Extraction de Connaissances

Ch. II & III : Représentation et Préparation des données

Ecole Centrale de Lyon

2017-2018

Alexander Saidi

Octobre 2017

1

2 Concepts, instances et attributs

- Rappel de l'essentiel du cours précédent
- Instance, attribut, concepts
- Préparation des données

A consulter (par vous mêmes) :

- Représentation de connaissances par diverses méthodes d'apprentissage
- Interprétation des résultats

2.1 ECD : un champ multidisciplinaire

Rappel :

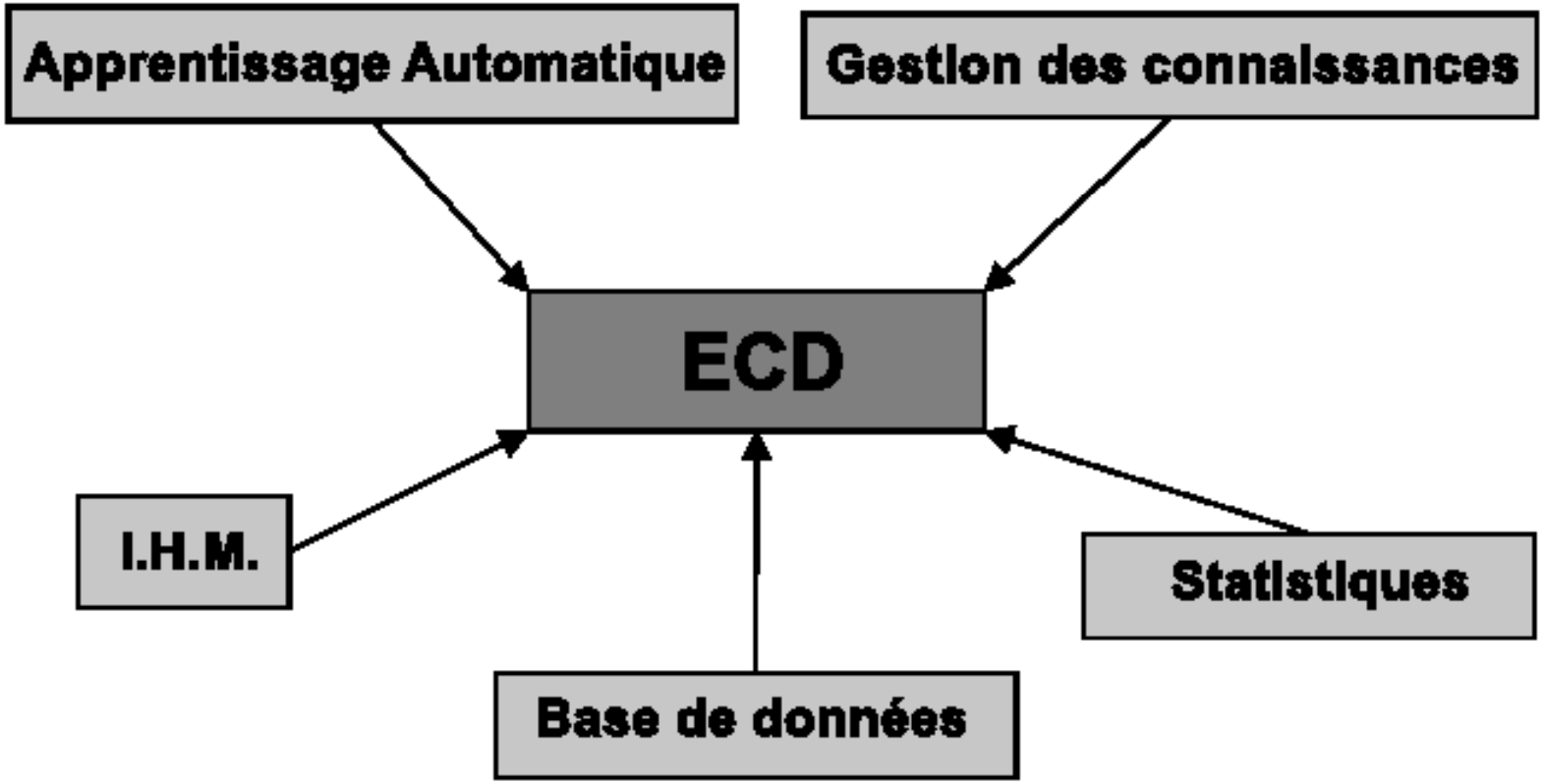


FIGURE 2.1 – ECD et autres disciplines

2.2 Généralisation (Induction) comme Recherche

- L'apprentissage automatique : apprentissage de **Concepts**.
- Sémantique : (des idées intuitives !) : généralisation par la recherche/fouille
  - ➥ *Recherche dans l'espace de description de concepts possibles pour en trouver un qui correspond aux données.*
  - ➥ *On énumère tous les ensembles de règles possibles puis on prend celui qui convient à notre base de données .*
- Trop complexe : le nombre d'ensembles possibles sans limite (mais **fini**).
  - → Le nombre d'attributs / valeurs par attribut / règles sont **finis**
  - → Avec discrétisation des numériques
- L'exemple météo : il y 4 x 4 x 3 x 3 x 2 = 288 possibilités pour chaque règle.
  - ➥ Même si l'on se limite aux ensemble de 14 règles (14 exemples) $288^{14} \backsimeq 2.7 \times 10^{34}$

• **La méthode :** énumérer les règles puis les tetster :

⇒ Un **exemple positif** élimine les concepts qui le contredisnet (qu'il ne "match" pas)

⇒ Un **exemple négatif** élimine ceux qu'il "match".

➥ Donc, chaque exemple peut réduire l'ensemble des descriptions.

• S'il en reste (un ou plusieurs), c'est le bon ensemble de concepts recherché

• Un nouvel exemple doit "matcher" tous les concepts restants

➥ S'il rate le "match", l'exemple sera hors le concept ($\rightarrow$ classification).

➥ S'il match certain et pas d'autres $\rightarrow$ ambiguïté.

➥ Dans ce cas, si la classe du nouvel exemple est connue d'avance, on peut élaguer l'ensemble de concepts qui le classifient mal (sans la remise en cause).

Comment fait-on d'habitude ?

➥ Outre les méthodes et modèles purement statistiques ...

➥ La plupart des méthodes de DM voit la généralisation comme recherche

• Au lieu d'énumérer et d'enlever celle qui ne convient pas :

Appliquer un processus *Hill Climbing* dans l'espace des concepts pour trouver la description qui "match" le mieux les exemples (selon un critère prédéfini).

➥ Processus qui avance de proche en proche

➥ *Hill Climbing* : le maximum local peut être différent du maximum global

➥ La plupart des algorithmes utilise des heuristiques

➥ On ne peut garantir la (meilleure intrinsèquement) description.

2.2.1 Enumération de l'espace de concepts : Espace de Versions

• L'espace de description de concepts consistents

• Complètement défini par 2 ensembles :

⇒ **L** : les descriptions *les plus spécifiques* qui couvrent tous les exemples positifs et aucun exemple négatif (de manière spécifique).

⇒ **G** : les descriptions *les plus générales* qui ne couvrent aucun exemple négatif mais tous les exemples positifs (de manière générale).

• **L** (least) et **G** (greatest) general descriptions

• On a juste besoin de maintenir et mettre à jour **L** et **G**

• **Inconvénients :** C'est encore coûteux en temps de calcul et..

Permet une définition théorique et claire mais ne résout pas de problème pratique !

□ **Exemple :**

Soit le vocabulaire : couleurs ∈ {rouge, vert}, animaux ∈ {vache, poule}

Ex. positifs	Ex. négatifs	L	G
<>		{ }	{< * , * >}
<vache, verte>		{<vache, verte>}	{< * , * >}
	<poule, rouge>	{<vache, verte>}	{<*, verte>, <vache, *>}
<poule, verte>		{<*, verte>}	{<*, verte>, <vache, *>}

• Si l'on ajoute <vache, rouge> ? :

→ Si ex. positif : ajouter <vache, *> à L

→ Si ex. négatif : retirer <vache, *> de G


Inconsistance si les exemples positifs et négatifs se contredisent.

