Chapitre 19 : apprentissage supervisé (apprendre à partir d'exemples)

06 mars

L'apprentissage automatique est un domaine de l'intelligence artificielle qui consiste à concevoir des modèles capables de s'adapter et de s'améliorer à partir de l'analyse d'une grande quantité de données.

On dit qu'un agent **apprend** s'il améliore ses performances lors de tâches futures après avoir fait des observations sur le monde. On remarque qu'un algorithme n'a pour sa part pas de capacité d'apprentissage et peut donc devenir inefficace face à des changements imprévus. Par exemple, il est trop compliqué de chercher à écrire un algorithme capable de prédire le prix du marché de demain alors que la conjoncture peut radicalement changer. L'apprentissage permet aussi de travailler sur des problèmes pour lesquels les algorithmes envisageables auraient des complexité inexploitables (comme pour les jeux) voire des problèmes pour lesquelles aucune solution algorithmique n'est connue (reconnaissance de visage par exemple).

Dans ce chapitre, nous allons uniquement parler d'apprentissage supervisé : il s'agit à partir de couples (entrée,sortie) d'apprendre une fonction qui permettra d'associer une sortie à une entrée donnée.

Si les valeurs de sortie possibles forment un ensemble fini alors on parle de problème de **classification** et si elles forment un ensemble infini alors on parle de problème de **régression.**

Par exemple, un problème de classification peut consister à reconnaitre un chiffre entre 0 et 9 écrit de manière manuscrite et un problème de régression peut consister à prévoir la distance de freinage dans une configuration de conduite donnée.

Le programme nous demande d'étudier quelques algorithmes qui permettent de travailler sur des problèmes de classification dans le cadre de l'apprentissage supervisé.

Ainsi notre cadre formel d'étude sera le suivant :

On cherche à apprendre une fonction $f : \mathbb{R}^d \to U$ avec U un ensemble fini.

On fixe un entier N qui correspond au nombre de données d'apprentissage et une famille de N couples $\{(x_i, y_i)\}_{1\leq i\leq N}$ avec chaque $x_i \in \mathbb{R}^d$ et chaque $y_i \in U$.

On cherche alors à déterminer une fonction f en cohérence avec les données d'apprentissage (c'est-à-dire qui minimise l'erreur entre les $f(x_i)$ et les y_i dans un sens à déterminer).

Notre propos ne sera pas de discuter les techniques permettant d'estimer la qualité concrète d'un modèle ni les manières de fixer les paramètres permettant d'obtenir des résultats jugés satisfaisants mais de présenter quelques grandes méthodes algorithmiques qui permettent d'aborder ce problème.

1 Classification par k-plus proches voisins

Dans cet algorithme, les données d'entraînement constituent directement le modèle, il n'y a pas de phase d'apprentissage. On se place dans $\mathbb{R}^d$ et on dispose donc d'un ensemble Z de données d'entraînement de la forme (x_i, y_i) où $x_i \in \mathbb{R}^d$ et y_i est une classe.

Pour classifier une donnée nouvelle x, on utilise les k plus proches voisins de x (au sens d'une distance δ sur $\mathbb{R}^d$), et on choisit la classe majoritairement représentée parmi les k voisins.

Dans cet algorithme, on peut jouer sur deux paramètres : la valeur de k et le choix de la distance δ.

Une implémentation possible est :

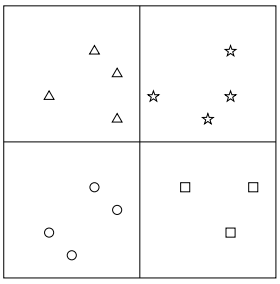
```
let plus_proches_voisins donnees k delta x =
    let distances = List.map (fun (xi, yi) -> (delta x xi, yi)) donnees in
        let plusProches = plus_petits k distances in
            mode (List.map snd plusProches);;
```

La fonction `plus_petits` peut être écrite comme un tri partiel (on obtient une complexité en $O(N \log N)$). Le calcul des distances coûte $O(dN)$ a priori. La fonction `mode` donne le mode (ou valeur modale) d'une liste, c'est-à-dire la valeur la plus présente, on peut utiliser un comptage en temps linéaire si les classes sont des entiers de 0 à $C-1$ par exemple, ou un comptage par dictionnaire dans le cas général. La complexité totale peut être estimée à $O(dN + N \log N + k)$ dans le meilleur cas pour chaque recherche.

