Une approche probabiliste pour le classement d'objets incomplètement connus dans un arbre de décision

Lamis Hawarah

Université Joseph Fourier – Grenoble I **École Doctorale MSTII**

Une approche probabiliste pour le classement d'objets incomplètement connus dans un arbre de décision

THÈSE

présentée et soutenue publiquement le 22 octobre 2008

pour obtenir le grade de

Docteur de l'Université Joseph Fourier – Grenoble I

(Spécialité : Intelligence Artificielle)

par

Lamis HAWARAH

Composition du jury

Président :	Jacques DEMONGEOT	Professeur, Université Joseph Fourier - Grenoble 1
Rapporteurs :	Omar BOUSSAID	Professeur, Université Lumière Lyon II
	Bruno CRÉMILLEUX	Professeur, Université de Caen
Examinateurs :	Ricco RAKOTOMALALA	Maître de Conférences, Université Lumière Lyon II
	Ana SIMONET	Maître de Conférences, UPMF de Grenoble (Co-directrice)
	Michel SIMONET	Chargé de recherche CNRS (Directeur de thèse)

Laboratoire des Techniques de l'Ingénierie Médicale et de la Complexité
Informatique, Mathématiques et Applications de Grenoble (TIMC-IMAG — UMR 5525)

Remerciements

Je tiens à remercier très sincèrement Ana Simonet et Michel Simonet, mes directeurs de thèse, pour m'avoir accueillie dans leur équipe et donné l'occasion de vivre l'expérience de réaliser une thèse de doctorat. Je les remercie également pour leur aide et leurs critiques constructives qui ont permis l'accomplissement de mon travail.

Je tiens à remercier Jacques Demongeot de m'avoir fait l'honneur de présider mon jury de thèse. Je suis très reconnaissante à Bruno Crémilleux et à Omar Boussaid pour avoir accepté de rapporter cette thèse à une période très chargée. Je les remercie également pour leur lecture approfondie de mon rapport ainsi que pour leurs questions et suggestions très enrichissantes. Je remercie également Ricco Rakotomalala pour avoir accepté d'évaluer mon travail ainsi que pour ses retours constructifs.

Je voudrais également remercier mes camarades qui ont partagé avec moi certains moment difficiles, Delphine, Houda, Dima, shokooh, Radja, Kinda, Gayo, Lina...
Merci à Delphine pour les moments passés ensemble, pour son aide et ses relectures de mes articles et de mon mémoire.

Enfin, je voudrais particulièrement remercier Samer pour son amour et son soutien, ainsi que pour son aide et sa patience en toute circonstance pendant toutes ces années.

Je tiens également à remercier toute ma famille, mes trois soeurs et mes deux frères, qui m'ont toujours encouragée. Un grand merci à mes parents à qui je dois tout.

À ma maman une partie du paradis qui m'appartient
ma précieuse perle qui illumine ma vie

À mes amours Samer et Karam

À l'esprit de mon père et mon grand-père

Table des matières

Liste des tableaux

Table des figures

/

Introduction

Dans de nombreux domaines, il est nécessaire de prendre des décisions critiques, dans un contexte parfois difficile et en un temps limité. Par exemple, un médecin, qui doit prendre une décision rapide pour traiter un cas urgent, fait appel à ses connaissances et expériences pour prendre sa décision. Mais il ne peut pas se souvenir de tous les dossiers qu'il a traités et étudiés depuis des années.

Les outils informatiques peuvent apporter une aide précieuse dans ce cas, car ils peuvent prendre en compte un grand nombre de cas déjà traités et proposer pour un nouveau cas une décision fondée sur la compilation de tous les cas passés.

Il peut arriver que les données issues du monde réel ne soient pas complètes. Elles peuvent contenir des informations non renseignées, par exemple parce qu'une personne a refusé de répondre à certaines questions, parce que certains tests ne peuvent pas être effectués, etc. L'ignorance des valeurs manquantes peut rendre la décision non représentative et donc être dangereuse.

L'extraction de connaissances à partir des données (ECD) est *un processus non trivial d'identification de structures inconnues, valides et potentiellement utiles dans les bases de données* [Fayyad *et al.*, 1996]. Son objectif est d'aider l'être humain à extraire les informations utiles (connaissances) à partir de données dont le volume croît très rapidement. Les étapes de ce processus sont, l'acquisition de données multiformes (textes, images, séquences vidéos, etc.), la préparation de données (pré-traitement), la fouille de données, et enfin la validation et mise en forme des connaissances.

La *Fouille de Données* (Data Mining) [Adriaans et Zantinge, 1996, Zighed et Rakotomalala, 2000] se situe dans le cadre de l'apprentissage inductif. Cette phase fait appel aux multiples techniques qui permettent de découvrir les connaissances auparavant cachées dans les données et d'aider à la décision.

Problématique

Le problème des valeurs manquantes est un problème connu dans le domaine de la fouille de données et de l'apprentissage automatique où, dans la base d'apprentissage, on rencontre des objets ayant des valeurs manquantes pour certains attributs. Cela arrive pendant la phase d'acquisition des données du processus de l'ECD. Les données sont manquantes parce qu'on ne les a peut-être pas enregistrées, ou bien que leur acquisition est trop coûteuse, etc.

Prendre une décision en présence de données manquantes est une tâche difficile. Par exemple, la santé est un domaine où l'incertitude prend une importance considérable ; prendre une seule décision dans l'incertitude peut être dangereux. La présence des probabilités est importante puisqu'elles fournissent un moyen de "résumer" l'incertitude [Russell et Norvig, 2003].

id	Temps	Température	Humidité	Vent	Classe
1	Ensoleillé	Elevée	Haute	Faux	A
2	Ensoleillé	Elevée	Haute	Vrai	A
3	Couvert	Elevée	Haute	Faux	B
4	Pluvieux	Moyenne	Haute	Faux	B
5	Pluvieux	Basse	Normale	Faux	B
6	Pluvieux	Basse	Normale	Vrai	A
7	Couvert	Basse	Normale	Vrai	B
8	Ensoleillé	Moyenne	Haute	Faux	A
9	Ensoleillé	Basse	Normale	Faux	B
10	Pluvieux	Moyenne	Normale	Faux	B
11	Ensoleillé	Moyenne	Normale	Vrai	B
12	Couvert	Moyenne	Haute	Vrai	B
13	Couvert	Elevée	Normale	Faux	B
14	Pluvieux	Moyenne	Haute	Vrai	A

TAB. 1 – La base *Météo* [Quinlan, 1993]

Les arbres de décision

Un arbre de décision est un modèle qui est à la fois descriptif et prédictif. À partir d'une base d'apprentissage, on construit l'arbre dont chaque chemin depuis la racine jusqu'à une feuille correspond à une règle de classement. Le fait de construire le modèle sous forme d'arbre rend cette méthode facile à comprendre et à utiliser par un utilisateur humain. La construction d'un arbre de décision se base sur la partition récursive de l'ensemble d'apprentissage en sous-ensembles plus homogènes que l'ensemble de base.

Les arbres de décision sont une des techniques de l'apprentissage automatique couramment utilisées pour la fouille de données. Une fois l'arbre construit, il est utilisé pour le **classement** pour prédire la classe d'un nouvel exemple ainsi que pour la **régression** quand la classe dans la base d'apprentissage est numérique. Nous nous intéressons dans cette thèse à l'utilisation de l'arbre de décision pour le classement.

Par exemple, l'arbre de décision donné dans la figure 1 est construit à partir du tableau 1. L'utilisateur peut facilement extraire les cinq règles suivantes :
Si *Temps* est *ensoleillé* et *Humidité* est *haute* alors **A**.
Si *Temps* est *ensoleillé* et *Humidité* est *normale* alors **B**.
Si *Temps* est *couvert* alors **B**.
Si *Temps* est *pluvieux* et *Vent* est *vrai* alors **A**.
Si *Temps* est *pluvieux* et *Vent* est *faux* alors **B**.
Lorsqu'un nouvel exemple arrive, on peut directement utiliser l'arbre ou les règles pour le classer. Par exemple, chaque objet dont l'attribut *Temps* prend la valeur *couvert* est classé comme **B**.

Objectif

Quand on classe un nouvel exemple où la valeur d'un de ses attributs est inconnue, par exemple *Humidité* sachant que *Temps* est *ensoleillé*, on ne peut plus continuer le classement parce qu'on ne sait pas la valeur d'*Humidité* qui doit être choisie pour descendre dans l'arbre.

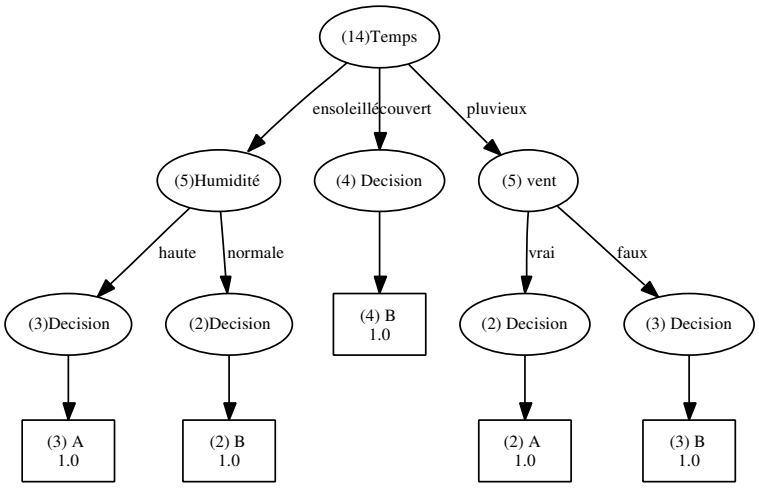

FIG. 1 – L'arbre de décision pour la *météo*

Dans ce cas, le processus de classement s'arrête. Les méthodes qui traitent les valeurs manquantes dans les arbres de décision remplacent le plus souvent un attribut manquant par une seule valeur, qui peut être la valeur la plus probable [Kononenko *et al.*, 1984] ou la plus semblable [Breiman *et al.*, 1984], etc. Ce type d'approches présente l'inconvénient d'oublier les autres valeurs possibles.

Notre objectif est de classer de manière probabiliste un objet ayant des valeurs manquantes. Pour cela, nous prédisons d'abord les attributs inconnus de cet objet de manière probabiliste. Ensuite, nous utilisons la distribution de probabilités obtenue pour chaque attribut manquant afin de pouvoir classer l'objet en question. Le calcul de la distribution de probabilités d'un attribut manquant est basé sur la dépendance entre cet attribut et les autres attributs dont il dépend.

Nous nous sommes particulièrement intéressés aux méthodes qui utilisent un arbre de décision pour chaque attribut inconnu afin de déterminer ses valeurs manquantes. Puisque les arbres de décision sont capables de déterminer la classe[1] d'une instance à partir des valeurs de ses attributs, on peut les utiliser pour déterminer les valeurs d'un attribut inconnu (qui joue alors le rôle de la classe) à partir des attributs dont il dépend. Dans ce cadre, nous pouvons déterminer la valeur d'un attribut manquant à partir des attributs dont il dépend, ce qui permet d'utiliser le maximum de l'information contenue dans l'objet pour le calcul des valeurs manquantes.

Notre travail se situe dans le cadre des *arbres de décision probabilistes* [Breiman *et al.*, 1984], [Quinlan, 1986], [Quinlan, 1990], [Quinlan, 1993]. Dans le but d'aider l'utilisateur à prendre sa décision, nous avons choisi de garder sur chaque feuille dans l'arbre de décision toutes les valeurs de classe avec leurs probabilités. Pour cela, nous utilisons un arbre de décision probabiliste plutôt qu'un arbre de décision classique, car ce dernier prend la valeur de classe la plus probable sur chaque feuille et considère comme mal classés les autres cas qui arrivent à cette feuille et qui n'appartiennent pas à sa classe. Le fait de fournir un résultat probabiliste donne à l'utilisateur une information plus fine sur la proportion des valeurs de classe sur chaque feuille. Conserver sur chaque feuille toutes les valeurs de classe avec leurs probabilités rend l'arbre plus simple à interpréter.

[1] La classe est l'attribut à prédire.

Description de nos travaux

Dans notre travail, nous nous sommes intéressés à une méthode appelée *Arbres des Attributs Ordonnés (AAO)*. Cette méthode traite les valeurs manquantes à la fois pendant les phases de construction et de classement. Pour chaque attribut elle construit un arbre de décision, appelé arbre d'attribut, qui fournit une valeur quand l'attribut est inconnu. L'ordre de construction de ces arbres est croissant en fonction de l'Information Mutuelle (IM[2]) entre chaque attribut et la classe. Le premier arbre d'attribut construit est celui qui correspond à l'attribut ayant l'IM minimale. Il est représenté par un seul nœud-feuille avec sa valeur la plus probable dans la base d'apprentissage. Pour les autres attributs on fournit, à partir de l'ensemble d'apprentissage initial, le sous-ensemble d'apprentissage qui contient les instances ayant des valeurs connues pour cet attribut. Ces instances sont décrites seulement par les attributs qui ont déjà été traités (c'est-à-dire les attributs pour lesquels on a déjà construit les arbres d'attributs et déterminé leurs valeurs manquantes dans la base d'apprentissage). Toutefois, cette méthode n'est pas applicable dans tous les cas, à cause de l'ordre imposé pour la construction, et elle ignore les dépendances entre les attributs. Elle présente aussi l'inconvénient, comme la plupart des méthodes (à l'exception de la méthode de Quinlan), de n'affecter l'objet incomplet qu'à une seule feuille de l'arbre, et donc à une seule classe. Or, dans le monde réel, et en particulier en médecine, un tel objet peut appartenir potentiellement à plusieurs classes et donc devrait pouvoir être associé à plusieurs feuilles dans l'arbre de décision.

Dans ce mémoire, nous décrivons une approche qui se compose de deux propositions :

- *les Arbres d'Attributs Ordonnés Probabilistes (AAOP)*, qui étendent la méthode d'*AAO* en construisant un arbre de décision probabiliste pour chaque attribut, au lieu d'un arbre de décision classique. Les attributs utilisés pour construire un arbre d'attribut selon *AAOP* sont les attributs déjà traités et dépendants de l'attribut courant, au lieu de tous les attributs déjà traités, comme dans la méthode *AAO*;
- *les Arbres d'Attributs Probabilistes AAP*, qui prennent en compte la dépendance entre les attributs lors de la construction des arbres de décision probabiliste. On construit pour chaque attribut un arbre de décision probabiliste en utilisant les attributs dont il dépend.

En conséquence, chaque attribut dans la base d'apprentissage possède deux arbres probabilistes, correspondant aux deux propositions précédentes.

Dans cette approche, un problème de *cycle* peut être rencontré : lorsque deux attributs mutuellement dépendants sont manquants. Pour le résoudre, nous proposons de calculer d'abord la probabilité de l'attribut le moins dépendant de la classe à partir de son arbre d'attribut ordonné probabiliste *AAOP*, puis la distribution de probabilités de l'autre attribut à partir de son arbre d'attribut probabiliste *AAP*.

Nous avons testé notre approche sur des bases de données réelles et avons comparé nos résultats de classement avec ceux donnés par C4.5 et *AAO*. Les matrices de confusion de ces méthodes sont également calculées pour analyser les résultats de classement de chaque valeur de classe.

Dans un deuxième temps, nous nous sommes intéressés à l'analyse du résultat de classement de chaque objet dans la base de test. Dans ce cadre, nous voulons calculer la fréquence de chaque objet de la base de test dans la base d'apprentissage. Pour cela, nous avons cherché pour

[2]L'Information Mutuelle est une mesure de dépendance entre deux attributs.

chaque objet de la base de test les objets qui sont ses plus proches voisins dans la base d'apprentissage, en calculant la distance entre eux. Cet algorithme est de la famille des méthodes des k plus proches voisins. Il utilise une fonction de distance inspirée par l'algorithme Relief proposé par [Kira et Rendell, 1992], qui est une mesure d'impureté basée sur l'analyse statistique pour sélectionner les attributs pertinents. Cette fonction prend en compte les attributs manquants que l'on peut rencontrer dans les objets de la base de test. Elle prend également en compte les attributs qualitatifs (symboliques). La fréquence obtenue pour chaque objet est comparée avec le résultat de classement donné par notre approche, C4.5 et *AAO* pour le même objet.

Finalement, nous calculons la complexité de construction de nos arbres *AAPs* et *AAOPs*, ainsi que la complexité de l'algorithme de classement proposé par notre approche, C4.5 et *AAO*.

Structure du mémoire

Le mémoire de thèse comporte trois parties.

La première partie est consacrée à l'état de l'art et elle se compose de deux chapitres. Nous introduisons dans le chapitre 1 les notions de probabilité et d'Information Mutuelle. Nous présentons dans le chapitre 2 la notion d'arbre de décision, les méthodes de construction d'un arbre de décision et les stratégies d'élagage. Ensuite, nous consacrons la dernière partie de ce chapitre aux arbres de décision probabilistes.

La deuxième partie de ce mémoire se compose également de deux chapitres. Le chapitre 3 est consacré au problème des valeurs manquantes dans les arbres de décision pendant la construction et le classement. Le chapitre 4 présente notre approche pour traiter ce problème de manière probabiliste.

La dernière partie de cette thèse est composée de 3 chapitres.

Le chapitre 5 présente l'expérimentation de notre approche sur plusieurs bases de données réelles, ainsi que la comparaison avec les résultats de classement obtenus par C4.5 et *AAO* pour les mêmes bases.

Dans le chapitre 6, nous proposons un algorithme qui calcule pour chaque instance de la base de test sa fréquence dans la base d'apprentissage. Ces fréquences sont comparées avec les distributions de probabilités données par notre approche, ainsi que C4.5 et *AAO*.

Finalement, le chapitre 7 présente le calcul de la complexité de construction de nos arbres de décision et la complexité du classement probabiliste d'un objet ayant des valeurs manquantes en utilisant notre approche, ainsi que C4.5 et *AAO*.

Première partie

État de l'art

Chapitre 1

Probabilité et Information Mutuelle

1.1 Introduction

L'objectif de ce chapitre est d'expliquer comment l'Information Mutuelle peut être utilisée pour évaluer l'importance de chaque attribut dans une base de données et plus précisément pour mesurer la réduction d'incertitude lors de la sélection d'un attribut pour construire un arbre de décision.

Nous commençons par présenter quelques éléments de la théorie des probabilités [Wehenkel, 2001] nécessaires dans le cadre de notre travail. Ensuite, nous décrivons l'entropie, qui est une mesure qui quantifie l'information et l'Information Mutuelle (IM) qui est une mesure issue de la théorie de l'information [Shannon, 1948, MacKay, 2003] et que nous utiliserons lors de la construction d'un arbre de décision.

1.2 Rappel de probabilités

Dans cette section, nous allons présenter rapidement les notions de probabilités utilisées au cours de notre travail.

1.2.1 Épreuves et événements

- Une expérience est dite aléatoire si ses résultats ne sont pas prévisibles avec certitude en fonction des conditions initiales, et donc si répétée dans des conditions apparemment identiques, elle peut conduire à des résultats différents.
- On appelle épreuve la réalisation d'une expérience aléatoire.
- On appelle événement la propriété du système qui, une fois l'épreuve effectuée, est ou n'est pas réalisée.
- Un événement impossible correspond à l'ensemble $\emptyset$ car il n'est jamais réalisé.
- Un événement certain correspond à l'espace entier Ω car il est toujours réalisé.
- Deux événements A et B sont incompatibles (deux à deux disjoints[3]) s'ils ne peuvent être réalisés simultanément ; $A \cap B = \emptyset$.

[3] On peut dire aussi que A et B sont mutuellement exclusifs.

1.2.2 Définition 1 (probabilité)

On appelle probabilité sur (Ω, ε) (où Ω est l'ensemble des événements et ε une classe de parties de Ω), ou loi de probabilité, une application P de ε dans [0,1] telle que :

- $P(\Omega) = 1$
- Pour tout ensemble dénombrable d'événements incompatibles $A_1, A_2, ..., A_n$ on a :

$$P(\cup A_i) = \sum_i P(A_i)$$

On appelle espace de probabilité le triplet (Ω, ε, P).

Quelques propriétés

- Une probabilité est un nombre compris entre 0 et 1 :
  P(e)=0 ; *e désigne l'événement impossible.*
  P(e)=1 ; *e désigne l'événement certain.*
- La somme des probabilités de tous les événements possibles est égale à 1.
- La somme des probabilités de deux événements contraires est égale à 1[4].
- Si A implique B ($A \subset B$ alors $P(A) \leq P(B)$).
- La fonction de probabilité est croissante.

1.2.3 Probabilité conditionnelle (Formule de Bayes)

Parfois, on a des connaissances partielles sur le résultat d'une expérience, ce qui peut influer sur le résultat expérimental. On utilise alors la notion de probabilité conditionnelle, qui correspond à la probabilité d'un événement étant donné des connaissances préalables. La probabilité d'un événement avant de prendre en compte la nouvelle connaissance est appelée probabilité *a priori*, et la probabilité *a posteriori* est la probabilité de l'événement sachant la nouvelle connaissance.

$$\text{Si } P(B) > 0 \text{ alors}^5 : P(A|B) = \frac{P(A \cap B)}{P(B)}$$

La formule de Bayes généralisée

Soient A_i, i= 1, 2,..., n des événements tels que : $P(A_1 \cap A_2 \cap .. \cap A_{n-1}) > 0$, alors :

$$P(A_1 \cap A_2 \cap .. \cap A_n) = P(A_1)P(A_2|A_1)...P(A_n|A_1 \cap A_2 \cap ... \cap A_{n-1})$$

1.2.4 Théorème des probabilités totales

On dit que $A_1, A_2, ..., A_n$ forment un système complet d'événements si les événements $A_1, A_2, ..., A_n$ constituent une partition de Ω tel que :

- $\Omega = \cup A_i$;
- $\forall i \neq j$, $A_i \cap A_j = \emptyset$ (A_i, A_j étant deux événements disjoints deux à deux) ;
- $\forall i$, $A_i \neq \emptyset$ ($P(A_i) > 0$).

$\forall B$ événement quelconque de Ω (figure 1.1) : $B = (B \cap A_1) \cup (B \cap A_2) \cup ...(B \cap A_n)$.

La loi de la **probabilité totale** est définie comme suit :

$$P(B) = \sum_i P(B \cap A_i) = \sum_i P(B|A_i)P(A_i)$$

[4] L'événement contraire (complémentaire) $\bar{A}$ d'un événement A est l'événement qui est réalisé si et seulement si A ne l'est pas ; $P(A) + P(\bar{A}) = 1$

[5] $B \neq \emptyset$.

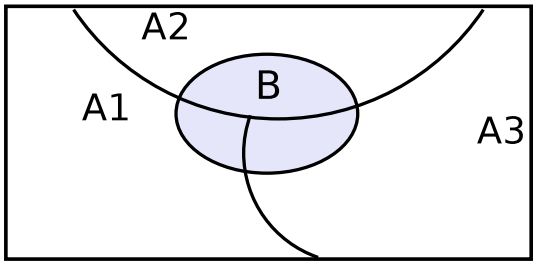

FIG. 1.1 – Probabilité totale

1.2.5 Indépendance

Deux événements A et B sont dits indépendants si : $P(A \cap B) = P(A)P(B)$ ou encore si P(B|A) = P(B)[6] et P(A|B)=P(A).
Deux événements indépendants ne signifie pas dire qu'ils sont incompatibles[7].

1.2.6 Théorème Bayésien

Soit un événement B qui peut dépendre de N événements A_i incompatibles deux à deux. Étant donnée la réalisation de l'événement B, quelle est la probabilité que ce soit A_i qui en soit la cause ?
On peut écrire que $B = \cup_{i=1}^{N}(B \cap A_i)$ car les événements A_i constituent un système complet.

- $P(A_i)$ est la probabilité a priori de A_i.
- $P(A_i|B)$ est la probabilité a posteriori de A_i sachant B.
- $P(B|A_i)$ est la probabilité de B sachant A_i.

D'après le théorème des probabilités totales, on a : $P(B) = \sum_i P(B \cap A_i)$
Et en appliquant le théorème des probabilités conditionnelles :

$$P(B \cap A_i) = P(B)P(A_i|B) = P(A_i)P(B|A_i)$$

Donc :

$$P(A_i|B) = \frac{P(A_i \cap B)}{P(B)} = \frac{P(A_i)P(B|A_i)}{\sum_j P(A_j)P(B|A_j)}$$

1.2.7 Indépendance conditionnelle

Soient A et B deux événements dans le même espace de probabilité (Ω, ε, P), et C un troisième événement du même espace tel que P(C)>0. On peut dire que A et B sont indépendants conditionnellement sachant C si :

$$P(A \cap B|C) = P(A|C)P(B|C) \tag{1.1}$$

[6] l'information sur la réalisation de A n'apporte rien à l'événement B

[7] Il ne faut pas confondre l'indépendance de deux événements A et B avec la propriété que ces événements sont deux à deux disjoints, c'est-à-dire que $A \cap B = \emptyset$. Si A et B sont deux à deux disjoints alors : $P(A \cap B) = P(\emptyset) = 0$ et donc $P(A \cap B) \neq P(A) \times P(B)$ sauf si un des deux événements est de probabilité nulle. On en conclut que des événements deux à deux disjoints ne sont pas indépendants sauf dans le cas trivial où au moins l'un d'entre eux est de probabilité nulle.

Par conséquent :

$$\begin{aligned} P(A \cap B|C) &= \frac{P(A \cap B \cap C)}{P(C)} \\ &= \frac{P(A|B \cap C)P(B|C)P(C)}{P(C)} \\ &= P(A|B \cap C)P(B|C) \end{aligned} \tag{1.2}$$

Donc, on peut déduire que $P(A|B \cap C) = P(A|C)$

1.3 Information Mutuelle

La théorie de l'information est due à Shannon [Shannon, 1948]. Le problème initial est celui de la communication entre une source et un récepteur. La source émet un message, qui passe par un canal, que le récepteur reçoit. Le message peut être de différents types : une séquence de lettres comme dans un télégraphe, un signal fonction du temps comme dans une radio ou un téléphone, une fonction du temps avec d'autres variables comme dans une télévision noir et blanc. Le canal est simplement le moyen de communication pour envoyer un signal de la source au récepteur ; ce peut être une paire de fils, un faisceau de lumière, etc. Le but est de quantifier l'information que contient chaque message émis. Par exemple : si le récepteur attend un message de type *oui/non*, la source envoie soit *oui* soit *non* et le récepteur reçoit une unité d'information (un bit) parce qu'on a un ensemble de deux possibilités, et que l'une d'elles se réalise.

Dans cette partie, nous commençons par présenter la notion de l'information et de la quantité d'information selon Shannon. Ensuite, nous expliquons la fonction d'Entropie ainsi que l'Information Mutuelle.

1.3.1 Approche intuitive

Dans le cas d'un ensemble Ω qui contient plusieurs possibilités, le message consiste à spécifier un élément de Ω : si Ω contient deux éléments, alors on transmet une unité d'information. Si Ω contient 2^n éléments, on peut spécifier un élément de Ω en envoyant n informations élémentaires. Par exemple, si Ω= {bleu, rouge, vert, jaune} on a 4 éléments = 2^2, et on peut numéroter les éléments de Ω et donner la décomposition en base 2 : 0 = bleu, 1 = rouge, 2 = vert, 3= jaune. Le codage est alors : bleu = 00, rouge = 01, vert = 10, jaune = 11 ; on constate que pour transmettre un message de Ω on a besoin de 2 bits d'information (2 unités d'information).
En s'appuyant sur le codage binaire, la quantité d'information nécessaire pour spécifier un élément parmi un ensemble Ω de possibilités revient à transmettre $log_2||\Omega||$ unités d'information.

En effet, lorsque la cardinalité de Ω est une puissance de 2 on retrouve bien le nombre de bits nécessaire pour transmettre un élément : $log_2(2^n) = n$. La formulation proposée par Shannon étend ce résultat à un nombre quelconque d'éléments.

Considérons maintenant l'information suivante : l'événement réalisé appartient à un sous-ensemble A de l'ensemble Ω des possibilités. Soit $\Omega = \{x_1, x_2, .., x_p, .., x_n\}$ et $A = \{x_1, x_2, .., x_p\}$. L'information considérée, I_1, peut s'écrire $x \in A$, sans que l'on sache de quel x_i il s'agit. Considérons les informations suivantes :

- $I : x = x_i$, avec $x_i \in \Omega$. La quantité d'information associée à I est $log_2||\Omega||$.
- $I_2 : x = x_j$, avec $x_j \in A$. La quantité d'information q associée à I_2 est $log_2||A||$.

On a $I_1 + I_2 = I$, ce qui peut se formuler : choisir un élément dans A (I_1) et préciser cet élément (I_2), correspond à la même quantité d'information que choisir un élément dans Ω (I). Ce qui s'écrit : $log_2||\Omega|| = I_\Omega(A) + log_2||A||$, où $I_\Omega(A)$ représente la quantité d'information de I_1 (l'événement réalisé appartient à A). On a donc : $I_\Omega(A) = log_2||\Omega|| - log_2||A|| = log_2(\frac{||\Omega||}{||A||})$

Par exemple, si Ω contient 8 éléments, codés {000, 001, 010, 011, 100, 101, 110, 111}, on a besoin de transmettre 3 unités d'informations pour spécifier un élément de Ω. Si on a précisé que cet élément appartient au sous-ensemble A = {000, 001, 010, 011} on n'a besoin de transmettre que 2 unités d'information pour spécifier un élément particulier de A. Donc, pour dire que l'événement appartient à A on a eu besoin de $log_2||\Omega|| - log_2||A||$, soit 3-2=1 unité d'information (dans cet exemple, il suffit de transmettre le premier bit).

La mesure d'information logarithmique précédente est algébrique, c'est-à-dire qu'elle ne prend pas en compte la probabilité de choix d'un message. Nous considérons maintenant le cas où Ω est un ensemble où tous les événements sont équiprobables. La probabilité qu'un élément quelconque de Ω soit dans la partie A est $\frac{||A||}{||\Omega||}$. La quantité d'information apportée par la réalisation d'un événement de A est : $log_2(\frac{||\Omega||}{||A||}) = -log_2(\frac{||A||}{||\Omega||}) = -log_2 P(A)$

1.3.2 Détermination d'une fonction pour mesurer la quantité d'Information

Supposons qu'on a un ensemble d'événements possibles avec une distribution de probabilités. Pour un événement x avec une probabilité p(x), soit h(x) la quantité d'information apportée par la réalisation de x. Pour trouver h(x)[8] :

- h doit être une fonction croissante f de l'improbabilité de x, parce que tant que la probabilité augmente la quantité d'information (incertitude) diminue. L'incertitude d'un événement x est l'inverse de sa probabilité et l'on peut poser :
$$h(x) = f(\frac{1}{P(x)})$$
- Un événement certain P(x)=1 n'apporte aucune information ; la quantité d'information est nulle et f doit vérifier : f(1)=0[9].
- La quantité d'information apportée par la réalisation de deux événements indépendants est la somme de leurs quantités d'information respectives, soit :
$$h(x,y) = f(\frac{1}{P(x,y)}) = f(\frac{1}{P(x)P(y)}) = f(\frac{1}{P(x)}) + f(\frac{1}{P(y)}) = h(x) + h(y)$$

Ces critères conduisent à utiliser la fonction logarithme[10]. Le choix de la base de logarithme définit l'unité d'information. On choisit la base 2 de telle sorte que l'alternative entre deux termes équiprobables apporte une unité d'information, que Shannon a appelée bit (**bi**nary digi**t**).

[8] Nous suivons ici la présentation faite par Louis Wehenkel dans son cours sur la théorie de l'information [Wehenkel, 2001].

[9] $\lim_{x \to 1} f(x) = 0$.

[10] Supposons maintenant que l'on a un ensemble d'événements possibles avec des probabilités $P_1, P_2, ..., P_n$. Pour un événement x avec une probabilité P_i, appelons H(x) la quantité d'information apportée par la réalisation de x. Pour déterminer la fonction H, Shannon a imposé trois critères :

- H doit être une fonction continue des P_i.
- Si tous les p_i sont égaux ($P_i = \frac{1}{n}$), H doit être une fonction monotone croissante de n.
- Si un choix est décomposé en deux choix successifs, H doit être la somme pondérée des H des choix constituants.

Shannon a montré que la seule fonction satisfaisant ces trois critères était de la forme :

$$H = -\sum_i P_i log P_i$$

$$h(x) = log\frac{1}{P(x)} = -logP(x) \text{ unités d'information.}$$

Dans la théorie de l'information, les concepts de quantité d'information et de quantité d'incertitude sont équivalents. Si on a une source d'information (S, P) où le choix d'un élément de S a une probabilité P, la probabilité de choisir x_i est $P(x_i)$. Avant que l'événement x_i ne se produise, on a une quantité d'incertitude sur son résultat, et après l'exécution de cet événement, on a une quantité d'information gagnée sur la source. Par exemple : si $P(x_1) = 1$ et $P(x_i) = 0$ pour i>1 alors x_1 est toujours choisi ; dans ce cas, on n'a pas d'incertitude et on n'a pas non plus d'information.
Shannon utilise les concepts de quantité d'information, de liberté de choix et d'incertitude, de manière équivalente. La quantité d'information moyenne mesure le nombre minimum de bits nécessaires en moyenne pour coder de façon binaire les symboles émis par une source. La quantité d'information associée à un événement peut également être vue comme une mesure de la réduction de l'incertitude suite à l'observation d'un événement [Roman, 1992].

Conclusion sur la quantité d'information

On voit que la quantité d'information est d'autant plus importante que l'événement était a priori improbable du point de vue de l'observateur. Inversement, si un événement est a priori très probable (à la limite certain) l'information qui lui est associée sera faible (à la limite nulle). Selon les propriétés du logarithme, nous avons :

- x > 1 $\Longrightarrow$ log x >0
- 0 <x $\leq$ 1 $\Longrightarrow$ log x $\leq$ 0

Donc :

- P(x) augmente $\Longrightarrow$ h(x) diminue
- P(x) > P(y) $\Longrightarrow$ log P(x) >log P(y) $\Longrightarrow$ - log P(x) < - log P(y)$\Longrightarrow$ h(x) < h(y)

Donc, nous pouvons dire que quand la probabilité augmente, l'information propre demandée pour quantifier l'événement diminue.

1.3.3 Entropie

On définit h(x) comme la quantité d'information apportée par la réalisation de l'événement x. Donc :

$$\text{h(x)= -log P(x)}$$

La quantité d'information moyenne pour une source d'information ou un ensemble d'événements X est définie comme l'espérance mathématique de l'information propre fournie par la réalisation de cet événement de l'ensemble :

$$H(X) = E\{h(X)\} = -\sum_i P(x_i)logP(x_i)$$

Propriétés

- L'entropie d'un attribut X à n valeurs est maximale et vaut log(n) lorsque la loi de X est uniforme. c'est-à-dire que l'incertitude de X est la plus grande si toutes les valeurs possibles ont la même probabilité de se réaliser. Exemple : pour un ensemble de 4 événements équiprobables $\frac{1}{4}$ l'entropie est : $H(X) = -\frac{4}{4} * log\frac{1}{4} = log4$.
- L'entropie augmente lorsque le nombre de valeurs possibles augmente.
- Shannon utilise l'entropie H(X) pour calculer le nombre moyen de bits nécessaires à la réalisation de X.