2.2.2 Algorithme candidate-elimination

Initialiser L et G

Pour tout exemple e :

Si e est positif :

Supprimer tout element de G qui ne couvre pas e

Pour tout $r \in L$ qui ne couvre pas e :

Remplacer r par toutes ses generalisations les plus spécifiques qui couvrent e

et qui sont plus specifiques que les elements dans G

Supprimer tout element de L qui est plus général que les autres elements de L

Si e est negatif :

Supprimer tout elements de L qui couvre e

Pour tout element $r \in G$ qui couvre e :

Remplacer r par toutes ses spécialisations les plus générales qui ne couvre pas e

et qui sont plus généraux les elements dans L

Supprimer tout element de G qui est plus spécifiques que les autres elements de G

2.2.3 Défis et Challenges de l'Extraction de Connaissances

- Echelle Larges (Scalability), WEB

- Réduction de Dimensions (complexité)

- Données Complexes et hétérogènes (texte + image + son + vidéo)

  ➥ e.g. classification pages WEB

- Qualité de données

- La propriété et la distribution des données

- Préservation de confidentialité (anonimisation)

- Streaming de données (on line, WEB, ...)

- etc...

2.3 Le processus EC : les étapes

1. Assemblage des données
2. Présentation au logiciel DM
3. Interpréter les résultats (& expertise)
4. Appliquer les résultats aux nouvelles instances .

• **Assemblage de données** :

→ Les données réparties, sur plusieurs DBs, multi sites.

→ Il faut souvent (au moins) quelques milliers d'instances pertinentes.

→ Leur représentativité.

→ Phase longue, difficile mais importante.

→ Les données peuvent être accédées via un entrepôt de données, une base de données ou même une feuille de calculs (WEB et DM Textuel, ...)

N.B. : Entrepôt de données (DataWare house) : données sur le même sujet (e.g. les clients) ; données optimisées pour les transactions, redondance possibles, ...

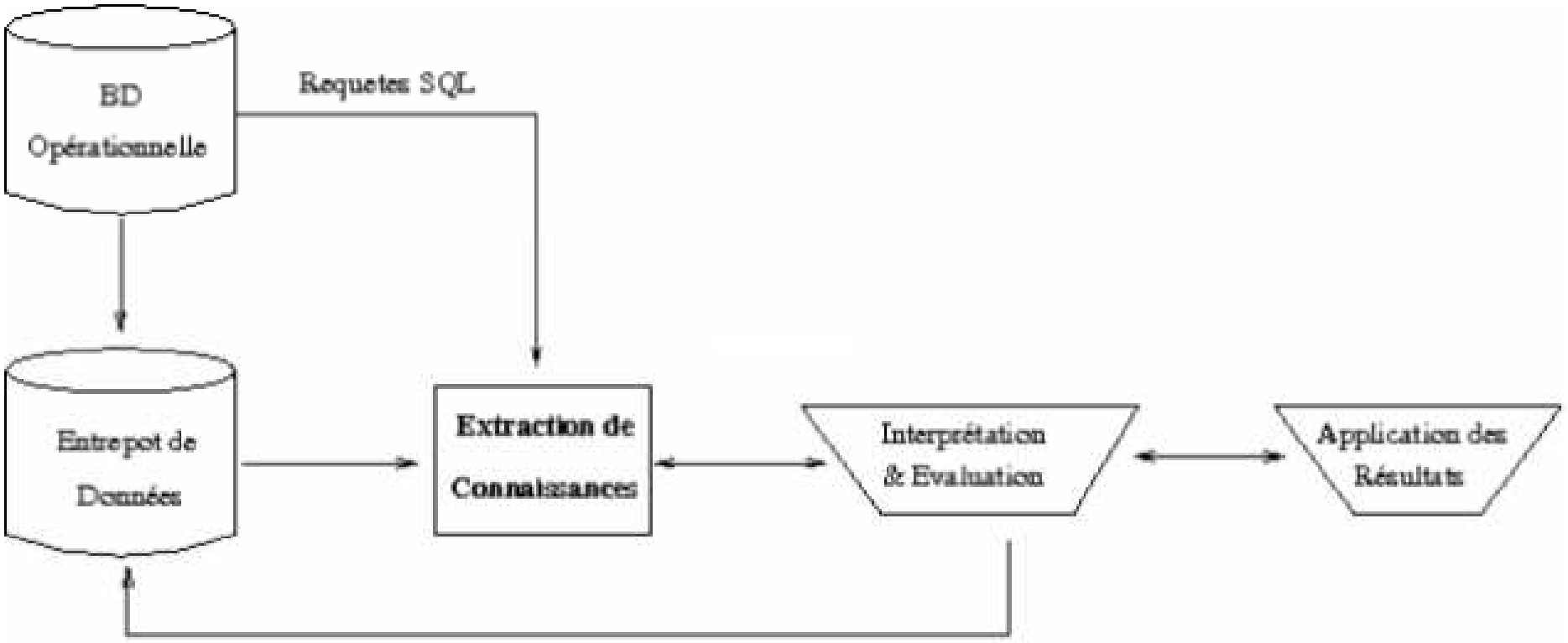


FIGURE 2.2 – Processus simplifié de DM

• **Méthodes DM** :

→ Supervisé ou non ?

→ Quelles données pour l'apprentissage / test ?

→ Quels attributs utiliser ? (c'est une aide)

→ Paramètres d'apprentissage (selon la méthode : Nb. clusters, support, confiance, ..)

• **Interprétation des résultats** :

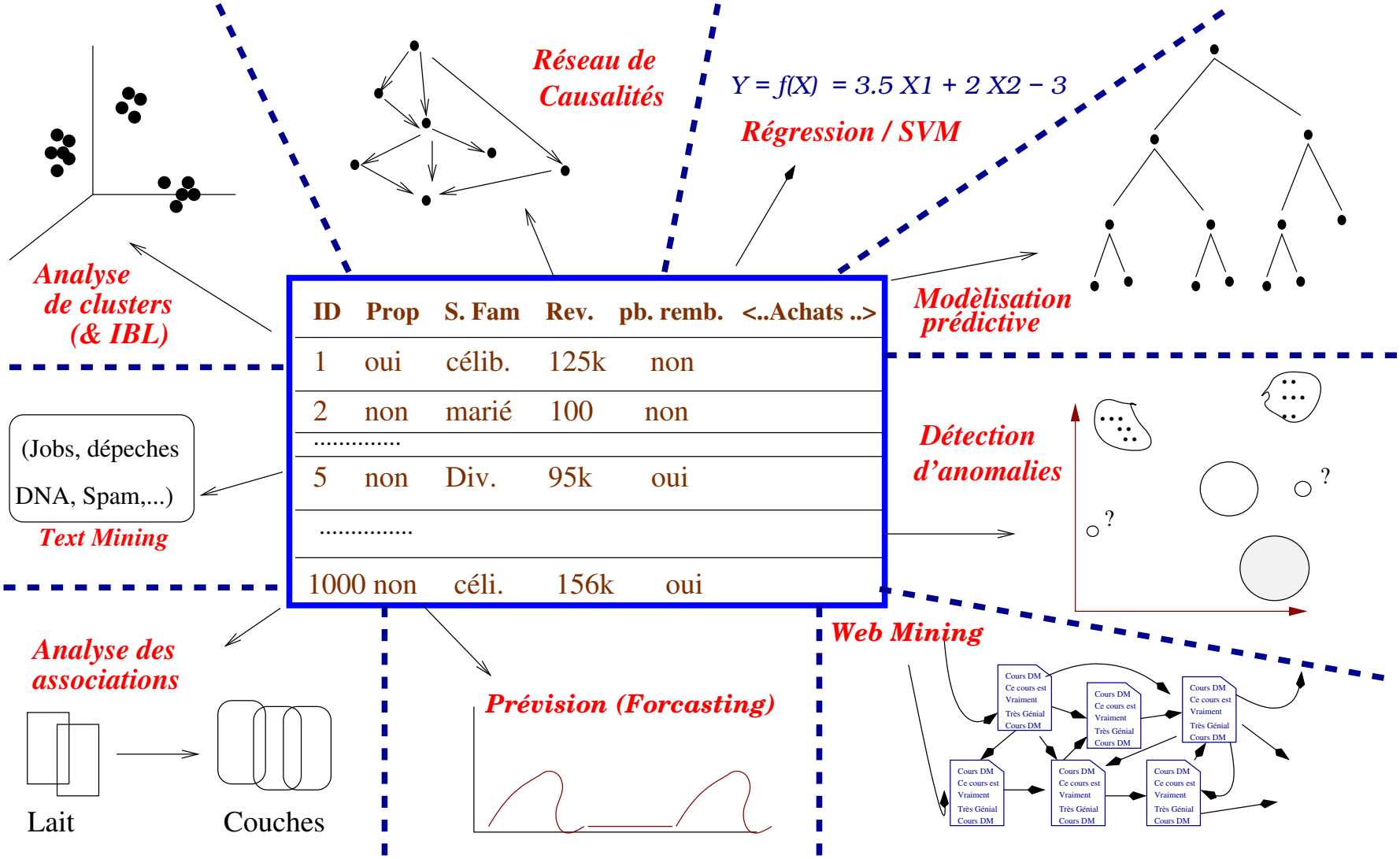
Découvertes utiles ? Itération de DM pour ajouter des attributs/instances, ...