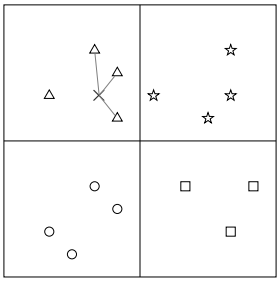
Remarque 1. *En cas d'ambiguïté dans la décision,* ie *s'il y a au moins deux classes modales dans les k plus proches voisins, on peut tirer au hasard entre les différentes classes modales. Pour éviter au maximum les ambiguïté, on peut également pondérer le vote en fonction de la distance à x.*

Remarque 2. *On choisit généralement k impair proche de $\sqrt{n}$.*

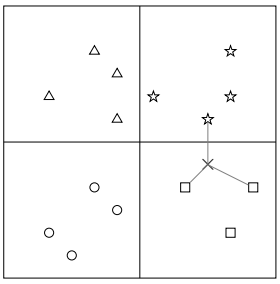
Exemple 1. *On souhaite classifier des données en quatre classes (ici modélisées par chaque quadrant). On dispose d'un jeu initial de 15 données étiquetées (par des étoiles, cercles, carrés ou triangles) réparties de la manière suivante :*



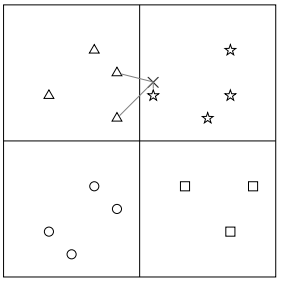
On suppose de plus que $K = 3$. On montre ici trois classifications possibles d'un point :



Le point est correctement classifié comme un triangle.



Le point est correctement classifié comme un carré.



Le point est incorrectement classifié comme un triangle.

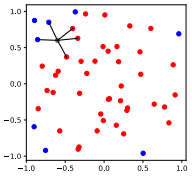
Remarque 3.
— *Dans l'exemple précédent, on peut voir que l'algorithme peut se tromper dans la classification, même pour un point déjà étiqueté.*
— *Le choix de la valeur de K peut changer les résultats obtenus. Il n'y a pas de méthode générale, mais on peut remarquer les faits suivants : si K est trop petit, le choix peut être trop sensible au bruit statistique ; si $K = N$, alors le résultat dépend uniquement de la répartition des classes des données étiquetées. Le choix de K peut être fait empiriquement, en utilisant les données étiquetées elles-mêmes.*

On remarque que le principe de l'algorithme KNN peut également servir pour faire de la régression : on peut identifier les k plus proches voisins et prédire avec la moyenne des classes associées à chaque valeur.

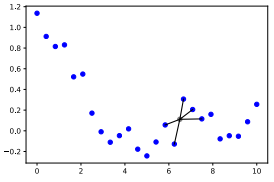
On peut ainsi :
— identifier $z_1, z_2, \dots, z_K$ des éléments distincts dans $\{x_1, \dots, x_N\}$ minimisant $\delta(x, z_i)$;
— renvoyer la moyenne $\frac{1}{K}\sum_{i=1}^{K} c(z_i)$ ou la moyenne pondérée $\dfrac{\sum_{i=1}^{K} \frac{c(z_i)}{\delta(x,z_i)}}{\sum_{i=1}^{K} \frac{1}{\delta(x,z_i)}}$.

Pour la moyenne pondérée, on utilise l'inverse de la distance pour donner plus de poids aux sommets les plus proches. On suppose alors que x n'est pas l'un des z_i.

Classification
Vote majoritaire



Régression
Moyenne



1.1 Représentation de Z par un arbre k-dimensionnel

La recherche naive des plus proches voisins présentée dans la partie précédente est fort couteuse. Pour obtenir les plus proches voisins de façon plus efficace, on peut représenter l'ensemble Z par une structure adaptée.

