

Chapitre 18 : Intelligence artificielle et théorie des jeux

Table des matières

1	Introduction	2
2	Apprentissage supervisé	2
2.1	Algorithme des k plus proches voisins	2
2.1.1	Introduction	2
2.1.2	Algorithme des k plus proches voisins	2
2.1.3	Arbres k -dimensionnels	4
2.1.4	Recherche des k plus proches voisins dans un arbre k -d	5
2.2	Arbres de décision	6
2.2.1	Introduction	6
2.2.2	Définition (<i>arbre de décision</i>)	6
2.2.3	Exemple	7
2.2.4	Entropie de SHANNON	7
2.2.5	Algorithme ID3	10

1 Introduction

L'expression *intelligence artificielle* est une manière floue de désigner des algorithmes chargés comme tous les autres de résoudre des problèmes. Certains d'entre eux doivent jouer à des jeux, d'autres doivent répartir des données en plusieurs catégories (on parle de *classification*) ou déterminer des valeurs numériques associées à des paramètres d'entrée (on parle de *problème de régression*). Quelques traits communs à ces algorithmes sont l'usage d'heuristiques afin d'essayer d'obtenir des réponses les meilleures possibles en temps raisonnable et l'exploitation d'une grande quantité de données afin d'en construire une représentation (on parle de *l'apprentissage d'un modèle de données*) qui sera exploité pour construire la réponse de l'algorithme. Dans ce chapitre, on se limite à l'étude des problèmes de classification et de la théorie des jeux.

2 Apprentissage supervisé

2.1 Algorithme des k plus proches voisins

2.1.1 Introduction

Pour résoudre un problème de classification, la méthode de l'apprentissage supervisé consiste à exploiter des données dont on connaît déjà la classe afin de construire un algorithme de classification prenant les caractéristiques d'une donnée en entrée et renvoyant la classe à laquelle cette donnée appartient probablement.

Les données manipulées sont en général représentées par des points de \mathbb{R}^d ou d est souvent grand. Par exemple, si on veut reconnaître des caractères manuscrits, l'algorithme peut prendre en entrée une image de 28×28 pixels en 256 niveaux de gris contenant un scan du caractère manuscrit à reconnaître, donc la donnée est représentée par un point de $\llbracket 0 ; 255 \rrbracket^d$, où $d = 28^2 = 784$.

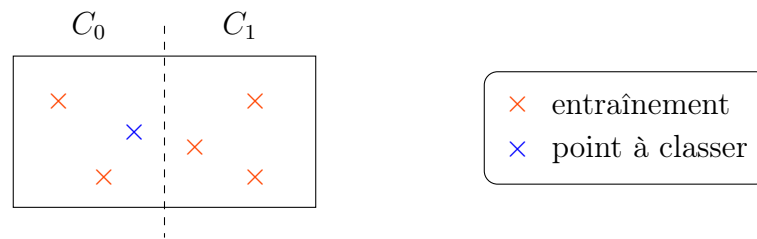
Si on veut répartir dans C classes des données de \mathbb{R}^d , le problème consiste à construire un algorithme réalisant une fonction $\mathbb{R}^d \rightarrow \llbracket 0 ; C - 1 \rrbracket$ en exploitant une heuristique s'appuyant sur un ensemble de couples $(x, y) \in \mathbb{R}^d \times \llbracket 0 ; C - 1 \rrbracket$, où y est la classe de x , appelées données d'entraînement.

2.1.2 Algorithme des k plus proches voisins

Idée : il est possible que des données proches appartiennent à la même classe, donc on pourrait pour un point donné, renvoyer la classe du point déjà étiqueté le plus proche.

Problème : la donnée d'entraînement la plus proche peut appartenir à une autre classe que celle du point, par exemple si les données d'entraînement sont mal réparties, ou bruitées, ou si le point considéré est proche de la frontière entre deux classes.