• **Application des Résultats** aux nouvelles instances .

→ Si l'on découvre que le produit X est presque toujours acheté avec le produit Y,

→ E.g., on découvre que ceux qui achètent les jeudis des couches pour bébé achètent aussi de la bière (WE en famille !).

2.4 Principales sorties de l'EC (rappel)



2.5 Les Entrées, Sorties, Instances

- **Les entrées** : instances et attributs
  - ➥ La fonction (EC) : **Entrées** $\rightarrow$ **Concepts**
  - ➥ Connaître ces entrées est plus important que le traitement !
- **Les Sorties = Concepts** : ce qui doit être appris = le résultat de l'apprentissage .
  - ➥ Doit être intelligible, compréhensible, discuté, critiqué.
  - ➥ Il est aussi opérationnel $\rightarrow$ doit pouvoir être appliqué aux données.
- **Instances** :
  - ➥ la forme des data en entrée (valeurs nominales/catégorielles pour les attributs),
  - ➥ chaque instance est individuelle et indépendante
  - ➥ chaque instance représente un exemple de ce qui doit être appris.

2.6 Préparation des données

• Récupération, intégration, normalisation, agrégation, assemblage, passage aux Bool.

Formats courants :

⇒ format **excel** (colonnes séparées par *tab* ou *virgule* etc.)

⇒ format **ARFF** (utilisé dans **Weka** et autres Extraction de Connaissances)

⇒ Autres formats propriétaires

2.6.1 Exemple format ARFF

```
% ARFF file for weather data with some numeric features

@relation weather

@attribute outlook {sunny, overcast, rainy}

@attribute temperature numeric

@attribute humidity numeric

@attribute windy {true, false}

@attribute play? {yes, no}

@data

sunny, 85, 85, false, no

sunny, 80, 90, true, no

overcast, 83, 86, false, yes

...
```

2.6.2 Normalisation des valeurs numériques

- Données numériques normalisées $\rightarrow$ ramenées dans un intervalle décidé (e.g. [0..1]).
- Exemples de normalisation : (x_i est une valeur)

  - on divise x_i par le $Max(x_i)$: $\boxed{\frac{x_i}{Max(x_i)}}$

  - on soustrait le min puis on divise par la distance max-min :

$$\boxed{\frac{x_i - Min(x_i)}{Max(x_i) - Min(x_i)}}$$

  - **Standardisation statistiques** : centrées réduites

    - on calcule la moyenne μ et l'écart type σ,

    - on enlève μ de toute valeur, puis diviser par σ : $\boxed{\frac{x_i - \mu}{\sigma}}$

    ➥ Pour les valeurs normalisées : moyenne=0 et écart type=1.

2.6.3 Les valeurs manquantes

- Valeurs *manquantes, fausses, inconnues, non pertinentes*, ...
- **Causes** :
  - ➥ données non produites en vue de DM,
  - ➥ erreurs de frappe,
  - ➥ changement de méthode de relevée,
  - ➥ fusion de plusieurs DBs différentes,
  - ➥ impossibilité de mesure, ...
- Parfois, le manque veut dire quelque chose :
  - ➥ Une valeur non mesurée sur une plante morte
  - ➥ quelque chose de plus que la simple absence :
    - ➥ morte avant la fin des mesures → temps important ?

../..

2.6.4 Valeurs erronées

- La présence d'une valeur négative alors que tout doit être positif

  $\Rightarrow$ par fois, -1 veut dire "absent" !

  ➥ Elle peut venir d'erreur de frappe/saisi (c.f. attribut nominal)

  ➥ Le "-" peut provoquer une nouvelle (fausse) valeur possible

  ➥ Ou un synonyme (*coca* et *coca-cola*, ...) !

- **outliers** dans des données numériques : souvent détectables.

- **Les redondances** et données dupliquées

  ➥ La répétition peut influencer les résultats

  ➥ Données plus importantes car 2 fois présentes ?

Le format ARFF permet de vérifier les types.

../..

3 Représentation de connaissances (KR)

Chapitre 3

(lecture pour les élèves)

3.1 Représentation des patterns Structurelles

• La sortie d'un système DM = connaissances (concepts, patterns)

• Structure Importante : $\rightarrow$ *Knowledge Representation*.

• Différentes manières de représenter la sortie ($\simeq$ une par technique).

• Comment représenter la sortie est $\rightarrow$ méthode d'inférence ?

• Exemples de sorties : **arbre de décision , règles de classification , équation** ...

Mais aussi :

• On peut avoir des règles plus complexes : *exceptions, relations entre des attributs*, ...

• Des formes particulières d'arbres pour prédire des valeurs numériques

• Les représentations à base d'instances s'intéressent beaucoup plus aux exemples qu'aux règles qui gouvernent les valeurs d'attributs.

• Quelques schémas produisent des **clusters** d'instances...

3.2 Tables de décision

• La plus simple des sorties : comme l'entrée = une **table de décision**.

⇒ On place la classe/décision dans la table

➥ Cette table remplace des règles

Outlook	**Humidity**	**Play**
Sunny	High	No
Sunny	Normal	Yes
Overcast	High	Yes
Overcast	Normal	Yes
Rainy	High	No
Rainy	Normal	No

TABLE 3.1 – Une table de Décision pour l'exemple météo

• "météo" : on regarde les conditions du temps pour dire si l'on joue ou pas.

⇒ Certains attributs pas importants pour la sortie

→ la table sera plus simple.

→ Lesquels ?

3.3 Arbres de décision

• La stratégie **Diviser pour Conquérir** (divide and conquer) → arbre de décision .

➥ Cf. les exemples lentilles et négociations salariales, ...

• Parcours d'un arbre de décision , test d'attribut p/r à une constante à chaque noeud.

• Autre : comparer deux attributs ou appliquer une fonction sur les attributs.

⇒ Une feuille de l'arbre donne :

→ une classification qui s'applique aux instances qui arrivent à ce noeud,

→ un ensemble de classification

→ une distribution de probabilités sur toutes les instances.

• Pour classifier une nouvelle instance, ... on fait les tests noeud par noeud...