Attention le k de arbre k dimensionnel n'est pas le k des k plus proches voisins mais la dimension !

On parlera aussi d'arbre $k-d$ où k correspond à la dimension (d pour nos notations) et d est une abréviation de dimensionnel.

Nous allons commencer par rappeler quelques définitions géométriques :

Définition 1. *Un ensemble P de $\mathbb{R}^d$ sera appelé pavé s'il existe des scalaires $(a_1,\dots,a_d)$ et $(b_1,\dots,b_d)$ tels que $P=\{x\in\mathbb{R}^d : \forall 1\leq i\leq d, a_i\leq x_i\leq b_i\}$. Il s'agit par exemple d'un rectangle dans $\mathbb{R}^2$ et d'un parallélépipède rectangle dans $\mathbb{R}^3$.*

Définition 2. *Soit $x\in\mathbb{R}^d$, si on fixe une coordonnée $1\leq k\leq d$ et un réel ℓ alors on peut partitionner un pavé P en deux sous ensembles $P\cap\{x\in\mathbb{R}^d : x_k\leq\ell\}$ et $P\cap\{x\in\mathbb{R}^d : x_k\geq\ell\}$ qui sont eux même des pavés situés de part et d'autre de l'hyperplan affine $\{x\in\mathbb{R}^d : x_k=\ell\}$ normal au vecteur e_k (k-ieme vecteur de la base canonique de $\mathbb{R}^d$).*

En particulier : si on fixe un vecteur $z\in\mathbb{R}^d$ et une coordonnée $1\leq k\leq d$ alors $P\cap\{x\in\mathbb{R}^d : x_k\leq z_k\}$ et $P\cap\{x\in\mathbb{R}^d : x_k\geq z_k\}$ correspondent aux points de P situés de part et d'autre de l'hyperplan affine passant par z et de vecteur normal e_k.

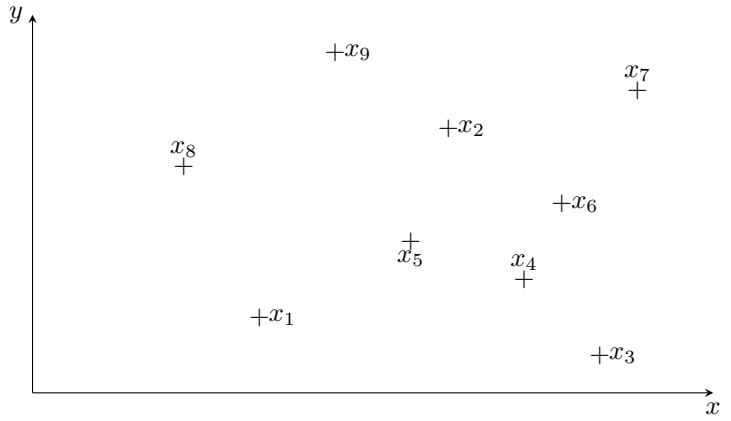
Définition 3. *Un arbre $k-d$ est un arbre binaire de recherche qui sépare l'espace des données qui est un pavé en deux sous-espaces qui sont eux même des pavés. Il facilite la recherche des plus proches voisins. Il sera étiqueté par les données $(x_1,\dots,x_N)$ d'entrainement.*

Un arbre $k-d$ satisfait la propriété suivante :
pour chaque noeud x de l'arbre, les points dans le sous arbre gauche et dans le sous arbre droit sont de part et d'autre de l'hyperplan passant par x et de vecteur normal e_{p+1} où p est la profondeur du noeud x modulo k (la dimension).