L'entropie conditionnelle

D'après Shannon dans la théorie de la communication, l'entropie conditionnelle H(X|Y) est la quantité d'information (incertitude) moyenne dans la source de message quand le signal est reçu. D'une autre façon, on peut dire que H(X|Y) est la quantité d'information apportée par la réalisation de X sachant que Y a été réalisée.

$$\begin{aligned} H(Y|X) &= \sum_i P(X=x_i)H(Y|x_i) \\ &= -\sum_i P(X=x_i)\sum_j P(Y=y_j|X=x_i)logP(Y=y_j|X=x_i) \\ &= -\sum_i \sum_j P(x_i)P(y_j|x_i)logP(y_j|x_i) \\ &= -\sum_i \sum_j P(y_j,x_i)logP(y_j|x_i) \end{aligned} \tag{1.3}$$

L'entropie jointe

H(X,Y) est la quantité d'information moyenne apportée par la réalisation de deux événements X, Y.

$$\begin{aligned} H(X,Y) &= -\sum_i \sum_j P(X=x_i,Y=y_j)logP(X=x_i,Y=y_j) \\ &= -\sum_i \sum_j P(X=x_i,Y=y_j)log(P(X=x_i)P(Y=y_j|X=x_i)) \\ &= -\sum_i \sum_j P(x_i,y_j)logP(y_j|x_i) - \sum_i \sum_j P(x_i,y_j)logP(x_i) \\ &= H(Y|X) - \sum_i P(x_i)logP(x_i) \\ &= H(Y|X) + H(X) \end{aligned} \tag{1.4}$$

Nous avons : $\sum_j P(Y=y_j,X=x_i) = P(X=x_i)$ Nous pouvons donc calculer l'entropie conditionnelle à partir de l'équation 1.4 comme suit :

$$\text{H(Y|X) = H(Y,X)-H(X)}$$

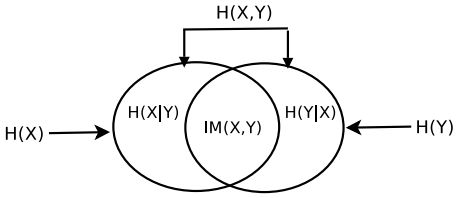

FIG. 1.2 – L'Information Mutuelle

1.3.4 Information Mutuelle

Nous avons vu précédemment que l'entropie d'une variable est la mesure de son degré d'incertitude. il semble naturel de dire que l'obtention d'information sur cette variable diminue son incertitude. La quantité d'information obtenue sur cette variable peut donc être mesurée par la diminution de son entropie. Nous avons également vu que l'entropie conditionnelle H(X|Y) peut être interprétée comme la quantité d'information demandée sur X, une fois que l'on possède déjà une information sur Y. Donc, le fait de connaître une information sur Y va réduire l'information demandée sur X. L'information que l'on peut gagner (figure 1.2) est appelée le gain d'information ou l'Information Mutuelle (IM).

L'information mutuelle IM(X,Y) est la réduction de l'incertitude sur X lorsque Y est connu :

$$\begin{aligned} IM(X,Y) &= H(Y) - H(Y|X) \\ &= H(X) - H(X|Y) \end{aligned}$$

L'information mutuelle pour deux variables aléatoires X, Y mesure la relation de dépendance entre les deux variables. Plus la valeur de l'information mutuelle augmente, plus la dépendance entre X et Y est forte.
Selon (Shannon,1949) :
Dans la théorie de l'information, on peut définir l'Information Mutuelle de la manière suivante : On a une source d'information, un émetteur des messages et un récepteur :
H(X) = l'incertitude moyenne des messages envoyés (émis) ou l'entropie de la source des messages.
H(Y) = l'incertitude moyenne (l'information ou l'entropie) des signaux reçus.
H(X/Y) = l'incertitude moyenne dans la source de messages quand le signal reçu est connu.
H(Y/X) = l'incertitude moyenne dans les signaux reçus quand les messages émis sont connus.
Si on fait la différence on gagne une quantité d'information ; on cherche toujours à trouver le gain d'information maximum. IM(X,Y) est le nombre moyen de bits nécessaires pour spécifier les messages émis moins le nombre de bits nécessaires pour spécifier les messages émis après avoir reçu un signal. On peut dire aussi que la capacité d'un canal est le maximum de l'Information Mutuelle moyenne IM(X,Y) où X,Y sont les variables aléatoires associées respectivement à l'entrée et à la sortie de ce canal.

Propriétés

– L'Information Mutuelle est symétrique.

$$
\begin{aligned}
IM(X,Y) &= H(Y) - H(Y|X) \qquad (1.5)\\
&= -\sum_j P(Y=y_j) log P(Y=y_j) + \sum_i \sum_j P(Y=y_j, X=x_i) log P(Y=y_j|X=x_i)\\
&= -\sum_j P(Y=y_j) log P(Y=y_j) + \sum_i \sum_j P(Y=y_j, X=x_i) log \frac{P(Y=y_j, X=x_i)}{P(X=x_i)}\\
&= -\sum_j P(y_j) log P(y_j) + \sum_i \sum_j P(y_j, x_i) log P(y_j, x_i) - \sum_i \sum_j P(y_j, x_i) log P(x_i)\\
&= -\sum_j \sum_i P(y_j, x_i) log P(y_j) + \sum_i \sum_j P(y_j, x_i) log P(y_j, x_i) - \sum_i P(x_i) log P(x_i)\\
&= +\sum_j \sum_i P(y_j, x_i)(log P(y_j, x_i) - log P(y_j)) - \sum_i P(x_i) log P(x_i)\\
&= +\sum_j \sum_i P(y_j, x_i) log \frac{P(y_j, x_i)}{P(y_j)} - \sum_i P(x_i) log P(x_i)\\
&= -\sum_i P(x_i) log P(x_i) + \sum_j \sum_i P(y_j, x_i) log P(x_i|y_j)\\
&= H(X) - H(X|Y)
\end{aligned}
$$

Nous avons : $\sum_i P(y_j, x_i) = P(y_j)$ et $\sum_j P(y_j, x_i) = P(x_i)$

– Le conditionnement diminue l'incertitude ; cela signifie que H(X) ≥ H(X|Y).
– L'Information Mutuelle moyenne de X et de Y est toujours positive, car H(X) ≥ H(X|Y) donc H(X)-H(X|Y) ≥ 0.
– IM(X,Y) = 0 si X et Y sont indépendants. Parce que H(X|Y) = H(X) alors IM(X, Y) = H(X)-H(X|Y) = H(X)- H(X) = 0.

On peut définir une mesure d'information comme dans la figure 1.2 pour n'importe quels ensembles de variables aléatoires. L'Information Mutuelle a pour but de minimiser l'entropie d'un système. Elle mesure l'information commune entre deux objets ; si cette quantité est faible alors la dépendance entre les deux objets est faible et inversement. On a vu aussi que l'Information Mutuelle est la différence entre l'incertitude initiale et l'incertitude conditionnelle.

Information Mutuelle conditionnelle

On peut introduire la notion d'Information Mutuelle (moyenne) entre deux variables X et Y conditionnellement à l'événement (Z=z) où Z est une troisième variable :

$$IM(X,Y|Z=z) = H(X|Z=z) - H(X|Y,Z=z) \qquad (1.6)$$

En multipliant les deux membres de l'équation 1.6 par P(Z=z) et en sommant sur toutes les valeurs possibles de Z, on obtient :

$$\sum P(Z=z) IM(X,Y|Z=z) = H(X|Z) - H(X|Y,Z)$$

Le membre de gauche pouvant être interprété comme IM(X,Y|Z), on définit l'Information Mutuelle entre X et Y sachant Z par :

$$IM(X,Y|Z) = H(X|Z) - H(X|Y,Z) = H(Y|Z) - H(Y|X,Z)$$
$$IM(X,Y|Z) = \sum_{i,j,k} P(x_i,y_j,z_k) log \frac{P(x_i,y_j,z_k)}{P(x_i|z_k)P(y_j,z_k)}$$

L'Information Mutuelle moyenne conditionnelle a les mêmes propriétés que l'Information Mutuelle moyenne.

1.3.5 Information Mutuelle Normalisée

Nous avons remarqué que d'une part, l'Information Mutuelle est une mesure de dépendance entre les attributs. Elle augmente quand la relation entre les attributs augmente. Par exemple, si la dépendance entre deux attributs (A et B) est assez importante, la quantité d'information demandée pour réaliser A diminue lorsque l'information sur B est connue. D'autre part, on sait que plus un attribut a de valeurs, plus son Information Mutuelle, relativement à la classe, a tendance à être élevée [Crémilleux, 1991]. Dans ce cas, l'Information Mutuelle favorise les attributs ayant un nombre de valeurs assez grand. Le problème se pose quand ces attributs ne sont pas assez pertinents et les informations qu'ils portent sont faibles. De plus, dans une base de données, nous pouvons avoir des attributs ayant un nombre de valeurs assez grand et qui classent parfaitement les objets autant que des attributs qui ne sont pas pertinents mais qui possèdent un nombre de valeurs assez grand.

L'Information Mutuelle entre deux attributs ne dépend pas seulement du nombre de valeurs des attributs mais également de la distribution des valeurs d'un attribut relativement à l'autre dans la base. La meilleure solution pour rendre l'Information Mutuelle insensible au nombre de valeurs de l'attribut testé est de la normaliser.

Pour normaliser le gain d'information, plusieurs propositions ont été faites :

- On peut diviser le gain par log m (où m est le nombre de valeurs de l'attribut testé). Des résultats expérimentaux [Kononenko *et al.*, 1984] ont montré que ce facteur semble pénaliser les attributs ayant un nombre élevé de valeurs.
- [Quinlan, 1986, Quinlan, 1993] utilise le ratio du gain qui consiste à diviser l'Information Mutuelle par l'entropie de l'attribut testé (à condition que celle-ci ne soit pas nulle).
- Si on divise l'Information Mutuelle par la cardinalité des espaces D_x et D_y, on obtient une normalisation qui favorise les attributs ayant peu de réponses possibles.
- Il y a également d'autres stratégies de normalisation de l'Information Mutuelle mais nous avons utilisé la normalisation proposée par Lobo et Numao [Lobo, 2000, Lobo et Numao, 2001]((*cf.* section 3.4.2)) qui divise l'Information Mutuelle entre deux attribut X et Y par $(log|D_x| + log|D_y|)$[11].

1.4 Conclusion

Ce chapitre était une introduction rapide sur les notions de probabilités. Nous avons également décrit l'entropie et l'Information Mutuelle qui sont des mesures fondamentales dans la théorie de l'information. L'entropie sert à quantifier l'incertitude d'un attribut et l'Information Mutuelle aide à mesurer la réduction de l'incertitude pour un attribut lorsqu'un autre attribut est connu. Par conséquence, elle aide à calculer le degré de dépendance entre deux attributs et à choisir le bon attribut lors de la construction d'un arbre de décision. Nous notons qu'il existe d'autres mesures comme l'*indice d'impureté de Gini*, qui sera expliqué dans le chapitre suivant.

Pour éviter de diminuer la capacité de cette mesure lorsqu'un attribut a un nombre de valeurs assez important, nous allons utiliser dans la suite de notre travail l'Information Mutuelle

[11] Le logarithme de cardinalité de X plus le logarithme de cardinalité de Y.

normalisée selon la méthodologie proposée par Lobo et Numao [Lobo, 2000, Lobo et Numao, 2001].

La suite de l'état de l'art sera organisé comme suit : dans un premier temps nous allons décrire des différents algorithmes utilisés pour construire un arbre de décision ainsi que les méthodes d'élagage utilisées. Nous allons également décrire les arbres de décision probabilistes. Dans un deuxième temps, nous expliquons les méthodes qui traitent les valeurs manquantes dans le cadre des arbres de décision. Ainsi, les méthodes statistiques disponibles pour la prise de compte des valeurs manquantes.

Chapitre 2

Arbres de Décision

2.1 Introduction

L'apprentissage inductif [Mitchell, 1997] consiste à extraire un modèle à partir d'un ensemble d'exemples de cas résolus par des experts d'un domaine. À partir de cet ensemble, appelé *ensemble d'apprentissage* (voir tableau 2.2), on génère un modèle qui sert à étudier de nouveaux exemples dans le même domaine. Chaque exemple (objet) de cet ensemble est représenté par un vecteur d'attributs et chaque attribut prend un ensemble de valeurs.

Si la classe de chaque vecteur est donnée, on se situe dans le cadre de l'*apprentissage supervisé*, dont les méthodes sont les arbres de décision, les règles d'associations, les réseaux bayésiens, etc. Dans ce cadre, on cherche à utiliser les exemples fournis déjà classés pour apprendre un modèle qui permet ensuite de déterminer la classe de tout nouvel exemple rencontré.

Si aucune classe n'est disponible, on se situe dans le cadre de l'*apprentissage non-supervisé*[12], qui cherche à regrouper les exemples *(clustering)* en mesurant la similarité entre eux (ex : les k-means, la classification ascendante hiérarchique, etc).

Le dernier type d'apprentissage est l'*apprentissage par renforcement*, qui désigne toute méthode adaptative permettant de résoudre le problème de décision séquentielle, c'est-à-dire un problème qui évolue au cours du temps. Les méthodes d'*apprentissage par renforcement* incluent la méthode des Différences temporelles (TD), le Q-Learning, le R-Learning [Russell et Norvig, 2003]. Ce type d'apprentissage est appliqué dans de nombreux domaines comme les jeux (dames, backgammon), la démonstration mathématique, la robotique, etc.

L'induction avec des arbres de décision est l'une des formes d'algorithme d'apprentissage les plus simples et pourtant les plus efficaces [Russell et Norvig, 2003].

Les *arbres de décision* sont une des techniques les plus populaires de l'apprentissage automatique et de la fouilles de données. L'apprentissage par arbre de décision se situe dans le cadre de l'*apprentissage supervisé*, où la classe de chaque objet dans la base est donnée. Le but est de construire un modèle à partir d'un ensemble d'exemples associés aux classes pour trouver une description pour chaque classe à partir des propriétés communes entre les exemples. Une fois ce modèle construit, on peut extraire un ensemble de règles de classement. Ce modèle ou les règles extraites sont ensuite utilisés pour classer de nouveaux objets dont la classe est inconnue. Le classement se fait en parcourant un chemin depuis la racine jusqu'à à une feuille. La classe renvoyée est celle qui est la plus fréquente parmi les exemples de la feuille.

[12] On trouve également l'*apprentissage semi-supervisé* où seulement les classes de certains exemples sont données.

Quantitative (numérique)			Qualitative (symbolique)			
Continue	Discrète		Nominale		Ordinale	
Taille, âge	Binaire	$\neg Binaire$	Binaire	$\neg Binaire$	Binaire	$\neg Binaire$
	{0,1}	{1,2,3,4}	{yes,no}	{A,B,AB,O}	{low, upper}	{+,++,+++}

TAB. 2.1 – Les types de données

Ce chapitre est une introduction générale aux arbres de décision. Nous commençons par décrire le type de données et introduire la notion d'arbre de décision. Nous décrivons ensuite l'algorithme général de construction d'un arbre de décision ainsi que les méthodes existantes. Nous passons ensuite à l'élagage d'un arbre de décision. Finalement, nous expliquons les arbres de décision probabilistes.

2.1.1 Les Types de données

Dans ce paragraphe, nous décrivons les types de données dans une base d'apprentissage. Nous pouvons diviser le type d'un attribut en deux grandes catégories :

- **Quantitative** (Numérique) : Si l'ensemble des valeurs qu'il peut prendre est un ensemble de nombres, fini ou infini, ou un intervalle de valeurs réelles. Un attribut X numérique peut être *discret* ou *continu* selon sa nature :
  - **Continu** : Si l'ensemble des valeurs qu'il peut prendre est réel ou un intervalle réel. Il s'agit donc d'un ensemble infini non dénombrable : on ne peut pas énumérer systématiquement l'ensemble de tous les points d'un intervalle réel. Par exemple, X peut être l'âge d'une personne prise au hasard, sa taille, son poids, etc.
  - **Discret** : Si l'ensemble des valeurs qu'il peut prendre est un ensemble numérique fini (comprenant un nombre fini d'éléments) ou un ensemble infini dénombrable (comprenant une infinité de nombres que l'on peut énumérer).
- **Qualitative** (symbolique) : Si l'ensemble des valeurs qu'il peut prendre est non numérique. X peut être par exemple la couleur des yeux d'une personne prise au hasard, sa région de naissance, son sexe, etc.

D'autre part, une donnée numérique ou symbolique peut être **ordinale** ($<$, $>$) si ses valeurs sont ordonnées. Par exemple, l'attribut dont les valeurs sont {bien, très-bien, excellente} est un attribut ordinal symbolique ; l'attribut dont les valeurs sont {1, 2, 3, 4, 5} est un attribut ordinal numérique (discret).

De plus, si les valeurs d'un attribut discret ou symbolique sont **binaires**, on parle d'un attribut binaire, par exemple l'attribut symbolique *sexe* qui prend les valeurs {masculin, féminin} ou un attribut discret qui prend les valeurs {0,1}.

Un attribut symbolique est dit **nominal** si l'ordre n'est pas important, comme le *groupe sanguin* {A, B, AB, O} ou l'*état civil* {marié, célibataire, divorcé, veuf}.

Les types de données sont résumés dans le tableau 2.1.

Un attribut quantitatif discret peut être traité comme une variable qualitative en considérant chaque valeur de l'attribut comme une modalité.

Dans notre travail, nous traitons les attributs symboliques et les attributs discrets. Si les attributs dans la base d'apprentissage sont continus, on applique des méthodes de discrétisation pour les rendre discrets. La discrétisation des attributs continus sera expliquée dans la troisième partie de ce mémoire.

2.1.2 Définition

Un arbre de décision est une structure qui est souvent utilisée pour représenter des connaissances. Il permet de remplacer ou d'assister un expert humain dans la détermination des propriétés d'un objet, c'est l'opération de *classement* (en anglais : *classification*). Un arbre de décision est une représentation d'une procédure de décision pour déterminer la classe d'un objet donné. En général, à chaque nœud interne de l'arbre, il y a un test (question) qui correspond à un attribut dans la base d'apprentissage, et une branche correspondant à chacune des valeurs possibles de l'attribut. À chaque nœud de feuille, il y a une valeur de classe. Les arbres de décision sont construits à partir d'un ensemble d'apprentissage. Un chemin de la racine à un nœud correspond à une série d'attributs (questions) avec leurs valeurs (réponses).
Par exemple, prenons l'arbre de décision donné dans la figure 2.1 et construit à partir de la base de météo (tableau 2.2) : les nœuds internes sont *Temps, Humidité* et *Vent*. Pour classer un objet dont le *temps* est *ensoleillé*, *l'humidité* est *haute* et le *vent* est *faux*, on part de la racine de cet arbre et on pose la question sur *Temps*. La réponse qui est la valeur *ensoleillé* nous aide à déterminer quelle branche on doit choisir pour descendre dans l'arbre. On continue le parcours et on pose une question sur *l'humidité*. La réponse à cette question est *haute*, ce qui nous conduit à arriver à une feuille ayant la valeur A, qui est la classe de cet objet.

2.2 Construction d'un arbre de décision

L'idée centrale qui préside à la construction d'un arbre de décision consiste à diviser récursivement les objets de l'ensemble d'apprentissage en utilisant des tests définis à l'aide des attributs jusqu'à ce que l'on obtienne des feuilles ne contenant (idéalement) que des objets appartenant tous à la même classe. Pour diviser l'ensemble d'apprentissage, on choisit des attributs qui vont minimiser l'impureté dans les sous-arbres ; autrement dit, qui maximisent l'information apportée par les réponses. c'est-à-dire que pour chaque attribut qui n'a pas encore été utilisé, on calcule l'impureté qui reste après son utilisation. Celui qui laisse le moins de désordre est choisi comme étant le prochain nœud de l'arbre de décision[13] ; on répète le processus sur chaque nouveau nœud. Le processus s'arrête quand les feuilles de l'arbre ainsi obtenu contiennent des exemples d'un seul concept (classe) ou quand aucun test n'apporte plus d'amélioration.

Dans toutes les méthodes de construction d'un arbre de décision, on trouve les trois opérateurs suivants (**divide-and-conquer strategy**) :

- Décider si un nœud est terminal : tous les exemples (un ou plus) appartiennent à la même classe (il y a moins d'un certain nombre d'erreurs).
- Sélectionner un test à associer à un nœud.
- Affecter une classe à une feuille. On attribue la classe majoritaire à une feuille.

La construction d'un arbre de décision se base sur le principe de **Top-Down Induction** [Quinlan, 1986] : On commence par construire la racine de l'arbre en continuant jusqu'à la feuille. A chaque étape un test sur le nœud courant est choisi, le choix d'un test se fait en mesurant l'impureté (l'incertitude). Le but de est de trouver l'arbre de décision le plus petit possible selon le principe du *Rasoir d'Occam*[14]. Pour choisir un attribut test, on a besoin de

[13] c'est-à-dire celui qui maximise le gain d'information ou la réduction d'impureté.

[14] Le rasoir d'Occam (ou Ockham) est un principe attribué au moine franciscain et penseur du XIVè siècle William d'Occam. Occam était le village du comté de Surrey où il était né. Le principe énonce : *"les entités ne devraient pas être multipliées sans nécessité." Donc, le principe déclare qu'on ne devrait pas faire plus de prétentions que le minimum requis.* L'énoncé le plus utile du principe pour les savants est *"quand on a deux théories en compétition qui permettent de prédire exactement les mêmes choses, celle qui est la plus simple est la*

calculer le désordre. Une des fonctions utilisée à cette fin est l'index de Gini qui est expliqué plus tard dans CART [Breiman *et al.*, 1984]. Une autre fonction possible est la formule classique de l'entropie, qui a été expliquée dans le chapitre précédent :

$$H(C) = \sum_{i=1}^{i=k} P_i log_2 P_i \tag{2.1}$$

où P_i est la probabilité de la classe C_i dans l'ensemble d'apprentissage. Cette formule explique la quantité d'information. S'il n'y a qu'une seule classe, l'entropie est nulle. Si les k classes sont équiprobables, elle vaut $log_2(k)$ où k est le nombre de classes. Pour partitionner l'ensemble d'apprentissage T, on va calculer l'information gagnante par attribut en utilisant la formule de l'entropie conditionnelle :

$$H(C|A) = \sum_{i=1}^{i=n} P(A = a_i) H(C|a_i) \tag{2.2}$$

où n est le nombre des valeurs possibles pour l'attribut A. Le gain d'information (l'information mutuelle) est :

$$gain(A, C) = IM(A, C) = H(C) - H(C|A) \tag{2.3}$$

On choisit l'attribut qui maximise le gain d'information, c'est-à-dire qui minimise l'entropie. Après avoir choisi un attribut test, on doit partitionner l'ensemble en sous-ensembles selon les valeurs possibles de cet attribut. On crée les branches correspondant à chaque valeur de l'attribut, on crée des nœuds pour chaque sous-ensemble non vide et on répète le processus pour chaque nouveau nœud jusqu'à ce qu'on arrive à un critère d'arrêt. Dans ce cas, on crée une feuille et on associe à cette feuille la classe la plus probable.

Nous pouvons résumer l'algorithme de construction d'un arbre de décision comme suit :

```
Construire-arbre(S, L):  (TDIDT)
  Entrées :
     S: un ensemble d'apprentissage.
     L: liste d'attributs.
  Sortie: un arbre de décision.
     Si un critère d'arrêt est vérifié
                          créer une feuille avec la classe la plus probable.
     Sinon:
       Choisir le meilleur attribut A dans L (gain d'information)
       Partitionner S en n sous-ensembles selon les valeurs de A.
       Créer un noeud pour A.
       Pour toutes les valeurs de A : construire-arbre(Si,L-A)
     Fin si.
```

Exemple

Considérons l'ensemble d'apprentissage donné dans le tableau 2.2 extrait de [Quinlan, 1993] : Pour construire l'arbre de décision à partir de l'ensemble d'apprentissage donné dans le tableau 2.2, on va chercher l'attribut le plus pertinent qui maximise le gain d'information (équation 2.3) :

IM(Décision, Temps) = H(Décision) - H(Décision/Temps) = 0.940 - 0.694 = 0.246

meilleure."

id	Temps	Température	Humidité	Vent	Décision
1	Ensoleillé	Elevée	Haute	Faux	A
2	Ensoleillé	Elevée	Haute	Vrai	A
3	Couvert	Elevée	Haute	Faux	B
4	Pluvieux	Moyenne	Haute	Faux	B
5	Pluvieux	Basse	Normale	Faux	B
6	Pluvieux	Basse	Normale	Vrai	A
7	Couvert	Basse	Normale	Vrai	B
8	Ensoleillé	Moyenne	Haute	Faux	A
9	Ensoleillé	Basse	Normale	Faux	B
10	Pluvieux	Moyenne	Normale	Faux	B
11	Ensoleillé	Moyenne	Normale	Vrai	B
12	Couvert	Moyenne	Haute	Vrai	B
13	Couvert	Elevée	Normale	Faux	B
14	Pluvieux	Moyenne	Haute	Vrai	A

TAB. 2.2 – La base *Météo*

id	Temps	Température	Humidité	Vent	Décision
1	Ensoleillé	Elevée	Haute	Faux	A
2	Ensoleillé	Elevée	Haute	Vrai	A
8	Ensoleillé	Moyenne	Haute	Faux	A
9	Ensoleillé	Basse	Normale	Faux	B
11	Ensoleillé	Moyenne	Normale	Vrai	B

TAB. 2.3 – Le sous-ensemble associé à la valeur *ensoleillé* de *Temps*

IM(Décision, Température) =H(Décision) - H(Décision/Température)= 0.940 -0.910 =0.030
IM(Décision, Humidité) = H(Décision) - H(Décision/Humidité)= 0.940 - 0.786 = 0.154
IM(Décision, Vent) =H(Décision) - H(Décision/Vent)= 0.940 - 0.891 = 0.049

L'attribut qui maximise le gain d'information est *Temps*. Donc, *Temps* est la racine de l'arbre de décision. L'ensemble d'apprentissage sera partitionné en trois sous-ensembles selon les trois valeurs de *Temps*, qui sont *ensoleillé*, *pluvieux* et *couvert*. Les trois sous-ensembles sont donnés dans les tableaux 2.3, 2.4, 2.5.

Pour continuer à construire l'arbre, on recommence le processus en choisissant d'autres attributs pertinents pour chaque sous-ensemble d'apprentissage (tableaux 2.3, 2.4, 2.5). Ce processus s'arrête quand les feuilles de l'arbre ainsi obtenu contiennent des exemples d'un seul concept ou quand aucun test ne donne d'améliorations.
Dans le sous-ensemble correspondant à la valeur *ensoleillé* de l'attribut *Temps* (tableau 2.3), l'attribut le plus pertinent est *Humidité*. Donc, le sous-ensemble est partitionné en deux sous-ensembles selon les valeurs *haute* et *normale* de l'*humidité*.
Dans le tableau 2.4, nous remarquons que tous les objets appartiennent à la classe B; dans ce cas, le processus s'arrête dans cette branche et on crée une feuille avec la classe B.
Dans le sous-ensemble correspondant à la valeur *pluvieux* de l'attribut *Temps* (tableau 2.5), c'est l'attribut *Vent* qui maximise le gain et le sous-ensemble est partitionné en deux sous-ensembles

id	Temps	Température	Humidité	Vent	Décision
3	Couvert	Elevée	Haute	Faux	B
7	Couvert	Basse	Normale	Vrai	B
12	Couvert	Moyenne	Haute	Vrai	B
13	Couvert	Elevée	Normale	Faux	B

TAB. 2.4 – Le sous-ensemble associé à la valeur *couvert* de *Temps*

id	Temps	Température	Humidité	Vent	Décision
4	Pluvieux	Moyenne	Haute	Faux	B
5	Pluvieux	Basse	Normale	Faux	B
6	Pluvieux	Basse	Normale	Vrai	A
10	Pluvieux	Moyenne	Normale	Faux	B
14	Pluvieux	Moyenne	Haute	Vrai	A

TAB. 2.5 – Le sous-ensemble associé à la valeur *pluvieux* de *Temps*

selon les valeurs *vrai* et *faux* du *Vent*.
L'arbre de décision final obtenu est donné dans la figure 2.1.

2.3 Les méthodes de construction d'un arbre de décision

En apprentissage automatique, il existe de nombreux systèmes d'induction qui construisent des arbres décision. Hunt, Martin et Stone [Hunt *et al.*, 1966] ont été les premiers à étudier l'apprentissage automatique à partir des exemples. Leur CLS (Concept Learning System framework) construit un arbre de décision qui essaie de minimiser le coût de classement d'un objet. Il y a deux types de coûts : 1) le coût de la détermination de la valeur d'une propriété de l'objet ; 2) le coût du mauvais classement. CLS utilise une stratégie appelée *lookahead* qui consiste à explorer l'espace de tous les arbres de décision possibles à une profondeur fixe et à choisir une action pour minimiser le coût dans cet espace limité puis avancer d'un seul niveau vers le bas dans l'arbre.

Quinlan [Quinlan, 1979] a proposé ID3, qui reprend certains concepts de CLS. ID3, qui sera détaillé par la suite, a été développé suite à un challenge sur l'étude de fins de parties d'échec. ACLS [Paterson et Niblett, 1982] est une généralisation de ID3. CLS et ID3 requièrent que chaque attribut utilisé pour décrire l'objet prenne ses valeurs dans un ensemble fini. En plus de ce type d'attributs, ACLS permet d'utiliser des attributs dont les valeurs peuvent être entières. ASSISTANT [Kononenko *et al.*, 1984], qui est un descendant d'ID3, permet d'utiliser des attributs continus et construit un arbre de décision binaire. ASSISTANT évite l'*Overfitting* (surajustement) en utilisant une technique d'élagage et il a donné lieu à ASSISTANT-86 [Cestnik *et al.*, 1987]. Un autre descendant d'ID3 est C4 [Quinlan *et al.*, 1987] et C4.5 [Quinlan, 1993] qui sera expliqué plus tard.

Il existe une autre famille de systèmes d'induction, comme l'algorithme de l'étoile AQ [Michalski, 1969], qui induit un ensemble de règles de décision à partir d'une base d'exemples. AQ construit une fonction R qui couvre les exemples positifs et rejette les négatifs. CN2 [Clark et Niblett, 1989] apprend un ensemble de règles non ordonnées de la forme *" SI PRÉMISSE ALORS*

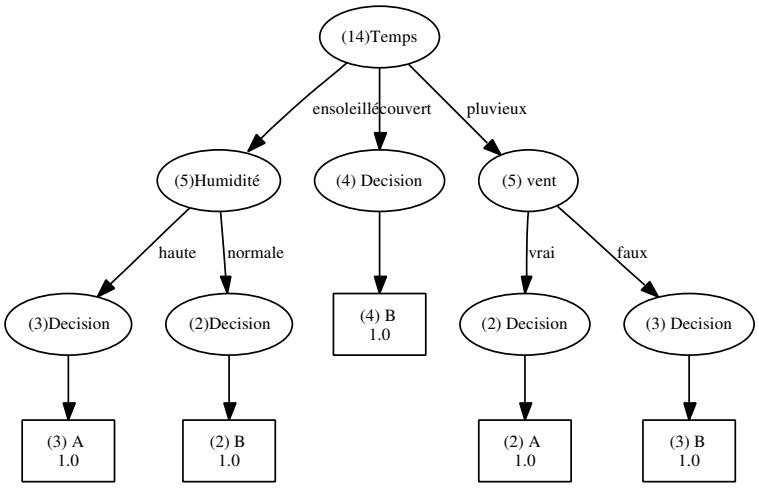

FIG. 2.1 – L'arbre de décision final pour la base *météo*

CLASSE" à partir d'un ensemble d'exemples. Pour ceci, CN2 exécute une recherche descendante (du général au spécifique) dans l'espace des règles, en recherchant la *meilleure* règle, puis enlève les exemples couverts par cette règle, et répète ce processus jusqu'à ce qu'aucune *bonne* règle ne puisse être trouvée. La stratégie de CN2 est semblable à celle de AQ en ce qu'elle élimine les exemples couverts par la règle découverte, mais elle s'en différencie également en ce sens qu'elle spécialise une règle de départ au lieu de la généraliser.

Les statisticiens ont attribué la paternité de la construction des arbres de décision à Morgan et Sonquist [Morgan et Sonquist, 1963], qui ont proposé la méthode AID (Automatic Interaction Detector) ; cette méthode s'applique à des problèmes d'apprentissage dont l'attribut à prédire (la classe) est quantitatif. Morgan et Sonquist ont été parmi les premiers à avoir utilisé les arbres de régression. Plusieurs extensions ont été proposés : ThAID (Theta AID) [Morgan et Messenger, 1973] et ChAID (Chi-2 AID) [Kass, 1980] qui utilise le khi-deux comme écart à l'indépendance pour choisir le meilleur attribut de partitionnement. Il existe également une méthode proposée par [Breiman *et al.*, 1984] appelée CART (Classification and regression Tree) qui construit un arbre de décision binaire. Elle sera également expliquée dans la suite.

Dans le reste de ce chapitre, nous allons expliquer les méthodes les plus populaires : ID3, C4.5 et CART. Ensuite, nous décrivons l'élagage d'un arbre de décision et nous expliquons rapidement les méthodes utilisées. Finalement, nous détaillons les arbres de décision probabilistes que nous avons utilisés dans la suite de notre travail.

2.3.1 ID3

Dans cette méthode [Quinlan, 1979], on génère tous les arbres de décision possibles pour classer correctement l'ensemble d'apprentissage et pour choisir l'arbre le plus simple (principe du rasoir d'Occam)[15]. Le nombre de ces arbres est très grand, mais fini. La structure de base pour cette méthode est itérative. Un sous-ensemble de l'ensemble d'apprentissage est choisi aléatoirement ; ce sous-ensemble est appelé fenêtre (window) et un arbre de décision est construit à partir de cette fenêtre. Cet arbre classe correctement tous les objets dans la fenêtre. On va classer le reste des objets en utilisant cet arbre. Si l'arbre donne des bonnes réponses pour tous

[15]L'arbre le plus simple est l'arbre le plus petit qui pose le moins de questions.

les objets restants (en dehors de la fenêtre) alors l'arbre est correct pour l'ensemble d'apprentissage entier et le processus est terminé ; sinon les objets qui sont mal classés sont ajoutés à la fenêtre et le processus continue, c'est-à-dire qu'un nouvel arbre de décision doit être construit. De cette façon, un arbre de décision correct est trouvé après quelques itérations. Le principe de construction d'un arbre de décision est celui donné au début de ce chapitre, et le choix d'un attribut se base sur l'idée de maximisation du gain d'information. Dans le cas spécial où on a un ensemble C d'objets qui ne contient aucun objet pour une valeur a_i d'un attribut A, le sous-ensemble correspondant à cette valeur est vide. ID3 étiquette cette feuille comme nulle et le classement de chaque objet qui arrive à cette feuille échoue. Une solution est alors générée à partir de l'ensemble C consistant à associer la classe la plus fréquente dans C à cette feuille. ASSISTANT [Kononenko *et al.*, 1984] arrête la construction sur un nœud lorsque tous ses attributs apportent un gain d'information inférieur à un seuil fixé.

2.3.2 C4.5

C4.5 est une extension de l'algorithme de base d'ID3. Dans cette méthode, il y a deux phases : La première consiste à construire l'arbre de décision jusqu'au bout en divisant récursivement l'ensemble d'apprentissage selon la méthodologie expliquée dans la section 2.2. Une fois l'arbre construit, une phase d'élagage est appliquée. L'idée est d'élaguer les branches qui augmentent le taux d'erreurs de classement dans l'arbre de décision. Le processus d'élagage sera expliqué dans la suite. La fonction d'impureté utilisée par C4.5 pour construire un arbre de décision est le ratio du gain (équation 2.4), qui consiste à diviser l'information mutuelle par l'entropie de l'attribut testé.

$$gainRatio(A, C) = \frac{IM(A, C)}{H(A)}$$

Contrairement à ID3, C4.5 traite le problème des attributs continus, le problème des valeurs manquantes ainsi que l'élagage d'un arbre de décision.