➥ Une nouvelle instance trouve (ou pas) de chemin !

Cas des attributs nominaux et Numériques :

• Attribut **nominal** $\rightarrow$ le nombre de fils d'un noeud = valeurs possibles .

➥ le test ne sera pas refait plus bas dans l'arbre.

• Parfois, les valeurs de l'attribut sont divisées en 2 sous-ensembles :

$\Rightarrow$ On prend l'une des 2 branches selon le sous-ensemble dans lequel on tombe.

$\Rightarrow$ L'attribut peut être testé plus d'une fois dans l'arbre.

• Attribut **numérique** : comparaison {=, >, <} p/r à une constante/attribut/...

➥ Tests multiples dans l'arbre possible.

➥ Tests à 3 branches possibles (transformable en 2 fois 2 branches) :

$\Rightarrow$ Entiers : =, >, <

$\Rightarrow$ Réels : expression d'intervalles par *]*, *[]* , *[* (au-dessous, dans, au-dessus)

- **Cas de valeur manquante :**

  ➥ Quelle branche prendre ?

- Dans certains cas, le manque est significatif :

  ➙ "absent" est une valeur (sens précis) comme une autre

- Dans d'autres cas, il faut un traitement spécial :

  Sol 1 ⇨ l'instance suit la **branche la plus populaire** ou

  Sol 2 ⇨ l'instance est divisée en pièces (pour suivre plusieurs branches) :

  → chaque pièces reçoit un poid entre 0 et 1 (selon le nombre d'instances de chaque branche).

  → classes déduites → combinaison des poids ayant conduit aux feuilles.

3.4 Règles de Classification

- Une alternative aux arbre de décision .
- Rappel : format :

  - **Antécédent** (pré condition) : tests comme dans un arbre de décision .
  - **Conséquence** : une/plusieurs classe(s) ou une distribution de probabilités .

- Précondition avec des & (ou d'autres expressions) logiques :
- L'ensemble des règles = un OU logique.

  ➥ Si plus d'une règle s'applique : **conflit**.

- Synonymie entre Règles de Classification et arbres de décision .

De l' arbre de décision aux Règles de Classification :

- Une règle par "path".

  $\Rightarrow$ Antécédents : les tests présents sur les noeuds

  $\Rightarrow$ Conséquence : la classe de la feuille

- Conversion simple et non ambiguë
- Pas d'ordre dans les règles
- Simplification des règles : enlever des tests (règles) redondants = **élagage**.

Des Règles à l'Arbre :


Passage plus difficile

• Les arbres ne savent pas bien exprimer la disjonction ("OU") entre les règles

➥ Dans un arbre, le "ou" supprime l'ordre sur les fils → peut poser problème .

➥ *Exemple* (avec 4 attributs booléens {a,b,c,d}) :

if a **and** b **then** x

if c **and** d **then** x

➥ Que faire des 2 autres dans chaque cas ?

⇒ Il faut casser la symétrie et choisir un simple test pour les noeuds de l'arbre.

⇒ L'arbre contiendra des sous-arbres identiques

Problème de sous-arbre dupliqués (*Replicated subtree problem*) :

• Si le test de **a** est choisi à la racine, la 2e règle doit être répétée 2 fois dans l'arbre

if a **and** b **then** x
if c **and** d **then** x

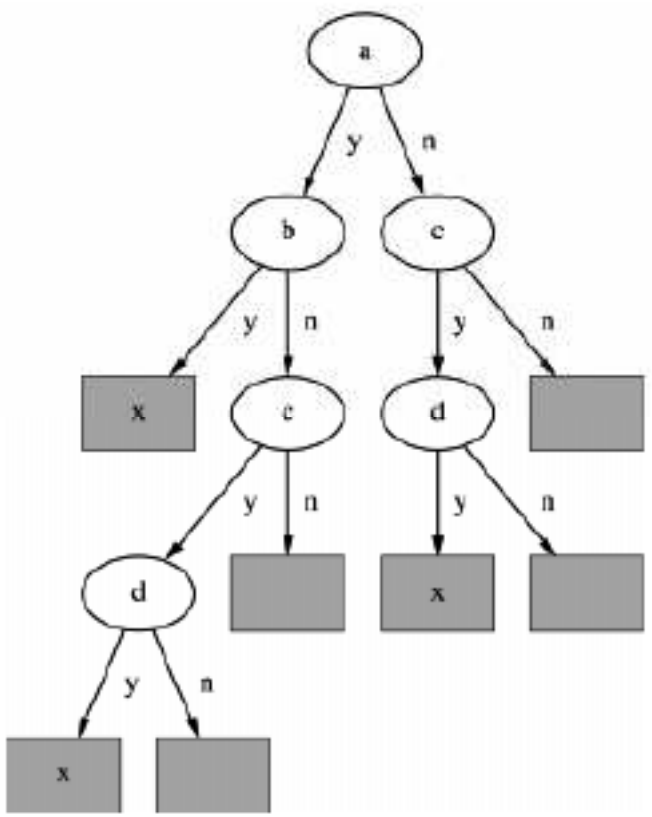


FIGURE 3.1 – Une table de Décision pour "Si a & b Alors x / Si c & d Alors x"

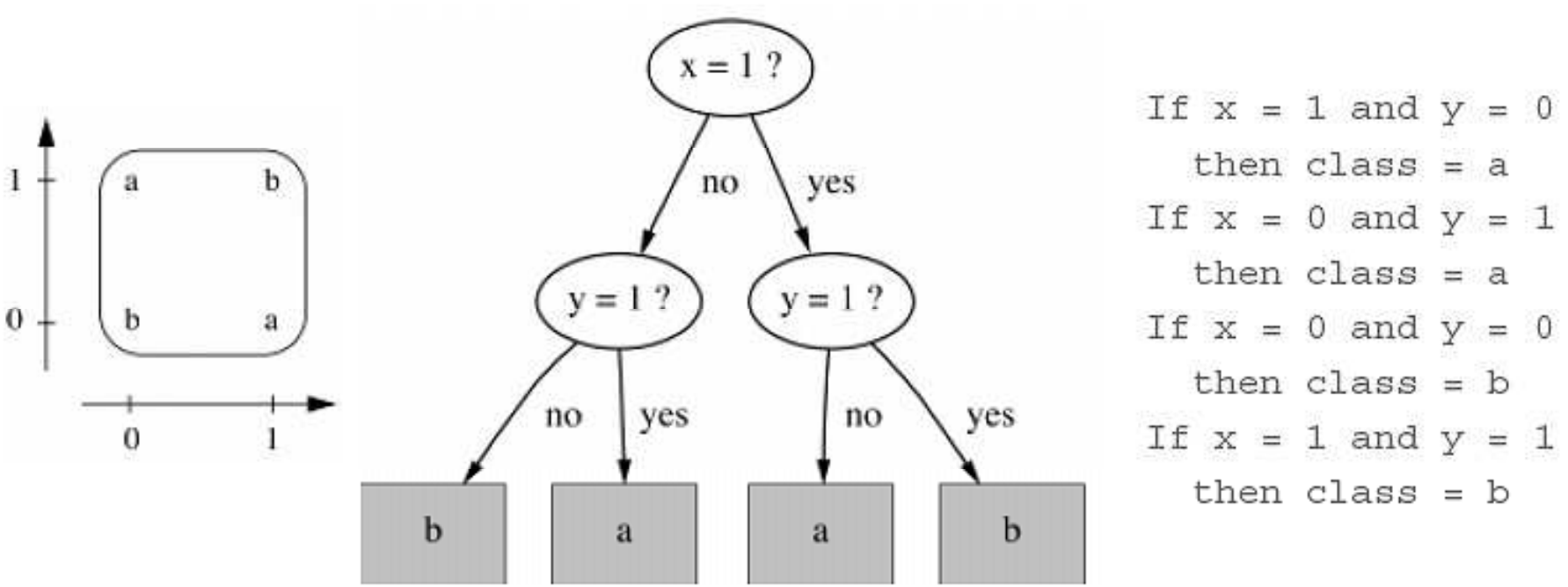


FIGURE 3.2 – L'arbre du problème OUx (cas favorable)

Règles à l'arbre : <u>cas favorable</u> (la figure 3.2)

⇒ diagramme à gauche = un **OuX** sur 'a' (ou **équivalence** sur 'b').

