 &\longrightarrow \mathbb{R}_+ \\ (A, B) &\longmapsto \min_{\substack{x \in A \\ y \in B}} \mathrm{d}(x, y) \end{aligned}$$

Pour A, B C trois parties deux à deux disjointes de Z :

$$\mathbb{D}(A \sqcup B, C) = \min\Big(\mathbb{D}(A, C),\, \mathbb{D}(B, C)\Big)$$

Définition 5.13 - *écart par lien maximal*

On suppose que (Z, d) est muni d'une structure d'espace métrique. L'écart suivant est appelé *écart par lien maximal* :

$$\begin{aligned} \mathbb{D} : \mathcal{P}(Z)^2 &\longrightarrow \mathbb{R}_+ \\ (A, B) &\longmapsto \max_{\substack{x \in A \\ y \in B}} \mathrm{d}(x, y) \end{aligned}$$

Pour A, B C trois parties deux à deux disjointes de Z :

$$\mathbb{D}(A \sqcup B, C) = \max\Big(\mathbb{D}(A, C),\, \mathbb{D}(B, C)\Big)$$

Définition 5.14 - *écart par lien moyen*

On suppose que (Z, d) est muni d'une structure d'espace métrique. L'écart suivant est appelé *écart par lien moyen* :

$$\begin{aligned} \mathbb{D} : \mathcal{P}(Z)^2 &\longrightarrow \mathbb{R}_+ \\ (A, B) &\longmapsto \frac{1}{|A||B|} \sum_{x \in A} \sum_{y \in B} \mathrm{d}(x, y) \end{aligned}$$

Pour A, B C trois parties deux à deux disjointes de Z :

$$\mathbb{D}(A \sqcup B, C) = \frac{1}{|A| + |B|}\Big(|A|\mathbb{D}(A, C) + |B|\mathbb{D}(B, C)\Big)$$

Implémentation - *algorithme d'apprentissage non supervisé des k-moyennes*

```
1  k_mean(k, Z):
2      // précondition : k ≤ Z
3      pour tout i ∈ ⟦1, k⟧:
4          c_i = élément aléatoire de Z, différent des précédents centres
5      initialiser (c_i)_{i∈⟦1,k⟧} les k centres des clusters comme
6              etant des éléments aléatoires de Z
7
8      tant que les centres sont modifiés:
9          décomposer Z = ⊔_{i=1}^{k} où P_i est
10                 l'ensemble des plus proches de c_i que des autres centres
11         pour tout i ∈ ⟦1, k⟧:
12             c_i = 1/|P_i| ∑_{z∈P_i} z
13     renvoyer (P_i)_{i∈⟦1,k⟧}
```