Pour éviter cet écueil, on considère plutôt la classe majoritaire parmi les classes des k données d'entraînement les plus proches du point d'entrée, pour un k fixé.

On parle alors de l'algorithme des k -plus proches voisins (k NN pour k -nearest neighbors).

Variables d'ajustement :

- En cas d'égalité, il faut choisir une classe, par exemple au hasard parmi les classes majoritaires.
- Le nombre k de voisins : si k est trop faible, l'algorithme sera trop sensible au bruit sur les données et si k est trop grand, l'algorithme renverra surtout la classe majoritaire parmi les données d'entraînement, donc effectue une mauvaise généralisation.
- La notion de distance : on utilise souvent la distance de MINKOWSKI

$$d(x, x') = \left(\sum_{i=1}^d |x_i - x'_i|^p \right)^{\frac{1}{p}}$$

qui donne la distance de MANHATTAN pour $p = 1$, et la distance euclidienne pour $p = 2$. Le programme se limite à la distance euclidienne.

Pour déterminer les k plus proches voisins d'un point donné, on peut exploiter une file de priorité :

Algorithm 1: Algorithme des k plus proches voisins

Input: données d'entraînement $(x_i, y_i)_{i \in [1 ; N]}$

Input: point à classer x

```

1  $F \leftarrow$  file de priorité max vide;
2 for  $i$  de 1 à  $k$  do
3    $\lfloor$  Insérer  $i$  dans  $F$  avec la priorité  $d(x, x_i)$ ;
4 for  $i$  de  $k + 1$  à  $N$  do
5   if  $d(x, x_i) < d(x, x_{\max F})$  then
6     Extraire le max de  $F$ ;
7      $\lfloor$  Insérer  $i$  dans  $F$  avec la priorité  $d(x, x_i)$ ;
8  $C \leftarrow \{C_i \mid i \in F\}$ ;
9 return un élément le plus fréquent de  $C$ ;
```

Complexité : $\mathcal{O}(N \log k)$ en temps, et $\mathcal{O}(k)$ en espace.

Remarque : si on a beaucoup de point à classer, cet algorithme est peu efficace car il nécessite de parcourir l'intégralité des données pour chaque point à classer. On pourrait

plutôt effectuer un prétraitement des données pour rendre plus efficace le calcul des k plus proches voisins d'un point donné.

2.1.3 Arbres k -dimensionnels

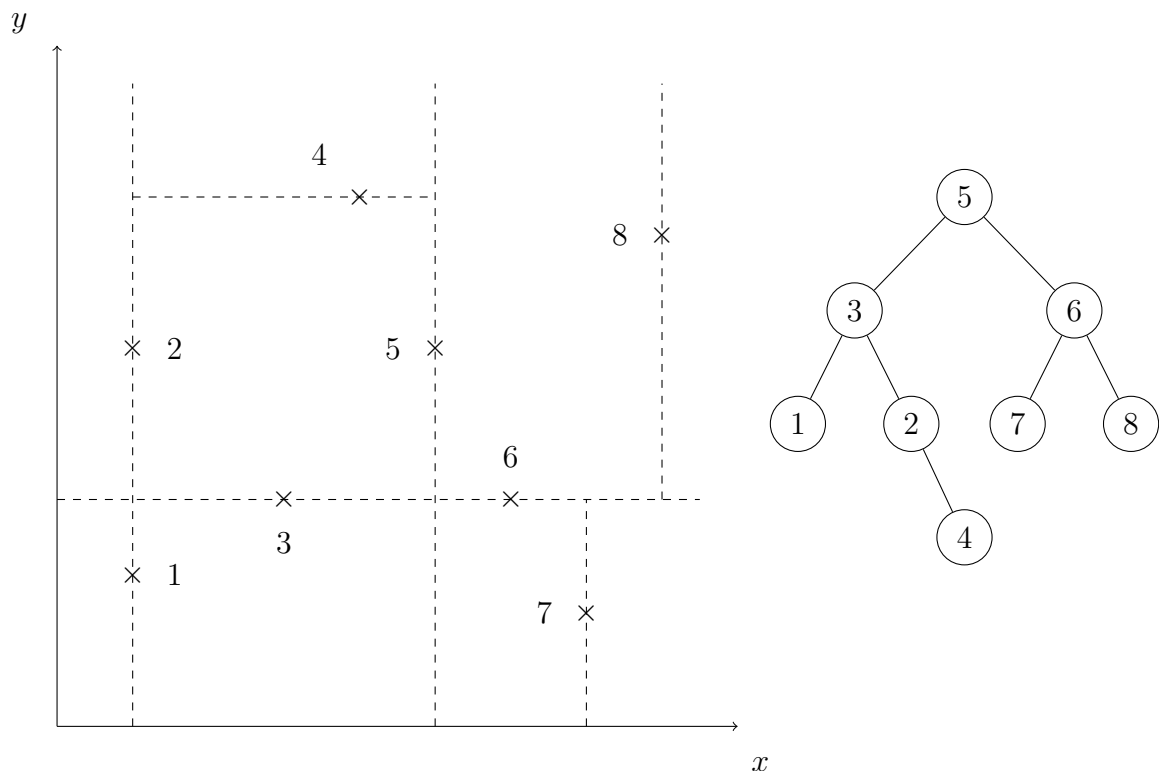
Attention : on ne parle pas du k de k NN, mais plutôt de la dimension de l'espace des données (d en 2.1.1, page 2), mais c'est la lettre k qui est utilisée dans la littérature.