2.3.3 CART

Cette méthode [Breiman *et al.*, 1984] permet d'inférer les arbres de décision binaires, c'est-à-dire que tous les tests étiquetant les nœuds de décision sont binaires. Les attributs peuvent être :

- binaires.
- continus à valeurs réelles.
- discrets ou qualitatifs, à valeurs dans un ensemble fini de modalités.

Un **Split**[16] est un test (question à poser) sur un attribut choisi pour diviser l'ensemble d'apprentissage ; chaque split dépend des valeurs d'un seul attribut.
Le problème de construction d'un arbre de décision binaire est de savoir comment utiliser la base d'apprentissage pour déterminer les tests (splits) binaires. L'idée principale est de sélectionner chaque split de manière à ce que les données dans chaque sous-ensemble descendant soient plus homogènes que les données dans le nœud père.

- Pour les attributs continus : toutes les questions possibles (splits) sont de la forme : $\{Isx_m \leq c\ ?\}\ c \in [-\infty, \infty]$
- Pour les attributs discrets et qualitatifs : toutes les questions possibles prennent la forme suivante : $\{Isx_m \in c?\}$ où c est un sous-ensemble de $\{b_1, b_2, ..., b_L\}$ par exemple.

[16] Les notions de test et de split (dichotomie) sont équivalentes.

Ces questions binaires forment tous les splits possibles. Par exemple, si on a 4 attributs x_1, x_2, x_3 qui sont des attributs continus et $x_4 \in \{b_1, b_2, b_3\}$, alors toutes les questions possibles sont de la forme :

Is $x_1 \leq 3.2$?

Is $x_3 \leq -6.8$?

Is $x_4 \in \{b_1, b_2\}$? etc.

Sur un nœud t, les instances qui répondent *oui* à une question posée sur ce nœud sont associées à la partie gauche de l'arbre et les instances qui répondent *non* à une question posée sur un nœud t sont associées à la partie droite de l'arbre. Le nombre de tests (splits) à explorer va dépendre de la nature des attributs :

- À un attribut binaire correspond un test binaire.
- À un attribut discret ou qualitatif ayant n modalités, on peut associer $2^{n-1} - 1$ tests binaires possibles parce que $\{x_m \in S\}$ et $\{x_m \notin S\}$ génèrent la même partition. Par exemple : si un attribut x_m prend ses valeurs dans l'ensemble $\{b_1, b_2, b_3\}$, alors on a 3 tests binaires possibles qui sont : $x_m \in \{b_1, b_2\}$, $x_m \in \{b_1, b_3\}$ et $x_m \in \{b_2, b_3\}$. Les ensembles compléments sont respectivement : $x_m \in \{b_3\}, x_m \in \{b_2\}, x_m \in \{b_1\}$.
- Pour un attribut continu, il y a une infinité de tests envisageables. On découpe l'ensemble des valeurs possibles en segments ; ce découpage peut être fait par un expert ou de façon automatique.

L'algorithme de construction d'un arbre de décision binaire en utilisant CART parcourt, pour chaque nœud, les M attributs $(x_1, x_2, ..., x_M)$ un par un, en commençant par x_1, et en continuant jusqu'à x_M. Pour chaque attribut, il explore tous les tests possibles (splits) et il choisit le meilleur split (dichotomie) qui maximise la réduction en impureté. Ensuite, il compare les M meilleurs splits pour choisir le meilleur d'entre eux. La fonction qui mesure l'impureté devra atteindre son maximum lorsque les instances sont équitablement réparties entre les différentes classes et son minimum lorsqu'une classe contient tous les exemples (le nœud est pur). Il existe différentes fonctions qui satisfont ces propriétés ; la fonction utilisée par CART est la fonction de Gini (indice d'impureté de Gini). La fonction de Gini sur un nœud t avec une distribution de probabilités des classes sur ce nœud P(j|t), j=1,..,J est :

$$G(t) = i(t) = \phi(p(1/t), p(2/t), ..., p(J/t)) = 1 - \sum_j P(j|t)^2 \tag{2.4}$$

Si un split s sur un nœud t partitionne le sous-ensemble associé à ce nœud en deux sous-ensembles gauche t_G avec une proportion p_G et droite t_D avec une proportion p_D, on peut définir la mesure de réduction d'impureté comme suit :

$$\Delta i(s,t) = i(t) - p_G \times i(t_G) - p_D \times i(t_D)$$

Par conséquent, sur chaque nœud, si l'ensemble de splits candidats est S, l'algorithme cherche, le meilleur split s^* tel que :

$$\Delta i(s^*, t) = max_{s \in S} \Delta i(s, t) \tag{2.5}$$

Supposons que nous avons obtenu quelques splits et que nous sommes arrivés à un ensemble de nœuds terminaux $\tilde{T}$. L'ensemble des splits, utilisés dans le même ordre, détermine l'arbre binaire T. Nous avons I(t) = i(t) p(t). Donc, la fonction d'impureté sur l'arbre est :
$I(T) = \sum_{t \in \tilde{T}} I(t) = \sum_{t \in \tilde{T}} i(t)p(t)$; i(t) est la mesure d'impureté sur le nœud t et p(t) est la probabilité qu'une instance appartienne au nœud t.
Il est facile de voir que la sélection des splits qui maximisent $\Delta i(s,t)$ est équivalente à la sélection

des splits qui minimisent l'impureté *I(T)* dans tout l'arbre. Si nous prenons n'importe quel nœud $t \in \tilde{T}$ et nous utilisons un split s qui partitionne le nœud en deux parties t_D et t_G. Le nouvel arbre $\acute{\tilde{T}}$ possède l'impureté suivante :

$$I(\acute{\tilde{T}}) = \sum_{\tilde{T}-\{t\}} I(t) + I(t_D) + I(t_G)$$

parce que nous avons partitionné le sous-ensemble arrivé à t en t_D et t_G. Donc, la réduction de l'impureté de l'arbre est :

$$\begin{aligned} I(T) - I(\acute{\tilde{T}}) &= \sum_{t\in\tilde{T}} I(t) - \sum_{\tilde{T}-\{t\}} I(t) - I(t_D) - I(t_G) \\ &= I(t) - I(t_D) - I(t_G) \end{aligned}$$

Cela dépend seulement du nœud t et du split s. Donc, pour maximiser la réduction d'impureté dans l'arbre sur un nœud t, on maximise :

$$\Delta I(s,t) = I(t) - I(t_G) - I(t_D) \tag{2.6}$$

Les proportions p_G, p_D sont définies comme suit : $p_D = p(t_D)/p(t)$, $p_G = p(t_G)/p(t)$ et $p_G + p_D = 1$. Donc, (2.6) peut être écrire comme suit :

$$\Delta I(s,t) = [i(t) - p_G \times i(t_G) - p_D \times i(t_D)]p(t) = \Delta i(s,t)p(t)$$

Puisque p(t) est la seule différence entre $\Delta i(s,t)$ et $\Delta I(s,t)$, le même split s^* maximise les deux expressions.

Le critère d'arrêt initial (stop splitting) utilisé par CART était très simple :
pour un seuil $\beta > 0$, un nœud est déclaré terminal (feuille) si max $\Delta I(s,t) \leq \beta$. L'algorithme associe à chaque nœud terminal la classe la plus probable.

2.3.4 Les critères d'arrêt de construction d'un arbre de décision

Il y a plusieurs critères d'arrêt possibles lors de la construction d'un arbre de décision.
- Tous les objets appartiennent à la même classe ou il n'y a plus d'attribut à utiliser.
- La profondeur de l'arbre atteint une limite fixée (=nombre d'attributs utilisés).
- Le nombre de feuilles atteint un maximum fixé.
- Le nombre d'instances par nœud est inférieur à un seuil fixé.
- Le gain d'information maximum obtenu est inférieur à un seuil fixé.
- La qualité de l'arbre n'augmente plus de façon sensible. Par exemple, aucun attribut n'améliore la qualité de l'arbre.

Le fait de construire l'arbre jusqu'au bout selon le premier critère ci-dessus, jusqu'à ce qu'aucun attribut ne reste à utiliser ou tous les objets appartiennent à la même classe, favorise les feuilles ayant peu d'objets et diminue la fiabilité de l'arbre lors du classement d'un nouvel exemple. Par conséquent, le taux d'erreurs de classement augmente. Le premier critère n'aide donc pas à construire l'arbre de décision le plus simple. Les autres critères aident à stopper la construction lorsqu'un seuil est atteint, comme le nombre d'instances par feuille, la profondeur de l'arbre, le nombre de feuilles dans l'arbre, etc. Ces critères se situent dans le cadre de l'*élagage* et plus précisément dans son premier type appelé *pré-élagage*. Les méthodes de l'élagage sont détaillées dans le paragraphe suivant.

2.4 Élagage

Un des problèmes connus lors des phases de construction et de classement est que la taille de l'arbre grandit de manière linéaire avec la taille de la base d'apprentissage. De plus, les arbres de décision complexes peuvent avoir des taux d'erreur très élevés à cause du sur-ajustement *(over-fitting)*[17] qui peut se produire lorsque l'ensemble d'apprentissage contient des données bruitées (*noise*[18]) ou qu'il ne contient pas certains exemples importants, ou encore lorsque les exemples sont trop spécifiques. L'élagage est une des solutions pour réduire ces taux d'erreurs en simplifiant l'arbre par suppression de quelques branches. Plusieurs techniques d'élagage [Mingers, 1989] ont été proposées pour éviter le sur-ajustement. On distingue deux approches principales : Le pré-élagage(*cf.* section 2.4.1) et le post-élagage(*cf.* section 2.4.2).

2.4.1 Pré-élagage

Le pré-élagage a pour but d'arrêter la construction de l'arbre de décision à l'avance même si les feuilles ne sont pas pures ; c'est-à-dire qu'on décide ou non de continuer à développer un certain nœud. Cette étape, nommée également *stopping* [Quinlan, 1993], cherche à trouver la meilleure division pour un sous-ensemble d'apprentissage et à l'évaluer en utilisant par exemple le gain d'information ou d'autres mesures d'impureté ; si cette évaluation ne dépasse pas un certain seuil, alors cette division est rejetée et l'arbre pour ce sous-ensemble est la feuille la plus convenable en choisissant la classe la plus fréquente dans ce sous-ensemble. Mais un seuil suffisamment élevé pour supprimer les attributs non pertinents risque de supprimer aussi les attributs pertinents. Il est donc possible d'utiliser d'autres critères d'arrêt (*cf.* section 2.3.4).
Une autre technique proposée par Quinlan et appelée *Chi-square pruning*, citée dans [Crémilleux, 1991, Quinlan, 1993, Kononenko, 1998] utilise le test du chi-deux pour déterminer si l'attribut sélectionné est pertinent ou non pour construire le sous-arbre. Autrement dit, on arrête le processus de construction sur un nœud lorsque tous les attributs candidats sont jugés indépendants de la classe. Le résultat de cette technique n'est pas tellement satisfaisant.
[Cestnik *et al.*, 1987] et [Kononenko *et al.*, 1984] utilisent des paramètres ad-hoc avec des seuils empiriques pour stopper la construction d'un sous-arbre si le nombre d'instances, l'estimation du meilleur attribut ou le taux d'erreurs de classement dans l'ensemble d'apprentissage sur le nœud courant devient assez petit.
Une autre idée a été introduite par [Bratko et Knononenko, 1987] qui vise à partitionner plusieurs fois l'ensemble d'instances associé au nœud courant en deux parties : une pour l'apprentissage et l'autre pour le test pour construire un seul niveau pour chaque partitionnement. Ensuite la base de test est classée dans le sous-arbre construit dont la profondeur est 1 et le taux d'erreurs de classement est calculé. Ce processus est répété pour chaque partitionnement. Si le taux moyen d'erreur dans le sous-arbre est plus grand que le taux d'erreur dans le nœud courant, alors ce nœud devient une feuille et la construction de l'arbre est ainsi stoppée pour ce nœud. Mais cette méthode n'est pas utilisable en pratique parce que sa complexité est très élevée.

[17] Si la base d'apprentissage est bruitée, les règles classeront parfaitement les exemples utilisés, mais moins bien les nouveaux exemples.

[18] Par exemple : La valeur d'un attribut est mal enregistrée dans la base ; on a deux objets identiques qui appartiennent à deux classes différentes (attributs inadéquats)

2.4.2 Post-élagage

Dans cette approche, l'arbre de décision est simplifié en supprimant un ou plusieurs de ses sous-arbres et en les remplaçant par des feuilles. On construit l'arbre de décision jusqu'au bout et ensuite on l'élague. On estime les erreurs de classification à chaque nœud. Le sous-ensemble est remplacé par une feuille (classe) ou par la branche la plus fréquente. On commence par le fond de l'arbre et on examine chacun des sous-arbres (non-feuille) ; si le remplacement de ce sous-arbre par une feuille ou par sa branche la plus fréquente conduit à prévoir un taux d'erreur plus bas, dans ce cas, on élague le sous-arbre. Les méthodes existantes sont :

Méthode du coût-complexité minimal

L'idée qui a été introduite par [Breiman *et al.*, 1984] est de construire une séquence d'arbres $T_0, T_1, ..., T_L$ qui minimisent une fonction appelée *Cost complexity metric*. Cette fonction combine deux facteurs : le taux d'erreur de classement et le nombre de feuilles dans l'arbre en utilisant un paramètre α. Pour différentes valeurs de α[19], cette technique génère différents arbres, depuis T_0 qui est l'arbre de décision initial jusqu'à T_L qui est une feuille. L'idée de base est que l'arbre T_{i+1} est obtenu par l'élagage de tous les nœuds de l'arbre T_i ayant la valeur la plus basse de α. Ensuite, le meilleur arbre construit est sélectionné. Pour estimer le taux d'erreurs pour chaque arbre, les auteurs proposent d'utiliser deux méthodes différentes, l'une basée sur la Validation Croisée et l'autre sur une nouvelle base de test.

Méthode de l'erreur réduite : Reduced error pruning

Au lieu de construire une séquence d'arbres puis choisir le meilleur d'entre eux comme dans la méthode précédente, Quinlan a suggéré une méthode plus directe [Quinlan, 1987b]. Cette méthode prend une base de test dont chaque instance est classée dans l'arbre initial T. Pour chaque sous-arbre S qui ne forme pas une feuille de l'arbre T, la méthode examine le taux de mal-classés en utilisant la base de test, d'une part quand le sous-arbre est élagué[20], et d'autre part quand il n'est pas élagué. Si le taux de mauvais classement après l'élagage diminue par rapport à celui avant l'élagage et si le sous-arbre S ne contient aucun sous-arbre avec la même propriété, on remplace le sous-arbre par une feuille. Le processus continue tant qu'aucune substitution n'augmente le nombre d'erreurs sur la base de test. Comme avec la méthode précédente, l'erreur réduite génère également une séquence d'arbres mais la logique suivie ici est plus claire parce que le dernier arbre construit avec cette méthode est le meilleur arbre élagué et est le plus petit par rapport aux taux d'erreurs de classement. L'inconvénient de ces deux méthodes est l'utilisation d'une base de test.

Méthode de l'erreur pessimiste : Pessimistic pruning

Cette méthode est également due à Quinlan [Quinlan, 1987b] et elle n'a pas besoin d'utiliser une base de test. Quand l'arbre initial T est utilisé pour classer les N instances dans la base d'apprentissage à partir de laquelle l'arbre a été généré et sur une feuille qui contient K cas dont J sont mal classés, le rapport $\frac{J}{K}$ est une estimation optimiste, qui n'est pas fiable, du taux de mal classés pour des nouveaux objets. Dans ce cas, Quinlan propose d'utiliser une correction continue d'une distribution binomiale dans laquelle J est remplacé par $J + \frac{1}{2}$.

[19] α est une réduction moyenne du nombre de mal classés par feuille.
[20] le sous arbre est remplacé par sa feuille la plus pertinente

Pour un sous-arbre S qui contient $L(S)$ feuilles, le taux pessimiste de mal classés avec la correction précédente sur ce sous-arbre est :

$$\frac{\sum(J+\frac{1}{2})}{\sum K} = \frac{\sum J + \sum \frac{1}{2}}{\sum K} = \frac{\sum J + \frac{L(S)}{2}}{\sum K} \tag{2.7}$$

Donc, il y a $\sum J + \frac{L(S)}{2}$ cas qui sont mal-classés parmi $\sum K$ dans le sous-arbre S où l'erreur standard des cas mal-classés peut être calculée comme suit :

$$SE = \sqrt{\frac{(\sum J + \frac{L(S)}{2}) \times (L(S) - (\sum J + \frac{L(S)}{2}))}{L(S)}} \tag{2.8}$$

Si S était remplacé par sa meilleure feuille, E est le nombre d'instances de la base d'apprentissage mal classées sur cette feuille ; cette méthode pessimiste remplace un sous-arbre par la meilleure feuille quand $E + \frac{1}{2}$ est inférieur à l'erreur standard corrigée (SE $+\sum J + \frac{L(S)}{2}$) du sous-arbre S. Tous les sous-arbres sont examinés seulement une seule fois pour savoir s'ils doivent être élagués ou non. Cependant, les sous-sous-arbres des sous-arbres élagués ne sont jamais testés. Cette méthode est plus rapide que les autres parce qu'elle teste chaque sous-arbre une seule fois.

L'élagage basé sur le principe MDL

Le principe MDL (Minimum Description Length) [Rissanen, 1984] consiste à choisir parmi plusieurs théories celle dont le codage est minimal. Ce principe est une approche naturelle pour éviter le problème d'*overfitting*. L'idée de base est qu'un sous-arbre est élagué si la *longueur de description* du classement de la base d'apprentissage dans tout l'arbre, augmentée de la *Description Length* de l'arbre, est supérieure à celle qu'on obtient si le sous-arbre est élagué. Plusieurs méthodes ont été proposées pour élaguer un arbre de décision en utilisant ce principe. Nous trouvons ces travaux dans [Quinlan et Rivest, 1989, Kovacic, 1994, Mehta *et al.*, 1995, Kononenko, 1998].

Minimum error pruning

Niblett et Bratko [Niblett et Bratko., 1986] ont proposé une méthode (voir également [Esposito *et al.*, 1997]) pour trouver un seul arbre élagué qui devrait donner un taux d'erreur minimum lorsqu'on l'utilise pour classer des nouveaux cas. Soit C l'attribut classe ayant k valeurs $c_1, c_2, .., c_k$. Dans cette méthode les valeurs de la classe sont équiprobables. Supposons que sur une feuille nous avons observé n objets dont n_j cas appartiennent à la classe j. Niblett et Bratko ont défini le taux d'erreur attendu, appelé l'erreur statique, comme suit :

$$E_k = \frac{(n-n_j+k-1)}{(n+k)}$$

En utilisant cette mesure la stratégie d'élagage est la suivante :
Sur chaque nœud (non feuille) dans l'arbre, Niblett et Bratko calculent le taux d'erreur attendu (l'erreur statique) si le sous-arbre est élagué et remplacé par une feuille. Ensuite ils calculent le taux d'erreur attendu si le sous-arbre n'est pas élagué en utilisant le taux d'erreur sur chaque branche ; cette erreur, appelée erreur dynamique, est la moyenne pondérée de l'erreur de tous ses fils. Cette procédure est récursive. Si l'élagage d'un nœud conduit à une erreur statique supérieure à l'erreur dynamique de ce nœud, le nœud n'est pas élagué. Dans le cas contraire,

quand l'erreur dynamique d'un nœud est supérieure à son erreur statique le nœud est élagué et remplacé par une feuille.
Plus tard, Cestnik et Bratko [Cestnik et Bratko, 1991] ont proposé une méthode, appelée *m-estimate*, qui est plus générale que la méthode précédente parce qu'elle prend en compte la probabilité a priori. *m-estimate* prend la forme suivante :

$$P = \frac{n_j + m * P_j}{n + m}$$

Où n_j est le nombre de cas de la classe j, n est le nombre de tous les cas sur ce nœud, P_j est la probabilité a priori de la classe j, m est le paramètre de l'estimation.

Méthode de la valeur critique

Pendant la construction d'un arbre de décision, une mesure d'impureté est utilisée pour déterminer l'attribut nœud. L'ensemble d'apprentissage est ensuite partitionné selon les valeurs de l'attribut test choisi. La stratégie d'élagage suivie dans cette méthode proposée par [Mingers, 1987, Mingers, 1989] consiste à spécifier une valeur critique et élaguer les nœuds qui n'arrivent pas à cette valeur, c'est-à-dire les nœuds dont la valeur de la mesure utilisée pour sélectionner l'attribut nœud ne dépasse pas la valeur critique. Cette méthode génère une séquence d'arbres élagués en augmentant la valeur critique. Un seul arbre est choisi, de la même manière que dans la méthode du *coût-complexité minimal*. La valeur critique choisie dépend de la mesure utilisée pendant la construction de l'arbre. Plus la valeur critique est élevée, plus l'arbre élagué est petit. Il peut arriver que les conditions d'élagage soient vérifiées par un nœud t mais pas par tous ses fils. Dans ce cas, on garde la branche T_t parce qu'elle contient des nœuds pertinents.

Error-based pruning

Cette méthode est proposée par Quinlan et implémentée en C4.5 [Quinlan, 1993]. L'idée est la suivante : si N est le nombre d'instances parvenues à une feuille et E le nombre d'objets mal classés sur cette feuille, alors le taux d'erreur optimiste observé est $\frac{E}{N}$ (observer E événements dans N essais). Pour trouver la probabilité qu'une erreur se produise dans l'ensemble de cas couverts par la feuille, Quinlan a utilisé les limites de confiance de la distribution binomiale[21]. Pour un niveau de confiance *cf*, on calcule la limite supérieure de cette probabilité $U_{cf}(E, N)$ en utilisant les limites de confiance de la distribution binomiale. Pour simplifier, C4.5 considère que le taux d'erreur prédit sur une feuille (la probabilité qu'une erreur se produise) est égal à cette borne $U_{cf}(E, N)$ où le niveau de confiance donné par Quinlan est 25%.
Pour décider si on élague un sous-arbre ou pas on doit d'abord :

- Calculer le taux d'erreur prédit sur chaque feuille de ce sous-arbre en utilisant la limite supérieure de confiance de la distribution binomiale : $U_{25\%}(E, N)$.
- Calculer le nombre d'erreurs prédites sur chaque feuille, qui est égal au nombre de cas sur cette feuille × le taux d'erreur prédit : $N \times U_{25\%}(E, N)$.
- Calculer le nombre total d'erreurs prédites sur toutes les feuilles du sous-arbre (la somme de toutes les erreurs des feuilles) : $\sum N \times U_{25\%}(E, N)$.

[21] $U_{cf}(E, N)$ est la limite supérieure de confiance de la distribution binomiale. Pour un niveau de confiance cf :

$$\sum_{x=0}^{x=E} (NE)\, U_{cf}^{x} (1 - U_{cf})^{N-x} = cf \tag{2.9}$$

N est le nombre d'instances sur une feuille, E est le nombre de mal classés sur cette feuille.

– Calculer le nombre d'erreurs prédites sur le nœud père de ces feuilles ; qui est le nombre de cas sur ce nœud × le taux d'erreur prédit sur ce nœud.
– Comparer les valeurs obtenues lors des deux étapes précédentes pour prendre une décision concernant l'élagage du sous-arbre : si l'erreur donnée par le nœud père est inférieure à l'erreur donnée par les feuilles alors on élague le sous-arbre de ce nœud père.

Indice de qualité

La plupart des méthodes d'élagage minimisent le taux d'erreur de classement [Crémilleux, 2000]. L'objectif de cette méthode est de ne pas élaguer systématiquement les sous-arbres ayant un taux d'erreur égal au taux d'erreur de leur racine. [Fournier et Crémilleux, 2000] ont proposé une méthode d'élagage, appelée *DI pruning*[22], qui prend en compte la complexité des sous-arbres, et qui est capable de garder les sous-arbres avec leurs feuilles pour extraire des règles de décision pertinentes, même si cela n'améliore pas le taux de bon classement. Pour cela, cette méthode définit un indice de qualité qui est un compromis entre la profondeur de l'arbre et son impureté. L'indice de qualité d'un arbre T de racine Ω est donné par l'équation suivante :

$$ID(T) = \sum_{i=1}^{m} \alpha_i (1 - \varphi(\Omega'_i)) f(prof_T(\Omega'_i))$$

où $\Omega'_1, .., \Omega'_m$ sont les feuilles de l'arbre T ayant Ω comme racine, α_i est le poids du noeud Ω'_i dans l'arbre, $(1 - \varphi)$ est une mesure de pureté [Fournier, 2001], et f est une fonction d'amortissement qui dépend de $prof(_T(\Omega))$ la profondeur du noeud dans l'arbre[23].

L'idée de cette méthode est de comparer ID(T) avec la pureté de sa racine. Quand ID(T) est inférieur à la pureté de sa racine ($DI(T) \leq 1 - \varphi(\Omega)$), on élague T et on le remplace par sa racine, qui devient une feuille dans l'arbre initial. Cette méthode prend en compte la qualité des données et a été implémentée dans le logiciel *UnDeT*[24][Fournier et Crémilleux, 2002].

2.5 Classement

Généralement, le processus du classement commence par la racine de l'arbre de décision, on descend dans l'arbre selon les valeurs des attributs nœuds dans l'objet à classer jusqu'à ce qu'on arrive à une feuille, et la classe associée à cette feuille est la classe de l'objet en question.

2.6 Conclusion

Nous avons expliqué dans la partie précédente les méthodes de construction d'un arbre de décision et les techniques existantes pour l'élaguer. La structure d'un arbre de décision est simple à expliquer et à interpréter, même à des profanes. Un médecin par exemple peut lire facilement un arbre de décision et classer un nouvel objet. De plus, nous avons vu que l'élagage rend d'une part l'arbre de décision plus simple et plus petit. D'autre part, il aide à éviter l'*Overfitting* lors du classement d'un nouveaux cas.

Un arbre de décision idéal est un arbre où toutes les instances d'apprentissage arrivant à une feuille appartiennent à la même classe avec un nombre d'instances par feuille supérieur à un seuil

[22] Depth-Impurity.
[23] La profondeur d'une feuille dans T est calculée à partir de la racine de l'arbre initial.
[24] `http://litis.univ-lehavre.fr/~fournier/UnDeT/Undet.html`

fixé. Cette catégorie d'arbres est rarement rencontrée dans le monde réel. Plus généralement, un arbre de décision se présente sous une forme où les instances d'une feuille appartiennent à plusieurs classes. Dans ce cas, on associe classiquement à chaque feuille la classe la plus probable. Nous avons vu également ci-dessus que le fait d'élaguer un arbre de décision conduit à supprimer des sous-arbres et à les remplacer par la feuille la plus pertinente. Par conséquent, une feuille dans un arbre élagué peut avoir des instances appartenant à des classes différentes. Dans ce cadre, au lieu d'associer la classe majoritaire à cette feuille, on peut donner comme résultat de classement la fréquence des classes sur cette feuille. Cette idée nous a conduit à étudier les arbres de décision probabilistes.

Dans la partie suivante, nous allons décrire les arbres de décision probabilistes où on associe à chaque feuille une distribution de probabilités de classes. Plusieurs chercheurs se sont intéressés aux arbres de décision pour améliorer leur performance en estimation de probabilités. Quelques-uns ont proposé de remplacer les feuilles par des classifieurs bayésiens naïfs, comme la méthode *NBTree* [Kohavi, 1996], d'autres utilisent la correction de Laplace [Domingos et Provost, 2000, Provost et Domingos, 2003], etc. Nous décrivons ces méthodes par la suite.

2.7 État de l'art sur les arbres de décision probabilistes

Selon [Breiman *et al.*, 1984], dans le cas du diagnostic en médecine, pour un patient qui pourrait avoir trois maladies m_1, m_2, m_3, il serait préférable d'estimer les probabilités relatives d'avoir m_1, m_2, m_3 plutôt que de lui affecter une seule maladie, et ceci même s'il n'y a pas d'information manquante. Ils ont donc proposé de construire des *Class Probability Trees*, où on associe à chaque feuille F la probabilité de chaque classe J sur F ; $P(J|F)$ avec J=1,...,n.
Quinlan a également proposé qu'une feuille puisse contenir des instances de différentes classes, en associant une distribution de classes à chaque feuille.

Selon [Quinlan, 1990, Quinlan, 1987a], pour un seul chemin (une seule règle de classement) de la racine de l'arbre jusqu'à une feuille F passant par les branches $B_1, B_2, ..., B_L$, où chaque branche correspond à une valeur d'attribut test (le résultat d'un noeud), la probabilité qu'un objet E arrive à une feuille F (c'est-à-dire qu'il passe par les branches $B_1, B_2, .., B_L$) est :
$P_E(F) = P_E(B_1, B_2, ..., B_L) = P_E(B_1) * P_E(B_2|B_1) * P_E(B_3|B_1, B_2) * ... * P_E(B_L|B_1, B_2, ..., B_{L-1})$
Si la valeur de chaque attribut est connue, chacune des probabilités précédentes est 0 ou 1. Si la valeur de l'attribut correspondant à la branche B_i n'est pas connue, sa probabilité est calculée par $P(B_i|B_1, B_2, ..., B_{i-1})$ à partir de l'ensemble d'apprentissage initial, i.e., la proportion des cas (instances) arrivés au ième test qui prennent la branche B_i. Parce qu'un cas E avec des valeurs manquantes peut appartenir à plusieurs feuilles, la probabilité que le cas E appartienne à une classe C est : $\sum_F P_E(F)P(C|F)$. Avant de décrire les méthodes qui traitent les arbres de décision probabilistes, nous allons rappeler quelques définitions.

Classifieur Bayésien Naïf

Supposons qu'on a un ensemble d'apprentissage ayant n attributs $A_1, .., A_n$. C est l'attribut classe et c est la valeur qu'elle peut prendre. Un exemple E est représenté par un vecteur $(a_1, a_2, ..., a_n)$ où a_i est la valeur de l'attribut A_i. Un classifieur bayésien naïf est défini comme suit [Su et Zhang, 2004] :

$$C_{nb}(E) = arg_c max P(c) \prod_{i=1}^{n} P(a_i|c) \tag{2.10}$$

où $P(a_i|c)$ est estimée à partir de la base d'apprentissage. Le classifieur bayésien naïf est basé sur l'hypothèse que tous les attributs sont indépendants étant donné la valeur de la classe (indépendance conditionnelle). Pendant la phase d'apprentissage, on construit un modèle dont la classe est la racine et les attributs sont les feuilles. Ainsi, on estime les probabilités : $P(A_i = v_j|C_k)$ et $P(C_k)$. Ces probabilités seront utilisées pendant la phase de test.

Par exemple, si on veut utiliser un classifeur bayésien naïf pour classer un exemple de la base *météo* où *Temps* est *ensoleillé*, *Humidité* est *haute*, *Température* est *élevée* et *Vent* est *faux*, on utilise les probabilités calculées dans la phase d'apprentissage et on applique l'équation 2.10 comme suit :

P(A) × P(ensoleillé|A)× P(haute|A)× P(faux|A)× P(élevée|A)

P(B) × P(ensoleillé|B) ×P(haute|B)×P(faux|B)× P(élevée|B)

La classe la plus probable est la classe retenue pour l'objet en question.

Conditional Log Likelihood CLL

Pour un ensemble d'apprentissage S ayant n instances et un ensemble d'attributs A, et où C est la classe. Le CCL d'un classifieur B est défini comme suit [Liang et Yan, 2006] :

$$CCL(B|S) = \sum_{s=1}^{n} logP_B(C/A)$$

Dans la suite, nous allons décrire les méthodes qui étudient la performance des arbres de décision en estimation de probabilité.

2.7.1 Probability Estimation Trees

Les études menées par plusieurs chercheurs sur les PETs (*Probability Estimation Trees*), citées dans [Domingos et Provost, 2000, Provost et Domingos, 2003], ont montré que les arbres de décision ne produisent pas une estimation de probabilité de classe très significative. Par conséquent, plusieurs chercheurs ont proposé des techniques pour améliorer cette estimation.
Le but d'un arbre de décision est de rendre les sous-ensembles d'apprentissage plus homogènes en fonction des valeurs de classe. Prenons l'exemple d'une feuille telle que 90% du sous-ensemble associé à cette feuille appartient à la même classe (par exemple classe positive[25]) et 10% de l'ensemble est considéré comme mal classé. En classement, chaque objet qui arrive à cette feuille est classé positif avec la probabilité 0.9. Un problème qui rend l'arbre de décision pauvre en estimation de probabilité est le suivant : quand la profondeur de l'arbre est assez grande, il peut arriver que le nombre d'instances par feuille diminue[26]. Par exemple, dans le cas où le sous-ensemble associé à la feuille ne contient que 5 instances de la classe positive la probabilité de la classe positive sur cette feuille est 1.0 (5/5). Dans ce cas, la question qui a été posée est : Est-ce que tous les objets arrivant à cette feuille sont de la classe positive ? une solution directe à ce problème est d'utiliser *la correction de Laplace* [Simonoff, 1998] : Si nous avons une feuille ayant k instances de la classe en question, N est le nombre total d'instances sur cette feuille et C est le nombre total de classes. L'estimation de probabilité comme présentée ci-dessus est $\frac{k}{N}$. La correction de Laplace utilisée pour estimer cette probabilité est $\frac{k+1}{N+C}$. Donc, si on prend

[25] La classe dans cet exemple prend deux valeurs : positive, négative.
[26] Dans ce cas là, la variance est très élevée.

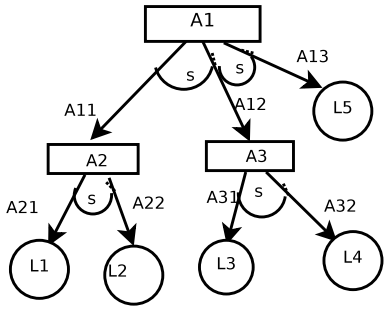

FIG. 2.2 – Exemple sur le facteur de confusion

l'exemple précédent où C=2, k=5, N=5 ; la probabilité sur cette feuille est 1.0. Avec la correction de Laplace, cette probabilité devient $\frac{5+1}{5+2}$= 0.86.
Un autre problème qui rend l'arbre de décision pauvre en estimation de probabilité réside dans l'algorithme de construction, qui cherche à construire l'arbre le plus petit qui donne un meilleur classement. Selon [Domingos et Provost, 2000, Provost et Domingos, 2003] le fait de construire l'arbre jusqu'au bout sans l'élaguer aide à obtenir de bons PETs(*Probability Estimation Trees*). [Domingos et Provost, 2000, Provost et Domingos, 2003] ont comparé C4.5 avec une autre version de C4.5 appelé C4.4 qui ne prend pas en compte l'élagage et utilise la correction de Laplace. Les résultats ont montré que C4.4 est meilleur que C4.5 au niveau de l'estimation de probabilité. Le *Bagging* améliore également la qualité d'estimation de probabilité [Domingos et Provost, 2000, Provost et Domingos, 2003]. Le *Bagging* construit L arbres de décision et calcule leurs moyennes en estimation de probabilité de classe. Chaque arbre est construit en utilisant la même base d'apprentissage. L'inconvénient du *Bagging* est que le coût de calcul est très élevé.