⇒ Il faut scinder sur un attribut d'abord (arbre du milieu de la figure).

⇒ Les règles reflètent la vrai <u>symétrie</u> (elles ne sont pas plus compactes que l'arbre)

Règles à l'arbre : <u>cas défavorable</u> :

• Règles plus compactes, surtout avec une règle 'par défaut' (mais pas l'arbre).

```
If x = 1 and y = 1
  then class = a
If z = 1 and w = 1
  then class = a
Otherwise class = b
```

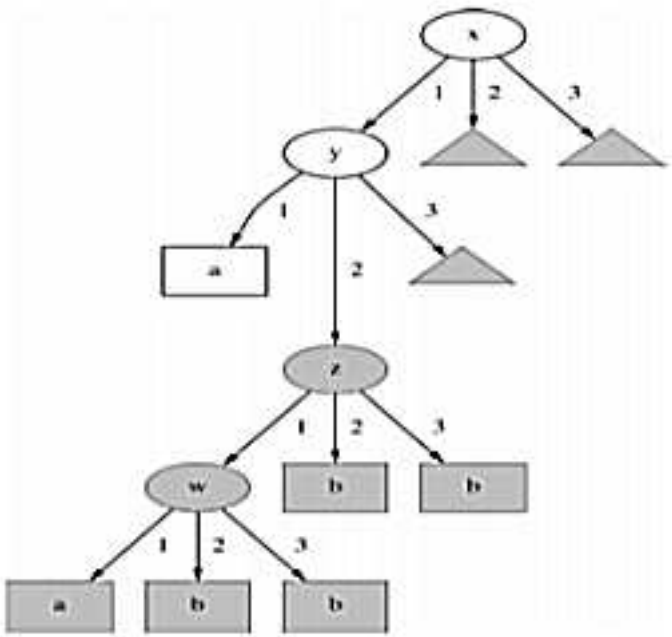


FIGURE 3.3 – L'arbre avec des sous-arbres dupliqués (cas défavorable)

• Cas plus extrême : 4 attributs $(x, y, z, w) \in \{1, 2, 3\}$ et deux classes a et b.

⇒ Chaque triangle gris = le sous-arbre à 3 niveaux gris à gauche en bas.

⇒ Une manière pénible d'exprimer un concept simple !

3.4.1 Règles : suite

- Les règles sont populaires : une règle = une petite "connaissance" indépendante.
- On peut ajouter facilement une règle à un ensemble de règles sans perturber le reste

  ➥ Mais ajouter un sous-arbre à un arbre → réorganisation de tout l'arbre.

- Un problème : l'indépendance des règles dépend de l'ordre :

  ⇒ Si exécutées dans l'ordre → *une liste de décision* (si-alors-sinon inhérent)

  ⇒ Si exécutées pas dans l'ordre → Recouvrement, conflit (plusieurs classes)

  ⇒ Ca n'arrive pas si règles obtenues depuis un arbre de décision

  ➥ Les redondances sont déjà codées dans l'arbre

Interprétation des règles :

- Si conflit (plusieurs conclusions pour une même instance)

  → Ne pas donner de conclusion du tout !

  → Prendre la règle la plus *populaire* (la plus appliquée)

    → Risque de résultats complètement différents

- Si aucune règle ne s'applique

  → Ne pas donner de conclusion du tout !

  → Prendre la classe la plus *fréquente* (de l'ensemble d'apprentissage)

Cas particulier : classes booléennes

• Hypothèse : Si l'instance n'est pas dans la classe "Oui", elle sera dans "Non"

⇒ Il suffit d'apprendre des règles pour la classe "Oui" + une règle par défaut pour la classe "Non"

Si x=1 & y=1 Alors classe = a

Si z=1 & w=1 Alors classe = a

Sinon classe = b

⇒ Hypothèse du monde clos (Close World Assumption = CW)

⇒ L'ordre d'interprétation n'a plus d'importance, pas de conflit

⇒ Règles sous la forme normale disjonctive (des **OU**s sur des **Et**s)

3.5 Règles d'association

- Prédiction de n'importe quel(s) attribut(s)
- Donnent les **régularités** (corrélations d'attributs) dans une base de données
- Pas destinées à être toutes utilisées comme un ensemble (on fait le tri)
- En général, beaucoup de règles d'association

  $\rightarrow$ Choisir les plus prédictives

  $\rightarrow$ *support* et *confiance* élevés

- **Support** = nombre d'instances correctement prédites
- **Confiance** = nombre prédictions correctes

  ➥ = une proportion de toutes les instances auxquelles la règle s'applique.

- Exemple :

  Si température = froide Alors Humidité = normale *{support=4, confiance = 100%}*

  ➔ *support*=correctement prédits

  = nombre de jours qui vérifient l'antécédent (4 dans l'exemple "météo")

  ➔ *confiance*= proportion des jours froides avec l'humidité normale

  = 100% dans la base de données (de cet exemple).

- On précise en général ces 2 valeurs (des minima)

  ➔ Dans l'exemple "météo", 58 règles avec $support \geq 2$ et $confiance \geq 95\%$

- **Le support peut être spécifié comme un pourcentage de toutes les instances.**

3.5.1 Interprétation des règles d'association

• Prudence ! Exemple ("météo")

Si venteux = faux & jouer = non Alors aspect = ensoleillé & humidité = forte. (1)

N'est pas un raccourci pour 2 règles séparées

Si venteux = faux & jouer = non Alors aspect = ensoleillé. (2)

Si venteux = faux & jouer = non Alors humidité = forte. (3)

• La règle originale (1) :

→ Le *support* S et la *confiance* C mini respectés

→ S exemples avec ***venteux = faux & jouer = non*** avec ***aspect = ensoleillé & humidité = forte***

→ Proportion d'au moins C de ces jours avec ***venteux = faux & jouer = non***

• Les règles (2) et (3) sont plus générales (moins contraintes)

• Par contre, la règle correcte (4) peut être déduite de (1) :

Si humidité = forte & venteux = faux & jouer = non Alors aspect = ensoleillé ***(4)***

⇒ La règle (4) a le même support S que la règle (1) rappelée ici :

Si venteux = faux & jouer = non Alors aspect = ensoleillé & humidité = forte. ***(1)***

⇒ La confiance de (4) sera au moins = C car le nombre de jours

humidité = forte & venteux = faux & jouer = non

est nécessairement plus petit que ***venteux = faux & jouer = non*** (plus large).

• Des liens entre des règles d'association particulières : certaines impliquent d'autres.

➥ Réduire le nombre de règles produites

➥ Si plusieurs règles sont liées, on prend la plus "forte"

→ Dans cet exemple, seule la règle (1) sera conservée.

3.6 Règles avec exception

- Idée : permettre des exceptions
- Exemple Iris :

  If petal-length $\geq$ 2.45 and petal-length $<$ 4.45 Then Iris-versicolor.

$\Rightarrow$ Soit une nouvelle instance avec le type donné par un expert (sera mal classée !) :

Taille Sépale	Largeur Sépale	Taille Pétale	Largeur Pétale	Type
5.1	3.5	2.6	0.2	Iris-setosa

$\Rightarrow$ Une solution : insérer des exceptions.

$\Rightarrow$ Permettent des modifications incrémentales (au lieu de refaire tout).

If petal-length $\geq$ 2.45 and petal-length $<$ 4.45 Then Iris-versicolor

EXCEPT if petal-width $<$ 1.0 then Iris-setosa.