- Définition : la structure d'arbre k -dimensionnel, ou arbre k -d, est une généralisation des la notion d'arbre binaire de recherche : un arbre binaire étiqueté par des éléments de \mathbb{R}^k est un arbre k -d si et seulement si pour tout nœud d'étiquette $x = (x_0, \dots, x_{k-1})$ de profondeur i ,

$$\begin{aligned} \forall x' = (x'_0, \dots, x'_{k-1}) \text{ étiquette du sous-arbre gauche, } & x'_j \leq x_j \\ \forall x' = (x'_0, \dots, x'_{k-1}) \text{ étiquette du sous-arbre droit, } & x'_j > x_j \end{aligned}$$

où $j = i \bmod k$.

- Exemple en dimension 2 :



- Remarque : dans cet exemple, on a fait en sorte de construire un arbre k -d équilibré en choisissant l'élément médian pour la coordonnée associée à la profondeur du nœud comme étiquette.

On écrit l'algorithme `créer_arbre` suivant :

Algorithm 2: créer_arbre

```

1 Function créer_arbre( $k, i, l$ ):
    Input: dimension  $k$ 
    Input: profondeur  $i$ 
    Input: liste de données  $l$ 

2   if  $l = []$  then
3     return l'arbre vide;
4   else
5     Extraire l'élément  $x$  de  $l$  médian pour la coordonnée  $i \bmod k$ ;
6      $l_<, l_> \leftarrow$  partition de  $l$  suivant le pivot  $x$ ;
7     return Noeud( $x$ , créer_arbre( $k, i + 1, l_<$ ), créer_arbre( $k$ ,
        ( $i + 1$ ),  $l_>$ ));

```

Complexité : à l'aide de l'algorithme de calcul de la médiane en temps linéaire (cf chapitre 7, 2.2), la complexité est celle du tri rapide dans le meilleur cas, i.e $\mathcal{O}(N \log N)$.

2.1.4 Recherche des k plus proches voisins dans un arbre k -d

Idee : pour trouver les k plus proches voisins d'un point $x \in \mathbb{R}^d$ dans un arbre k -d T de dimension d , on procède initialement comme pour la recherche de x dans un ABR (en comparant la bonne coordonnée à chaque profondeur) puis on remonte dans l'arbre en sélectionnant les k voisins, parfois en redescendant des un sous-arbre que l'on avait ignoré.