2.7.2 Le facteur de confusion

[Ling et Yan, 2003] ont proposé une méthode pour produire un arbre de décision plus efficace en estimation de probabilité. L'idée de cette méthode est d'estimer la probabilité moyenne qu'une instance appartient à toutes les feuilles dans l'arbre au lieu d'estimer la probabilité qu'une instance appartienne à une seule feuille. Ils ont introduit un paramètre appelé *le facteur de confusion s* sur chaque noeud interne dans l'arbre où s<1. Ce facteur est considéré comme représentant la probabilité des erreurs qui se produisent quand les valeurs de l'attribut changent. Donc, quand un objet est classé dans l'arbre, il possède une probabilité faible (s) d'aller dans d'autres branches dans l'arbre. Donc, la probabilité qu'un objet e appartienne à une autre feuille dans l'arbre est $P_i * s^j$ où s le facteur de confusion, j le nombre de fois que les valeurs des attributs ont changé par rapport à l'objet e, et P_i la probabilité attachée à cette feuille. Par exemple, dans l'arbre de la figure 2.2, il y a deux attributs qui arrivent à la feuille L_1 : A_1 avec la valeur A_{11} et A_2 avec la valeur A_{21}. Si l'objet e à classer arrive à cette feuille, la contribution de la feuille L_1 est $1 * P(L_1)$, et la contribution de la feuille L_2 est $s * P(L_2)$, parce que l'attribut A_2 change sa valeur dans cette feuille. La contribution de la feuille L_3 dépend de la valeur de l'attribut A_3 dans l'objet e. Si la valeur de A_3 est A_{31}, L_3 a une seule valeur différente et sa contribution est $s * P(L_3)$. Si la valeur de A_3 est A_{32}, L_3 a deux valeurs différentes et sa contribution est $s * s * P(L_3)$. La probabilité finale P_e est la probabilité moyenne sur toutes les feuilles dans l'arbre, $P_e = \frac{\sum P_i * s^j}{\sum s^j}$. Cette méthode a été testée sur plusieurs bases de données artificielles et réelles. Les expérimentations ont montré que cet algorithme est meilleur que C4.5 et C4.4 quand

il utilise la correction de Laplace, l'élagage et un facteur de confusion s égal à 0.3.

2.7.3 NBTree

[Kohavi, 1996] a proposé une méthode hybride, appelée *NBTree*, qui utilise les avantages des arbres de décision et de l'algorithme Naïve-Bayes. Elle construit un arbre de décision dont chaque nœud correspond à un seul attribut. Ensuite, un classifieur bayésien naïf est construit pour chaque feuille en utilisant les données associées à cette feuille. Cette méthode calcule l'utilité de chaque nœud en discrétisant les attributs continus. La 5-validation croisée est utilisée pour estimer l'exactitude de l'utilisation de l'algorithme Naïve-Bayes sur ce nœud. Cette méthode calcule tout d'abord l'utilité de chaque attribut test pour choisir l'attribut qui sera le nœud racine ayant l'utilité maximale C_1. Ensuite, l'utilité D_1 du test choisi est également calculée, elle est la somme pondérée de l'utilité de ses nœuds, où le poids accordé à un nœud est proportionnel au nombre de cas qui arrivent jusqu'à ce nœud. Si cette utilité D_1 n'est pas meilleure que l'utilité C_1 du nœud courant, on crée une feuille sous la forme d'un classifieur bayésien naïf pour le nœud courant. Dans le cas contraire, l'attribut sera choisi comme racine et les instances seront partitionnées en sous-ensembles selon ses valeurs. Le processus sera appliqué récursivement sur chaque sous-ensemble. Pour éviter les tests ayant peu d'instances, NBTree considère qu'un test est significatif si la réduction d'erreur est supérieure à 5% et si le nombre d'instances sur le nœud est au moins 30. Cette méthode est implémentée dans Weka [Witten Ian et Frank, 2005]. Selon [Kohavi, 1996], les performances de cette méthode sont bien meilleures que celles de C4.5 et de l'algorithme Naïve-Bayes. En revanche, la complexité en temps de construction est très élevée par rapport à un arbre de décision standard. *NBTree* ignore les instances ayant des valeurs manquantes.

2.7.4 Flexible NBTree

[Wang *et al.*, 2004] ont proposé une version plus flexible de *NBTree*, appelée *Flexible NBTree*, qui étend *NBTree* sur deux points : 1) elle utilise une stratégie de post-discrétisation pour construire l'arbre de décision. Pour cela, elle choisit d'abord le test le plus pertinent, ensuite elle discrétise les attributs continus ; 2) elle utilise un algorithme Naïve-Bayes général sur chaque nœud-feuille pour traiter les attributs continus et nominaux au lieu d'un algorithme Naïve-Bayes standard qui traite les attributs pré-discrétisés et les attributs nominaux. Selon [Wang *et al.*, 2004] la méthode *Flexible NBTree* est plus performante que *NBTree*.

2.7.5 Les estimateurs de la densité sur l'arbre de décision

Cette méthode combine les arbres de décision et l'estimation par la méthode du noyau (kernel estimation). [Smyth *et al.*, 1995] ont utilisé la *Kernel Density Estimation*[27] pour améliorer la performance d'un arbre de décision en estimation de probabilité de classe. De plus, l'utilisation de l'information fournie par la structure de l'arbre de décision peut réduire le nombre de variables utilisées dans l'estimation de la densité par la méthode du noyau.

L'idée de cette méthode est d'estimer sur chaque nœud de l'arbre la probabilité a posteriori de classe sachant les données. Pour cela, elle utilise un estimateur de densité par la méthode du noyau où l'estimateur de densité utilise seulement les attributs qui ont été utilisés pendant la partition des branches pour arriver à ce nœud. Cette méthode est applicable dans les deux phases (construction et exploitation) d'un arbre de décision. Seul l'algorithme de classement

[27] La Kernel Density Estimation est une technique non-paramétrique pour estimer la densité de probabilité.

est expliqué dans [Smyth *et al.*, 1995], où on utilise un algorithme standard comme C4.5 ou CART pour générer l'arbre. Ensuite, une estimation de la densité par la méthode du noyau est associée à chaque feuille dans l'arbre. Pour chaque nouvel objet, le classement se fait par parcours de l'arbre de décision depuis la racine jusqu'à une feuille. Losqu'on arrive à une feuille, cette méthode estime la probabilité de classe de l'objet en question en générant une estimation de densité locale pour chaque valeur de classe ; on utilise seulement le sous-ensemble appartenant à la classe et les attributs ayant déjà été utilisés pour arriver à cette feuille. Cette méthode combine les meilleures caractéristiques des arbres et de l'estimateur de densité. La méthode proposée cherche à trouver les bonnes dimensions via la structure de l'arbre et elle utilise ces dimensions pour construire l'estimation de densité locale. Les résultats expérimentaux ont montré que cette méthode fournit des améliorations par rapport à l'utilisation des arbres de décision seuls ou des méthodes de densité seules.

2.7.6 Curtailment

Une autre méthode appelée *Curtailment* [Zadrozny et Elkan, 2001] se base sur l'idée suivante : corriger l'estimation de probabilité sur chaque feuille en la décalant vers la probabilité moyenne, i.e. le taux de base b. Cette méthode associe à chaque instance test qui arrive à une feuille ayant un nombre d'instances assez petit l'estimation de probabilité du nœud-père de cette feuille si le nœud-père contient assez d'exemples pour estimer la probabilité. Si le nœud père de cette feuille continue à avoir peu d'exemples, la méthode cherche à estimer la probabilité de cette instance à partir du nœud-grand-père et ainsi de suite jusqu'à ce qu'on arrive à la racine de l'arbre. La fréquence observée sur la racine de l'arbre est le taux de base de l'ensemble d'apprentissage. Quand on classe une instance en utilisant cette méthode, on arrête la recherche quand on arrive à un nœud ayant un nombre d'exemples inférieur à v[28]. La fréquence du nœud-père de ce nœud est associée à l'instance en question.

Dans les méthodes d'élagage, un nœud est remplacé par sa feuille la plus pertinente, i.e. toutes les branches d'un nœud sont supprimées et remplacées par une seule feuille. En revanche, cette méthode[29] élimine quelques branches et en garde d'autres pour le même nœud en fonction du nombre d'instances associées aux fils de ce nœud et leur fiabilité en estimation de probabilité. Elle élimine donc les branches ayant un nombre d'exemples inférieur à v. Par conséquent, selon les valeurs des attributs dans un objet, un nœud peut être utilisé comme nœud interne pour quelques exemples et comme feuille pour d'autres exemples, parce qu'une ou plusieurs de ses branches ont été supprimées.

2.7.7 Lazy Option Tree

[Margineantu et Dietterich, 2002] ont combiné deux méthodes, *Lazy Tree* [Friedman *et al.*, 1996] et *Option Tree* [Buntine, 1993, Kohavi et Kunz, 1997], pour obtenir *Lazy Option Tree*. *Lazy Decision Tree* construit l'arbre de décision quand une instance test arrive ; cet arbre se compose d'un seul chemin depuis la racine à une feuille. Par conséquent, l'arbre est construit pour classer seulement cette instance test. Les tests utilisés dans un tel arbre sont satisfaits seulement par l'instance test. En conséquence, le temps demandé pour classer toutes les instances dans la base est assez élevé par rapport à un algorithme standard. Pour estimer la probabilité de classe sur la

[28] v est un paramètre de la méthode. v peut être choisi en utilisant la validation-croisée ou en utilisant une heuristique telle que $bv = 10$, b est le taux de base. $v=200$ a été choisi pour toutes les expérimentations.

[29] Cette méthode n'est pas considérée comme une méthode d'élagage parce qu'on ne supprime pas toutes les branches d'un nœud mais seulement celles ayant peu d'exemples.

feuille dans *Lazy Decision Tree*, on calcule la proportion des instances de la base d'apprentissage qui appartiennent à la classe k.

Une autre technique qui a été utilisée par les auteurs est *Option Tree* [Buntine, 1993, Kohavi et Kunz, 1997]. Un *Option Tree* est une généralisation de l'arbre de décision standard avec des options sur chaque nœud. Sur chaque nœud interne, au lieu d'avoir un seul attribut test avec ses sous-arbres correspondant à ses valeurs, il y a plusieurs tests optionnels avec leurs sous-arbres respectifs. L'estimation de probabilité finale est la moyenne des probabilités estimées sur chacune de ces options.

L'algorithme *Lazy Option Tree* sélectionne sur chaque nœud les meilleurs tests b_t (i.e. ayant le meilleur gain). Ce nombre b_t doit être inférieur à un nombre maximum de tests *MaxTests* par nœud. De plus, un gain minimum *MinG* pour chaque test est fixé par cet algorithme. Si le meilleur test a comme gain la valeur g, chaque test associé à ce nœud doit avoir un $gain > MingG \times g$. Un nombre minimum d'exemples par feuille doit également être fixé. Cet algorithme vérifie et supprime un type particulier de tests pour éviter les redondances [Margineantu et Dietterich, 2002]. Par exemple, si nous avons deux tests U et V, supposons qu'un nœud A possède deux options. Une option teste U et mène à un nœud-fils B ; une autre option teste V et mène un nœud-fils C. On suppose maintenant que le nœud B choisit V comme option. Cette méthode ne permet pas au nœud C de choisir U comme option parce que cela va créer deux chemins identiques dans l'*Option Tree*. Avant de choisir des options sur chaque nœud, cette méthode associe des poids aux instances de chaque classe pour équilibrer les fréquences de classe pondérée.

Les auteurs ont de plus utilisé le Bagging pour améliorer l'estimation de probabilité. Ils ont comparé les *Bagged Lazy Option trees (B-LOTs)* avec les *Bagged Probability Estimation Trees (B-PETs)* [Domingos et Provost, 2000]. Les résultats de l'expérimentation sur des données réelles ont montré que l'estimation de probabilité produite par *B-LOTs* est meilleure que celle de *B-PETs*. Cependant, l'expérimentation sur des données artificielles a montré que la performance de *B-PETs* est meilleure.

2.7.8 Conditional Independence Trees (CITree)

[Zhang et Su, 2004] ont proposé un algorithme de construction d'un arbre de décision basé sur l'indépendance conditionnelle, appelé *Conditional Independence Trees CITree* et inspiré par les méthodes proposées par [Kohavi, 1996, Zadrozny et Elkan, 2001]. Contrairement à un arbre de décision probabiliste qui représente seulement une distribution de probabilités conditionnelle des attributs rencontrés dans le chemin depuis la racine de l'arbre jusqu'à une feuille, *CITree* représente une distribution de probabilités jointe pour tous les attributs sur chaque feuille. *CITree* définit explicitement les dépendances conditionnelles entre les attributs dans chaque chemin dans l'arbre et les indépendances conditionnelles entre le reste des attributs. Les feuilles dans un *CITree* sont des classifieurs bayésiens naïfs (Naïve-Bayes). *CITree* est vu comme une combinaison entre un arbre de décision et le modèle Naïve-Bayes.

Dans un arbre de décision probabiliste, une feuille L représente la probabilité conditionnelle $P(C|A_p(L))$ où C est la classe sur cette feuille. $A_p(L)$ sont les attributs dans le chemin de la racine de l'arbre jusqu'à L ; $A_l(L)$ désigne le reste des attributs. S'il existe une représentation de la probabilité conditionnelle du reste des attributs $P(A_l(L)|A_p(L), C)$ sur chaque feuille L, on l'appelle distribution conditionnelle locale sur L, et chaque feuille, dans ce cas, représente une distribution de probabilités jointe complète sur tous les attributs sachant la classe C. Cette distribution est donnée par l'equation 2.11.

$$P(A|C) = \alpha P(C|A_p(L))P(A_l(L)|A_p(L), C) \tag{2.11}$$

où α est un facteur de normalisation ; $P(A_l(L)|A_p(L), C)$ est la probabilité conditionnelle des attributs restants $A_l(L)$ sachant $A_p(L)$ et la classe C.
Dans ce contexte, un arbre de décision probabiliste est appelé *arbre de probabilité jointe* si chacune de ses feuilles représente la distribution de probabilités conditionnelle $P(C|A_p(L))$ ainsi que $P(A_l(L)|A_p(L), C)$. De plus, un *arbre de probabilité jointe* est appelé *arbre d'indépendance conditionnelle (CITree)* si l'indépendance conditionnelle locale donnée dans l'équation 2.12 est vraie sur chaque feuille dans l'arbre.

$$P(A_l(L)|A_p(L), C) = \prod_{i=1}^{n} P(A_l i|A_p(L), C) \tag{2.12}$$

Pour construire un *CITree*, on choisit les attributs qui rendent l'indépendance conditionnelle locale parmi le reste des attributs vraie autant que possible même si l'impureté des feuilles est élevée. Cet arbre reste un bon *CITree* tant que le reste des attributs sont indépendants. L'algorithme de construction d'un *CITree* est un algorithme récursif : à chaque étape il faut choisir le meilleur attribut qui maximise l'indépendance conditionnelle entre le reste des attributs. Donc, pour chaque attribut A dans la base, cet algorithme partitionne l'ensemble d'apprentissage en sous-ensembles S_i selon les valeurs de A. Ensuite, il crée un classifieur bayésien naïf pour chaque sous-ensemble S_i et il évalue le *CITree* en fonction de l'indépendance, comme expliqué ci-dessus. Une fois choisi l'attribut A_{max} qui maximise l'indépendance, le processus est répété dans chaque sous-ensemble S_i de l'attribut A_{max}, et l'attribut A_{max} est ajouté comme nœud dans l'arbre.

Cette méthode utilise l'approche *pessimistic error-based* comme méthode de post-élagage. Les résultats d'expérimentation [Zhang et Su, 2004] ont montré que la méthode *CITree* est plus performante que C4.5 et l'algorithme Naïve-Bayes. Le problème de valeurs manquantes est traité en utilisant un mécanisme implémenté dans Weka. Mais cette méthode n'est pas assez générale pour représenter n'importe quelle indépendance conditionnelle [Su et Zhang, 2005]. De plus, un problème de duplication peut se produire avec cette méthode. Ainsi, [Su et Zhang, 2005] ont proposé une autre méthode, dont l'idée est de partitionner l'ensemble des attributs en sous-ensembles *CI* en utilisant l'information mutuelle pour calculer la dépendance entre chaque paire d'attributs. Ils ont utilisé un seuil basé sur le principe de *MDL* pour filtrer les attributs ayant une faible dépendance. Ensuite, un *CITree* est construit pour chaque *CI*. Une autre proposition *AUC-CITree* [Su et Zhang, 2004] a été faite, qui construit un arbre de décision dont les feuilles sont des classifieurs bayésiens naïfs et les attributs sont choisis en fonction de *AUC (The Area Under the ROC Curve)*. De plus, [Liang et Yan, 2006] ont proposé une autre méthode, appelée *CLLTree*, qui construit un arbre de décision avec des classifieurs bayésiens naïfs sur les feuilles et les attributs sont choisis en fonction de leur *Conditional Log Likelihood (CLL)*.

2.8 Conclusion

Dans ce chapitre, nous avons décrit les méthodes qui construisent un arbre de décision ainsi que les méthodes d'élagage (pré-élagage, post-élagage). Nous avons également expliqué les arbres de décision probabilistes.

Nous avons vu que le fait de laisser l'arbre de décision croître jusqu'au bout provoque plusieurs problèmes, comme l'*Overfitting*. En outre, plus l'arbre est profond, plus le nombre d'instances par feuille diminue. En revanche, plusieurs travaux ont montré que l'élagage diminue la performance de l'arbre de décision en estimation de probabilité. Dans notre travail, nous allons utiliser une stratégie de pré-élagage basée sur l'information mutuelle entre chaque attribut et la classe. La justification de ce choix sera présentée dans le chapitre 4.

L'inconvénient d'un arbre de décision est qu'il est pauvre en estimation de probabilité de classe, nous avons montré dans ce chapitre que les chercheurs se sont intéressés aux arbres de décision et ont essayé d'améliorer sa performance en estimation de probabilité. Cette idée a été l'objectif de plusieurs travaux ; quelques chercheurs ont gardé la structure de l'arbre mais ont utilisé d'autres mesures pour estimer la probabilité de classe sur chaque feuille comme *l'estimation de densité basée sur la méthode de noyau*, *la correction de Laplace*, *le facteur de confusion*. D'autres ont modifié sa structure en générant un Naïve-Bayes par feuille comme dans *CITree*, *AUC-CITree*, *CLLTree* et *NBTree*, ou en supprimant quelques branches d'un nœud comme dans *Curtailment*. La méthode *Lazy Option Tree* est intéressante lorsqu'on rencontre une valeur manquante car on peut utiliser une autre option pour continuer le classement, mais elle est très coûteuse puisqu'on doit construire un arbre pour chaque instance test. Ces méthodes ne traitent pas le problème des valeurs manquantes dans les données. D'ailleurs, nous pensons que le fait de changer la structure de l'arbre en remplaçant une feuille par un Naive-Bayes augmente le coût de construction. D'ailleurs, l'arbre de décision standard reste plus facile à interpréter qu'un NBTree. La méthode du noyau ignore les instances ayant des attributs inconnus parce que la complexité de calcul devient très élevée.

La question que cette thèse traite n'est pas l'amélioration de la performance d'un arbre de décision en estimation de probabilité. Le problème essentiel est de classer un objet ayant des valeurs manquantes dans un arbre de décision de manière probabiliste. Mais l'étude théorique précédente nous a aidé à réfléchir à la manière dont nous allons procéder lors de la construction d'un arbre de décision pour, d'un côté, éviter l'*overfitting* et d'un autre côté assurer une bonne estimation de probabilité.

Dans la partie suivante, nous allons décrire dans un premier temps les méthodes qui traitent les valeurs manquantes dans les arbres de décision. Nous allons expliquer ensuite l'approche proposée pour traiter ce problème pendant la phase de classement.

Deuxième partie

Le traitement des valeurs manquantes dans les arbres de décision

Chapitre 3

Le problème des valeurs manquantes

3.1 Introduction

Le problème des valeurs manquantes est un problème connu dans le domaine de la fouille de données [Grzymala-Busse et Hu, 2001, Delavallade et Dang, 2007, Heckerman, 1997, Ragel et Crémilleux, 1999, Zheng et Low, 1999], où, dans la base d'apprentissage, on rencontre des objets ayant des valeurs manquantes pour certains attributs. La donnée manquante est considérée comme une donnée complexe dont le problème fournit un thème lié à la fouille de données complexes [Boussaid *et al.*, 2005, Cloppet *et al.*, 2005].

Nous étudions ce problème dans le cadre des arbres de décision pendant la phase de classement. Nous avons expliqué dans le chapitre précédent qu'un arbre de décision est construit à partir d'un ensemble d'apprentissage selon l'approche divide-and-conquer. Une fois l'arbre construit, il est utilisé pour classer de nouveaux objets. Pour cela, on parcourt l'arbre en commençant par la racine et en suivant les branches correspondant aux valeurs de l'objet, jusqu'à une feuille. La classe associée à cette feuille est appelée la classe de l'objet.

Les arbres de décision sont confrontés au problème des données manquantes, à la fois lors de leur construction et lors du classement d'objets. Lors de la construction, l'existence de valeurs manquantes pose problème pour le calcul du gain d'information, nécessaire au choix de l'attribut test, ainsi que pour la partition de l'ensemble d'apprentissage selon l'attribut test choisi. Le classement d'un objet avec des valeurs manquantes soulève également des problèmes lorsqu'un n œud correspondant à un attribut manquant[30] est rencontré dans le parcours de l'arbre.

Dans ce chapitre, nous allons tout d'abord décrire les types de valeurs manquantes. Ensuite, nous étudions les méthodes qui traitent ce problème dans les deux phases (construction et classement).

3.2 Les types des valeurs manquantes

L'algorithme de base pour construire un arbre de décision fait l'hypothèse que la valeur d'un attribut test pour n'importe quelle instance est connue. Cette hypothèse se présente pendant la partition de l'ensemble d'apprentissage en sous-ensembles selon les valeurs possibles pour l'attribut test et pendant l'évaluation des tests possibles. De plus, classer un nouvel objet dans un arbre de décision nécessite de suivre la branche appropriée à l'objet dans chaque noeud (test) de l'arbre. Chaque noeud étant basé sur un seul attribut, la valeur prise par le noeud ne peut être déterminée que si la valeur de l'attribut dans l'objet est connue.

[30] Quand la valeur de l'attribut associé au nœud est inconnue dans l'objet à classer.

Mais les données réelles peuvent avoir des valeurs manquantes qui peuvent se produire parce que la valeur n'est pas pertinente pour un cas particulier, que la valeur n'a pas été enregistrée lors la collection de données ou qu'elle n'est pas déchiffrée par la personne responsable de mettre les données dans la machine.

Le problème des valeurs manquantes peut se présenter à la construction et au cours de l'exploitation de l'arbre ; on rencontre trois types de problèmes [Quinlan, 1993] :

- Comment choisir un attribut pour partitionner l'arbre s'il y a des attributs ayant des valeurs manquantes ?
- Après avoir choisi un attribut (test), comment partitionner l'ensemble d'apprentissage si l'attribut choisi possède une valeur manquante ? Comment associer à une branche les instances avec des valeurs inconnues pour cet attribut ?
- Après la construction de l'arbre : comment classer un nouvel objet qui a des valeurs manquantes ?

Il y a des méthodes qui traitent les problèmes des valeurs manquantes pendant la construction de l'arbre et qui ne fonctionnent pas pendant la phase de l'utilisation (classement), et d'autres qui les traitent pendant les deux phases (construction et classement).

On considère deux types des valeurs manquantes :

- Valeur manquante aléatoire : lorsqu'elle affecte indifféremment toutes les valeurs d'un attribut. Exemple : Une erreur de transmission, un oubli à la saisie, une impossibilité d'effectuer une mesure suite à la panne d'un appareil.
- Valeur manquante non aléatoire (informative) : lorsqu'elle affecte uniquement une certaine valeur d'un attribut. Exemples : un four qui tombe en panne dès que la température de 500 degrés Celsius est dépassée. Seules les valeurs supérieures à ce seuil sont manquantes. La valeur de température est dépendante de la panne du four.

La méthode la plus simple pour traiter les valeurs manquantes pour un attribut est d'ignorer les instances qui les contiennent et de ne pas les prendre en compte lors du calcul du gain. On calcule le gain à partir du sous-ensemble qui contient seulement les instances avec des valeurs connues pour cet attribut. Si le nombre d'instances ayant des valeurs manquantes est assez grand, la taille de l'ensemble d'apprentissage diminue, ce qui rend la base non représentative.

Si on supprime les attributs ayant des valeurs inconnues, la taille de la base ne change pas mais c'est la description de la base[31] qui sera modifiée. Dans ce cas, la perte d'information sera très grande importante et la base ne sera pas intéressante pour représenter le domaine d'application.

Une méthode a été proposée par (Quinlan, 1986) pour traiter les valeurs manquantes informatives ; elle consiste à traiter la valeur manquante comme une autre valeur possible pour chaque attribut (c'est-à dire une valeur à part entière). On ajoute à chaque noeud une branche destinée à collecter les cas arrivés à ce noeud où la valeur est manquante. On a alors deux options pour calculer le gain d'information pour choisir l'attribut test :

- Soit l'on considère que la valeur manquante n'apporte aucune information, et dans ce cas les instances où l'attribut est inconnu ne sont pas prises en compte.
- Dans le cas contraire, où la valeur manquante apporte de l'information, les instances correspondantes doivent être prises en compte pour le calcul du gain.

[31] La description de la base est l'ensemble des attributs qui décrivent les objets.

id	Temps	Température	Humidité	Vent	Decision
1	Ensoleillé	Elevée	?	Faux	A
2	Ensoleillé	Elevée	Haute	Vrai	A
3	Couvert	Elevée	Haute	Faux	B
4	Pluvieux	Moyenne	Haute	Faux	B
5	Pluvieux	Basse	Normale	Faux	B
6	Pluvieux	Basse	Normale	Vrai	A
7	Couvert	Basse	Normale	Vrai	B
8	Ensoleillé	Moyenne	Haute	Faux	A
9	Ensoleillé	Basse	Normale	Faux	B
10	Pluvieux	Moyenne	Normale	Faux	B
11	Ensoleillé	Moyenne	Normale	Vrai	B
12	Couvert	Moyenne	Haute	Vrai	B
13	Couvert	Elevée	Normale	Faux	B
14	Pluvieux	Moyenne	Haute	Vrai	A

TAB. 3.1 – La base *Météo*

Cette méthode pose problème lorsque l'attribut inconnu n'est pas informatif et qu'il tombe dans cette branche. La valeur qui sera associée à cet attribut n'est pas appropriée.

3.3 Le traitement des valeurs manquantes dans les arbres de décision

Nous présentons maintenant les méthodes qui ont été proposées pour traiter le problème des valeurs manquantes dans un arbre de décision, dans les phases de construction et de classement.

3.3.1 La méthode de majorité

La méthode de majorité [Kononenko *et al.*, 1984] choisit, pour un attribut dont la valeur est manquante, la valeur la plus fréquente pour cet attribut parmi toutes les instances qui appartiennent à la classe de l'objet dont cet attribut est inconnu. Pour remplir la valeur manquante d'un attribut A dans un objet appartenant à la classe N, on choisit la valeur la plus fréquente pour cet attribut parmi toutes les instances qui appartiennent à la classe N. Cette méthode ne fonctionne qu'à la construction de l'arbre, parce qu'elle est reliée à la classe à laquelle l'objet qui possède une valeur manquante appartient. On réduit l'ensemble d'apprentissage à un sous-ensemble qui contient seulement les instances (objets) qui appartiennent à la classe de l'objet manquant. Si l'attribut A appartient à la classe N, alors pour chaque valeur A_i possible pour A on calcule :

$$P(A = A_i | classe = N) = \frac{P(A = A_i, Classe = N)}{P(Classe = N)} \tag{3.1}$$

Par exemple, prenons la base donnée dans le tableau 3.1, où l'attribut *Humidité* de la première instance est manquant. Cette instance appartient à la classe A. On calcule la probabilité que *Humidité* prenne la valeur *haute* et la probabilité que *Humidité* prenne la valeur *normale* sachant la classe A, à partir du sous-ensemble donné dans le tableau 3.2.

id	Temps	Température	Humidité	Vent	Decision
2	Ensoleillé	Elevée	Haute	Vrai	A
6	Pluvieux	Basse	Normale	Vrai	A
8	Ensoleillé	Moyenne	Haute	Faux	A
14	Pluvieux	Moyenne	Haute	Vrai	A

TAB. 3.2 – Le sous-ensemble de la base *Météo* appartenant à la classe A

$P(\texttt{humidité} = haute|classe = A) = \frac{3}{4} = 0.75$
$P(\texttt{humidité} = normale|classe = A) = \frac{1}{4} = 0.25$
On associe donc la valeur *haute* à l'attribut *Humidité* dans l'objet 1. Une autre méthode similaire à la *méthode majorité* a été proposée [Fortesa *et al.*, 2006].

Une nouvelle méthode a été proposée [Delavallade et Dang, 2007], qui utilise l'entropie et l'Information Mutuelle pour prédire la valeur manquante dans les données. Cette méthode fonctionne seulement pendant la phase d'apprentissage car la classe est prise en compte lors du calcul de l'Information Mutuelle.

3.3.2 La méthode de Shapiro

Cette méthode est proposée par Shapiro et décrite par [Quinlan, 1986] consiste à construire un arbre de décision pour déterminer les valeurs manquantes d'un attribut A. Elle utilise le sous-ensemble S' de l'ensemble d'apprentissage S contenant les instances ayant des valeurs connues pour A et pour la classe. La classe dans cette méthode est considérée comme un attribut, et l'attribut A joue le rôle de classe. L'arbre construit à partir de l'ensemble S' est utilisé pour classer chaque instance appartenant à l'ensemble S-S', et les valeurs inconnues de A sont alors déterminées. Par exemple, prenons la base donnée dans le tableau 3.1 ; cette méthode va construire un arbre de décision dont la classe est l'attribut *Humidité*, et la classe *Décision* devient un attribut qui participe à la construction de l'arbre. Les instances utilisées sont celles ayant une valeur connue pour *Humidité*. Ces sont les instances de 2 jusqu'à 14.

L'idée de cette méthode est intéressante parce qu'elle utilise l'arbre de décision pour déterminer les valeurs manquantes, mais sa difficulté augmente quand l'objet à classer contient plusieurs attributs ayant des valeurs inconnues : si on rencontre, pendant la construction d'un arbre de décision pour un attribut inconnu, un autre attribut manquant, on doit construire également son arbre de décision, et ainsi de suite. Cela rend cette méthode peu pratique quand il y a beaucoup d'attributs manquants, parce que :

- Si nous utilisions à chaque fois qu'on rencontre une valeur manquante la base d'apprentissage initiale, les attributs ne sont pas tous disponibles parce que nous aurions un risque de blocage. Donc, le nombre de d'attributs utilisés pour classer l'ensemble devient trop petit.
- Si nous prenons uniquement le sous-ensemble d'apprentissage parvenu au nœud où on a rencontré un nouvel attribut inconnu, la taille de ce sous-ensemble peut être petite parce qu'on élimine trop d'instances dans la base d'apprentissage pendant le processus récursif de construction des arbres de décision pour ces attributs.

3.3.3 La valeur la plus commune

On complète la valeur manquante par la valeur la plus fréquente dans tout l'ensemble d'apprentissage [Quinlan, 1989, Clark et Niblett, 1989] pendant la construction ainsi que pendant le

id	Temps	Température	Humidité	Vent	Decision
O	Ensoleillé	?	Haute	Faux	?

TAB. 3.3 – Une instance de la base *Météo*

classement, comme suit :

- Pendant la construction d'un arbre de décision, quand une instance de la base d'apprentissage contient des valeurs manquantes, on remplace l'attribut manquant par sa valeur la plus probable dans le sous-ensemble d'apprentissage ayant des valeurs connues pour cet attribut, sans prendre en compte la classe de cette instance. Par exemple, prenons l'ensemble précédent donné dans le tableau 3.1. L'attribut *Humidité* est remplacé par sa valeur *normale* la plus probable dans la base sans prendre en compte la classe parce que :
  $P(\texttt{humidité} = haute) = \frac{6}{13} = 0.46$
  $P(\texttt{humidité} = normale) = \frac{7}{13} = 0.53$
- Pendant le classement d'un nouvel objet ayant des valeurs manquantes. On suppose que la base d'apprentissage ne possède pas de valeurs manquantes parce qu'elle est complète ou parce qu'on a déjà rempli ses valeurs manquantes. Dans ce cas, on remplace la valeur manquante par sa valeur la plus probable dans toute la base. Par exemple, prenons l'objet donné dans le tableau 3.3. Il contient une valeur manquante pour l'attribut *Température*.
  $P(\texttt{Température= élevée}) = \frac{4}{14} = 0.28$
  $P(\texttt{Température} = moyenne) = \frac{6}{14} = 0.42$
  $P(\texttt{Température} = basse) = \frac{4}{14} = 0.28$
  Donc, l'attribut *Température* sera remplacé par sa valeur *moyenne* la plus probable.

L'inconvénient de la première situation est que s'il y a beaucoup de valeurs manquantes dans la base pour le même attribut, la probabilité donnée ne sera pas valide. L'inconvénient de la deuxième situation est que la valeur manquante de l'attribut *Température* sera remplacée par *moyenne* dans chaque instance à classer où *Température* est inconnue. Il y a une autre version de cette méthode qui remplit la valeur manquante sur un noeud par la valeur la plus fréquente dans le sous-ensemble parvenu jusqu'au noeud.

3.3.4 L'approche de C4.5

L'utilisation de l'arbre pour classer un objet avec des valeurs manquantes a aussi fait l'objet de quelques études, comme *l'approche probabiliste* de C4.5 [Quinlan, 1993], qui consiste à associer un poids à chaque valeur possible d'un attribut. Si cette valeur est connue son poids est 1, et 0 pour toutes les autres valeurs possibles de cet attribut ; si la valeur est inconnue, le poids associé à cette valeur d'attribut est sa fréquence dans le sous-ensemble d'apprentissage correspondant au noeud de cet attribut. Cette approche remplace la valeur manquante par une distribution de probabilités, et le résultat du classement d'un objet avec valeurs manquantes est une distribution de probabilités de classe. Cette méthode fonctionne pendant la construction d'un arbre de décision à partir d'une base ayant des valeurs inconnues ainsi que pendant le classement d'un nouvel objet avec des valeurs manquantes.