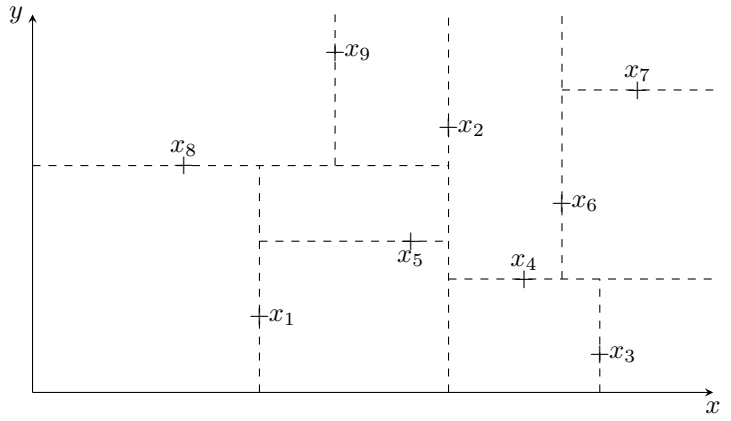
Afin d'obtenir un arbre équilibré, on choisira comme valeur de noeud l'élément dont la coordonnée selon laquelle on fait le découpage est médiane dans l'ensemble des valeurs considérées.

En dimension 2, on remarque que les profondeurs paires correspondent aux abscisses et les profondeurs impaires aux ordonnées.

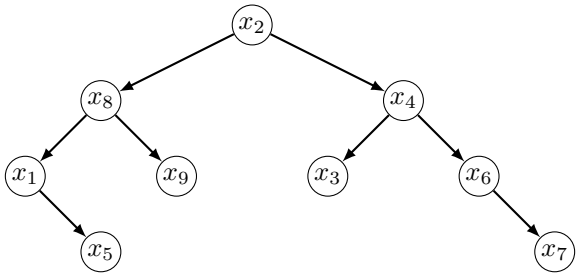
Exemple 2. *On suppose que $d=2$ et on cherche à créer un arbre d-dimensionnel pour le jeu de données suivant :*



On obtient alors le découpage spatial suivant, en alternant découpage vertical et découpage horizontal :



Ce qui donne l'arbre d-dimensionnel suivant :



Une implémentation possible est :

```
type point_t = float array;;
type arbrekd = Vide | Noeud of point_t * arbrekd * arbrekd;;

let arbrekd_creer donnees =
    let d = Array.length (List.hd donnees) in
    let rec arbrekd_creer_aux k donnees =
        if donnees = []
        then Vide
        else
            let pivot, reste = extrait_median donnees k in
                let inf, sup = List.partition (fun x -> x.(k) < pivot.(k)) reste in
                    Noeud(pivot,
                          arbrekd_creer_aux ((k+1) mod d) inf,
                          arbrekd_creer_aux ((k+1) mod d) sup)
    in arbrekd_creer_aux 0 donnees;;
```

Coût de construction d'un arbre $k-d$: avec un calcul de médiane en $O(N\log(N))$ on obtient une construction moyenne en $O(N\log^2(N)))$. Cependant cette construction n'est faite qu'une seule fois, il s'agit d'un prétraitement, la complexité significative est celle de la recherche de plus proches voisins d'un nouveau vecteur dont on veut prédire la classe.

Comment trouver les plus proches voisins à l'aide d'un arbre $k-d$?

Entrée : $x = (a_1, \dots, a_d) \in \mathbb{R}^d$
Début algorithme
  $FP \leftarrow$ file de priorité vide.
  $\ell \leftarrow 0$.
  Function `Traiter(`z`)`
    $\delta_z \leftarrow \delta(x, z)$.
    $p_{\max} \leftarrow$ Priorité_max(FP).
    Si $\ell < K$ ou $\delta_z < p_{\max}$ **Alors**
      Inserer dans FP l'élément z avec priorité δ_z.
      $\ell \leftarrow \ell + 1$.
    Si $\ell > K$ **Alors**
      Extraire l'élément de priorité maximale dans FP.
      $\ell \leftarrow \ell - 1$.
  Function `Explore(`A, p`)`
    Si A non vide **Alors**
      `Traiter(`*racine de* A`)`.
      $j \leftarrow (p \mod d) + 1$.
      $b_j \leftarrow$ valeur de la j-ème composante de racine de A.
      Si $a_j \leq b_j$ **Alors**
        $A_1 \leftarrow$ fils gauche de A.
        $A_2 \leftarrow$ fils droit de A.
      Sinon
        $A_1 \leftarrow$ fils droit de A.
        $A_2 \leftarrow$ fils gauche de A.
      `Explore(`A_1, $p+1$`)`.
      $p_{\max} \leftarrow$ Priorité_max(FP).
      Si $\ell < K$ ou $p_{\max} > |a_j - b_j|$ **Alors**
        `Explore(`A_2, $p+1$`)`.
  `Explore(`A, 0`)`.
  Renvoyer les éléments de FP.