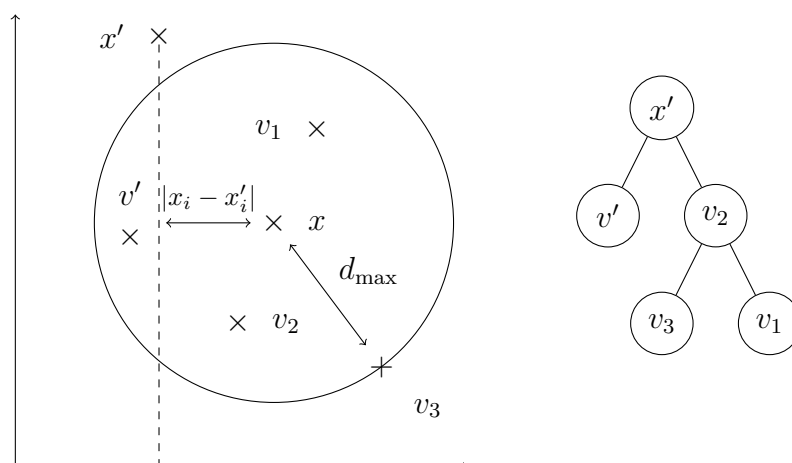
Exemple : on suppose être au niveau d'un nœud d'étiquette x' de profondeur i et tel que $x_i > x'_i$.

On cherche donc récursivement les k voisins dans le sous-arbre droit.

S'il y a moins de k nœuds dans ce sous-arbre, il faudra considérer x' comme voisin et peut-être aussi les nœuds du sous-arbre gauche.

Même si l'appel récursif sélectionne k voisins, on peut devoir considérer x' ou les nœuds du sous-arbre gauche si $|x_i - x'_i|$ est inférieure à la distance maximale entre x et les voisins sélectionnés.

Par exemple, si $k = 3, d = 2$, et i correspond aux abscisses,



D'où l'algorithme :

Algorithm 3: recherche_voisins, visite	
1	Function recherche_voisins(x, T, k):
2	$F \leftarrow$ file de priorité max vide;
3	visite($F, x, T, 0, k$);
4	return les éléments de F ;
5	Function visite(F, x, T, i, k):
6	if $T = \text{Noeud}(x', k, r)$ then
7	if $x_i \leq x'_i$ then
8	$t_1, t_2 \leftarrow l, r$;
9	else
10	$t_1, t_2 \leftarrow r, l$;
11	visite($F, x, t_1, i + 1 \bmod d, k$);
12	if $ F < k$ ou priorité max $F \geq x_i - x_j $ then
13	if $d(x, x') < \text{priorité max } F$ then
14	Extraire max F ;
15	Insérer x' dans F avec la priorité $d(x, x')$;
16	visite($F, x, t_2, i + 1 \bmod d, k$);

Dans le pire cas, on visite les N nœuds de l'arbre (donc on n'a rien gagné par rapport au premier algorithme) mais le plus souvent on ne visite que de l'ordre de $\log N$ nœuds.

2.2 Arbres de décision

2.2.1 Introduction

Un arbre de décision est un outil permettant d'implémenter un algorithme de classification dont le fonctionnement est le suivant : étant donné un point à classer, on descend récursivement dans l'arbre en effectuant à chaque nœud un test sur une coordonnée dont le résultat détermine la branche à parcourir. La feuille atteinte donne la classe du point.

2.2.2 Définition (*arbre de décision*)

Un *arbre décision* est un arbre étiqueté tel que les étiquettes des nœuds internes sont des coordonnées et celles des feuilles sont des classes, de telle sorte que les fils d'un nœud donné correspondent aux différentes valeurs possibles pour la coordonnée associée au nœud.

Si la coordonnée est *catégorique*, i.e ne peut prendre qu'un nombre fini de valeur, alors il y a autant de fils que de valeurs. Si la coordonnée est *numérique*, les fils correspondent à des intervalles de valeurs disjoints.