Pour choisir l'attribut test ayant des valeurs manquantes dans la base, le gain d'information est calculé à partir de l'ensemble ayant des valeurs connues pour cet attribut. Ce gain est multiplié par la probabilité que cet attribut est connu.

id	Temps	Température	Humidité	Vent	Decision
1	Ensoleillé	Elevée	Haute	Faux	A
2	Ensoleillé	Elevée	Haute	Vrai	A
3	Couvert	Elevée	Haute	Faux	B
4	Pluvieux	Moyenne	Haute	Faux	B
5	Pluvieux	Basse	Normale	Faux	B
6	Pluvieux	Basse	Normale	Vrai	A
7	Couvert	Basse	Normale	Vrai	B
8	Ensoleillé	Moyenne	Haute	Faux	A
9	Ensoleillé	Basse	Normale	Faux	B
10	Pluvieux	Moyenne	Normale	Faux	B
11	Ensoleillé	Moyenne	Normale	Vrai	B
12	?	Moyenne	Haute	Vrai	B
13	Couvert	Elevée	Normale	Faux	B
14	Pluvieux	Moyenne	Haute	Vrai	A

TAB. 3.4 – La base *Météo* ayant une valeur manquante pour *Temps*

$$gain(A, C) = \texttt{probabilité que A est connu} \times (H(C) - H(C|A))$$

Par exemple, prenons l'ensemble d'apprentissage pour la base météo donnée dans le tableau 3.4, l'attribut *Temps* dans l'objet numéro 12 est inconnu *?*

Donc, pour construire un arbre de décision à partir de cette base, on doit calculer le gain d'information entre chaque attribut et la classe, à partir de l'ensemble d'apprentissage ayant des valeurs connues pour *Temps* :
H(Décision)= $-\frac{8}{13}log(\frac{8}{13}) - \frac{5}{13}log(\frac{5}{13}) = 0.961$.
H(Décision|Temps)=$\frac{5}{13}(-\frac{2}{5}log(\frac{2}{5}) - \frac{3}{5}log(\frac{3}{5})) + \frac{3}{13}(-\frac{3}{3}log(\frac{3}{3}) - \frac{0}{3}log(\frac{0}{3})) + \frac{5}{13}(-\frac{3}{5}log(\frac{3}{5}) - \frac{2}{5}log(\frac{2}{5}))$
=0.747
IM(Décision, Temps) =$\frac{13}{14}$(H(Décision) - H(Décision|Temps)) = $\frac{13}{14}$(0.961 - 0.747) = 0.199
IM(Décision, Température) =$\frac{14}{14}$ (H(Décision) - H(Décision|Température))= 0.940 -0.910 =0.030
IM(Décision, Humidité) = $\frac{14}{14}$(H(Décision) - H(Décision|Humidité))= 0.940 - 0.786 = 0.154
IM(Décision, Vent) =$\frac{14}{14}$(H(Décision) - H(Décision|Vent))= 0.940 - 0.891 = 0.049

Par conséquent, l'attribut *Temps* est choisi comme la racine de l'arbre.

Ensuite, on partitionne l'ensemble d'apprentissage en trois sous-ensembles selon les valeurs de *Temps*. Les 13 objets seront partitionnés sans problème. En revanche, l'objet 12 sera affecté aux trois sous-ensembles avec les poids suivants : $\frac{5}{13}$ pour *ensoleillé*, $\frac{3}{13}$ pour *couvert* et $\frac{5}{13}$ pour *pluvieux*. Ces trois sous-ensembles sont donnés dans les tableaux (3.5, 3.6, 3.7)

Ensuite, le sous-ensemble dans le tableau 3.5 est partitionné selon les valeurs de l'attribut *Humidité*. Dans ce cas, on a deux sous-ensembles : l'un correspond à la valeur *haute* avec 3 objets de la classe A et $\frac{5}{13}$ de la classe B; l'autre correspond à la valeur *normale* avec 2 objets de la classe B et 0 de la classe A.

Le sous-ensemble dans le tableau 3.6 contient $3 + \frac{3}{13}$ objets de la classe B et 0 de la classe A.

id	Temps	Température	Humidité	Vent	Décision	poids
1	Ensoleillé	Elevée	Haute	Faux	A	1
2	Ensoleillé	Elevée	Haute	Vrai	A	1
8	Ensoleillé	Moyenne	Haute	Faux	A	1
9	Ensoleillé	Basse	Normale	Faux	B	1
11	Ensoleillé	Moyenne	Normale	Vrai	B	1
12	?	Moyenne	Haute	Vrai	B	5/13

TAB. 3.5 – Le sous-ensemble associé à la valeur *ensoleillé* de *Temps*

id	Temps	Température	Humidité	Vent	Décision	poids
3	Couvert	Elevée	Haute	Faux	B	1
7	Couvert	Basse	Normale	Vrai	B	1
12	?	Moyenne	Haute	Vrai	B	3/13
13	Couvert	Elevée	Normale	Faux	B	1

TAB. 3.6 – Le sous-ensemble associé à la valeur *couvert* de *Temps*

id	Temps	Température	Humidité	Vent	Décision	poids
4	Pluvieux	Moyenne	Haute	Faux	B	1
5	Pluvieux	Basse	Normale	Faux	B	1
6	Pluvieux	Basse	Normale	Vrai	A	1
10	Pluvieux	Moyenne	Normale	Faux	B	1
14	Pluvieux	Moyenne	Haute	Vrai	A	1
12	?	Moyenne	Haute	Vrai	B	5/13

TAB. 3.7 – Le sous-ensemble associé à la valeur *pluvieux* de *Temps*

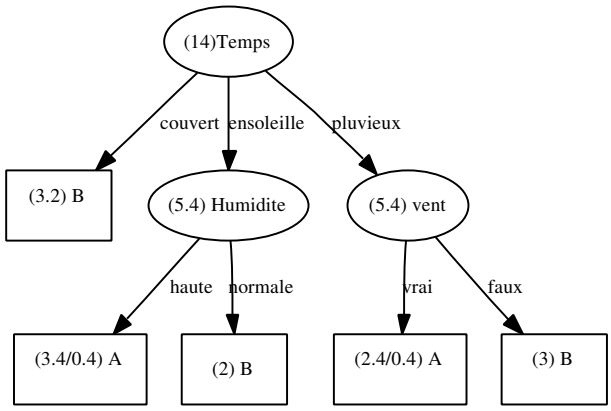

FIG. 3.1 – L'arbre de décision final pour la base *météo* ayant une valeur manquante

id	Temps	Température	Humidité	Vent	Decision
O	Ensoleillé	moyenne	?	Faux	?

TAB. 3.8 – Une instance à classer de la base *Météo*

Le sous-ensemble dans le tableau 3.7 est partitionné selon les valeurs de l'attribut *Vent*. Dans ce cas on a deux sous-ensembles : l'un correspond à la valeur *vrai* avec 2 objets de la classe A et $\frac{5}{13}$ de la classe B; l'autre correspond à la valeur *faux* avec 3 objets de la classe B et 0 de la classe A. L'arbre de décision correspondant à cette méthode est donné dans la figure 3.1.

Dans la figure 3.1, quand *Temps* est *ensoleillé* et *Humidité* est *haute*, nous avons une feuille avec (3.4/0.4)A ; cela signifie que nous avons 3.4 objets qui sont arrivés à cette feuille dont 3 objets qui appartiennent à la classe A et 0.4 objets qui appartiennent à la classe B.

Nous allons montrer maintenant comment classer un objet ayant des valeurs manquantes avec cette méthode : Supposons que nous voulons classer l'objet donné dans le tableau 3.8. Nous commençons par la racine de l'arbre (figure 3.1), qui correspond à l'attribut *Temps* ayant *ensoleillé* comme valeur dans l'objet à classer. Nous descendons dans l'arbre en suivant cette branche, et nous arrivons à l'attribut *Humidité*, qui est inconnu dans l'objet à classer.
Si l'*Humidité* est *haute*, l'objet appartient à la classe A avec la probabilité $\frac{3}{3.4}$ (88%) et à la classe B avec la probabilité $\frac{0.4}{3.4}$(12%) ; si l'*Humidité* est *normale*, l'objet appartient à la classe B avec la probabilité 1 (100%).
Cette méthode calcule la probabilité d'associer l'objet à la classe A et la probabilité d'associer l'objet à la classe B comme suit :

$$\begin{aligned} P(A) &= P(A|haute)P(haute) + P(A|normale)P(normale) \\ &= (\frac{3}{3.4}) \times (\frac{3.4}{5.4}) + 0 \times (\frac{2}{5.4}) = 0.88 \times 0.63 = 0.56 \\ P(B) &= P(B|haute)P(haute) + P(B|normale)P(normale) \\ &= (\frac{0.4}{3.4}) \times (\frac{3.4}{5.4}) + 1 \times (\frac{2}{5.4}) = 0.44. \end{aligned}$$

Nous trouvons que cette méthode est très intéressante parce qu'elle classe l'objet de manière probabiliste au lieu de lui associer simplement la valeur la plus probable [Saar-Tsechansky et Provost, 2007]. Cependant, elle ne prend pas en compte les dépendances éventuelles entre les attributs. La probabilité d'un attribut manquant est calculée en sachant seulement les valeurs des attributs déjà rencontrés dans son chemin. De plus, dans le cas d'un arbre de décision classique construit selon C4.5, on connaît pour chaque feuille la valeur de classe la plus probable, le nombre d'objets arrivant à cette feuille ainsi que le nombre d'objets mal classés, mais on ne connaît pas la distribution des valeurs de classe considérées comme mal classées sur cette feuille. Par ailleurs, conserver toutes les valeurs de classe sur chaque feuille est particulièrement important lorsque la classe possède un nombre de valeurs supérieur à deux.

3.3.5 L'approche proposée par CART

La méthode CART [Breiman *et al.*, 1984] utilise le *surrogate splits* pour traiter les valeurs manquantes. L'idée consiste à utiliser un autre attribut, appelé l'attribut de substitution, pour

décider quelle branche (gauche ou droite) choisir pour continuer le classement.
L'idée du *surrogate split* est la suivante. Lorsqu'une donnée parvient jusqu'à un nœud et que la valeur de l'attribut testé par ce nœud est manquante, CART utilise un autre attribut pour déterminer dans quelle branche on doit envoyer la donnée. C'est-à-dire qu'on définit une mesure de similarité entre deux splits s et s' sur un même nœud t (ce nœud regroupe un sous-ensemble des attributs) ; Si le meilleur split sur ce nœud est s en utilisant l'attribut x_m, on va chercher le split s', sur un autre attribut que x_m, qui est le plus similaire à s. Le split s' fait une partition de l'ensemble aussi proche que possible de celle réalisée par s ; on appelle s' le meilleur surrogate de s. De la même façon, on définit le deuxième meilleur surrogate, le troisième meilleur surrogate, et ainsi de suite.

Si un objet contient une valeur manquante pour x_m, pour décider dans quelle branche (gauche ou droite) il sera envoyé, on utilise le meilleur *surrogate split* s' ; si l'attribut du split s' contient une valeur manquante dans cet objet, on utilise le deuxième meilleur surrogate, et ainsi de suite.

3.3.6 Comment définir un surrogate split (attribut de substitution)

Pour un nœud donné t, supposons que s^* est le meilleur split sur ce nœud et il partitionne l'ensemble parvenu à t en deux parties t_D et t_G. Pour un attribut x_m, supposons que S_m est l'ensemble des splits sur x_m et $\bar{S}_m$ est l'ensemble des splits complémentaires. Pour un split $s_m \in S_m \cup \bar{S_m}$ qui partitionne le nœud t en $t_D^{'}, t_G^{'}$, $N_j(GG)$ est le nombre d'instances sur t qui seront envoyées à gauche par les deux splits s_m et s^*. Donc, ces instances tombent dans $t_G \cap t_G^{'}$. La probabilité qu'une instance tombe dans $t_G \cap t_G^{'}$ est estimée comme suit :

$$p(t_G \cap t_G^{'}) = \sum_j \pi(j) N_j(GG)/N_j$$

où $\pi(j)$ est la probabilité d'avoir la classe j, j =1,...,J. $\pi(j) = \frac{N_j}{N}$, N_j est le nombre d'instances de la classe j sur le nœud t. Donc, la probabilité $p_{GG}(s_m, s^*)$ que s_m et s^* ensemble envoient une instance dans la partie gauche t_G est :

$$p_{GG}(s_m, s^*) = p(t_G \cap t_G^{'})/p(t)$$

de la même façon, on peut définir $p_{DD}(s_m, s^*)$.
La probabilité que s_m prévoie exactement s^* est :

$$p(s^*, s_m) = p_{GG}(s_m, s^*) + p_{DD}(s_m, s^*)$$

Un split $\tilde{s_m} \in S_m \cup \bar{S_m}$ est appelé *surrogate split* sur l'attribut x_m pour s^* si :

$$p(s^*, \tilde{s_m}) = max_{s_m} p(s^*, s_m)$$

Supposons que s^* envoie les instances dans la partie gauche de l'arbre avec une probabilité p_G et dans la partie droite avec la probabilité p_D. Si une nouvelle instance arrive, elle sera envoyée dans la partie t_G si $p_G = max(p_G, p_D)$. Dans le cas contraire, elle sera envoyée dans t_D.

De plus, CART cherche à trouver le degré de la corrélation entre les deux splits s^* et $\tilde{s_m}$; pour cela, il définit une mesure d'association $\lambda(s^*, \tilde{s_m})$ comme suit :

$$\lambda(s^*, \tilde{s_m}) = \frac{min(p_G, p_D) - (1 - p(s^*, s_m))}{min(p_G, p_D)} \tag{3.2}$$

Cette mesure est la réduction d'erreur obtenue en utilisant $s_{\tilde{m}}$ pour prédire s^*. Si $\lambda(s^*, s_{\tilde{m}}) \leq 0$, $s_{\tilde{m}}$ sera supprimé comme *surrogate split* de s^* parce qu'il n'est pas assez pertinent pour prédire s^*.

Pendant la construction d'un arbre de décision, le meilleur split s^*_m de l'attribut x_m est calculé à partir des objets ayant des valeurs connues pour x_m, i.e., cette méthode ignore les instances ayant des valeurs manquantes lors du calcul de réduction de l'impureté. Si le meilleur split trouvé pour un nœud est s^* et un objet dans la base d'apprentissage possède une valeur manquante pour ce split, on cherche dans cet objet un attribut, par exemple x_m parmi ceux qui n'ont pas de valeurs manquantes, tel que $s_{\tilde{m}}$ possède la plus haute corrélation $\lambda(s^*, s_{\tilde{m}})$ avec s^*. On utilise $s_{\tilde{m}}$ pour attribuer l'objet à une branche.

En général, lorsqu'un attribut manquant est envoyé dans un sous-arbre d'un noeud en suivant une branche déterminée par l'attribut de substitution, cela revient à compléter cet attribut manquant par la modalité[32] qui étiquette la branche choisie.

3.3.7 La génération de chemin dynamique

Cette méthode est proposée par [White, 1987] et mentionnée dans [Liu *et al.*, 1997]. Son idée se base sur la génération de chemin (la règle) nécessaire pour classer le cas courant, au lieu de générer l'arbre de décision complètement à l'avance. Si une valeur manquante est présentée dans un nouvel objet, l'attribut avec cette valeur n'est pas utilisé pour classer ce cas. Le processus de construction d'une règle de classement pour classer un nouvel objet O est le suivant : à chaque étape, on cherche l'attribut le plus pertinent pour choisir la bonne branche. Si la valeur de l'attribut choisi est inconnue dans l'objet O, alors cet attribut n'est pas utilisé, et l'algorithme essaie avec le deuxième attribut le plus pertinent qui permet de continuer le classement. Le chemin généré (la règle de classement) utilisé est dynamique. Pour classer chaque objet, une règle de classement est construite en fonction des attributs disponibles dans cet objet, sans prendre en compte l'attribut ayant des valeurs inconnues. Par exemple, si on prend la base dans le tableau 3.4, l'objet 12 possède une valeur inconnue pour l'attribut *Temps*, et la règle de classement construite pour classer cet objet n'utilise pas cet attribut, même s'il est le plus informatif sur cet ensemble d'apprentissage ; elle cherche le deuxième attribut le plus pertinent. Seuls les attributs ayant des valeurs connues dans l'objet sont utilisés pour construire la règle. Une règle est utilisée pour classer un seul objet. Cette méthode est utilisée seulement dans le classement et le coût de calcul est très élevé. L'approche proposée avec cette méthode pour traiter les valeurs manquantes dans la base de test est également désignée sous le nom de *Lazy Decision Tree* [Friedman *et al.*, 1996].

3.3.8 Lazy Decision Tree

Les algorithmes de construction d'un arbre de décision comme C4.5 et CART construisent le meilleur arbre de décision pendant la phase d'apprentissage ; cet arbre est utilisé ensuite pour classer les nouveaux objets. Par ailleurs, la méthode d'*arbre de décision Lazy* [Friedman *et al.*, 1996] consiste à sélectionner le meilleur arbre de décision pour chaque instance à tester. En pratique, seul le chemin de classement est réellement pris en compte [Crémilleux et Robert, 2000].

[32]Les arbres produits par CART sont des arbres binaires, où tous les tests étiquetant les noeuds de décision sont binaires. Le nombre de tests à explorer va dépendre de la nature des attributs. *À* un attribut binaire correspond un test binaire. *À* un attribut discret ou qualitatif ayant n modalités, on peut associer autant de tests qu'il y a de partitions en deux classes, soit $2^{n-1} - 1$ tests binaires possibles. Enfin, dans le cas d'attributs continus, il y a une infinité de tests envisageables. Dans ce cas, on découpe l'ensemble des valeurs possibles en segments.

L'algorithme *Lazy Decision Tree* fonctionne comme suit :
En entrée, nous avons un ensemble d'apprentissage T et une instance test I. Si toutes les instances appartiennent à la même classe l, on associe cette classe à l'instance I. Si les attributs dans toutes les instances possèdent les mêmes valeurs, on associe à l'instance I la classe la plus probable dans la base. Autrement, on sélectionne un attribut test X; si sa valeur dans l'objet à classer est x, on associe les instances de l'ensemble d'apprentissage pour lesquelles l'attribut X prend la valeur x dans l'ensemble T et on répète l'algorithme. Le choix de l'attribut test est basé sur l'information mutuelle.

Cette méthode ne prend pas en compte les attributs ayant des valeurs manquantes dans l'instance à classer. De plus, les instances dans l'ensemble d'apprentissage sont exclues seulement quand leurs valeurs sont inconnues pour un attribut dans un chemin.

Nous nous sommes intéressés à une méthode particulière appelée *Arbres d'Attributs Ordonnés* proposée par [Lobo et Numao, 1999], [Lobo et Numao, 2000], qui construit un arbre de décision pour chaque attribut dans la base d'apprentissage selon un ordre de construction croissant en fonction de l'Information Mutuelle relativement à la classe. Cette méthode est utilisée dans la phase d'apprentissage ainsi que dans la phase de classement. Elle est expliquée en détail dans la section suivante.

3.4 La méthode des arbres d'Attributs Ordonnés (AAO)

Les Arbres d'Attributs Ordonnés (AAO) sont une méthode d'apprentissage supervisé proposée par Lobo et Numao pour traiter le problème des valeurs manquantes, à la fois dans les phases de construction et de classement [Lobo et Numao, 1999], [Lobo et Numao, 2000]. L'idée générale de cette méthode est de construire un arbre de décision, appelé *arbre d'attribut*, pour chaque attribut dans la base en utilisant un sous-ensemble d'apprentissage contenant les instances ayant des valeurs connues pour cet attribut. Le fait de construire un arbre d'attribut pour un attribut et remplir ses valeurs manquantes dans la base d'apprentissage permet l'utilisation de cette base après ce remplissage pour construire un autre arbre de décision pour un autre attribut et remplir également ses valeurs manquantes. Donc, l'ordre de construction des arbres de décision pour les attributs et la détermination de leurs valeurs manquantes devient important. L'ordre proposé par Lobo et Numao [Lobo et Numao, 1999] est basé sur l'*Information Mutuelle* (IM)[Shannon, 1948][33]. Cette méthode commence par traiter l'attribut le moins dépendant de la classe parce que les attributs ayant des informations mutuelles maximales relativement à la classe ont plus de chances de participer à la construction de l'arbre de décision final. En revanche, les attributs ayant des informations mutuelles minimales relativement à la classe ont moins de chances de participer à la construction de l'arbre de décision final [Lobo et Numao, 1999].

[33]On rappelle que l'Information Mutuelle mesure la force de la relation entre deux attributs ou entre un attribut et la classe. L'IM entre deux attributs catégoriels X et Y est définie comme suit :

$$IM(X,Y) = -\sum_{x \in D_x} P(x) log_2 P(x) + \sum_{y \in D_y} P(y) \sum_{x \in D_x} P(x|y) log_2 P(x|y) \tag{3.3}$$

D_x et D_y sont les domaines des attributs catégoriels X et Y. *P(X)* et *P(Y)* sont les probabilités de $x \in D_x$ et $y \in D_y$, respectivement. *P(x|y)* est la probabilité conditionnelle que X prenne la valeur x sachant que Y est connu et prend la valeur y.

3.4.1 L'algorithme de construction des Arbres d'Attributs Ordonnés

Pour un attribut donné, son arbre d'attribut est un arbre de décision dont les feuilles représentent les valeurs de cet attribut. Ces arbres sont construits selon un ordre de construction croissant en fonction de l'IM entre chaque attribut et la classe. Cette méthode commence par calculer l'IM entre chaque attribut dans la base d'apprentissage et la classe. Ensuite, les attributs sont ordonnés par ordre croissant d'IM. Le premier arbre d'attribut construit est celui qui correspond à l'attribut ayant l'IM minimale. Il est représenté par un seul noeud-feuille avec sa valeur la plus probable dans la base d'apprentissage.
Pour les autres attributs, on fournit, à partir de l'ensemble d'apprentissage initial, le sous-ensemble d'apprentissage qui contient les instances ayant des valeurs connues pour cet attribut. Ces instances sont décrites seulement par les attributs qui ont déjà été traités (c'est-à-dire les attributs pour lesquels on a déjà construit les arbres d'attributs et déterminé leurs valeurs manquantes dans la base d'apprentissage). Pour construire l'arbre d'attribut pour un attribut A_i, les autres attributs A_j qui vérifient la condition : $IM(A_i, C) < IM(A_j, C)$ sont enlevés. L'algorithme utilisé pour la construction est un algorithme standard de construction d'un arbre de décision. Pendant le calcul de l'IM pour un attribut, les instances ayant des valeurs inconnues pour cet attribut sont ignorées [Lobo et Numao, 2001].

L'arbre d'attribut est utilisé pour déterminer la valeur de l'attribut pour des instances où elle est inconnue. Il est utilisé dans deux cas distincts : 1) lors de la construction de l'arbre de décision, pour déterminer la valeur de l'attribut pour les instances de la base d'apprentissage où cet attribut est inconnu ; 2) lors du classement d'instances incomplètes, pour déterminer la valeur de l'attribut lorsque celle-ci est manquante.

Lors d'un classement, les valeurs des attributs inconnus de l'objet sont calculées successivement, par ordre d'IM croissante, en utilisant leurs arbres d'attributs.

Nous présentons la méthode *AAO* en utilisant l'exemple pris par [Quinlan, 1993]. La base est donnée dans le tableau 2.2. L'information mutuelle entre chaque attribut et la classe est :
IM(Decision, Temps) = 0.246
IM(Decision, Température) =0.030
M(Decision, Humidité) = 0.154
M(Decision, Vent) = 0.049
Les attributs sont ordonnés par ordre croissant en fonction de l'IM : *Température*, *Vent*, *Humidité*, *Temps*. Les arbres sont construits en utilisant l'algorithme ID3 (Quinlan 1986) et le logiciel Weka[34]. Le nombre de cas sur chaque nœud est indiqué entre parenthèses. Dans cet exemple, on suppose qu'il n'y a pas de valeurs manquantes dans la base d'apprentissage initiale ; les arbres sont construits à partir d'une base d'apprentissage complète et ils sont utilisés seulement pendant le classement d'un objet ayant des valeurs manquantes. L'arbre de décision final est donné dans la figure 3.3.

Le premier arbre construit selon de la méthode *AAO* est donc l'arbre de *Température* ; la base d'apprentissage de l'attribut *Température* est donnée dans la figure 3.9. Son arbre est composé d'un seul noeud, ayant pour valeur *moyenne*, qui est la valeur la plus probable (figure 3.2). Selon l'ordre de construction imposé, l'arbre de *Vent* est construit en utilisant seulement l'attribut *Température*. Les arbres pour les attributs *Humidité* et *Temps* sont ensuite construits dans cet ordre. Nous rappelons que le fait d'associer la valeur la plus probable à une feuille élimine les autres valeurs possibles. Par exemple, cette méthode remplace l'attribut *Température* par la

[34] www.cs.waikato.ac.nz/ml/weka/index.html.

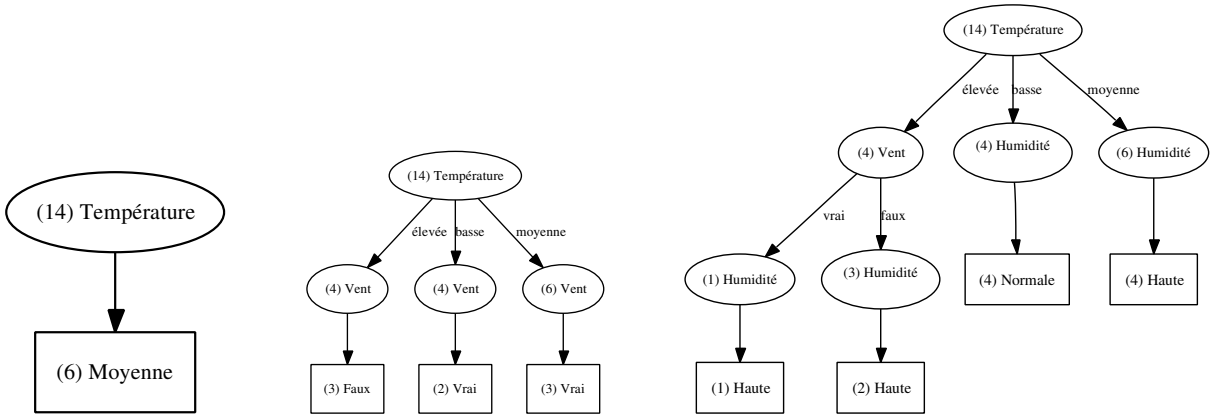

FIG. 3.2 – Les Arbres d'Attributs Ordonnés pour Température, Vent et Humidité

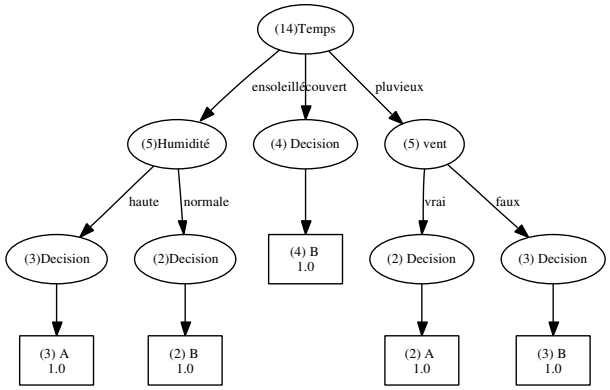

FIG. 3.3 – L'arbre de décision final pour la base *météo*

id	Température
1	Elevée
2	Elevée
3	Elevée
4	Moyenne
5	Basse
6	Basse
7	Basse
8	Moyenne
9	Basse
10	Moyenne
11	Moyenne
12	Moyenne
13	Elevée
14	Moyenne

TAB. 3.9 – La base pour l'attribut *Température*

id	Température	Vent
1	Elevée	Faux
2	Elevée	Vrai
3	Elevée	Faux
4	Moyenne	Faux
5	Basse	Faux
6	Basse	Vrai
7	Basse	Vrai
8	Moyenne	Faux
9	Basse	Faux
10	Moyenne	Faux
11	Moyenne	Vrai
12	Moyenne	Vrai
13	Elevée	Faux
14	Moyenne	Vrai

TAB. 3.10 – La base pour l'attribut *Vent*

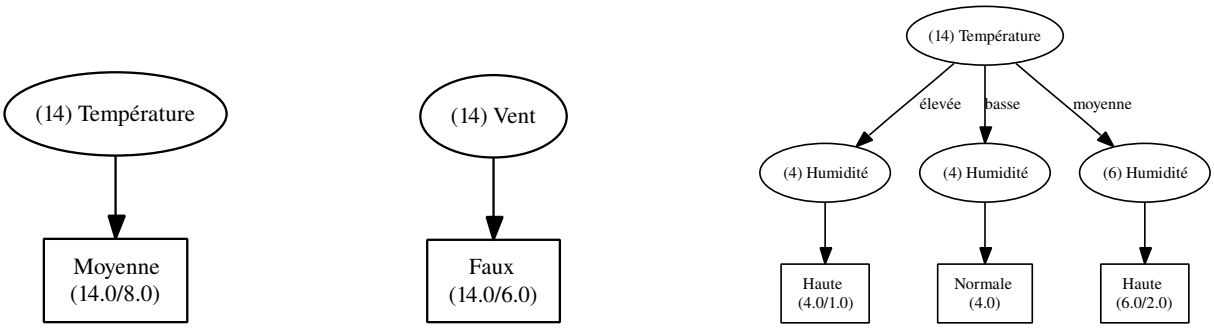

FIG. 3.4 – Les Arbres d'Attributs Ordonnés pour Température, Vent et Humidité construits selon C4.5

valeur *moyenne* dans tout objet dont l'attribut *Température* est inconnu. D'autre part, pour l'arbre de *Vent* et pour la valeur *basse* de l'attribut *Température*, la base d'apprentissage dispose de deux cas où *Vent* est *faux* et deux cas où *Vent* est *vrai*. Dans cette situation, ID3 a choisi arbitrairement la valeur *vrai*. Nous avons également construit les arbres d'attributs de la base *météo* en utilisant C4.5 (figure 3.4). Nous remarquons que l'arbre d'attribut de *Vent* construit selon C4.5 est constitué d'un seul noeud-feuille avec sa valeur la plus probable, qui est *faux*. Dans ce cas, on ignore qu'il y a de dépendance entre *Vent* et *Temps* et on remplace l'attribut *Vent* par *faux* dans chaque objet où *Vent* est inconnu.

L'ordre imposé par cette méthode ne garantit pas que l'arbre d'un attribut soit construit à partir des attributs dont il dépend. L'étude qui a été faite par Lobo et Numao [Lobo et Numao, 2001] a montré que les relations entre attributs d'une base d'apprentissage doivent vérifier certaines conditions pour que la méthode *AAO* soit applicable. Lobo et Numao ont analysé l'information mutuelle pour trouver ces conditions. Dans la suite de ce chapitre, nous allons expliquer l'information mutuelle normalisée utilisée par Lobo et Numao ainsi que leurs expérimentations. Ensuite, nous décrivons leur analyse concernant l'information mutuelle.

3.4.2 L'information Mutuelle Normalisée

En général, plus un attribut a de valeurs, plus son information mutuelle, relativement à la classe, a tendance à être élevée. L'objectif de la normalisation de l'Information Mutuelle est de la rendre insensible au nombre de valeurs de l'attribut testé. [Lobo et Numao, 2001] ont normalisé l'Information Mutuelle en se basant sur ses propriétés :
L'Information Mutuelle entre deux attributs X et Y est symétrique, donc :

$$\begin{aligned} IM(X,Y) &= IM(Y,X) \\ IM(X,Y) &= H(X) - H(X|Y) \leq H(X) \\ IM(Y,X) &= H(Y) - H(Y|X) \leq H(Y) \end{aligned}$$

D'où :

$$IM(X,Y) \leq \frac{H(X) + H(Y)}{2}$$

En considérant que $log|D_x|$ et $log|D_y|$ sont les bornes supérieures de H(X) et H(Y) respectivement, la borne supérieure de l'Information Mutuelle devient :

$$IM(X,Y) \leq \frac{log|D_x| + log|D_y|}{2}$$

qui s'écrit donc :

$$\frac{2IM(X,Y)}{log|D_x| + log|D_y|} \leq 1$$

D'où l'Information Mutuelle normalisée :

$$IM_N(X,Y) \equiv \frac{2IM(X,Y)}{log|D_x| + log|D_y|} \tag{3.4}$$

Dans notre expérimentation, nous avons utilisé l'Information Mutuelle Normalisée proposé ci-dessus [Lobo et Numao, 2001].

3.4.3 Comparaison des performances

Lobo et Numao ont testé leur méthode sur 24 bases d'apprentissage [Lobo et Numao, 2001] et ont comparé sa performance avec 3 autres méthodes : la méthode de majorité [Kononenko *et al.*, 1984], la méthode probabiliste [Quinlan, 1993] et la méthode des arbres d'attributs non ordonnés [Quinlan, 1989, Quinlan, 1986]. Ils ont également introduit des valeurs manquantes artificielles dont les taux sont : 0, 10, 20, 30, 40, 50, 60 et 70 % dans chaque base utilisée.
Ensuite, pour chaque base incomplète, quatre bases d'apprentissage complètes (sans valeurs manquantes) sont générées. Chacune est générée en traitant ses valeurs manquantes en utilisant une de quatre méthodes mentionnées ci-dessus. Dans ce cas, on obtient, pour chaque base incomplète, quatre bases complètes. Ensuite, la performance de chaque base complète est évaluée en utilisant l'algorithme C4.5 [Quinlan, 1993]. La différence de performance entre la méthode des *Arbres d'Attributs Ordonnées* et les autres méthodes est alors calculée.

Le fait d'avoir 8 taux de valeurs manquantes pour chaque base et 3 autres méthodes conduit à avoir 24 tests de comparaison de performances entre la méthode des *Arbres d'Attributs Ordonnés* et les 3 autres méthodes [Lobo et Numao, 2001]. Par conséquent, on compare la performance

d'*AAO* avec les performances de trois d'autres méthodes. Pour chaque base, on a 24 tests de comparaison. Au total, on a donc 576 tests de comparaison pour toutes les bases.

Pour déterminer si la différence est significative ou non, Lobo et Numao ont utilisé la méthode de la *10-validation croisée* et *paired t-test*[35], avec un niveau de confiance de 95%. Lobo et Numao ont distingué trois catégories de performances :

- *wins* (gagnante) si la différence de performance est favorable pour la méthode *AAO*.
- *loses* (perdante) si la différence de performance est défavorable pour la méthode *AAO*.
- *unset* (non définie) si la différence de performance n'est pas significatives (favorable ou défavorable).

Pour chaque base, les résultats des tests sont comptés séparément. Donc, selon les tests effectués par Lobo et Numao [Lobo et Numao, 2001] : la méthode *AAO* est *gagnante* pour 22% des tests, *perdante* pour 4.7% des tests et *non définie* pour 73.1% de tests. Ils ont également trouvé que quand le test de signification est favorable, il y a une réduction du taux d'erreurs considérable. Ces résultats montrent que la méthode des *Arbres d'Attributs Ordonnés* n'est pas suffisamment générale pour être applicable sur toutes les bases d'apprentissage. Pour cela, Lobo et Numao ont étudié les raisons qui sont derrière le succès ou l'échec de cette méthode lors de son utilisation pour déterminer les valeurs manquantes.

3.4.4 Identification des domaines

Lobo et Numao ont analysé l'Information Mutuelle pour étudier les relations entre les attributs et entre les attributs et la classe pour chaque base d'apprentissage. Pour cela ils ont défini l'Information Mutuelle Marginale comme suit :

$$I\bar{M}_N(F,Y) \equiv \sum_{X \in F-\{Y\}} \frac{IM_N(X,Y)}{||F-\{Y\}||} \tag{3.5}$$

L'information Mutuelle Marginale est l'Information Mutuelle moyenne entre un attribut Y et le reste des attributs dans un ensemble F. C'est une mesure de certitude de l'attribut Y quand l'information contenue dans le reste de l'ensemble d'attributs est connue.
Ils ont également défini l'ensemble de l'Information Mutuelle sur un ensemble d'attributs F :

$$I\bar{M}_N(F) \equiv \sum_{Z \in F} \frac{I\bar{M}_N(F,Z)}{||F||} \tag{3.6}$$

C'est une mesure moyenne de certitude pour chaque attribut dans l'ensemble F. Pour une base d'apprentissage décrite par un ensemble d'attributs $A_1, .., A_n$ et une classe C, il est possible de caractériser les données de cette base par tracer les relations de l'Information Mutuelle entre les attributs et la classe en utilisant $IM_N(A_i, C)$ et $I\bar{M}_N(A, A_i)$.