Expliquée en français, l'idée est la suivante :

- On garde en mémoire une file de priorité contenant les plus proches voisins de x parmi les points étudiés, dans une limite de K au maximum.
- Chaque fois qu'on découvre un nouveau point a, on l'ajoute à la file de priorité si elle en contient moins que K, ou si ce nouveau point est plus proche de x que le voisin le plus loin. On extrait ce voisin le plus loin si nécessaire.
- On explore une partie de l'arbre d-dimensionnel selon l'idée suivante : pour chaque nœud a de l'arbre, on explore le fils gauche ou droit en premier selon où se trouve x par rapport à l'hyperplan de découpage en a. Après cette exploration, deux cas sont possibles :
  - soit on a déjà trouvé K voisins, et le plus loin d'entre eux est plus proche de x que ne l'est l'hyperplan de découpage, auquel cas on n'explore pas l'autre fils (car tous les points qui s'y trouvent seront trop loin) ;
  - sinon, on explore l'autre fils.

On peut montrer que la complexité d'une recherche des K plus proches voisins dans un arbre d-dimensionnel de taille N est en moyenne en $O(K(\log K + d) + d \log N)$ (même si elle peut atteindre $O(N(\log N + d))$ dans le pire cas).

Exemple 3. *On dispose de* $N = 10$ *points dans le plan, de coordonnées :*

$$x_1(11, 0.5)\,;\ x_2(8.5, 6)\,;\ x_3(7, 8.5)\,;\ x_4(9, 4)\,;\ x_5(15, 1.5)$$

$$x_6(14.5, 8)\,;\ x_7(3.5, 7)\,;\ x_8(3, 2.5)\,;\ x_9(14, 5.5)\,;\ x_{10}(2, 5)$$

1. *Représenter les points dans le plan et faire le découpage de construction de l'arbre* 2*-dimensionnel correspondant, puis dessiner cet arbre.*
2. *Détailler la recherche des* 2 *plus proches voisins de* $x = (4, 5)$ *et* $y = (8, 1)$.

1.2 Matrice de confusion

La matrice de confusion est un outil statistique pour mesurer l'efficacité d'un algorithme de classification. Elle indique pour chaque couple (i, j) de classes le nombre d'exemples appartenant à la classe i et/mais classés dans la classe j.

Définition 4. *Soit $f : \mathbb{R}^d \to \{1, \dots m\}$ une fonction correspondant à un algorithme de classification et $c : \mathbb{R}^d \to \{1, \dots, m\}$ la fonction exacte de classification. Pour un jeu de N données inconnues X et $i, j \in \{1, \dots, m\}^2$, on note m_{ij} le cardinal de $\{x \in X : f(x) = j$ et $c(x) = i\}$. La matrice de terme général m_{ij} est appelée matrice de confusion.*

Exemple 4. *On considère un jeu de 1000 points choisis aléatoirement dans l'ensemble du carré de l'exemple précédent auquel on applique l'algorithme KNN avec $K = 3$. On obtient la matrice de confusion suivante :*

$$\begin{pmatrix} 224 & 13 & 0 & 0 \\ 1 & 248 & 0 & 0 \\ 26 & 3 & 191 & 20 \\ 0 & 40 & 0 & 234 \end{pmatrix}$$

en supposant que les numéros corrects de classe sont organisés comme : $\begin{array}{|c|c|}\hline 2 & 4 \\ \hline 1 & 3 \\ \hline \end{array}$.