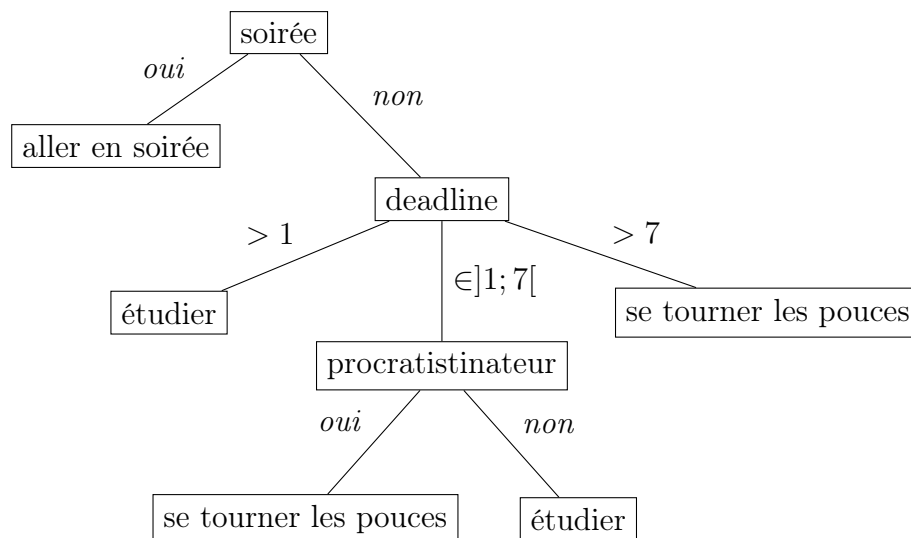


2.2.3 Exemple

On veut savoir comment passer la soirée, sachant qu'il y a trois activités possibles : aller en soirée, étudier, ou se tourner les pouces. On représente les étudiants par des triplets dont les coordonnées sont :

- un booléen indiquant s'il y a une soirée organisée par son cercle d'amis ;
- Un réel > 0 indiquant le nombre de jour avant la prochaine deadline (exemple : devoir) ;
- Un booléen indiquant si l'étudiant est procrastinateur.

On pourrait construire l'arbre suivant :



L'étudiant $(V, 0.2, 5, F)$ va en soirée

L'étudiant $(F, 4, V)$ va se tourner les pouces.

Remarque : dans le cadre de l'apprentissage supervisé, la question est de savoir comment construire un arbre de décision à partir de données étiquetées. Il faut donc choisir la structure de l'arbre et les tests effectués à chaque nœud. Il est possible de construire un arbre qui classe parfaitement toutes les données d'entraînement ou alors de s'auto-riser des erreurs pour tenir compte d'un éventuel bruit sur les données d'entraînement.

2.2.4 Entropie de SHANNON

- Idée : pour construire l'arbre de décision, une idée serait de placer en racine la coordonnée qui discrimine au mieux entre les différentes classes pour le jeu de données et de construire les sous-arbres récursivement. On utilise pour cela la notion d'*entropie de SHANNON*, qui est une mesure de l'information contenue dans un jeu de données.

- Définition (*entropie de SHANNON*) : on considère un ensemble S de N données réparties dans C classes C_0, \dots, C_{C-1} .

L'entropie de SHANNON de S est la quantité

$$H(S) = - \sum_{i=0}^{C-1} \frac{|C_i|}{N} \log_2 \frac{|C_i|}{N}$$

- Remarque : L'entropie de SHANNON est l'espérance du nombre de bits d'informations que l'on obtient en tirant aléatoirement uniformément un point de S .

En effet, $\frac{|C_i|}{N}$ est la probabilité d'obtenir un élément de la classe C_i , et $-\log_2 \frac{|C_i|}{N} = \log_2 N - \log_2 |C_i|$ est la différence entre le nombre de bits nécessaires pour représenter l'intégralité des données et le nombre de bits nécessaires pour distinguer entre elles les données de la classe C_i . C'est donc le nombre de bits qui restent afin de donner de l'information sur l'appartenance à la classe C_i .