Lobo et Numao ont également défini :

- Le Ratio de l'Information Mutuelle $IM_r(X,Y)$, donné dans l'équation 3.7, qui mesure le pourcentage entre l'Information Mutuelle moyenne entre un attribut X et le reste des attributs $\{F-\{Y\}\}$ et l'Information Mutuelle entre cet attribut et la classe Y.

$$IM_r(X,Y) = \frac{I\bar{M}_N(F-\{Y\},X)}{IM_N(X,Y)} \tag{3.7}$$

[35]Nous trouvons dans l'annexe A, la méthode de la Validation croisée ainsi que *paired t-test*.

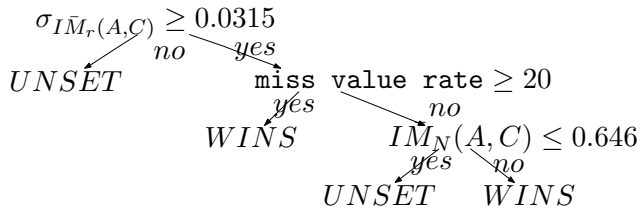

FIG. 3.5 – L'arbre de décision construit en utilisant les indicateurs de l'IM, avec leurs écarts-types

– Le Ratio de l'Information Mutuelle Marginale $I\bar{M}_r(F,Y)$, donné dans l'équation 3.8, qui est la moyenne du Ratio d'IM.

$$I\bar{M}_r(F,Y) = \sum_{X \in F-\{Y\}} \frac{IM_r(X,Y)}{||F-\{Y\}||} \tag{3.8}$$

– Les écarts types de tous les indicateurs précédents $\sigma_{I\bar{M}_N(A,C)}$, $\sigma_{I\bar{M}_N(A)}$, $\sigma_{I\bar{M}_r(A,C)}$.
Nous présentons dans l'équation 3.10 seulement l'écart type [36] du Ratio de l'Information Mutuelle Marginale :

$$\sigma_{I\bar{M}_r(A,C)} = \sqrt{\sum_{i=1}^{||A||} \frac{\left(IM_r(A_i,C) - I\bar{M}_r(A,C)\right)^2}{||A||}} \tag{3.10}$$

Ainsi, pour chaque base de données les indicateurs : $I\bar{M}_N(A,C)$, $I\bar{M}_N(A)$, $I\bar{M}_r(A,C)$ et leurs écarts types $\sigma_{I\bar{M}_N(A,C)}$, $\sigma_{I\bar{M}_N(A)}$, $\sigma_{I\bar{M}_r(A,C)}$ sont calculés.

Lobo et Numao créent alors une nouvelle base de données. Chaque instance dans une telle base concerne une base d'apprentissage [37] testée et un taux de valeurs manquantes [38]. Elle est décrite par le taux de valeurs manquantes et les indicateurs de l'Information Mutuelle avec leurs écarts types. La classe C dans cette base prend trois valeurs : *wins* si la méthode AAO a eu au moins une différence significative favorable et aucune différence défavorable par rapport aux trois d'autres méthodes ; *loses* si la méthode AAO a eu au moins une différence significative défavorable et aucune différence favorable par rapport aux trois d'autres méthodes ; et *unset* dans les autres cas. Par conséquent, une nouvelle base d'apprentissage est créée, qui contient 192 instances parce que l'on a 24 bases d'apprentissage et 8 taux de valeurs manquantes.
Lobo et Numao ont utilisé C4.5 pour construire l'arbre de décision correspondant à cette base. L'arbre obtenu (figure 3.5) contient l'écart-type du Ratio de l'Information Mutuelle Marginale $\sigma_{\bar{M}_r(A,C)}$ comme racine ; le taux de valeurs manquantes et l'Information Mutuelle Marginale relativement à la classe comme deux noeuds internes [Lobo et Numao, 2001].

En conclusion, pour que cette méthode soit applicable sur une base d'apprentissage, il faut que l'écart-type du Ratio de l'Information Mutuelle Marginale $\sigma_{\bar{M}_r(A,C)}$ soit supérieur ou égal à

[36] L'écart type est la mesure de dispersion autour de la moyenne d'un ensemble de données :

$$\sigma_X = \sqrt{\sum_{i=1}^{N} \frac{(x_i - \bar{x})^2}{N}} \tag{3.9}$$

[37] Nous avons 24 bases d'apprentissage.
[38] Nous avons 8 taux de valeurs manquantes.

0.0315 et que le taux de valeurs manquantes soit supérieur à 20%. Ou bien il faut que $\sigma_{I\bar{M}_r(A,C)}$ soit supérieur ou égal à 0.0315 et que l'Information Mutuelle marginale $I\bar{M}_r(A,C)$ soit supérieure à 0.646.

Pour que $\sigma_{I\bar{M}_r(A,C)}$ soit maximal, il faut que le terme $(IM_r(A_i,C) - I\bar{M}_r(A,C))$ soit grand pour plusieurs attributs, c'est-à-dire que la valeur du Ratio de l'Information Mutuelle $IM_r(A_i,C)$ pour chaque attribut soit très écartée de la moyenne $I\bar{M}_r(A,C)$. Cela peut arriver quand chaque attribut dans la base satisfait au moins l'une des deux conditions :

- *Les attributs ayant des faibles relations avec la classe ont des fortes relations avec le reste des attributs. Autrement dit,* $IM_r(A_i,C)$ *est élevé.*
- *Les attributs ayant des fortes relations avec la classe ont des faibles relations avec le reste des attributs. Autrement dit,* $IM_r(A_i,C)$ *est petit.*

Pour mieux comprendre les deux conditions précédentes, nous prenons un exemple de la base *météo* : Nous calculons le Ratio de l'Information Mutuelle entre l'attribut *Temps* et la classe :

$$IM_r(Temps, \texttt{Décision}) = \frac{I\bar{M}_N(F - \{\texttt{Décision}\}, Temps)}{IM_N(Temps, \texttt{Décision})} \tag{3.11}$$

Le terme $I\bar{M}_N(F - \{\texttt{Décision}\}, Temps)$ est calculé à partir de l'équation 3.5 :

$$IM_r(Temps, \texttt{Décision}) = \frac{\frac{IM_N(\texttt{Température}, Temps) + IM_N(\texttt{Humidité}, Temps) + IM_N(Vent, Temps)}{3}}{IM_N(Temps, \texttt{Décision})} \tag{3.12}$$

À partir de l'équation 3.12 et pour que $IM_r(Temps, \texttt{Décision})$ soit maximal, il faut que la moyenne des Informations Mutuelles entre *Temps* et le reste des attributs {*Température*, *Humidité* et *Vent*} soit supérieure à l'Information Mutuelle entre *Temps* et la classe *Décision*. Dans ce cas, nous pouvons dire que les relations entre *Temps* et le reste des attributs sont plus fortes que la relation entre *Temps* et la classe.

Si l'IM entre *Temps* et *Décision* est supérieur à la moyenne des IM entre *Temps* et le reste des attributs, nous pouvons dire que la relation entre *Temps* et *Décision* est plus forte que celles entre *Temps* et le reste des attributs. Pour que la méthode *AAO* soit applicable sur la base *météo*, il faut que tous les attributs dans cette base vérifient une des deux conditions précédentes. Autrement, cette méthode n'est pas applicable sur la base.

En général, les attributs dans une base ne vérifient pas forcément les conditions précédentes. Par exemple, nous pouvons trouver une base dont quelques attributs sont fortement dépendants de la classe et dépendants également des autres attributs dans la base, ou bien le contraire.

La méthode des *Arbres d'Attributs Ordonnés* n'est pas applicable sur toutes les bases mais nous avons vu que Lobo et Numao sont arrivés à identifier les domaines où leur méthode peut être appliquée.

3.5 Méthodes statistiques

Nous n'avons pas étudié dans ce mémoire les méthodes statistiques qui traitent les valeurs manquantes pour construire des bases complètes[39]. Nous pouvons néanmoins rappeler que les procédures d'imputation statistiques [Roderick et Donald, 2002, Saar-Tsechansky et Provost, 2007] peuvent être basées sur des modèles :

- *explicites*, qui sont issus en général de la théorie statistique, comme la régression linéaire, le modèle linéaire général, l'imputation moyenne, etc.
- *implicites*, comme les méthodes de type *hot-deck*, qui remplacent la valeur manquante par une valeur observée chez un objet partageant les mêmes valeurs pour les autres attributs, ou les méthodes de type *cold-deck*, qui remplacent la valeur manquante par une valeur prise d'une autre base d'apprentissage du même domaine.

L'imputation multiple est une méthode statistique pour remplir les valeurs manquantes et rendre une base d'apprentissage complète. L'idée de base est de remplacer la valeur manquante par plusieurs valeurs plausibles m. Dans ce cas, nous aurons m bases complètes. Chaque base sera analysée de manière identique en utilisant une même méthode standard. Ensuite, Les résultats seront combinés. Une procédure d'imputation multiple peut être basée sur des modèles explicites ou implicites. Plus le nombre k d'imputations est grand, plus les estimateurs seront précis. Pour analyser et combiner les résultats, il faut d'abord pour chaque analyse calculer et enregistrer les estimations et les erreurs standard ainsi que la variance et l'écart-type. Un test de signification de l'hypothèse nulle est également réalisée.

3.6 Conclusion

Nous avons présenté dans ce chapitre les méthodes qui traitent les valeurs manquantes dans les arbres de décision. Elles remplacent un attribut manquant par une seule valeur, qui peut être la valeur la plus probable [Kononenko *et al.*, 1984] ou la plus semblable [Breiman *et al.*, 1984], etc. Ce type d'approche présente l'inconvénient d'oublier les autres valeurs possibles.

Les méthodes *Lazy decision trees* et *la génération de chemin dynamique* ignorent les instances ayant des valeurs manquantes ; l'arbre est construit en utilisant seulement les attributs connus dans l'objet à classer. Dans des telles méthodes, si le nombre d'attributs manquants est assez grand dans l'instance à classer, on est obligé d'utiliser seulement les attributs connus qui ne seraient pas forcément en forte relation avec la classe, ce qui conduit à un taux de mauvais classement très élevé.

La méthode proposée par CART pour traiter les valeurs manquantes est intéressante parce qu'elle calcule la corrélation entre les tests utilisés sur chaque nœud dans l'arbre. Cependant, le fait de construire un arbre binaire élimine les autres valeurs des attributs, ce qui n'est pas pratique dans les domaines où les attributs ne sont pas continus et où toutes leurs valeurs sont importantes. La méthode proposée par C4.5 est intéressante parce le résultat de classement est probabiliste mais elle ne prend pas en compte la dépendance éventuelle entre les attributs.

Nous avons également détaillé dans ce chapitre la méthode des *Arbres d'Attributs Ordonnées*, qui traite les valeurs manquantes dans les données, à la fois dans les phases de construction et de classement. Nous nous sommes intéressés particulièrement à cette méthode parce qu'elle construit un arbre de décision pour chaque attribut dans la base d'apprentissage. Elle commence

[39] Nous trouvons le logiciel *SOLAS* qui est un logiciel statistique pour traiter les valeurs manquantes. `http://www.statsol.ie/html/solas/solas_home.html`

par l'attribut le moins dépendant de la classe parce que c'est l'attribut ayant le moins de chances de participer à la construction de l'arbre de décision final. Par ailleurs, l'ordre imposé par cette méthode ne garantit pas que l'arbre d'un attribut soit construit à partir des attributs dont il dépend. Nous avons vu que cette méthode n'est pas applicable sur toutes les bases de données ; il faut que les relations entre les attributs d'une base vérifient certaines conditions pour que cette méthode soit applicable.

Nous avons observé que les méthodes que nous venons de présenter, à part la méthode C4.5, remplacent un attribut inconnu par une seule valeur et, pour un objet ayant des valeurs manquantes, donne une seule classe (la plus probable).

Parce que notre objectif est d'aider l'utilisateur à prendre sa décision en présence des données manquantes, notre approche vise à réaliser une détermination probabiliste des valeurs manquantes, en prenant en compte les dépendances entre l'attribut manquant et les autres attributs de l'objet, ce qui permet d'utiliser le maximum de l'information contenue dans l'objet pour le calcul des valeurs manquantes. De plus, nous voulons un résultat de classement sous la forme d'une distribution de probabilités plutôt que simplement la valeur la plus probable, ce qui constitue une information plus précise.

Nous rappelons que avons présenté dans ce chapitre les méthodes qui traitent les attributs inconnus dans le cadre d'un arbre de décision pendant la construction et le classement. Dans la suite, nous nous intéressons exclusivement au second problème, c'est-à-dire *le classement d'objets incomplets* dans un arbre de décision. Nous allons décrire dans le chapitre suivant notre approche, qui associe à un objet ayant des valeurs manquantes un résultat de classement sous forme d'une distribution de probabilités et qui prend en compte les dépendances entre les attributs lors du calcul de probabilité. Nous expliquons également l'algorithme de classement proposé.

Chapitre 4

L'approche proposée

4.1 Motivation

Pour traiter le problème de valeurs manquantes pendant le classement dans un arbre de décision, nous proposons que :

- La valeur manquante d'un attribut soit prédite sous forme d'une distribution de probabilités en utilisant les autres attributs dont il dépend.
- Le résultat de classement d'un objet ayant des valeurs manquantes soit également sous forme d'une distribution de probabilités de classe au lieu de la classe la plus probable.

Nous avons vu dans le chapitre précédent que le fait d'associer à un attribut manquant sa valeur la plus probable éliminait les autres valeurs possibles. Pour rendre compte de ces autres valeurs, nous voulons un résultat sous la forme d'une distribution de probabilités plutôt que simplement la valeur la plus probable, ce qui constitue une information plus précise. Nous utiliserons l'arbre de décision dans sa structure la plus simple en gardant sur chaque feuille la distribution de probabilités de classe.

Lorsque nous avons étudié l'élagage, nous n'avons pas comparé empiriquement les différentes méthodes pour choisir la meilleure, parce que notre approche est basée sur la dépendance entre chaque attribut et la classe ainsi que entre les attributs eux-mêmes. Les méthodes de post-élagage proposées ne prennent pas en compte la dépendance entre les attributs. L'algorithme de construction d'un arbre de décision cherche principalement à trouver l'arbre le plus petit. Ainsi, le post-élagage remplace des nœuds ou des branches jugées inutiles par une seule feuille. Cependant, plusieurs travaux [Domingos et Provost, 2000, Provost et Domingos, 2003, Crémilleux, 2000] ont montré que l'élagage diminue la performance de l'arbre de décision en estimation de probabilité et qu'il peut supprimer des règles de décision pertinentes. D'un autre côté, l'élagage est utile pour éviter l'overfitting et il permet d'éviter les feuilles ayant un trop petit nombre d'objets.

Dans ce cadre, nous allons utiliser une stratégie de pré-élagage qui élimine à l'avance les attributs jugés non pertinents par rapport à la classe. Dans notre stratégie, nous fixons un seuil minimal de l'information mutuelle pour sélectionner l'attribut nœud. Pour construire un arbre de décision, nous calculons l'information mutuelle entre chaque attribut et la classe et nous choisissons ceux ayant d'information mutuelle supérieure au seuil fixé. Ce sont les attributs les plus dépendants de la classe et qui vont participer à la construction de l'arbre. Ensuite, l'arbre est construit jusqu'au bout en utilisant ces attributs.

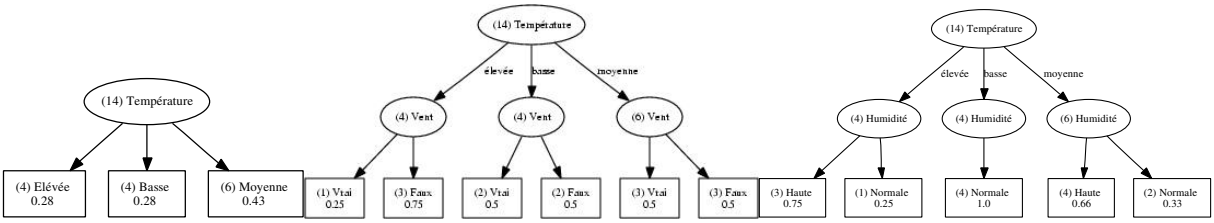

FIG. 4.1 – Les AAOPs pour Température, Vent et Humidité

Dans notre travail, nous nous sommes intéressés aux méthodes qui utilisent les arbres de décision pour trouver l'attribut manquant, et en particulier à la méthode des *Arbres d'Attributs Ordonnés* [Lobo et Numao, 1999], [Lobo et Numao, 2000]. Parce que les arbres de décision sont capables de déterminer la classe d'une instance à partir des valeurs de ses attributs, on peut les utiliser pour déterminer les valeurs d'un attribut inconnu (qui joue alors le rôle de la classe) à partir des attributs dont il dépend. Dans ce cadre, nous déterminons la valeur d'un attribut manquant à partir des attributs dont il dépend, ce qui permet d'utiliser le maximum de l'information contenue dans l'objet pour le calcul des valeurs manquantes.

Nous présentons dans ce chapitre notre approche, qui se compose de deux parties :
1) les *Arbres d'Attributs Ordonnés Probabilistes (AAOP)*, qui étendent la méthode précédente *AAO* en construisant un arbre de décision **probabiliste** pour chaque attribut au lieu d'un arbre de décision classique ; 2) *les Arbres d'Attributs Probabilistes (AAP)*, qui prennent en compte la **dépendance** entre les attributs lors de la construction des arbres de décision probabilistes et non selon l'ordre croissant de l'information mutuelle.

4.2 Arbres d'Attributs Ordonnés Probabilistes (AAOP)

Notre première proposition [Hawarah *et al.*, 2004] est une extension de la méthode des *Arbres d'Attributs Ordonnés* [Lobo et Numao, 1999], [Lobo et Numao, 2000]. Elle consiste à construire pour chaque attribut un arbre d'attribut selon la méthode *AAO*. Cependant, contrairement à Lobo, qui, en suivant la méthodologie classique, associe à chaque feuille la valeur la plus probable, nous conservons dans chaque feuille d'un arbre d'attribut la distribution des fréquences des valeurs de l'attribut courant[40]. De plus, les attributs utilisés pour construire un arbre d'attribut selon *AAOP* sont les attributs déjà traités et **dépendants** de l'attribut courant au lieu d'utiliser tous les attributs déjà traités. La distribution de probabilités sur chaque feuille d'un arbre d'attribut probabiliste permet de déterminer le classement probabiliste des valeurs d'un attribut manquant. En conséquence, elle permet le classement probabiliste d'un objet avec des attributs manquants. On appelle cette proposition *Arbres d'Attributs Ordonnés Probabilistes (AAOP)*. Le résultat du classement permettant de déterminer une valeur manquante est une distribution de probabilités des valeurs de l'attribut. Ainsi, le classement d'un objet incomplet en utilisant les *AAOPs* est une distribution probabiliste de classe au lieu d'une seule valeur de classe. Ces arbres sont utilisés pendant le classement d'un objet avec des valeurs d'attributs manquantes.

Si on classe un objet dans l'arbre de décision de la figure 4.4, qui correspond à la base d'apprentissage complète, où *Temps* est *ensoleillé*, *Vent* est *faux*, *Température* est *élevée* mais

[40]L'attribut courant est l'attribut pour lequel on construit l'arbre d'attribut.

Humidité est inconnue. Les attributs sont ordonnés par ordre croissant selon leurs information mutuelle par rapport à la classe : *Température*, *Vent*, *Humidité*, *Temps*. Les probabilités des valeurs de l'attribut *Humidité* sont calculées à partir de son *arbre d'attribut* donné dans la figure 4.1 (à droite) ; cet arbre est construit sans l'attribut *Vent* car l'information mutuelle entre *Vent* et *Humidité* est nulle dans cette base. Dans notre exemple, la distribution de probabilités des valeurs de l'attribut *Humidité* sont : *normal* avec la probabilité 0.25 et *haute* avec la probabilité 0.75. Contrairement aux *arbres d'attributs ordonnés* de Lobo, les *AAOPs* ont l'avantage de permettre d'aboutir à des résultats probabilistes. Cependant, de notre point de vue, ces *AAOPs* pose le problème du critère du choix des attributs à prendre en compte, qui est leur IM par rapport à la classe. En particulier, seuls les attributs dépendants de l'attribut courant et ayant, par rapport à la classe, une IM inférieure à celle de cet attribut, sont pris en compte. En conséquence, tous les autres attributs qui dépendent de l'attribut courant ne participent pas à la construction de son arbre.

Les *Arbres d'Attributs Probabilistes (AAPs)*, présentés ci-dessous, constituent notre deuxième proposition d'extension des arbres d'attributs de Lobo. Contrairement aux *AAOPs*, les *AAPs* prennent en compte toutes les dépendances entre attributs.

4.3 La deuxième proposition : Les Arbres d'Attributs Probabilistes (AAP)

La méthodologie des arbres d'attributs probabilistes (AAP) [Hawarah *et al.*, 2004], [Hawarah *et al.*, 2005] est une méthodologie qui, pour chaque attribut, construit un arbre d'attribut probabiliste en utilisant les attributs dont il dépend. Afin de déterminer les dépendances entre les attributs, nous calculons l'IM entre chaque couple d'attributs de la base. En effet, l'IM entre deux attributs est la réduction moyenne de l'incertitude sur un attribut sachant l'autre. Ainsi, pour un attribut A_i, les attributs dont il dépend sont calculés par l'expression :

$$Dep(A_i) = \{A_j \mid IM(A_i, A_j) > Seuil^{41}\}$$

En général, les deux arbres *AAOP* et *AAP* de l'attribut le plus dépendant de la classe sont identiques (figure 4.2).

L'*AAP* d'*Humidité* est construit en utilisant les attributs *Temps*, *Température* (figure4.3). Ainsi, les deux arbres *AAOP* et *AAP* de l'attribut le plus dépendant de la classe *Temps* sont identiques (figure 4.2).

En conséquence, dans notre travail nous construisons pour chaque attribut dans la base d'apprentissage deux arbres d'attributs.

4.3.1 Le problème de cycle

Dans la deuxième approche, un problème de *cycle* peut être rencontré lorsque deux attributs mutuellement dépendants sont manquants. Pour le résoudre, nous proposons la solution suivante : on calcule d'abord la probabilité de l'attribut le moins dépendant de la classe à partir de son *Arbre d'Attribut Ordonné Probabiliste (AAOP)* construit selon la première proposition ; puis la distribution de probabilités de l'autre attribut à partir de son *Arbre d'Attribut Probabiliste (AAP)* construit selon la deuxième proposition. C'est à dire qu'en cas de *cycle*, les deux propositions sont utilisées conjointement pour classer le même objet ayant des valeurs manquantes.

[41] Seuil à fixer (*cf.* section 5.2).

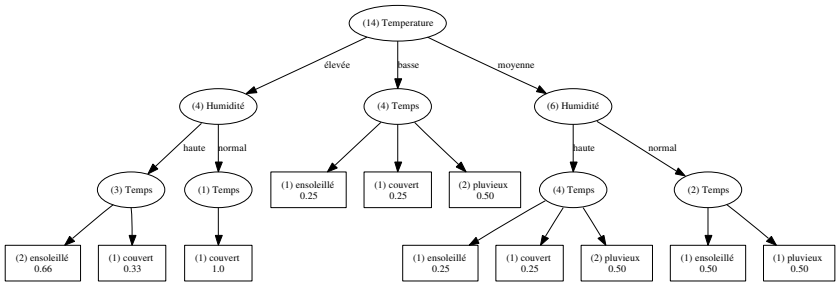

FIG. 4.2 – L'AAP/AAOP de l'attribut Temps

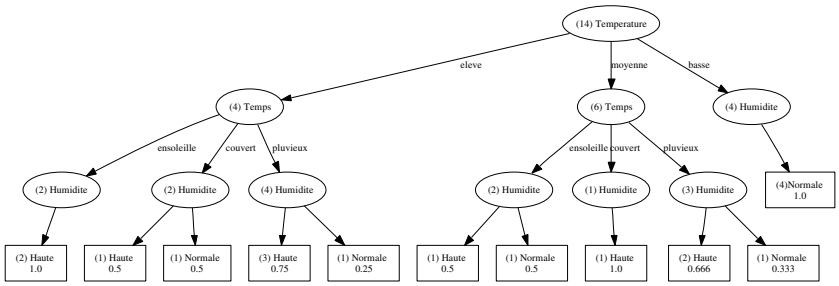

FIG. 4.3 – L'AAP de l'attribut Humidité

Pourquoi cet ordre

Si nous avons deux attributs manquants et dépendants en même temps, par exemple *Temps* et *Humidité* et que *Temps* est plus dépendent de la classe que *Humidité*, à partir de la figure 4.4 nous savons que :

P(Temps = ensoleillé, Humidité= haute) = P(ensoleillé) P(haute|ensoleillé)
Ou bien :

P(Temps = ensoleillé, Humidité= haute) = P(haute) P(ensoleillé|haute)
Les deux cas donnent la même réponse mais nous utilisons toujours le deuxième cas parce que : L'*AAP* de *Temps* est construit en utilisant *Température* et *Humidité* (figure 4.2). L'*AAOP* de *Temps* est également construit en utilisant *Température* et *Humidité*. L'*AAP* d'*Humidité* est construit en utilisant *Temps* et *Température* (figure 4.3). L'*AAOP* de *Humidité* est construit en utilisant *Température* seulement (figure 4.1). On voit que *Temps* et *Humidité* dépendent d'un autre attribut *Température*. Sachant que *Température* est *élevé*, le premier cas nous donne :

P(ensoleillé, haute)= P(élevé) P(ensoleillé|élevé) P(haute|ensoleillé, élevé)
Le deuxième cas nous donne :

P(ensoleillé, haute) = P(élevé) P(haute|élevé) P(ensoleillé|haute, élevé)
Pour qu'on puisse utiliser le premier cas, il faudrait que l'*AAOP* de *Temps* soit construit sans *Humidité*[42] or l'*AAOP* et *AAP* de *Temps* sont construit avec *Humidité*. Dans cette situation, nous utilisons le deuxième cas parce que nous n'avons pas un arbre pour *Temps* sans l'attribut *Humidité*.

Désormais, nous allons appeler notre approche *AAP* pour indiquer les deux familles d'arbres *AAOPs* et *AAPs*.

[42]Pour calculer la probabilité de *Temps* sans *Humidité*.

4.3.2 Choix entre un AAOP ou un AAP

- L'*AAP* de l'attribut le plus dépendant de la classe[43] et l'*AAOP* du même attribut sont identiques.
- Si l'attribut le plus dépendant de la classe et un autre attribut qui est également dépendant de la classe sont manquants et dépendants en même temps, dans ce cas, on appelle pour le premier attribut (racine) son *AAP* et pour le deuxième son *AAOP*. Lorsque, l'*AAP* et l'*AAOP* pour la racine sont identiques, on utilise seulement l'approche *AAOP*.
- Si l'attribut le plus dépendant de la classe et un autre attribut qui dépend de la classe sont manquants et indépendants, on appelle pour le premier attribut (racine) son *AAP* et pour le deuxième son *AAP*.
- Quand un attribut qui dépendant de la classe et qui participe à la construction de l'arbre de décision final est manquant et dépendant de l'attribut racine qui est connu, on appelle son AAP construit en utilisant l'attribut racine, ce qui n'est pas le cas dans l'approche *AAOP*.
- Si nous avons deux attributs manquants et indépendants, mais qui participent à la construction de l'arbre de décision final nous appelons pour chacun son AAP. La différence entre AAP et AAOP dans ce cas est la suivante : AAP pour un attribut est construit en utilisant ses attributs dépendants, et AAOP pour le même attribut est construit en utilisant ses attributs dépendants ayant des IM inférieures à son IM avec la classe. Dans ce cas, on ignore les autres attributs.
- Il peut arriver que les *AAOPs* et *AAPs* sont identiques pour tous les attributs.

4.4 Classement d'une instance avec des attributs manquants

Pour classer un objet ayant des valeurs manquantes dans l'arbre de décision probabiliste final[44], on commence par parcourir l'arbre de décision, en partant de sa racine jusqu'à ce qu'on arrive à une feuille en suivant les branches correspondant aux valeurs des attributs de l'instance à classer. Une fois que l'on rencontre une valeur manquante pour un attribut test (noeud), on explore toutes les branches correspondant aux valeurs de cet attribut. Dans ce cas, on arrive à plusieurs feuilles dans l'arbre, et pas seulement à une seule feuille comme dans le classement classique. Pour cela, il faut calculer les probabilités de classe sur chacune de ces feuilles. Supposons que l'on ait deux valeurs de classe A et D et que, pour un chemin de la racine de l'arbre jusqu'à une feuille F, on passe par les branches B_1, B_2,.., B_n. Alors :
P(classe A sur une feuille F) = P(A| chemin de la racine à F) = $P(A|B_1, B_2, .., B_n)$
P(classe D sur une feuille F) = P(D| chemin de la racine à F) = $P(D|B1, B2, .., Bn)$

$$P(\texttt{A dans tout l'arbre}) = \sum_i P(A|F_i) * P(F_i)$$

$$P(\texttt{D dans tout l'arbre}) = \sum_i P(D|F_i) * P(F_i)$$

où i = 1,..,m (m le nombre de feuilles dans l'arbre)
Le calcul précédent est inspiré de la méthodologie utilisée par [Quinlan, 1986], [Quinlan, 1990], [Quinlan, 1993] lors du classement d'une instance avec des valeurs manquantes dans l'arbre de

[43] L'attribut qui forme la racine de l'arbre de décision final.
[44] L'arbre de décision final est l'arbre de classement qui correspond à toute la base d'apprentissage.

décision.
La probabilité $P(A|F_i)$ est la probabilité de classe A attachée à cette feuille ; la probabilité $P(F_i)$ est la probabilité jointe des attributs dans le chemin de la racine de l'arbre jusqu'à F_i.
Pour simplifier, considérons que le chemin de la racine jusqu'à F_i passe par les branches B_1, B_2 : $P(F_i) = P(B_1, B_2) = P(B_1) * P(B_2|B_1)$ sachant que B_1 est moins dépendant de la classe que B_2.

4.4.1 Calcul de la probabilité jointe $P(B_1, B_2)$ dans notre approche

Pour calculer cette probabilité jointe quand B_1 et B_2 sont manquants en même temps, on distingue les cas suivants :

- B_1 et B_2 sont indépendants :
  $P(B_2|B_1) = P(B_2)$ et $P(B_1, B_2) = P(B_1) * P(B_2)$
  En conséquence, l'AAP de B_1 est construit sans B_2 et l'AAP de B_2 est construit sans B_1. On calcule la probabilité de l'attribut B_1 à partir de son AAP. La probabilité de l'attribut B_2 est également calculée à partir son AAP.
- B_1 et B_2 sont dépendants mais l'$AAOP$ de B_1 est construit sans B_2 parce que B_1 est moins dépendant de la classe que B_2 : $P(B_1|B_2) \neq P(B_1)$. La probabilité de l'attribut B_1 est calculée à partir de son $AAOP$. L'attribut B_1 participe à la construction de l'AAP de l'attribut B_2. Donc, la probabilité $P(B_2|B_1)$ est calculée à partir de l'AAP de B_2.
- B_1 et B_2 sont dépendants, B_1 est le moins dépendant de la classe, un autre attribut manquant G dépend de B_1 et B_2, G est moins dépendant de la classe que B_1 et B_2[45].

$$
\begin{aligned}
P(B_1) &= \sum_i P(B_1|G_i) * P(G_i) \qquad (4.1)\\
P(B_2|B_1) &= \sum_i P(B_2|B_1, G_i) * P(G_i|B_1)\\
P(B_1, B_2) &= \sum_i P(B_1, B_2, G_i)\\
&= \sum_i P(G_i) * P(B_1|G_i) * P(B_2|B_1, G_i)
\end{aligned}
$$

  Les probabilités de G_i et de B_1 sont calculées à partir de leur $AAOPs$. La probabilité de B_2 est calculée à partir de AAP.
- B_1 et B_2 sont indépendants mais ils dépendent d'un autre attribut G. G est moins dépendant de la classe que B_1 et B_2 :

$$
\begin{aligned}
P(B_1, B_2) &= \sum_i P(B_1, B_2, G_i) \qquad (4.2)\\
&= \sum_i P(G_i) * P(B_1|G_i) * P(B_2|G_i)
\end{aligned}
$$

  B_2 est indépendant de B_1 conditionnellement à G. La probabilité de G_i est calculée à partir de son $AAOP$. Les probabilités de B_1 et de B_2 sont calculées à partir de leur $AAPs$.

[45] Nous pouvons également calculer la probabilité jointe donnée dans l'équation 4.1 comme suit : $P(B_1, B_2) = P(B_1) * P(B_2|B_1) = P(B_1) * \sum_i P(B_2|B_1, G_i) * P(G_i|B_1)$
$= \sum_i P(B_2|B_1, G_i) * P(G_i|B_1) * P(B_1) = \sum_i P(B_2|B_1, G_i)P(B_1|G_i)P(G_i)$

Généralement, le classement probabiliste dans un arbre de décision est basé sur : $P(C|A_{ch}(F))$ où A_{ch} désigne l'ensemble des attributs rencontrés dans le chemin ch de la racine de l'arbre jusqu'à la feuille F.
En cas de valeurs manquantes dans l'objet à classer, nous calculons la probabilité que l'objet appartienne à la classe C dans tout l'arbre de décision et non seulement dans un seul chemin.

4.4.2 L'algorithme de classement

Dans ce paragraphe, nous allons décrire l'algorithme de classement que nous avons proposé pour traiter les attributs manquants dans un objet à classer. Le principal algorithme est *Classer-Instance* que nous décrivons de manière générale, sans entrer dans les détails de programmation :

classer-Instance

```
Classer-instance(I: Instance à classer, Arbre: arbre de décision,
                           L: liste d'attributs manquants)
                           Sortie: une distribution de probabilités
Début
Amin, A: attribut,
ProbAmin: réel,
Dist, ResultatClassement: un tableau de valeurs réelles,

 Si la racine d'Arbre est une feuille
     pour i=0 jusqu'à nbClasse
       //nbClasse est le nombre de valeurs de la classe
     Dist[i] = La probabilité de classe i sur cette feuille

     Si le nombre d'attributs manquants != 0
                récupérer l'attribut Amin le moins dépendant de la classe

                Si il existe d'autres attributs manquants dans I
                  qui sont plus dépendants de la classe que Amin

                    ProbAmin= Calculer-Probabilité(AAOP, Amin, L)

                  Sinon

                    ProbAmin= Calculer-Probabilité(AAP, Amin, L)

               pour k=0 jusqu'à nbClasse
               ResultatClassement[k]+ = Dist[k] * ProbAmin

               retourner   ResultatClassement

    Sinon return Dist

 Sinon
      Si l'attribut noeud  A est inconnu
         Ajouter A à la liste d'attributs manquants L
         pour i=1 jusqu'à n   // n est le nombre de valeurs de l'attribut A
         Classer-instance (I, sous-arbre(vi), L)

      Sinon
         Classer-instance(I, sous-arbre-Correspondant, L)
fin;
```

Cette méthode sert à parcourir l'arbre de décision final qui correspond à l'ensemble d'apprentissage entier, à explorer ses branches en cas de valeurs manquantes et à appeler l'arbre de l'attribut le moins dépendant de la classe trouvé dans un chemin depuis la racine de l'arbre à une feuille.