On peut interpréter de cette matrice que les éléments de la classe 2 sont presque toujours correctement bien identifiés (une seule erreur sur 249), contrairement à ceux des classes 3 et 4. Le taux d'erreurs total est $\frac{13+1+26+3+20+40}{1000} = 10,3\%$.

De cette matrice, on peut notamment déduire le taux de bonne prédiction (somme des valeurs sur la diagonale divisée par la somme des valeurs totales).

Dans le cas où il y a deux classes appelées Positifs, et Négatifs, cette matrice indique donc le nombre de vrais positifs, faux négatifs, faux positifs et vrais négatifs. On peut alors s'intéresser au taux de vrais positifs (aussi appelé *sensibilité*), au taux de vrais négatifs (aussi appelé *spécificité*) et la *précision* du test (nombre de vrais positifs sur nombre de positifs).

Pour construire une matrice de confusion, on sépare les données d'entraînement en deux ensembles : un vrai ensemble d'entraînement et un ensemble de test. On entraîne ensuite l'algorithme sur les données sélectionnées et on construit la matrice de confusion sur l'ensemble de test.

Lorsqu'on utilise des algorithmes d'apprentissage supervisé itératif (la lecture des données entraîne un ajustement des coefficients caractéristiques de l'algorithme : ici k et δ, on répète cette lecture plusieurs fois), on peut obtenir une situation de *surapprentissage* où l'algorithme obtient de très bons résultats de classification sur l'ensemble d'entraînement mais des résultats qui se dégradent pour l'ensemble test : l'algorithme se comporte comme s'il avait appris les données d'entraînement sans avoir réussi à généraliser. On peut se reposer sur la matrice de confusion pour déterminer à quel moment arrêter l'apprentissage sur les données d'entraînement.

2 Classification par arbre de décision

Un arbre de décision est un outil permettant de classifier ou de décider en fonction d'une succession de questions. Il s'agit d'un arbre dans lequel chaque nœud interne correspond à une question, chaque fils à une réponse à cette question et chaque feuille à une classe ou une décision finale. C'est un objet très simple et utilisé notamment dans les techniques de ventes de démarcheurs téléphoniques ou encore dans la représentation d'algorithmes simples. Pour classer un objet donné à partir d'un tel arbre, on le parcourt en suivant le choix obtenu à chaque nœud.

Pour construire l'arbre, il s'agit donc de construire itérativement un nœud en choisissant un test et en construisant ses fils jusqu'à ce que toutes les données d'entraînement relatives au nœud aient la même classe et qu'on ait donc une feuille.

Ici, on va décrire un algorithme permettant de construire un tel arbre à partir d'un jeu de données. Les questions (nœuds internes) correspondront à une coordonnées des valeurs d'entrainement (les x_i). Dans certains cas, ces valeurs forment un ensemble fini et on aura autant de sous arbres que de valeurs possibles et dans d'autres cas une coordonnée pourra avoir une infinité de valeurs et dans ce cas on aura des sous arbres qui correspondront à des intervalles.

On remarque que considérer les composantes dans un ordre quelconque n'est pas judicieux. En effet : considérons l'exemple où les $x_1 = (0, \dots, 0)$, $x_2 = (1, 0, \dots, 0), x_3 = (1, 1, 0, \dots, 0), \dots, x_N = (1, \dots, 1)$ avec $c(x_i) = 1, \forall i < N$ et $c(x_N) = 2$.

On va donc choisir l'ordre sur les coordonnées par rapport auxquelles construire l'arbre de décision afin de maximiser l'homogénéité de la classification obtenue par la partition. On veut choisir une coordonnée telle que les sous arbres obtenus aient des classes les plus homogènes possibles. Pour cela on va utiliser un algorithme glouton qui fait un choix optimal à chaque étape mais on a besoin d'une mesure de cette notion d'homogénéité.

2.1 Entropie

L'entropie est une mesure de l'incertidude, ainsi diminuer l'entropie revient à acquérir de l'information.