En particulier,

- Si $C = 1$, alors $H(S) = 0$: la représentation des données sert uniquement à les distinguer entre elles et n'apporte aucune information sur la classe.

- L'entropie est maximale lorsque les données sont uniformément réparties $\left(|C_i| = \frac{N}{C}\right)$.

$H(S) = -\log_2 C$, et on obtient autant de bits que nécessaire pour distinguer les C classes.

- Définition (*gain d'un attribut*) : On considère un ensemble S de N données en dimension d , et une coordonnée (ou un attribut) $i \in \llbracket 1 ; d \rrbracket$ dont on note m le nombre de valeurs possibles.

Le *gain* de l'attribut i est la quantité

$$G(S, i) = H(S) - \sum_{j=1}^m \frac{|S_j|}{N} H(S_j)$$

où S_1, \dots, S_m est la partition de S suivant les valeurs de la coordonnée i .

- Remarque : le gain de l'attribut correspond à l'espérance de la perte d'entropie lorsque l'on fixe la valeur de l'attribut.

$$\begin{aligned} G(S, i) &= H(S) - \sum_{j=1}^m \frac{|S_j|}{N} H(S_j) \\ &= \sum_{j=1}^m \frac{|S_j|}{N} H(S) - \sum_{j=1}^m \frac{|S_j|}{N} H(S_j) \\ &= \sum_{j=1}^m \frac{|S_j|}{N} (H(S) - H(S_j)) \end{aligned}$$

où $\frac{|S_j|}{N}$ est la probabilité de tirer un point dont la coordonnée i prend la $j^{\text{ème}}$ valeur, et $H(S) - H(S_j)$ correspond à la différence entre l'entropie du jeu de données initial et l'entropie des données restantes après le choix de la $j^{\text{ème}}$ valeur.



Plus un attribut est discriminant pour les classes, plus l'entropie des données restantes après le choix d'une valeur pour l'attribut doit être faible. On cherche donc à maximiser le gain de l'attribut.

- Proposition (Inégalité de GIBLS) :

Si $(p_i)_{i \in \llbracket 0 ; k-1 \rrbracket}$, et $(q_i)_{i \in \llbracket 0 ; k-1 \rrbracket}$ sont deux distributions de probabilité sur un ensemble de cardinal k , alors

$$-\sum_{i=0}^{k-1} p_i \log_2(p_i) \leq -\sum_{i=0}^{k-1} p_i \log_2(q_i)$$

□

$\forall x \in \mathbb{R}_+^*$, $\ln(x) \leq x - 1$ (concavité de \ln).

Donc

$$\begin{aligned} \sum_{i=0}^{k-1} p_i \ln\left(\frac{q_i}{p_i}\right) &\leq \sum_{i=0}^{k-1} p_i \left(\frac{q_i}{p_i} - 1\right) \\ &= \sum_{i=0}^{k-1} (q_i - p_i) \\ &= 1 - 1 = 0 \end{aligned}$$

Donc

$$\sum_{i=0}^{k-1} p_i \ln q_i \leq \sum_{i=0}^{k-1} p_i \ln p_i$$

ce qui conclut car $\frac{-1}{\ln 2} < 0$. ■

- Corollaire :

Si S est un ensemble de N données en dimension d réparties dans les clauses C_0, \dots, C_{C-1} , et si $i \in \llbracket 1 ; d \rrbracket$ est un attribut pouvant prendre une valeur, alors $G(S, i) \geq 0$.