Calculer-Probabilité

Dans cette méthode, on calcule la probabilité qu'un attribut manquant prenne une valeur v. On distingue quatre cas pendant le parcours de son arbre d'attribut :

- *Si on est arrivé à une feuille sans valeurs manquantes* : on retourne la probabilité d'avoir la valeur v sur cette feuille. On fournit ensuite la liste L_1 qui contient les attributs qui ne sont pas dans la liste L mais qui sont manquants et dépendants des attributs dans la liste L avec une IM inférieure à celle des attributs de L. On ajoute à la liste L_1 les autres attributs qui sont à la fois manquants et dépendants des attributs dans la liste L_1. Ces nouveaux attributs possèdent une IM inférieure à celle des attributs de L_1.

  1. Si la liste L_1 est vide, on ordonne les attributs de L par ordre croissant en fonction de leur IM. Pour chaque attribut, on parcours son arbre (AAP ou AAOP selon la situation) pour obtenir sa probabilité P [46].

  2. Si la liste L_1 n'est pas vide, il faut traiter ses attributs un par un, et pour chaque attribut il faut traiter toutes ses valeurs de manière récursive en appelant la méthode *Traiter-Liste*.

  Enfin, on calcule la probabilité finale.
- *Si on est arrivé à une feuille avec valeurs manquantes* : on récupère la probabilité que l'attribut en question possède une valeur v sur cette feuille et on rappelle la méthode avec l'arbre *AAOP* de l'attribut manquant le moins dépendant de la classe qu'on a trouvé.
- *Si on a rencontré une valeur manquante* : on ajoute son attribut à la liste L et on rappelle la méthode pour chacune de ses valeurs [47].
- *Sinon* : on continue le classement en rappelant la méthode avec le sous-arbre approprié.

[46] Après avoir calculé la probabilité qu'un attribut A prenne la valeur v, on rend cet attribut connu dans l'objet à classer pour qu'on puisse calculer la probabilité qu'un autre attribut$B$ prenne une valeur sachant A. À la fin de traitement, on rend à nouveau l'attribut A inconnu et ainsi de suite.

[47] Quand on ajoute un attribut à la liste, on ajoute le couple (attribut, index) où l'index est son index dans la base d'apprentissage. Par exemple, un attribut *Temps* peut prendre 3 valeurs : *ensoleillé, couvert* et *pluvieux*. L'index de *Temps* peut prendre 3 valeurs : 0, 1, 2, ce qui correspond aux trois valeurs possibles de l'attribut *Temps*.

Traiter-Liste

```
Traiter-Liste(L, L1: Listes d'attributs manquants,
                     nbListe: Taille de la liste)
                      sortie: probabilitéFinale,
Début,
prob: une probabilité,
A: attribut,
   récupérer le premier attribut A dans la liste L1
   qui correspond à l'index 0
   pour i=1 jusqu'à nbValeur
   // nbValeur est le nombre de valeurs de l'attribut A
    Début
     ajouter Ai à la liste L,
     nbListe --,
     Traiter-Liste(L1,L,nbListe)

         Si nbListe ==0
          Début
           ordonner les attributs dans la liste L par ordre croissant
           en fonction de leur IM
           pour chaque attribut dans L
           prob = prob * la probabilité de l'attribut calculée
                  à partir de son AAOP ou AAP
         fin
        nbListe ++,
     probabilitéFinale = ProbabilitéFinale + prob,
     retirer Ai de la liste L,
     fin,
retourner probabilitéFinale,
fin,
```

Les principales étapes de classement

Les principales étapes pour classer un objet ayant des valeurs inconnues pour certains attributs en utilisant notre approche sont :

1. **Principe** : Nous parcourons l'arbre de décision final de sa racine jusqu'à ce qu'on arrive à une feuille. À chaque fois qu'on rencontre un attribut manquant pour un noeud dans cet arbre on doit parcourir tous les chemins qui correspondent aux valeurs de l'attribut inconnu. Dans ce cas, nous pouvons voir un arbre de décision comme un ensemble de chemins ; chaque chemin commence à la racine de l'arbre et va jusqu'à une feuille. Le nombre possible des chemins est donc le nombre de feuilles dans un arbre de décision. Dans le pire des cas, on parcourt tous les chemins dans cet arbre. Le traitement est donc effectué chemin par chemin.
2. **Traitement** : Pendant le parcours d'un chemin dans l'arbre de décision final, nous enregistrons dans une liste tous les attributs manquants rencontrés dans ce chemin.

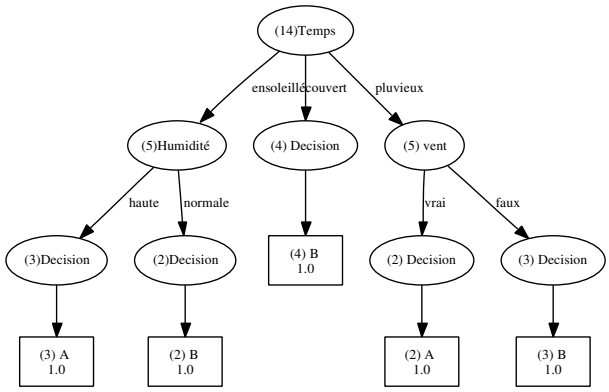

FIG. 4.4 – L'arbre de décision final pour la base *météo*

3. Quand nous arrivons à une feuille dans l'arbre de décision final, nous récupérons d'abord la distribution de probabilités de classe associée à cette feuille.
4. Ensuite, parmi tous les attributs manquants rencontrés dans le chemin, nous traitons l'attribut le moins dépendant de la classe :

   a- Si cet attribut n'est pas dépendant d'autres attributs manquants ayant des IM supérieurs à son IM, nous appelons son AAP. Sinon nous appelons son AAOP.

   b- Nous parcourons l'arbre (AAOP/AAP) de cet attribut ; si on trouve d'autres attributs manquants, on les ajoute à la liste contenant des valeurs manquantes ; on explore également tous les chemins correspondant aux valeurs de l'attribut inconnu jusqu'aux feuilles.

   c- Quand nous arrivons à une feuille dans le dernier arbre d'attribut, nous récupérons la probabilité associé à cette feuille.

   d- Nous fournissons une autre liste L_1 qui contient les autres attributs manquants dans l'instance, car les nouveaux attributs manquants ajoutés ne sont pas dans la liste L mais sont dépendants des attributs dans L et ont une IM inférieure à celle des attributs de L. Cette étape facilite le calcul de probabilité. Si cette liste n'est pas vide, on appelle la méthode *Traiter-Liste* et on passe ensuite à *g.* Sinon, on passe à *e.*

   e- Nous ordonnons tous les attributs manquants dans la liste par ordre croissant en fonction de leur IM avec la classe.

   f- Nous appelons successivement leurs arbres (AAP/ AAOP) pour récupérer leurs probabilités.

   g- Nous revenons à *b* pour un autre chemin dans cet arbre.
5. Quand nous avons parcouru tous les chemins dans cet arbre, nous retournons à 1 pour traiter un autre chemin dans l'arbre de décision final.

Le calcul de probabilité est donné dans la section précédente (*cf.* section 4.4).

Exemple

Si nous prenons l'exemple de *Météo*, pour classer un objet dont *Temps*, *Humidité* et *Température* sont inconnus. *Vent* est *vrai*. En parcourant l'arbre de décision donné dans la figure 4.4,

nous avons :

$$\begin{aligned} P(A) &= P(A|\texttt{ensoleillé, haute})P(\texttt{ensoleillé, haute}) \\ &+ P(A|\texttt{pluvieux, vrai})P(\texttt{pluvieux, vrai}) \\ P(B) &= P(B|\texttt{ensoleillé, normale})P(\texttt{ensoleillé, normale}) \\ &+ P(B|\texttt{couvert})P(\texttt{couvert}) \\ &+ P(B|\texttt{ pluvieux, faux})P(\texttt{pluvieux, faux}) \end{aligned} \tag{4.3}$$

Puisque *Temps* et *Humidité* sont inconnus, nous avons quatre chemins à parcourir **(étape 1)** :

- chemin1 : ensoleillé, haute
- chemin2 : ensoleillé, normal
- chemin3 : couvert
- chemin4 : pluvieux, vrai

Donc, quand nous parcourons le premier chemin *(ensoleillé, haute)* **(étape 2)**, nous arrivons à une feuille où les probabilités des valeurs de classe sont : P(A|ensoleillé, haute) =1 et P(B|ensoleillé, haute) =0 **(étape 3)**. Dans ce chemin il y a deux valeurs manquantes, *ensoleillé* et *haute*. Nous commençons par traiter l'attribut le moins dépendant de la classe ; qui est ici *haute* **(étape 4)**. Donc :

P(ensoleillé, haute)= P(haute) P(ensoleillé| haute)

Puisque *Humidité* et *Temps* sont inconnus et dépendants en même temps, nous appelons l'arbre *AAOP* de *Humidité* donné dans la figure 4.1 **(étape a)**. En parcourant cet arbre nous trouvons un autre attribut manquant qui est *Température*, et nous parcourons toutes les branches correspondant aux valeurs ce cet attribut **(étape b)**. Les chemins que nous rencontrons dans cet arbre sont :

- chemin 5 : élevée
- chemin 6 : basse
- chemin 7 : moyenne

Nous avons maintenant 3 attributs manquants *Temps*, *Humidité* et *Température*. Nous pouvons écrire la probabilité de *haute* comme suit (figure 4.1) :

$$\begin{aligned} P(haute) &= P(\texttt{haute|élevée})P(\texttt{élevée}) \\ &+ P(\texttt{haute|basse})P(\texttt{basse}) \\ &+ P(\texttt{haute|moyenne})P(\texttt{moyenne}) \end{aligned} \tag{4.4}$$

Donc, en parcourant le premier chemin *élevée* dans l'arbre de *Humidité* nous arrivons à une feuille où P(haute|élevée) = 0.75 **(étape c)**.

Nous cherchons s'il y a d'autres attributs manquants qui dépendent de *Temps*, *Humidité* ou *Température* et que nous n'avons pas encore rencontrés **(étape d)**. Dans notre exemple, il n'y a pas de tels attributs.

Nous passons à **l'étape e**. Nous ordonnons les attributs manquants rencontrés en ordre croissant en fonction de leurs IMs avec la classe : *élevée*, *haute*, *ensoleillé*. Nous calculons pour chaque valeur sa probabilité à partir de son arbre ; pour cela, nous pouvons écrire la probabilité de chemin 1 *(ensoleillé, haute)* :

$$\begin{aligned} P(\texttt{haute, ensoleillé}) &= P(\texttt{élevée})P(\texttt{haute|élevée})P(\texttt{ensoleillé|élevée, haute}) \\ &+ P(\texttt{basse})P(\texttt{haute|basse})P(\texttt{ensoleillé| basse,haute}) \\ &+ P(\texttt{moyenne})P(\texttt{haute|moyenne})P(\texttt{ensoleillé|moyenne, haute}) \end{aligned} \quad (4.5)$$

Dans **l'étape f**, nous calculons seulement la probabilité :

P(élevée) P(haute|élevée) P(ensoleillé| élevée, haute) = $0.28 \times 0.75 \times 0.66 = 0.1386$.

Nous notons que P(élevée) est calculé à partir de son AAOP donné dans la figure 4.1 ; P(haute|élevée) est calculé à partir de son AAOP donné dans la figure 4.1 et P(ensoleillé| élevée, haute) est calculée à partir de son AAP donné dans la figure 4.2.

Dans **l'étape g**, nous revenons à **l'étape b** pour parcourir un autre chemin dans l'arbre *AAOP* de *Humidité* (équation 4.4). Dans cette étape, nous parcourons le deuxième chemin dans l'arbre d'*Humidité*. Ce chemin correspond à *basse*, donc : P(haute| basse) = 0 **(étape c)**. Il n'y a pas d'autres attributs manquants qui vérifient **l'étape d**. Dans ce cas, les attributs manquants sont : basse, haute, ensoleillé, ce qui nous conduit à calculer la probabilité :

P(basse) P(haute|basse) P(ensoleillé| basse, haute)= $0.28 \times 0 \times 0.25 = 0$

Retour à **l'étape b** pour le dernier chemin dans l'arbre d'*Humidité*. Ce chemin correspond à *moyenne*. De la même façon, nous calculons la probabilité (deuxième terme de l'équation 4.4) :

P(moyenne) P(haute|moyenne) P(ensoleillé| moyenne, haute) =$0.43 \times 0.66 \times 0.25 = 0.071$

en conséquence :

P(haute, ensoleillé)= $0.1386 + 0 + 0.071 = 0.2096$.

Donc, à cette étape, nous avons :

P(A) = 0.2096, P(B) = 0.

Nous retournons à l'**étape 1** pour traiter un autre chemin dans l'arbre de décision final (équation 4.3). Ce chemin correspond à *(ensoleillé, normal)*. Donc, P(A|ensoleillé, normale) =0 et P(B|ensoleillée, normale)=1. Et ainsi de suite.

4.4.3 L'algorithme de construction

Nous avons utilisé l'algorithme ID3 [Quinlan, 1986] pour construire nos arbres *AAOPs* et *AAPs*, car cet algorithme construit un arbre de décision à partir d'une base d'apprentissage complète sans élagage. Nous avons ajouté à cet algorithme le fait de donner des résultats probabilistes sur chaque feuille au lieu d'associer à chaque feuille la valeur de classe la plus probable. Nous avons appelé cet algorithme ID3-Probabiliste, et chaque feuille contient toutes les valeurs de classe avec leur probabilités.

Lorsque nous construisons un *AAOP* pour un attribut A, nous donnons à ID3-probabiliste la base d'apprentissage qui contient toutes les instances, mais seulement les attributs qui sont dépendants de A et dont l'IM est inférieure à celle de A. Lorsque nous construisons un *AAP* pour un attribut A, nous donnons à ID3-probabiliste la base d'apprentissage qui contient toutes les instances ainsi que les attributs qui sont dépendants de A. Nous construisons également l'arbre de décision final qui correspond à la base d'apprentissage entière en utilisant seulement les attributs qui sont dépendants de la classe. Le fait d'éliminer certains attributs avant la construction est un pré-élagage.

Dans l'expérimentation que nous allons présenter dans la partie suivante, les arbres *AAOs* de Lobo et Numao sont construits en utilisant C4.5, qui utilise un post-élagage.

4.5 Conclusion

Le travail que nous avons proposé est basé sur l'utilisation des arbres de décision pour estimer des probabilités en cas de valeurs manquantes dans l'objet à classer. Nous avons vu que les branches dans un arbre de décision représentent les probabilités jointes et les feuilles représentent les probabilités conditionnelles. Ainsi, nous pouvons calculer la probabilité totale de chaque classe. En cas de valeurs manquantes, on fait appel à des arbres de décision pour prédire l'attribut inconnu.

Dans ce chapitre, nous avons expliqué notre approche, qui est dérivée de la méthode des *Arbres d'Attributs Ordonnées* proposée par Lobo et Numao, pour traiter les valeurs manquantes dans les données. Nous avons fait deux propositions : la première étend *(AAO) Arbres d'Attributs Ordonnés* avec des données probabilistes. La deuxième construit un arbre de décision probabiliste pour chaque attribut dans la base d'apprentissage en fonction de ses attributs dépendants. Chaque feuille dans un tel arbre contient une distribution de probabilités pour les valeurs de l'attribut en question, au lieu de sa valeur la plus probable. Ainsi, chaque attribut dans la base d'apprentissage possède deux arbres d'attributs, le premier construit selon l'approche *AAOPs* et le second selon la deuxième proposition *AAPs*.

Nous avons proposé de remplacer une valeur manquante par une distribution de probabilités et un objet incomplet par une distribution de probabilités de classe. Nous avons construit nos arbres en utilisant la dépendance entre les attributs. Par exemple, un médecin peut estimer la valeur d'un attribut en connaissant les valeurs des autres attributs dont il dépend. Le fait de prendre en compte la dépendance entre les attributs pour prédire la valeur inconnue conduit à des résultats de classement meilleurs et plus fiables que dans le cas contraire. Nous allons développer cette idée dans le chapitre suivant. Nous avons également expliqué dans ce chapitre l'algorithme de classement d'un objet incomplet en utilisant notre approche et le calcul de probabilité utilisé pendant le processus de classement.

Dans la partie suivante de cette thèse, nous allons présenter l'expérimentation de notre approche sur des bases d'apprentissage réelles et nous allons comparer nos résultats avec ceux donnés par C4.5 ainsi que par *AAO*. Ensuite, nous mesurerons la qualité de nos résultats de classement et nous calculons la complexité de notre approche.

Troisième partie

Expérimentation et Évaluation

Chapitre 5

Expérimentations

5.1 Introduction

Dans le chapitre précédent, nous avons expliqué notre approche, qui est basée sur la dépendance entre les attributs et qui donne un résultat de classement probabiliste. Nous présentons dans ce chapitre des études expérimentales réalisées sur des bases de données réelles issues du *(UCI Repository of machine learning databases)* [Newman *et al.*, 1998].

Nous partons d'une base d'apprentissage complète dont on construit les arbres *AAPs* et *AAOPs* ainsi que l'arbre de C4.5 et les arbres *AAOs* [48]. Nous utilisons l'algorithme *ID3* avec une extension probabiliste pour construire les arbres de décision.

Pendant la phase de classement, des bases de test ayant des valeurs manquantes sont également fournies pour tester notre approche. Les résultats de classement donnés par notre approche seront comparés avec ceux donnés par la méthode *AAO* et par la méthode C4.5 en utilisant les mêmes bases de test.

Le résultat de classement de chaque objet est une distribution de probabilités de classe. Les valeurs d'un attribut manquant ne sont pas remplies dans la base de test, parce qu'une distribution de probabilités est associée à chaque attribut dont la valeur est inconnue. Ces distributions de probabilités ont servi à calculer la probabilité de classe. Or, nous ne voulons pas remplacer une valeur manquante d'un attribut par sa valeur la plus probable. L'utilisation de la méthode de *Validation-croisée* n'est donc pas possible parce que la base de test n'est pas complète. Nous pouvons utiliser la *validation croisée* seulement pour comparer notre arbre de décision final avec ceux construits selon C4.5 et *AAO*.

Notre approche n'est applicable que sur des attributs discrets ou qualitatifs. Pour cela, nous discrétisons les attributs continus en utilisant une méthode de discrétisation supervisée [Witten Ian et Frank, 2005, Dougherty *et al.*, 1995].

Dans ce chapitre, nous détaillons également la *Matrice de Confusion* des résultats de classement obtenus par chaque méthode sur chaque base de test.

Dans un premier temps, nous classons les objets dans la base de test selon C4.5, *AAO*, et notre approche. Nous comparons les résultats de ces trois méthodes. Ensuite, nous calculons la matrice de confusion des résultats de chaque méthode. Nous calculons les mesures standards à partir de ces matrices, Précision, Recall, F-measure, etc. Ces mesures nous aident à analyser les

[48] Nous n'avons pas traité la question de valeurs manquantes dans la base d'apprentissage ; en effet, nous nous intéressons seulement à cette question pendant le classement. Par conséquent, notre méthode n'est pas applicable quand la base d'apprentissage contient des valeurs manquantes puisque l'on ne peut pas affecter une valeur unique à un attribut dont la valeur est inconnue.

résultats de chaque méthode pour chaque valeur de classe.

5.2 Choix d'un seuil

Le seuil[49] est fixé en calculant l'Information Mutuelle (IM) entre chaque attribut et la classe. Dans notre approche, on détermine un seuil qui est supérieur à l'Information Mutuelle minimale trouvée (non nulle) et proche de l'Information Mutuelle moyenne. Par exemple, si l'IM minimale est 0.01, l'IM maximale 0.2, et la moyenne de toutes les valeurs d'IM 0.05, on peut tester la méthode sur plusieurs seuils : 0.02, 0.03, 0.04 et 0.05.

- Le fait de fixer un seuil conduit à éliminer avant la construction d'un arbre d'attribut probabiliste les attributs qui sont indépendants de l'attribut courant.
- Si on choisit un seuil très faible, on augmente le nombre d'attributs dépendants. De plus, on diminue le nombre d'instances par feuille. En revanche, le choix d'un seuil assez élevé nous conduit à diminuer le nombre d'attributs dépendants. Pour cela, nous testons notre approche sur plusieurs seuils pour la même base de test et nous comparons les résultats.
- Dans le cas où le nombre de valeurs par attribut est élevé, nous étudions l'information mutuelle normalisée et nous pouvons choisir un seuil égal ou supérieur à la moyenne. L'information Mutuelle Normalisée utilisé est celle proposée par Lobo et Numao ((*cf.* section 3.4.2), équation 3.4).

5.3 Évaluation du modèle

Pour évaluer notre approche, nous commençons par la tester sur plusieurs bases réelles. Ensuite, nous évaluons sa performance en utilisant la *Matrice de Confusion*. Finalement, nous comparons les résultats donnés par notre approche avec les résultats donnés par la méthode C4.5 et par la méthode AAO sur les mêmes bases. Nous commençons par expliquer *la matrice de confusion*.

5.3.1 Matrice de Confusion

La matrice de confusion [Kohavi et Provost, 1998, Witten Ian et Frank, 2005, Tan *et al.*, 2006] contient des informations concernant le classement actuel [50] dans la base de test ainsi que le classement prédit [51] par le système de classement utilisé (par exemple, notre approche, C4.5 ou AAO). La performance du système de classement est évaluée en utilisant les données de la matrice. Le tableau 5.1 présente la matrice de confusion pour deux valeurs de classe (positive, négative). Ainsi,

- a est le nombre de classements corrects des instances de classe **négative**.
- b est le nombre de classements incorrects des instances de classe **négative**.
- c est le nombre de classements incorrects des instances de classe **positive**.
- d est le nombre de classements corrects des instances de classe **positive**.

À partir de la matrice de confusion ci-dessus, plusieurs mesures appelées mesures d'exactitude par classe, sont définies :

[49] L'expert du domaine peut également fixer le seuil.

[50] La classe actuelle d'une instance dans la base de test est la classe associée à cette instance dans la base de test.

[51] Le résultat de classement en utilisant C4.5, AAO ou notre approche.

		Predicted	
		Negative	Positive
Actual	Négative	a	b
	Positive	c	d

Tab. 5.1 – La martice de Confusion

– Accuracy (exactitude - taux de bon apprentissage) : la proportion des instances qui sont bien classées.

$$Accuracy = \frac{a+d}{a+b+c+d} \tag{5.1}$$

– True Postive Rate (TP Rate) ou (Recall - rappel) : la proportion des instances de classe positive qui sont correctement classées.

$$TPrate = Recall = \frac{d}{c+d} \tag{5.2}$$

C'est donc le rapport entre le nombre de bien classés et le nombre total d'instances qui devraient être bien classées. Il se calcule en utilisant la deuxième ligne de tableau 5.1. Si le *rappel* est à 1, cela signifie que toutes les instances positives ont été trouvées.

– True Negative Rate (TN Rate) : la proportion des instances négatives qui sont correctement classées.

$$TNrate = \frac{a}{a+b} \tag{5.3}$$

– False Postive Rate (FP Rate) : la proportion des instances négatives qui sont incorrectement classées comme positives.

$$FPrate = \frac{b}{a+b} \tag{5.4}$$

Il se calcule en utilisant la première ligne du tableau 5.1.

– False Negative Rate (FN Rate) : la proportion des instances positives qui sont incorrectement classées comme négatives.

$$FNrate = \frac{c}{c+d} \tag{5.5}$$

– Precision (p) : la proportion des instances classées positives correctement parmi toutes les instances classées positives.

$$Precision = \frac{d}{b+d} \tag{5.6}$$

La précision se calcule en utilisant la deuxième colonne du tableau 5.1. Si la valeur de Precision est à 1, cela exprime le fait que toutes les instances classées positives l'étaient vraiment.

– $F_{measure}$: c'est une mesure globale qui regroupe Precision et Recall dans une seule matrice.

$$F_{measure} = \frac{2 \times Recall \times Precision}{Recall + Precision} = \frac{2rp}{r+p} \tag{5.7}$$

Cette mesure permet de regrouper en une seule valeur les performances du classifieur (pour une classe donnée) pour ce qui concerne le Recall et la Precision.

Dans notre expérimentation et pour chaque base de test, nous avons calculé la matrice de confusion sur les résultats de classement donnés par notre approche, ainsi que par C4.5 et *AAO*.

Nous avons choisi de présenter les résultats de classement dans un tableau de 5 colonnes :

- la première colonne contient le nom de la base ainsi que les noms des méthodes utilisées *AAP, C4.5, AAO*.
- la deuxième colonne contient le seuil, qui prend différentes valeurs quand la méthode utilisée est *AAP*.
- la troisième colonne contient le pourcentage des instances de la base de test qui sont bien classées selon les trois méthodes.
- la quatrième colonne contient le pourcentage des instances de la base de test qui sont mal classées selon les trois méthodes.
- la cinquième colonne est celle qui contient le pourcentage des instances dont nous ne pouvons pas décider le résultat de classement parce que nous avons au moins deux valeurs de classe qui ont la même probabilité maximale. Par exemple, si la classe prend deux valeurs, alors le résultat de classement d'une telle instance est 0.5 pour chaque valeur de classe. Si la classe prend quatre valeurs, dans ce cas, le résultat de classement de chaque valeur de classe est 0.25, ou bien 0.5 pour seulement deux valeurs de classe et 0 pour les autres, ou encore 0.33333 pour trois valeurs de classe et 0 pour la quatrième, etc.

Donc, dans un premier temps, nous considérons les résultats de classement situés dans la quatrième colonne comme mal classés. Dans le chapitre suivant, nous analysons le résultat de classement de chaque objet.

5.4 Études Expérimentales

Dans cette étude, nous commençons par tester notre approche sur des bases de données réelles sur plusieurs seuils [Hawarah *et al.*, 2006c, Hawarah *et al.*, 2006d]. Il y a trois étapes :

La phase de construction : Nous construisons nos arbres *AAPs* et *AAOPs* en utilisant une base d'apprentissage sans valeurs manquantes. L'arbre C4.5 et les arbres *AAOs* sont également construits.

La phase de classement : Cette phase a pour but de classer les nouvelles instances ayant des valeurs manquantes en utilisant notre approche *AAP*, la méthode C4.5 et la méthode*AAO*. Dans cette phase, une base de test incomplète est utilisée.

La phase de comparaison : Une fois le classement effectué, nous comparons les résultats donnés par *AAP* avec ceux donnés par C4.5 et *AAO* pour les mêmes instances.

Nous allons tester notre approche sur différentes bases de données. Nous avons distingué trois types de bases : 1) les bases dont les attributs (discrets ou qualitatifs) sont indépendants ; 2) les bases dont les attributs (discrets ou qualitatifs) sont dépendants ; 3) les bases dont les attributs sont continus et dépendants.

Le choix d'une base de test

Les bases de test sont générées à partir des bases d'apprentissage. Nous avons précisé que les bases d'apprentissage utilisées ne contiennent pas de valeurs manquantes. Pour fournir une base de test, nous sélectionnons de manière aléatoire des instances de la base d'apprentissage. Nous remplaçons les attributs pertinents (qui dépendent de la classe) par des valeurs manquantes.

Nous rappelons que les arbres *AAOPs* et *AAPs* sont construits en utilisant l'algorithme ID3-Probabiliste. Les arbres *AAOs* sont construits en utilisant C4.5.

Zoo	Seuil	bien classés	mal classés	50%
	0.1	94.18%	05.63%	
AAPs	**0.2**	**97.18%**	02.81%	
	0.3	87.32%	12.67%	
C4.5		**63.38%**	**36.61%**	
AAO		**88.73%**	**11.26%**	

TAB. 5.2 – Résultats de test de *AAP*, *C4.5* et *AAO* sur la base *Zoo*

ID	Attributs	Taux de valeurs manquantes
1	feathers	43.50%
2	eggs	5.61%
3	**milk**	36.23%
4	airborne	15.49%
5	aquatic	22.53%
6	predator	15.49%
7	toothed	07.04%
8	backbone	19.71%
9	breathes	14.08%
10	venomous	16.90%
11	fins	09.85%
12	legs	22.53%
13	tail	21.12%

TAB. 5.3 – Le taux de valeurs manquantes dans la base de test *Zoo*

5.4.1 Bases ayant des attributs dépendants

La dépendance entre les attributs est calculée en utilisant l'Information Mutuelle Normalisée. Deux attributs sont considérés comme dépendants si leur Information Mutuelle Normalisée est supérieure ou égale à un certain seuil. Chaque base est testée en utilisant notre approche sur plusieurs seuils((*cf.* section 5.2)).

La base Zoo

Nous avons testé notre méthode sur la base *Zoo* ayant 101 instances sans valeurs manquantes et 17 attributs. La classe prend les 7 valeurs suivantes : *mammal, bird, reptile, fish, amphibian, insect, invertebrate*. La base de test utilisée possède 71 instances dont le taux de valeurs manquantes est donné dans le tableau 5.3. Nous pouvons remarquer que les résultats de classement donnés par *AAP* sont meilleurs que ceux donnés par C4.5 et *AAO* (tableau 5.2).

Nous présentons *les Matrices de Confusion* générées pour chaque méthode sur la base *Zoo*. Ces matrices nous permettent de voir en détail les résultats de classement pour chaque valeur de classe.

L'exactitude de notre approche sur la base *Zoo* est $\frac{67}{71} = 0.941$, ce qui correspond à notre résultat dans le tableau 5.2.

a	b	c	d	e	f	g	<–classified as
27	0	0	0	0	0	0	| a = mammal
0	16	0	0	0	0	1	| b = bird
1	0	2	0	1	0	0	| c = reptile
0	0	0	10	0	0	0	| d = fish
0	0	0	0	3	0	0	| e = amphibian
0	1	0	0	0	3	0	| f = insect
0	0	0	0	0	0	6	| g = invertebrate

TAB. 5.4 – La matrice de Confusion de AAP sur la base Zoo

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
1	0.023	0.964	1	0.982	mammal
0.941	0.019	0.941	0.941	0.941	bird
0.5	0	1	0.5	0.667	reptile
1	0	1	1	1	fish
1	0.015	0.75	1	0.857	amphibian
0.75	0	1	0.75	0.857	insect
1	0.015	0.857	1	0.923	invertebrate

TAB. 5.5 – Les détails des résultats de AAP

		Classe prédite	
		non bird	bird
Classe actuelle	non bird	53	1
	bird	1	16

TAB. 5.6 – La martice de Confusion pour *bird* selon *AAP*

a	b	c	d	e	f	g	<–classified as
27	0	0	0	0	0	0	| a = mammal
6	1	10	0	0	0	0	| b = bird
1	0	2	0	0	0	1	| c = reptile
1	0	0	9	0	0	0	| d = fish
1	0	0	1	0	0	1	| e = amphibian
1	0	0	0	0	3	0	| f = insect
2	0	0	0	1	0	3	| g = invertebrate

TAB. 5.7 – La matrice de Confusion de C4.5 sur la base *Zoo*

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
1	0.273	0.692	1	0.818	mammal
0.059	0	1	0.059	0.111	bird
0.5	0.149	0.167	0.5	0.25	reptile
0.9	0.016	0.9	0.9	0.9	fish
0	0.015	0	0	0	amphibian
0.75	0	1	0.75	0.857	insect
0.5	0.031	0.6	0.5	0.545	invertebrate

TAB. 5.8 – Les détails des résultats de C4.5

À partir du tableau 5.4, nous pouvons calculer pour chaque valeur de classe TP Rate, FP Rate, Precision, Recall et F-Measure. Par exemple, à partir du tableau 5.4 la matrice de confusion de la classe *bird* est donnée dans le tableau 5.6. Donc, nous pouvons calculer pour *bird* les mesures suivantes :

TP Rate pour *bird*= Recall de *bird*= $\frac{16}{16+1} = 0.941$

FP Rate pour *bird* = $\frac{1}{53+1} = 0.0185 = 0.019$

Precision de *bird* = $\frac{16}{16+1} = 0.941$

F-Measure de *bird* = $\frac{2\times0.941\times0.941}{0.941+0.941} = 0.941$

Nous pouvons calculer de la même manière ces mesures pour les autres valeurs de classe.

À partir du tableau 5.7, nous pouvons calculer :

L'exactitude de C4.5 sur la base *Zoo* = $\frac{45}{71} = 0.63$.

TP Rate pour *bird*=Recall de *bird*= $\frac{1}{1+16} = 0.0588$

FP Rate pour *bird* = $\frac{0}{54+0} = 0$

Precision de *bird* = $\frac{1}{1+0} = 1$

F-Measure de *bird* = $\frac{2\times0.0588\times1}{1+0.0588} = 0.111$

À partir de la matrice de confusion pour la méthode AAO donnée dans le tableau 5.9, nous pouvons calculer :

L'exactitude d'AAO sur la base *Zoo* = $\frac{63}{71} = 0.88$

TP Rate pour *bird*=Recall de *bird*= $\frac{13}{13+4} = 0.765$

FP Rate pour *bird* = $\frac{0}{54+0} = 0$

Precision de *bird* = $\frac{13}{13+0} = 1$

a	b	c	d	e	f	g	<–classified as
25	0	2	0	0	0	0	| a = mammal
0	13	3	0	1	0	0	| b = bird
0	0	2	0	2	0	0	| c = reptile
0	0	0	10	0	0	0	| d = fish
0	0	0	0	3	0	0	| e = amphibian
0	0	0	0	0	4	0	| f = insect
0	0	0	0	0	0	6	| g = invertebrate

TAB. 5.9 – La matrice de Confusion de AAO sur la base Zoo

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
0.926	0	1	0.926	0.962	mammal
0.765	0	1	0.765	0.867	bird
0.5	0.075	0.286	0.5	0.364	reptile
1	0	1	1	1	fish
1	0.044	0.5	1	0.667	amphibian
1	0	1	1	1	insect
1	0	1	1	1	invertebrate

TAB. 5.10 – Les détails des résultats de AAO

F-Measure de *bird* = $\frac{2\times0.765\times1}{1+0.765} = 0.867$

En comparant les rappels de *bird* obtenus avec chaque méthode, nous trouvons 94% avec *AAP*, 5% avec C4.5 et 76% avec *AAO*. Cela signifie que 94% des instances de classe *bird* ont été trouvées avec *AAP*, 5% seulement davec *C4.5* et 76% avec *AAO*. Nous pouvons constater que la performance de classement quand la classe prend la valeur *bird* est la meilleure lorsqu'on utilise l'approche *AAP*.

Comparer les résultats de chaque méthode sur la base *Zoo*

Nous pouvons remarquer à partir des tableaux 5.4, 5.7, 5.9 que :
- Le nombre des instances ayant comme classe la valeur *mammal* est 27,donc notre approche et C4.5 ont bien classé les 27 instances. *AAO* a bien classé 25 instances parmi les 27.
- Le nombre des instances ayant comme classe la valeur *bird* est 17 parmi lesquelles notre approche a bien classé 16 instances. C4.5 a bien classé une seule instances parmi les 17 et AAO a bien classé 13 instances.
- Le nombre des instances ayant comme classe la valeur *reptile* est seulement 4, donc notre approche, C4.5 et *AAO* ont bien classé 2 instances.
- Le nombre des instances ayant comme classe la valeur *fish* est 10 ; notre approche et *AAO* ont bien classé les 10 instances. C4.5 a classé correctement 9 instances parmi les 10.
- Le nombre des instances ayant comme classe la valeur *amphibian* est 3 que notre approche et *AAO* ont classé correctement. Par contre, C4.5 a mal classé les 3 instances.
- Le nombre des instances ayant comme classe la valeur *insect* est 4, donc notre approche a classé correctement les 4 instances. C4.5 et *AAO* ont bien classé 3 instances.
- Le nombre des instances ayant comme classe la valeur *invertebrate* est 6 que notre approche

ID	attribut	valeurs	ID	attribut	valeurs
1	hair	false	10	breathes	true
2	feathers	?	11	venomous	false
3	eggs	true	12	fins	false
4	milk	false	13	legs	2
5	airborne	false	14	tail	true
6	aquatic	false	15	domestic	false
7	predator	false	16	catsize	true
8	toothed	false	17	type	bird
9	backbone	true			

TAB. 5.11 – L'instance de la base *Zoo* à classer

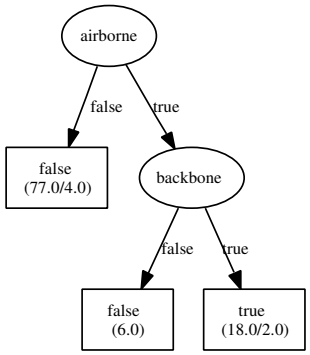

FIG. 5.1 – L'AAO pour *feathers*

et AAO ont classé correctement. C4.5 a bien classé seulement 3 instances.