Un exemple plus complexe : exceptions sur les exceptions...

default : Iris-setosa

except if petal-length $\geq$ 2.45 and petal-length $<$ 5. 355 and petal-width $<$ 1.75

then Iris-versicolor

except if petal-length $\geq$ 4.95 and petal-width $<$ 1.55

then Iris-virginica

else if sepal-length $<$ 4.95 and sepal-width $\geq$ 2.45

then Iris-virginica

else if petal- length $\geq$ 3.35

then Iris-virginica

except if petal-length $<$ 4.85 and sepal-length $<$ 5.95

then Iris-versicolor

Intérêts des Exceptions :

— modification incrémentale,

— compréhension plus simple,

— ajoute des connaissances du domaine,

— cas particuliers traités,

— Simplification de la compréhension de larges ensembles de règles

— Ajoute simple

— Contextualisation des règles

— Propriétés locales pour l'appréhension

— N'existe pas dans les règles *normales*

— ...

3.7 Règles Relationnelles

- En général : un test sur un attribut p/r à une constante → **propositionnel**.
- Pas toujours suffisant, e.g. exprimer des relations entre exemples (cf. Ex. Famille).

- Exemple : Grisés : *debout*, Autre : *couché*

  ⇒ Concept recherché : **debout**

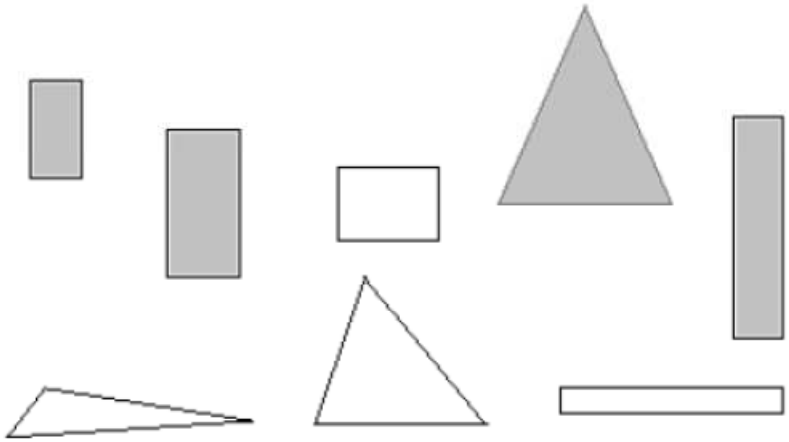


FIGURE 3.4 – Exemple de formes géométriques

- Les 4 formes grisées = exemples positifs (debout),

• Les autres négatifs (couchés).

Largeur	**Hauteur**	**Nb. côtés**	**Classe**
2	4	4	debout
3	6	4	debout
4	3	4	couché
7	8	3	debout
7	6	3	couché
2	9	4	debout
9	1	4	couché
10	2	3	couché

• Propositionnelle : **Si Largeur $\geq$ 3.5 & Hauteur $<$ 7.0 Alors couché**

Si Hauteur $\geq$ 3.5 Alors debout

•**Une façon courante** de fixer les seuils numériques :

➔ **largeur** 3.5 = la moitié de la largeur de la plus fine forme couchée (4 cm) et La largeur de la plus "grosse" forme debout dont la hauteur est moins de 7 (3 cm).

➔ **hauteur** 7.0 = la moitié de la hauteur du plus grand bloc couché (6 cm) et de la plus courte forme debout dont largeur $>$ 3.5 (8 cm).

• Ces règles marchent sur les exemples mais elles ne sont pas bonnes.

➥ Beaucoup de nouvelles instances mal classées (e.g. largeur = hauteur = 2).

$\Rightarrow$ On remarque : les blocs "debouts" sont plus haut que large ➔ **relationnel**

Si Largeur $>$ ***Hauteur Alors Couché.***

Si Hauteur $>$ ***Largeur Alors Debout.***

3.7.1 Règles avec relations

- Règles **relationnelles** : expriment une relation entre attributs
- Les opérateurs $=, <, >$ sur les numériques et $=, \neq$ sur les nominaux
- Ces relations peuvent figurer dans les arbres de décision .
- L'apprentissage de ces règles est coûteux (peu de système le propose)

$\Rightarrow$ Une solution : ajouter un attribut extra (e.g. booléen **Largeur** $>$ **Hauteur** ?)

➥ Phase de préparation des données

$\Rightarrow$ Autre exemple : BD familiale (vue plus haut) et la relation *soeur*.

Variables dans les règles (1er ordre) :

• Rendre le rôle l'instance explicite :

Si largeur(bloc) $>$ ***hauteur(bloc) Alors couché(bloc).***

Si hauteur(bloc) $>$ ***largeur(bloc) Alors debout(bloc).***

• Si l'on considère l'instance comme une structure

⇒ E.g. Une pile de blocs avec *top* pour le bloc au dessus :

Si hauteur(pile.top) $>$ ***largeur(pile.top) Alors debout(pile.top).***

⇒ Si "reste" = le reste de la pile,

⇒ Exprimer "toute la pile est composée de blocs debouts" par la règle récursive :

Si hauteur(pile.top) $>$ ***largeur(pile.top) and debout(pile.reste) Alors debout(pile).***

Si pile=vide Alors debout(pile).

Programmation Logique Inductive

- Ces ensembles de règles = programmes logiques du domaine de ML :

  $\Rightarrow$ "inductive logic programming" (ILP).

- Permet les définitions récursives de règles

- Difficile à "apprendre" des règles (assez théorique)

- Idéal pour la formalisation

  $\Rightarrow$ transformation possible

  ➥ Les techniques ILP peuvent éviter la récursivité

3.8 Arbres pour la prédiction numérique

• Exemple "performances CPU" du chapitre 1

• **Régression** : calcul d'une expression (équation) pour prédire une valeur numérique

• **Arbre de Régression** = arbre de décision où une = une prédiction numérique

⇒ La conclusion est **la moyenne des valeurs** de tous les exemples d'apprentissage qui arrivent à cette feuille (dont la performance est connue)

• **Arbre de Modèle** = Arbre de Régression où les feuilles = une Régression linéaire

⇒ Approximation de fonctions contiguës (appelée "linear patches")

• Rappel : la régression linéaire de l'exemple "performances CPU" :

PRP = -56.1 + 0.049 MYCY + 0.015 MMIN + 0.006 MMAX + 0.630 CACH - 0.270 CHMIN + 1.46 CHMAX.

⇒ L'arbre de régression de l'exemple : ../..