□

$$\begin{aligned} G(S, i) &= H(S) - \sum_{j=1}^m \frac{|S_j|}{N} H(S_j) \\ &= -\sum_{k=0}^{C-1} \frac{|C_k|}{N} \log_2 \frac{|C_k|}{N} - \sum_{j=1}^m \frac{|S_j|}{N} \left(-\sum_{k=0}^{C-1} \frac{|C_k \cap S_j|}{|S_j|} \log_2 \frac{|C_k \cap S_j|}{|S_j|} \right) \\ &= \sum_{k=0}^{C-1} \sum_{j=1}^m \frac{|C_k \cap S_j|}{N} \log_2 \frac{|C_k|}{N} - \left(-\sum_{j=1}^m \sum_{k=0}^{C-1} \frac{|C_k \cap S_j|}{N} \log_2 \frac{|C_k \cap S_j|}{|S_j|} \right) \end{aligned}$$

Or

$$\log_2 \frac{|C_k \cap S_j|}{|S_j|} = \log_2 \frac{|C_k \cap S_j|}{N} = -\log_2 \frac{|S_j|}{N}$$

Donc

$$G(S, i) = - \sum_{k=0}^{C-1} \sum_{j=1}^m \frac{|C_k \cap S_j|}{N} \left(\log_2 \frac{|C_k|}{N} + \log_2 \frac{|S_j|}{N} \right) - \left(- \sum_{j=1}^m \sum_{k=0}^{C-1} \frac{|C_k \cap S_j|}{N} \log_2 \frac{|C_k \cap S_j|}{N} \right)$$

On note $(p_{j,k})_{(j,k) \in [1; n] \times [0; C-1]} = \left(\frac{|C_k \cap S_j|}{N} \right)_{j,k}$ et $(q_{j,k})_{j,k} = \left(\frac{|C_k| |S_j|}{N^2} \right)_{j,k}$ et on remarque que $(p_{j,k})_{j,k}$ et $(q_{j,k})_{j,k}$ sont des distributions de probabilité, donc l'inégalité de GIBLS donne :

$$- \sum_{j,k} p_{j,k} \log_2 p_{j,k} \leq - \sum_{j,k} p_{j,k} \log_2 q_{j,k}$$

d'où $G(S, i) \geq 0$. ■

2.2.5 Algorithme ID3

- Principe : l'algorithme ID3 (pour *Iterative Dichotomiser 3*) consiste à construire un arbre de décision en choisissant pour racine l'attribut de gain maximal et en construisant récursivement les sous-arbres en considérant comme jeu de données la partition des données selon la valeur de l'attribut choisi.
- Cas binaire : on énonce l'algorithme ID3 dans le cas où les attributs sont binaires.

Algorithm 4: ID3, cas binaire

Input: L'ensemble S des données

Input: L'ensemble A des attributs que l'on s'autorise à utiliser dans les nœuds

```

1 if  $S = \emptyset$  then
2   | Construire une feuille contenant la classe la plus représentée parmi les
   | données qui ont servi à construire son père;
3 else if tous les éléments de  $S$  appartiennent à la même classe  $C$  then
4   | Construire une feuille contenant  $C$ ;
5 else if  $A = \emptyset$  then
6   | Construire une feuille contenant la classe la plus représentée parmi les
   | données de  $S$ ;
7 else
8   |  $i \leftarrow$  attribut de  $A$  qui maximise  $G(S, i)$ ;
9   |  $S_0 \leftarrow$  données de  $S$  dont l'attribut  $i$  vaut 0;
10  |  $S_1 \leftarrow$  données de  $S$  dont l'attribut  $i$  vaut 1;
11  | Construire le nœud d'étiquette  $i$ , de sous-arbre gauche  $\text{ID3}(S_0, A \setminus \{i\})$ ,
   | de sous-arbre droit  $\text{ID3}(S_1, A \setminus \{i\})$ ;

```

- Exemple :



Attributs				Classe
E	C	F	L	R
1	1	1	1	1
1	1	1	0	0
1	1	0	1	1
1	1	0	0	0
1	0	1	1	1
1	0	1	0	0
1	0	0	1	0
1	0	0	0	0
0	0	1	1	0
0	0	1	0	0
0	0	0	1	1
0	0	0	0	0