Définition 5. *Soit S un ensemble partitionné en k classes $C_1, \dots, C_k$. On définit l'entropie de Shannon de l'ensemble S par :*

$$H(S) = -\sum_{i=1}^{k} \frac{|C_i|}{|S|} \log(\frac{|C_i|}{|S|})$$

Ici, pour un jeu de données, la partition en classes est celle induite par les valeurs de classe des données. Ainsi, $C_j = \{x_i | y_i = j\}$.

Intuitivement, cette quantité mesure la "dispersion" dans les classes. On peut facilement observer les cas extrêmes : s'il n'y a qu'une seule classe alors l'entropie est nulle et si les classes sont toutes de même cardinal alors l'entropie est maximale. Plus cette quantité est grande, moins on ne dispose d'homogénéité relativement aux classes de ses éléments.

Ici on va donc choisir la coordonnée selon laquelle faire la partition des données afin d'obtenir le meilleur gain d'information c'est-à-dire l'entropie moyenne la plus faible des ensembles correspondant aux sous-arbres.

Définition 6. *Soit $k \in \{1, \dots d\}$ une coordonnée et $\{a_1, \dots, a_m\}$ les valeurs possibles de cette coordonnée. On définit le gain d'entropie d'un ensemble de données S par rapport à la coordonnée k par $G(S, a) = H(S) - \sum_{i=1}^{m} \frac{|S_{x_k=a_i}|}{|S|} H(S_{x_k=a_i})$ où $S_{x_k=a_i}$ est l'ensemble des données dont la $k$ieme coordonnée vaut a_i et dont on calcule l'entropie relativement aux classes du jeu de données.*

2.2 Algorithme ID3

On notera que le programme restreint l'utilisation de cet algorithme à l'obtention d'arbres de décision binaires c'est-à-dire qui correspondent à des problèmes où les vecteurs avec lesquels on travaille sont dans $\{0, 1\}^d$.

L'algorithme ID3 est un algorithme glouton qui vise à maximiser localement le gain d'information, il ne garantit pas d'obtenir le meilleur arbre de décision global.

Il se présente sous la forme suivante :

— si tous les éléments de l'ensemble E ont la même classe alors on crée une feuille de cette classe.
— si $E = \emptyset$, on crée une feuille correspondant à la classe majoritaire du nœud parent.
— si aucune variable ne permet de discriminer les éléments restants alors on construit une feuille de la classe majoritaire.
— sinon, on fixe k la coordonnée qui maximise le gain $G(E, k)$ et on construit un noeud k ayant pour fils les arbres construits récursivement sur les $E_{x_k=a+i}$ pour chaque valeur possible a_i.

Exemple 5. *Deux joueurs de tennis sont en vacances dans un hôtel de luxe. Après avoir observé leur routine pendant deux semaines où les joueurs ont parfois joué ou non, le personnel de l'hôtel qui garantit un service impeccable souhaite faire une prévision pour les jours à venir, pour anticiper quels jours vont jouer les joueurs en fonction des conditions météorologiques. Les jours de jeu en fonction de la météo sont les suivants :*

Ciel	*Température*	*Humidité*	*Vent*	*Ont-ils joué au tennis ?*
Soleil	*Chaud*	*Élevée*	*Faible*	*Non*
Soleil	*Chaud*	*Élevée*	*Fort*	*Non*
Nuages	*Chaud*	*Élevée*	*Faible*	*Oui*
Pluie	*Moyen*	*Élevée*	*Faible*	*Oui*
Pluie	*Frais*	*Normale*	*Faible*	*Oui*
Pluie	*Frais*	*Normale*	*Fort*	*Non*
Nuages	*Frais*	*Normale*	*Fort*	*Oui*
Soleil	*Moyen*	*Élevée*	*Faible*	*Non*
Soleil	*Frais*	*Normale*	*Faible*	*Oui*
Pluie	*Moyen*	*Normale*	*Faible*	*Oui*
Soleil	*Moyen*	*Normale*	*Fort*	*Oui*
Nuages	*Moyen*	*Élevée*	*Fort*	*Oui*
Nuages	*Chaud*	*Normale*	*Faible*	*Oui*
Pluie	*Moyen*	*Élevée*	*Fort*	*Non*