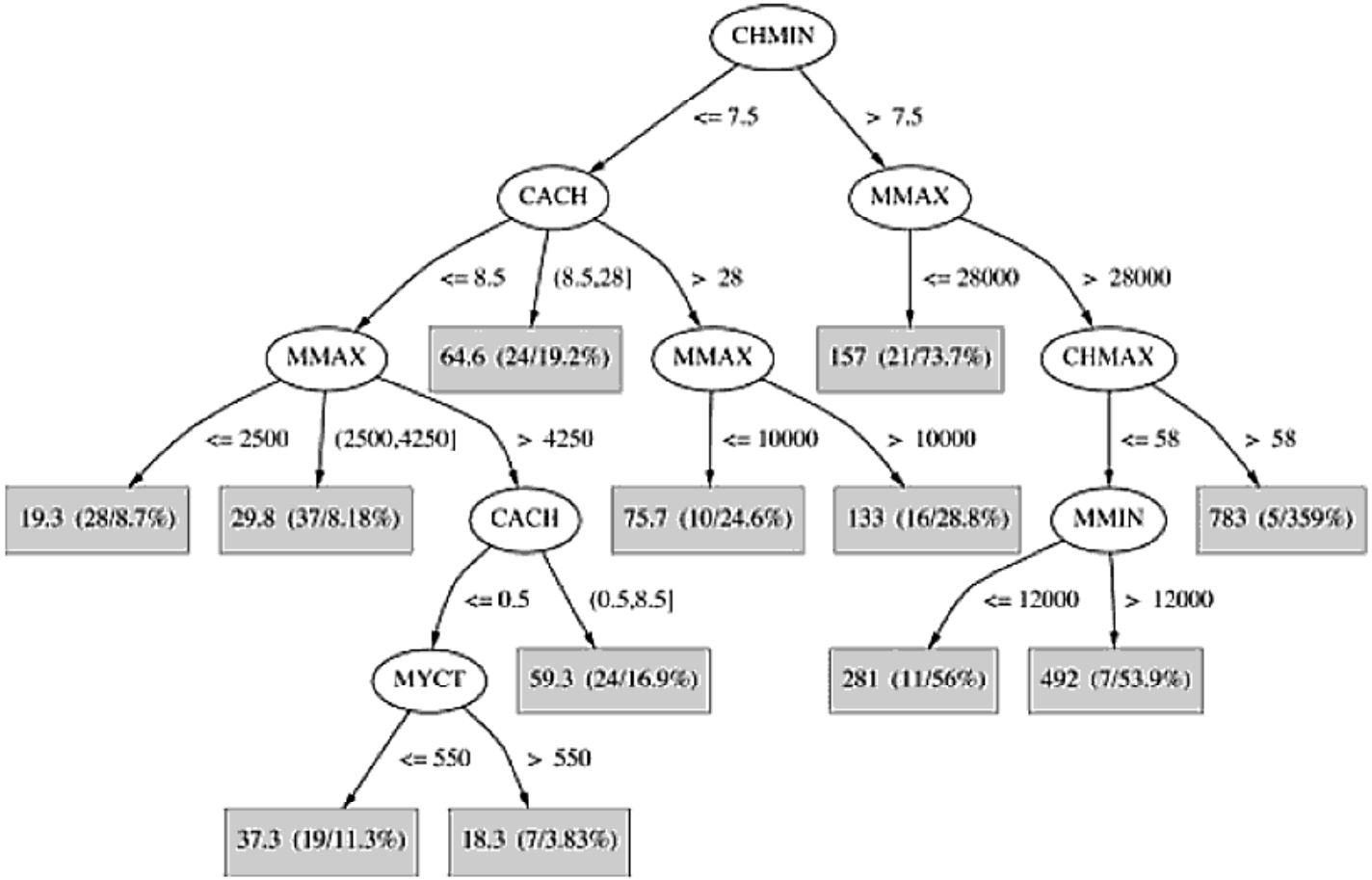


FIGURE 3.5 – Arbre de Régression de l'exemple CPU

⇒ **PRP** = performances, **CHMIN/CHMAX**= Nb. mini/maxi de canaux, **CACH** = taille cache,

MMIN/MMAX = Ram mini/maxi, **MYCT** = cycle (en ns.)

Arbre de Modèle pour l'exemple CPU :

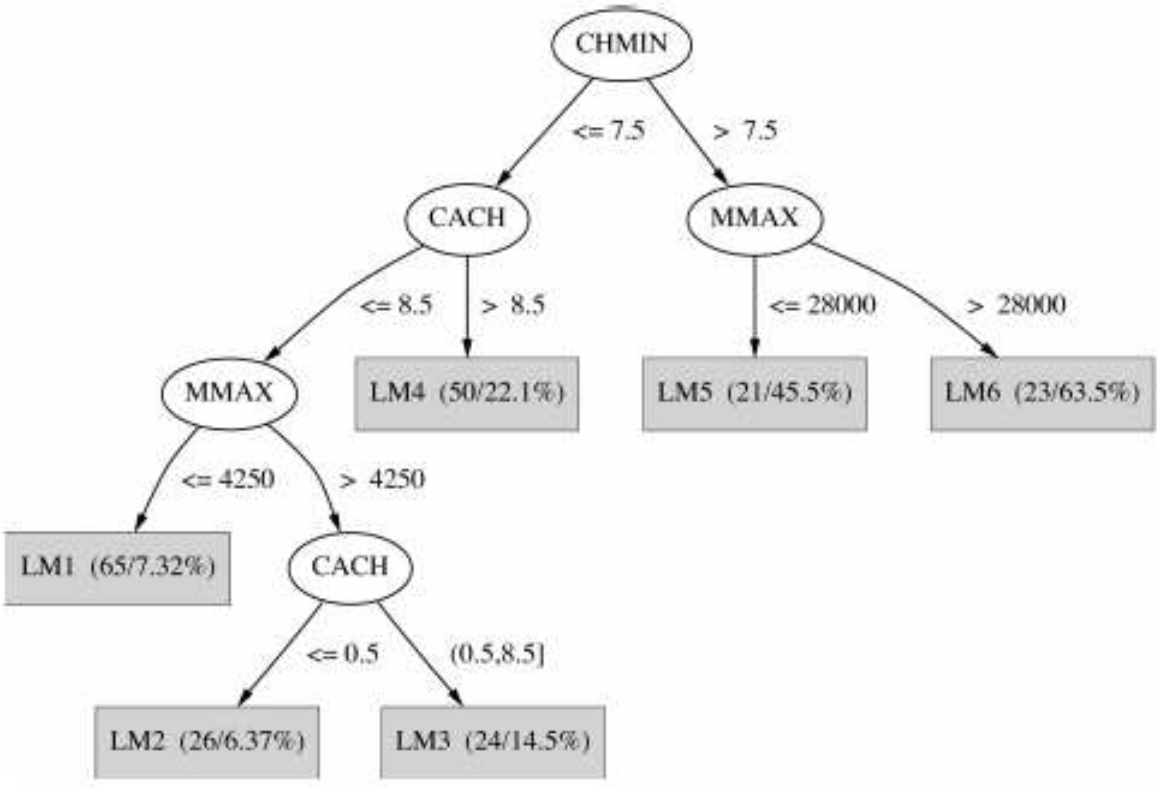


FIGURE 3.6 – Arbre de Modèle de l'exemple CPU

⇒ Les LM sont des modèles linéaires au niveau des feuilles.

⇒ E.g. **LM1 : PRP=8.29 + 0.004 MMAX + 2.77 CHMIN**

Remarques :

• Les données de ce problème pas bien présentées par un modèle linéaire simple.

⇒ La moyenne des valeurs absolues des erreurs (prédictions vs. réalité) est **significativement moins importante** pour l'arbre que pour l'équation.

➥ L'arbre de regression est plus juste

⇒ Mais l'arbre est touffu et difficile à interpréter (large taille).

• L'arbre de modèle est plus petit et plus simple à lire que l'arbre de décision

➥ Les moyennes d'erreurs y sont **plus petites**.

3.9 Representation à base d'exemple (IBL)

- La forme la plus simple d'apprentissage = la *pleine mémorisation* ("rote learning").

  ⇒ Recherche dans les instances mémorisées pour trouver la **plus ressemblante**

  ⇒ La connaissance est représentée par les instances mêmes

  ⇒ La question clé est donc la **ressemblance**

  ⇒ La fonction **distance** définit ce qui est "appris"

  ⇒ l'apprentissage à base d'instances est une méthode **différée** (*lazy learning*)

  ➥ Le "moment" d'apprentissage diffère.

  ⇒ Par opposition aux autres méthodes **actives** (*eager learning*)

- Méthodes : **le plus proche voisin**, **k plus proches voisins**

  ⇒ La classe majoritaire des k plus proches voisins

  ➥ ou la moyenne des distances si données numériques

Calcul de la distance :

⇒ Triviale avec un seul attribut numérique

➥ la différence entre les 2 valeurs (ou une fonction des 2).

• Si plusieurs attributs numériques : distance **euclidienne**

➥ avec une normalisation éventuelle = égalisation de l'importance

• Si attributs nominaux : distance=1 si valeurs $\neq$, 0 si $=$

Inconvénients de l'Apprentissage à base d'instances :

⇒ ne donne pas explicitement la structure de ce qui est appris

⇒ les instances combinées avec le métrique distance

→ les frontières de l'espace d'instances qui distinguent une classe de l'autre

→ c'est une forme de structure de la connaissance :/..

3.9.1 Apprentissage à base d'instances : suite

- Un des problèmes importants de l'Apprentissage Automatique :

  ⇒ **déterminer les attributs importants**

  ⇒ Associer des poids aux attributs est une technique

  ⇒ Trouver la **bonne pondération** de l'attribut à partir d'un ensemble d'apprentissage

- Parfois, inutile de stocker **toutes** les instances (temps de calcul, mémoire, ...)

- Certains espaces d'attribut sont plus stables que d'autres (p/r aux classes)

  ⇒ densité à l'intérieur vs. aux frontières

  ⇒ seulement quelques instances suffisent dans ces région stables

- Décider **quelles instances sauvegarder** est un autre problème clé de l'IBL.

3.9.2 IBL : Apprentissage de prototypes

- Seules les instances impliquées dans la décision sont stockées
- Idée : utiliser seulement des exemples prototypiques

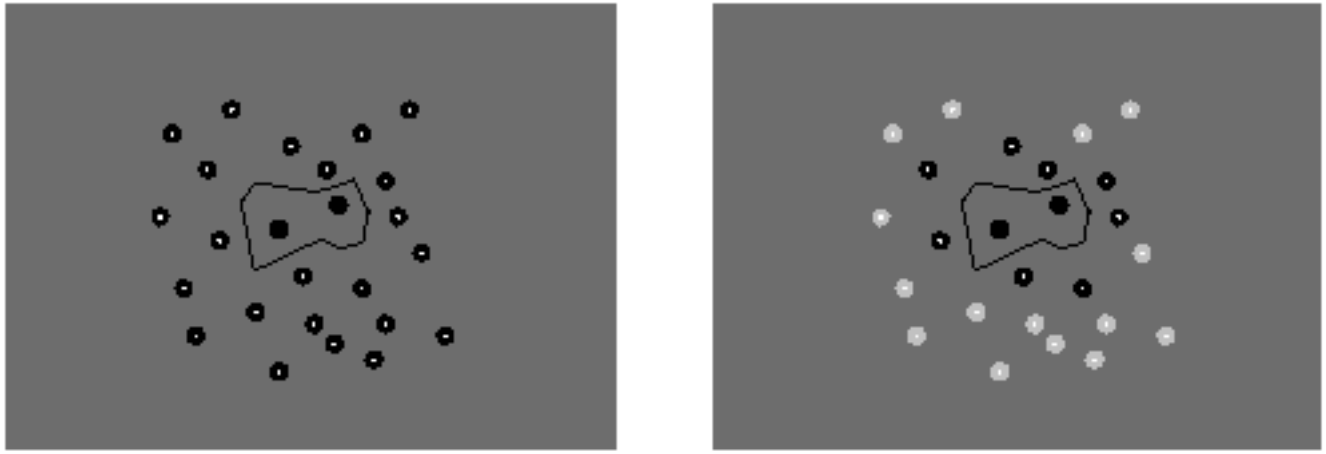


FIGURE 3.7 – Partitionnement de l'espace d'instances : Ex. gris pâles éliminés

⇒ Si 2 classes → perpendiculaire bissectrice

⇒ Si Plus : Polygone de 9 côtés qui sépare la classe de cercles pleins des autres cercles.

➥ Polygone implicite dans l'application de la règle du plus-proche-voisin

IBL : Généralisation Rectangulaire :

- Généralisation explicite de quelques instances (création de rectangles)
- La règle plus-proche-voisin utilisée hors les rectangles
- Les rectangles $\simeq$ règles (souvent plus conservatifs que les règles habituelles)

  ➥ Les rectangles imbriqués sont des règles avec exception

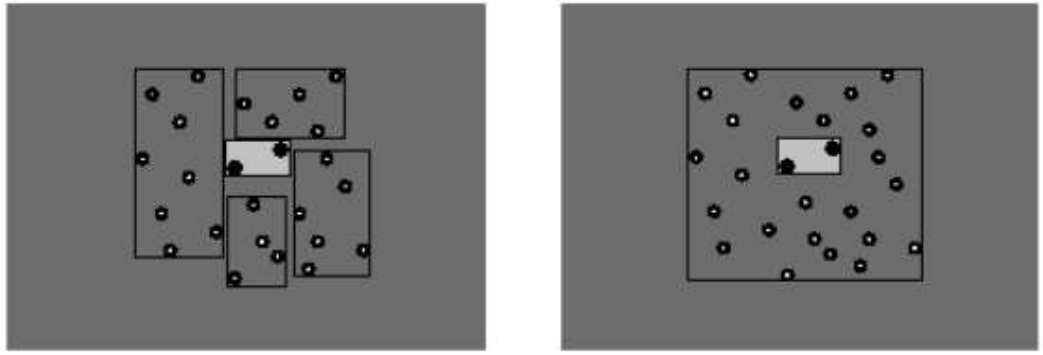


FIGURE 3.8 – Partitionnement de l'espace d'instances (suite)

$\Rightarrow$ Les test sur les dimensions d'un rectangle (dedans=classe, dehors $\rightarrow$ test PPV)

➥ correspond aux tests sur différents attributs

3.10 Clusters (classification non Supervisée)

- La sortie (vs. classifieur) : un diagramme de regroupement
- Forme simple : un numéro de cluster associé à chaque instance.
- Espace 2D partitionné (= table/arbrorescence/....)
- Algorithme : Si une instance dans plusieurs clusters → **diagramme de "Venn"**

FIGURE 3.9 – (a) : Représentation 2D et (b) : diagramme de Venn (recouvrement)

➥ plusieurs clusters pour une instance avec des probabilités d'appartenance

- **Autre algorithme** → arborescence (**Dendrogram**=clustrum)

  ➥ Au sommet : peu de clusters, ensuite sub-divisés, ...

  ➥ Au niveau des feuilles, les éléments regroupés dans un cluster.

- Le clustering est souvent suivi de l'inférence d'un arbre de décision (ou des règles)

  ➥ affectation des instances aux classes

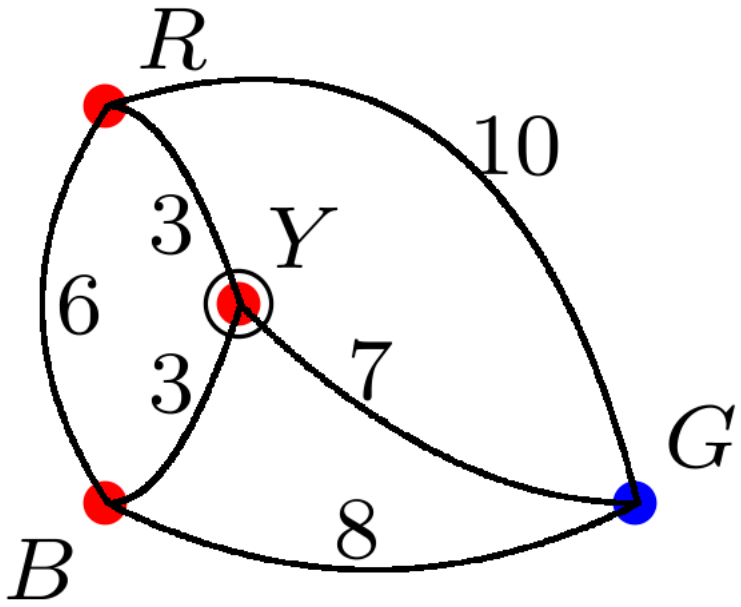


FIGURE 3.10 – Affectation avec probabilité et Dendrogramme (Dendron =arbre en grecque)

Table des matières