Dans la suite, pour montrer la différence entre notre approche et la méthode AAO, nous avons choisi un exemple très simple. Dans l'exemple donné dans le tableau 5.11, le seul attribut manquant est *feathers*. Le résultat de classement en utilisant AAO est *reptile*. Le résultat de classement en utilisant AAP est *bird* avec une probabilité égale à 1.

Pourquoi AAO n'a pas bien classé cette instance ?

La méthode AAO cherche à trouver la valeur de l'attribut *feathers* à partir de son arbre qui est construit en utilisant les attributs *airborne* et *backbone* (figure 5.1). L'attribut *airborne* prend la valeur *false* et l'attribut *backbone* prend la valeur *true* dans l'instance à classer.
Dans ce cas, la valeur de l'attribut *feathers* est *false*. En classant l'objet dans l'arbre de décision final utilisé par AAO et construit en utilisant la méthode C4.5 (figure 5.2) et en prenant en compte les valeurs des autres attributs dans l'instance à classer, nous arrivons à la classe *reptile*. Nous notons que l'attribut *feathers* n'est pas l'attribut le plus dépendant de la classe mais C4.5 l'a utilisé comme racine de son arbre.

Pour classer la même instance en utilisant AAP, nous allons utiliser l'arbre de *feathers* donné dans la figure 5.3 et construit en utilisant ses attributs dépendants, qui sont *toothed* et *legs*. En parcourant cet arbre et en sachant que *toothed* est *false* et *legs* vaut 2, nous pouvons constater que la valeur de *feathers* est *true*.

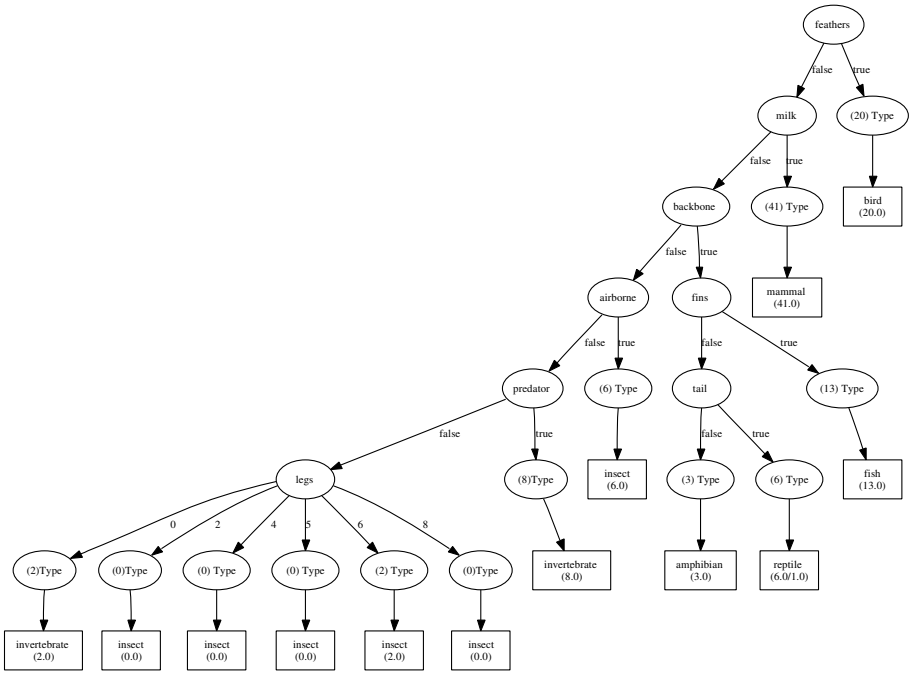

FIG. 5.2 – L'arbre de décision final utilisé par *AAO* et construit en utilisant C4.5 pour la base *Zoo*

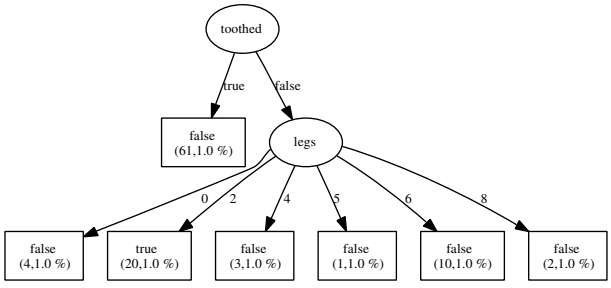

FIG. 5.3 – L'*AAP* pour *feathers*

Mushroom	Seuil	bien classés	mal classés	50%
PAT	**0.1**	**80.82%**	**19.17 %**	
	0.2	75.34%	24.65 %	
C4.5		**58.90%**	**41.09%**	
AAO		**67.12%**	**32.87%**	

TAB. 5.12 – Résultats de test de *AAP*, *C4.5* et *AAO* sur la base *Mushroom*

ID	Attributs	Taux de valeurs manquantes
1	bruises ?	27.39%
2	odor	75.34%
3	gill-attachment	08.21%
4	gill-size	09.58%
5	gill-color	08.21%
6	stalk-shape	26.02%
7	stalk-root	52.05%
8	stalk-surface-above-ring	04.10%
9	stalk-surface-below-ring	04.10%
10	stalk-color-below-ring	08.21%
11	veil-type	23.28%
12	ring-type	19.17%
13	spore-print-color	27.39%

TAB. 5.13 – Le taux de valeurs manquantes dans la base de test *Mushroom*

Le résultat n'est équivalent à celui donné par l'*AAO* de *feathers*. Donc, si on classe cette instance en utilisant l'arbre donné dans la figure 5.2, nous obtenons la valeur *bird* comme résultat de classement.

De plus, si nous utilisons l'arbre de décision final donné dans la figure 6.2 et construit selon l'approche *AAP* pour un seuil 0.1, nous obtenons également *bird* comme résultat de classement. Dans ce dernier arbre, l'attribut *milk* est la racine de l'arbre parce que c'est l'attribut le plus dépendant de la classe. Ainsi, le classement est fait sans prendre en compte la valeur manquante de l'attribut *feathers* parce que nous n'avons pas rencontré cet attribut pendant le parcours de l'arbre. Nous pouvons constater qu'avec notre approche, nous gardons l'attribut le plus dépendant de la classe comme racine de l'arbre, ce qui conduit à des résultats de classement plus précis.

La base *Mushroom*

Nous avons testé notre approche ainsi que C4.5 et *AAO* sur la base *Mushroom*. La base d'apprentissage possède 5644 instances sans valeurs manquantes et 22 attributs nominaux. La classe dans cette base prend deux valeurs : *e, p*. La base de test utilisée contient 73 objets dont le taux de valeurs manquantes est donné dans le tableau 5.13. Le tableau 5.12 contient les résultats de test quand le seuil est 0.1 et 0.2. Nous remarquons que dans les deux cas les résultats de l'approche *AAP* sont meilleurs que ceux obtenus par C4.5 et par *AAO*.

a	b	<–classified as
33	5	| a = e
13	22	| b = p

TAB. 5.14 – La matrice de Confusion de AAP sur la base *Mushroom*

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
0.868	0.371	0.717	0.868	0.786	e
0.629	0.132	0.815	0.629	0.71	p

TAB. 5.15 – Les détails des résultats de AAP sur la base *Mushroom*

a	b	<–classified as
33	5	| a = e
25	10	| b = p

TAB. 5.16 – La matrice de Confusion de C4.5 sur la base *Mushroom*

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
0.868	0.714	0.569	0.868	0.688	e
0.286	0.132	0.667	0.286	0.4	p

TAB. 5.17 – Les détails des résultats de C4.5 sur la base Mushroom

a	b	<–classified as
35	3	| a = e
21	14	| b = p

TAB. 5.18 – La matrice de Confusion de AAO sur la base *Mushroom*

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
0.921	0.6	0.625	0.921	0.745	e
0.4	0.079	0.824	0.4	0.538	p

TAB. 5.19 – Les détails des résultats de AAO sur la base *Mushroom*

ID	Attributs	Le taux de valeurs manquantes
1	scaling	12.96%
2	polygonal-papules	5.40%
3	follicular-papules	7.40%
4	family-history	7.40%
5	fibrosis-of-the-papillary-dermis	51.85%
6	elongation-of-the-rete-ridges	25.92%
7	spongiform-pustule	7.40%
8	focal-hypergranulosis	14.81%
9	vacuolisation-and-damage-of-basal-layer	24.07%
10	spongiosis	22.22%
11	perifollicular-parakeratosis	18.51%
12	band-like-infiltrate	24.07%
13	Age	16.66%

TAB. 5.20 – Le taux de valeurs manquantes dans la base de test *Dermatology*

Dermatology	Seuil	bien classés	mal classés	50%
PAT	**0.07**	**90.74%**	**09.25%**	
	0.09	**88.88%**	**11.11%**	
	0.1	**88.88%**	**11.11%**	
	0.2	77.77%	22.22%	
C4.5		**75.92%**	**24.07%**	
AAO		**85.18%**	**14.81%**	

TAB. 5.21 – Résultats de test de *AAP*, *C4.5* et *AAO* sur la base *Dermatology*

En ce qui concerne le Recall et la Precision, nous remarquons à partir des tableaux 5.15, 5.17, 5.19 que *AAP* est le classifieur le plus performant quand la classe prend la valeur *p*. Le pourcentage des instances de classes *p* qui sont correctement classées est 62% avec *AAP*, 28% avec C4.5 et 40% avec *AAO*. Le pourcentage des instances qui sont correctement classées *p* parmi toutes les instances classées *p* est 81% avec *AAP*, 66% avec C4.5 et 82% avec *AAO*.

La base *Dermatology*

La base d'apprentissage *Dermatology* possède 358 objets sans valeurs manquantes et 34 attributs discrets dont la classe qui prend les valeurs suivantes : *1, 2, 3, 4, 5, 6*. La base de test possède 54 objets dont le taux de valeurs manquantes est donné dans le tableau 5.20. L'Information Mutuelle Normalisée moyenne est 0.19. Nous avons testé notre approche sur plusieurs seuils : 0.09, 0.1 et 0.2. Le tableau 5.21 contient les résultats de test en utilisant notre méthode, C4.5 et *AAO* sur la base *Dermatology*. Nos résultats sont meilleurs que les résultats de C4.5 et *AAO*. Nous remarquons à partir du tableau 5.21 ainsi que des matrices de confusion pour la base *Dermatology* que notre approche a bien classé 49 instances parmi les 54 pour un seuil 0.07.C4.5 a bien classé 41 instances et *AAO* a bien classé 46 instances.

En comparant le tableau 5.23 avec le tableau 5.27, on voit que la performance de *AAP* est

a	b	c	d	e	f	<–classified as
13	0	0	0	0	0	| a = 1
0	9	0	0	0	0	| b = 2
0	0	11	0	0	0	| c = 3
0	0	0	7	0	0	| d = 4
2	3	0	0	8	0	| e = 5
0	0	0	0	0	1	| f = 6

TAB. 5.22 – La matrice de Confusion de AAP sur la base *Dermatology* pour le seuil 0.07

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
1	0.049	0.867	1	0.929	1
1	0.067	0.75	1	0.857	2
1	0	1	1	1	3
1	0	1	1	1	4
0.615	0	1	0.615	0.762	5
1	0	1	1	1	6

TAB. 5.23 – Les détails des résultats de AAP sur la base *Dermatology*

a	b	c	d	e	f	<–classified as
13	0	0	0	0	0	| a = 1
3	6	0	0	0	0	| b = 2
2	1	8	0	0	0	| c = 3
2	0	0	5	0	0	| d = 4
4	1	0	0	8	0	| e = 5
0	0	0	0	0	1	| f = 6

TAB. 5.24 – La matrice de Confusion de C4.5 sur la base *Dermatology*

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
1	0.268	0.542	1	0.703	1
0.667	0.044	0.75	0.667	0.706	2
0.727	0	1	0.727	0.842	3
0.714	0	1	0.714	0.833	4
0.615	0	1	0.615	0.762	5
1	0	1	1	1	6

TAB. 5.25 – Les détails des résultats de C4.5 sur la base *Dermatology*

a	b	c	d	e	f	<–classified as
12	0	0	1	0	0	| a = 1
2	6	0	1	0	0	| b = 2
0	0	11	0	0	0	| c = 3
1	0	0	6	0	0	| d = 4
2	1	0	0	10	0	| e = 5
0	0	0	0	0	1	| f = 6

TAB. 5.26 – La matrice de Confusion de AAO sur la base *Dermatology*

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
0.923	0.122	0.706	0.923	0.8	1
0.667	0.022	0.857	0.667	0.75	2
1	0	1	1	1	3
0.857	0.043	0.75	0.857	0.8	4
0.769	0	1	0.769	0.87	5
1	0	1	1	1	6

TAB. 5.27 – Les détails des résultats de AAO sur la base *Dermatology*

meilleure que celle de *AAO* en ce qui concerne la mesure *F-Measure*, pour toutes les valeurs de classe sauf la valeur 5 où la performance de *AAO* est meilleure. Nous remarquons que le *rappel* de la classe *a* avec C4.5 est 1 parce que C4.5 a bien classé les 13 instances de classe *a*. Cependant, le pourcentage de la Précision est 54% parce qu'il y a 11 instances qui sont classées incorrectement *a*.

La base *Vote*

Nous avons testé notre approche sur la base *Vote* [Newman *et al.*, 1998]. La base d'apprentissage contient 232 objets avec 16 attributs nominaux binaires[52]. La classe prend deux valeurs (*Democrat* et *Republican*). Cependant, nous avons utilisé une base de test qui contient 240 objets avec des valeurs manquantes. Le taux de valeurs manquantes est donné dans le tableau 5.28. La moyenne de l'Information Mutuelle Normalisée est 0.26. Pour cela, nous avons testé notre méthode sur plusieurs seuils :0.2, 0.3, 0.4 et 0.5. Le résultat est donné dans le tableau 5.29.

La colonne 50% dans le tableau 5.29 contient le pourcentage des objets ayant la probabilité 0.5 pour chaque valeur de classe[53]. Dans le même tableau, on trouve les résultats de classement donnés par l'arbre de C4.5 et *AAO* pour la même base de test. Nous observons que quand nous réduisons le seuil, nous améliorons notre résultat pour cette base. Pour tous les seuils testés, nos résultats sont meilleurs que ceux donnés par C4.5. Mais nos résultats sont égaux à ceux donnés par *AAO*. Les matrices de confusion pour la base *Vote* sont présentées dans l'annexe B ((*cf.* section B.1)).

[52]Chacun de ces attributs prend deux valeurs : y ou n.
[53]Parce que la classe prend deux valeurs

ID	Attributs	Taux de valeurs manquantes
1	handicapped-infants	05.00%
2	water-project-cost-sharing	24.58%
3	adoption-of-the-budget-resolution	04.58%
4	**physician-fee-freeze**	48.33%
5	**el-salvador-aide**	08.75%
6	religious-groups-in-schools	04.58%
7	anti-satellite-test-ban	07.50%
8	aid-to-nicaraguan-contras	07.08%
9	mx-missile	10.41%
10	immigration	02.50%
11	synfuels-corporation-cutback	11.25%
12	**education-spending**	17.08%
13	superfund-right-to-sue	13.75%
14	crime	08.33%
15	duty-free-exports	14.58%
16	export-administration-act-south-africa	50.20%

TAB. 5.28 – Le taux de valeurs manquantes dans la base de test *Vote*

Vote	Seuil	bien classés	mal classés	50%
PAT	**0.2**	**91.25%**	**08.33%**	**0.41%**
	0.3	90.00%	09.16%	0.83%
	0.4	88.33%	11.66%	
	0.5	87.08%	12.91%	
C4.5		**83.75%**	**16.25%**	
AAO		**91.66%**	**08.33%**	

TAB. 5.29 – Résultats de test de *AAP*, *C4.5* et *AAO* sur la base *vote*

Attribut	Information Mutuelle	Information Mutuelle Normalisée
Age	0.020728822	0.009942060
menopause	0.011550725	0.008936861
tumor-size	0.061456810	0.026807988
inv-nodes	0.082421073	0.035069516
node-caps	0.055882090	0.055882090
deg-malig	0.088532845	0.068498359
breast	0.001232732	0.001232732
breast-quad	0.008641386	0.005202633
Irradiat	0.034703629	0.034703629
Class	0.871825271	0.871825271

TAB. 5.30 – L'Information Mutuelle et l'Information Mutuelle Normalisée entre les attributs et la classe dans la base *breast-cancer*

La base *Breast-cancer*

Nous présentons les résultats de test de notre méthode ainsi que C4.5 et AAO sur la base *Breast-cancer* [Newman *et al.*, 1998]. La base possède 286 objets dont 9 ayant des valeurs manquantes. Elle contient 9 attributs discrets et nominaux, parmi lesquels *age* possède 9 valeurs, *tumor-size* 12, *inv-nodes* 13, *breast-quad* 5, et chacun des attributs *menopause* et *deg-malig* 3 valeurs. La classe dans cette base peut prendre deux valeurs (*no-recurrence-events, recurrence-events*). Pour la construction des arbres $AAOPs$ et $AAPs$, nous avons utilisé la base précédente sans les 9 objets ayant des valeurs manquantes.

L'arbre de la figure 5.5 est celui qui correspond à la base d'apprentissage *Breast-cancer* construit selon notre approche AAP pour un seuil 0.04. Dans la figure 5.4, nous avons l'arbre correspondant à la même base d'apprentissage, mais construit selon la méthode C4.5. L'Information Mutuelle Normalisée est utilisée pour étudier la dépendance entre les attributs, ainsi que pendant la construction des arbres, car le nombre de valeurs pour certains attributs dans cette base est assez élevé. Le tableau 5.30 contient l'Information Mutuelle et l'Information Mutuelle Normalisée sur la base Breast-cancer. On remarque que l'Information Mutuelle Normalisée n'a pas trop pénalisé les attributs ayant un nombre élevé de valeurs.

Nous avons utilisé une base de test qui contient 92 objets, dont le taux de valeurs manquantes est 60.86% pour l'attribut *node-caps*, 39.13% pour l'attribut *deg-malig*, 16.30% pour *irradiat* et 3.23% pour *tumor-size*. Les résultats de classement sont présentés dans le tableau 5.31. Nous remarquons que nos résultats sont meilleurs que ceux donnés par C4.5 et par AAO. Les matrices de confusion pour la base *Breast-cancer* sont présentées dans l'annexe B ((*cf.* section B.2)). AAP a bien classé 62 objets de la classe *no-recurrence-events* parmi 66 et 10 objets de la classe *recurrence-events* parmi 26. C4.5 a bien classé 63 objets de la classe *no-recurrence-events* parmi 66 et 2 objets de la classe *recurrence-events* parmi 26. AAO a bien classé 60 objets de la classe *no-recurrence-events* parmi 66 et 5 objets de la classe *recurrence-events* parmi 26. Donc, AAP classe mieux que les autres méthodes les instances appartenant à la classe *recurrence-events*.

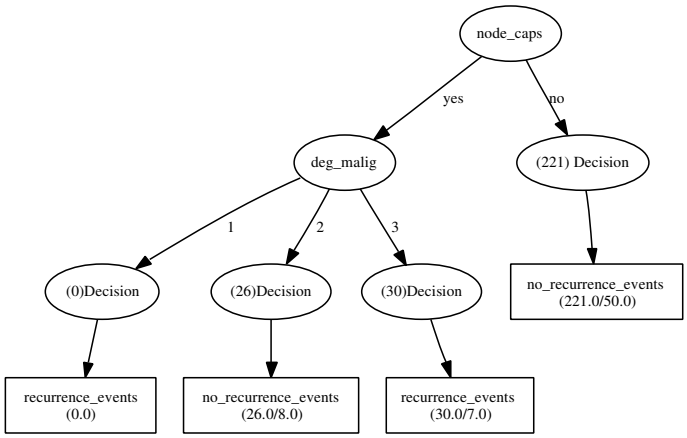

FIG. 5.4 – C4.5 *Breast-cancer*

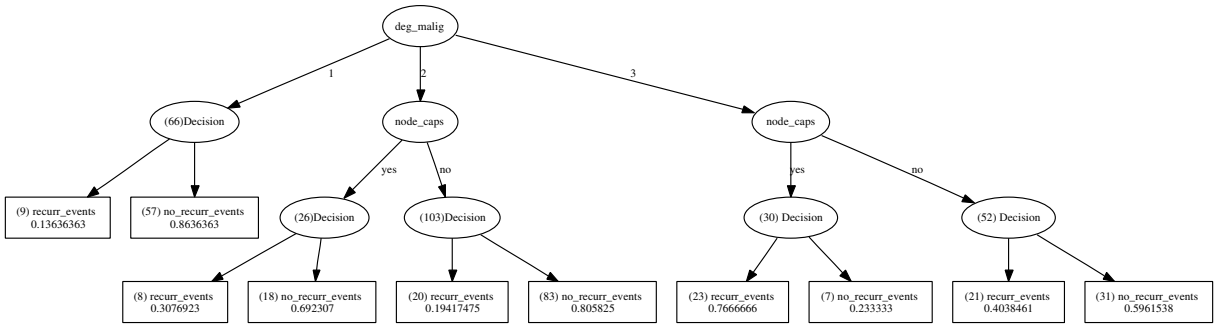

FIG. 5.5 – L'AAP pour *Breast-cancer* pour un seuil 0.04

Breast-cancer	Seuil	bien classés	mal classés	50%
AAPs	**0.02**	**76.08%**	**19.56%**	**4.34%**
	0.03	70.65%	23.91%	5.43 %
	0.04	71.73%	28.26%	
C4.5		**70.65%**	**29.34%**	
AAO		**70.65%**	**29.34%**	

TAB. 5.31 – Résultats de test de *AAP*, *C4.5* et *AAO* sur la base *Breast-cancer*

lymphography	Seuil	bien classés	mal classés	50%
PAT	0.06	91.93%	8.06%	
	0.07	**95.16%**	**4.83%**	
	0.08	83.87%	12.90%	
C4.5		**79.03%**	**19.35%**	**01.61%**
AAO		**79.03%**	**19.35%**	**01.61%**

TAB. 5.32 – Résultats de test de *AAP*, *C4.5* et *AAO* sur la base *Lymphography*

Splice	Seuil	bien classés	mal classés	50%
PAT	**0.03**	**96.36%**	**3.63 %**	
	0.04	94.00%	04.50%	0.90%
C4.5		**93.63%**	**6.36%**	
AAO		**97.27%**	**02.72%**	

TAB. 5.33 – Résultats de test de *AAP*, *C4.5* et *AAO* sur la base *Splice*

La base *Lymphography*

Nous avons testé notre approche sur la base *lymphography* [Newman *et al.*, 1998]. La base d'apprentissage contient 148 objets avec 18 attributs discrets et nominaux. La classe dans cette base prend quatre valeurs : *normal, metastases, malign-lymph, fibrosis*. Nous avons également utilisé une base de test qui possède 62 objets. Le taux de valeurs manquantes dans cette base est : 56% pour l'attribut *block-of-affere*, 30% pour *lym-nodes-dimin*, 40% pour *changes-in-node*, 12% pour *early-uptake-in*, 13% pour *special-forms*, 11% *changes-in-stru*,17% pour *defect-in-node* et 11% pour l'attribut *lym-nodes-enlar*. Les résultats de l'approche *AAP* sont meilleurs que ceux obtenus par C4.5 et *AAO* (tableau 5.4.1). Les matrices de confusion sont présentées dans l'annexe B (*cf.* section B.3)). L'approche *AAP* a bien classé 57 instances parmi les 62 pour un seuil 0.06 et 59 instances quand le seuil est 0.07. C4.5 et *AAO* ont bien classé 49 instances parmi les 62.

La base *Splice*

La base d'apprentissage de *Splice* contient 3190 instances sans valeurs manquantes et 60 attributs nominaux. La classe prend trois valeurs : 1) EI (intron -> exon) ; 2) EI (exon -> intron) ; 3) N (Neither). La base de test possède 110 instances pour lesquelles le taux de valeurs manquantes est 15% pour l'attribut *attribute-17*, 9% pour *attribute-24*, 23% pour *attribute-28*, 17% pour *attribute-29*, 28% pour *attribute-30*, 25% pour *attribute-31*, 12% pour *attribute-32*, 18% pour *attribute-34*, 21% pour *attribute-35* et 18% pour *attribute-49*. Ainsi que 8% pour chacun des attributs *attribute-22*, *attribute-25*, *attribute-36* et *attribute-3*. Les résultats de test donnés dans le tableau 5.33 montrent que l'approche *AAP* est proche de *AAO* et meilleure que C4.5. Les matrices de confusion sont présentées dans l'annexe B (*cf.* section B.4)). Les attributs qui participent à la construction de l'arbre de décision final pour la base *splice* sont : attribute-30, attribute-29, attribute-32, attribute-35, attribute-28, attribute-24, attribute-22, attribute-33, attribute-19, attribute-48 ,attribute-34, attribute-31, attribute-18, attribute-20, attribute-14, attribute-17, attribute-26.

Nursery	Seuil	Bien classés	Mal classés	50%
PAT	**0.001**	**68.25%**	**23.80%**	**07.93%**
	0.02	**67.46%**	**32.53%**	
	0.03	**67.46%**	**32.53%**	
C4.5		**68.25%**	**25.39%**	**06.34%**
AAO		**72.22%**	**27.77%**	

TAB. 5.34 – Résultats de test de *AAP*, *C4.5* et *AAO* sur la base *Nursery*

5.4.2 Bases ayant des attributs indépendants

Dans notre travail, deux attributs sont considérés comme indépendants si leur Information Mutuelle Normalisée est nulle ou inférieure à un seuil choisi ((*cf.* section 5.2)). Nous avons testé notre approche sur deux bases ayant des attributs indépendants : *Nursery* et *Car*.

La base *Nursery*

La base de données *Nursery* est dérivée d'un modèle hiérarchique de décision développé à l'origine pour ranger des demandes d'écoles maternelles. Elle a été employée pendant les années 80, où il y avait un taux de demandes excessif à ces écoles à Ljubljana en Slovénie. Les demandes rejetées ont fréquemment eu besoin d'une explication objective. La décision finale a dépendu de trois sous-problèmes : 1) L'emploi des parents et de la nurse de l'enfant ; 2) La structure de famille et sa situation financière ; 3) L'image sociale et de santé de la famille.
Nous présentons le test de notre approche, C4.5 et *AAO* sur la base *Nursery* [Newman *et al.*, 1998]. La base d'apprentissage contient 12960 instances sans valeurs manquantes et 8 attributs nominaux. La classe prend 5 valeurs : *not-recom, recommend, very-recom, priority, spec-prior*. La base de test contient 126 instances pour lesquelles le taux de valeurs manquantes est 35% pour l'attribut *parents*, 37% pour *has-nur*, 39% pour *health* et 13% pour *form*.

Dans le tableau 5.34, nous remarquons que nos résultats sont proches des ceux obtenus par C4.5. Nos résultats ne s'améliorent pas beaucoup quand nous diminuons le seuil, parce que tous les attributs dans cette base sont indépendants. Donc, chaque arbre d'attribut est un seul noeud-feuille. Pour le seuil 0.001, tous les attributs dans la base d'apprentissage sont pris en compte pour construire l'arbre de décision final[54]. Dans ce cas, nous remarquons que les résultats d'*AAP* et C4.5 sont identiques. Les résultats donnés par *AAO* sont meilleurs parce que cette méthode a bien classé les 6 objets de la classe *very-recom*. Les détails des matrices de confusion sont présentés dans l'annexe B ((*cf.* section B.6)).

La base *Car*

La base d'apprentissage de la base *Car* contient 1728 instances sans valeurs manquantes et 6 attributs nominaux. La classe prend les quatre valeurs suivantes : *unacc, acc, good, vgood*. La base de test possède 236 instances pour lesquelles le taux de valeurs manquantes est : 33% pour l'attribut *buying*, 29% pour *maint*, 16% pour *doors*, 17% pour *persons*, 19% pour *lug-boot* et 11% pour l'attribut *safety*. Les résultats de test sont donnés dans le tableau 5.35. Les résultats de C4.5 sont meilleurs que les notres et ceux de *AAO*. La colonne 50% indique qu'il y a au moins deux valeurs de classe ayant la probabilité maximale.

[54]Parce que l'Information Mutuelle entre chaque attribut et la classe est supérieure à 0.001.

Car	Seuil	Bien classés	Mal classés	50%
PAT	**0.01**	**86.01%**	**05.50%**	**08.47%**
	0.03	86.44%	09.74%	03.81%
	0.04	83.89%	15.25%	00.84%
C4.5		**87.71%**	**07.20%**	05.08%
AAO		**81.77%**	**17.37 %**	0.84%

TAB. 5.35 – Résultats de test de *AAP*, *C4.5* et *AAO* sur la base *Car*

AAO a bien classé toutes les instances de la classe *unacc*. Cependant, elle a classé une seule instance de la classe *vgood* et aucune instance de la classe *good*. C4.5 a bien classé 179 instances de la classe *unacc* parmi 183 et 26 instances de la classe *acc* parmi 31. Les détails des matrices de confusion sont donnés dans l'annexe B ((*cf.* section B.5)).

5.4.3 Bases ayant des attributs continus

L'approche que nous avons proposée est applicable seulement sur des bases de données dont les attributs sont discrets ou qualitatifs. Dans ce cadre, les attributs continus doivent être discrétisés.

Discrétisation des attributs continus

Le but de la discrétisation est de trouver les bonnes bornes (seuils) pour découper les valeurs des attributs continus. Il y a deux types de discrétisation :
1) Discrétisation non-supervisée [Dougherty *et al.*, 1995, Risvik, 1997] : ne prend pas en compte la corrélation entre les attributs et la classe. Exemple de méthodes : Equal with Discretization, Equal frequency Discretization, etc.
2) Discrétisation supervisée : prend en compte les corrélations entre les attributs et la classe. Exemples : 1R algorithlm [Holte, 1993], ChiMerge [Kerber, 1992], Entropy-based discretization, Naive algorithm [Dougherty *et al.*, 1995, Risvik, 1997], etc.

Nous avons utilisé la méthode *Entropy-based discretization* avec MDLP [55] implémentée en Weka pour discrétiser les attributs continus [Fayyad et Irani, 1992]. Cette méthode utilise l'entropie comme critère de découpage et le principe de MDL comme critère d'arrêt. Pour découper les valeurs d'un attribut continu en intervalles, on calcule l'entropie de la classe en utilisant chaque seuil (borne) possible pour cet attribut et on choisit comme discrétisation binaire le seuil qui donne l'entropie minimale. Le processus est répété récursivement sur les partitions obtenues jusqu'à ce qu'un certain critère basé sur le MDLP soit vérifié.

Nous avons discrétisé les attributs continus dans seulement deux bases : *Iris* et *Breast-w*.

La base *Iris*

La base d'apprentissage *Iris* contient 150 instances et 4 attributs continus. La classe de cette base prend 3 valeurs : *Iris Setosa, Iris Versicolor, Iris Virginica*. L'objectif de cette base est de classer les fleurs d'iris. La base de test utilisée contient 51 instances. Le taux de valeurs manquantes est 31% pour l'attribut *sepallength*, 27% pour *sepalwidth*, 27% pour *petallength* et

[55] Le principe MDL (Minimum Description Length) consiste à choisir parmi plusieurs théories (hypothèses) celle dont le codage (théorie plus exceptions) est minimal.

Attribut	Valeurs discrètes
sepallength	(-inf-5.55],(5.55-6.15],(6.15-inf)
sepalwidth	(-inf-2.95],(2.95-3.35],(3.35-inf)
petallength	(-inf-2.45],(2.45-4.75],(4.75-inf)
petalwidth	(-inf-0.8],(0.8-1.75],(1.75-inf)

TAB. 5.36 – Les valeurs discrètes pour les attributs de la base *Iris*

Car	Seuil	Bien classés	Mal classés	50%
PAT	**0.2**	**96.07%**	**03.92%**	
	0.3	**96.07%**	**03.92%**	
	0.4	**96.07%**	**03.92%**	
C4.5		**43.13%**	**01.96%**	**54.90%**
AAO		**96.07%**	**03.92 %**	

TAB. 5.37 – Résultats de test de *AAP*, *C4.5* et *AAO* sur la base *Iris*

54% pour *petalwidth*. Les résultats de test donnés par notre approche, C4.5 et *AAO* sont présentés dans le tableau 5.37. Nous remarquons que nos résultats et les résultats de *AAO* sont équivalents. Par contre, les résultats donnés par C4.5 sont médiocres parce que l'arbre de décision final de la base d'apprentissage *Iris* construit selon C4.5 est composé de deux attributs : sa racine qui est l'attribut *petalwidth*, et la classe. Dans ce cas, chaque objet ayant l'attribut *petalwidth* est classé *Iris-Setosa* avec une probabilité 0.33, *Iris-Versicolor* avec une probabilité 0.36 et *Iris-Virginica* avec la probabilité 0.31, parce que ces probabilités ne prennent pas en compte que la fréquence de l'attribut *petalwidth*. L'arbre de décision final construit selon *AAP* pour cette base contient les attributs : *petalwidth*, *petallength* et *sepallength*.
Les valeurs discrètes de ces attributs sont données dans le tableau 5.36.

Les détails des matrices de confusion sont présentés dans l'annexe B ((*cf.* section B.7)). Nous avons appliqué C4.5 sur la base *Iris* discrétisée selon la méthode *Entropy-based discretization*.

La base *Breast-w*

La base d'apprentissage contient 683 instances et 9 attributs continus. La classe de cette base prend deux valeurs : *benign, malignant*. La base de test possède 88 instances dont le taux de valeurs manquantes est : 16% pour l'attribut *Clump-Thickness*, 46% pour *Cell-Size-Uniformity*, 27% pour *Cell-Shape-Uniformity*, 11% pour *Marginal-Adhesion* , 14% pour *Single-Epi-Cell-Size*, 60% pour *Bare-Nuclei*, 14% pour *Bland-Chromatin*, 21% pour *Normal-Nucleoli*. Le résultats de test sont donnés dans le tableau 5.39. Les valeurs discrètes de ces attributs sont données dans le tableau 5.38. Les détails des matrices de confusion sont présentés dans l'annexe B (*cf.* section B.8).

Root Mean Squared Error (RMSE) : Erreur Quadratique Moyenne

Comme nous l'avons vu précédemment le résultat de classement selon notre approche est une distribution de probabilités de classe au lieu de la valeur de classe la plus probable. Ces résultats sont sous forme numérique. Pour comparer les exactitudes de deux estimateurs de probabilités,

Attribut	Valeurs discrètes
Clump-Thickness	(-inf-4.5],(4.5-6.5],(6.5-inf)
Cell-Size-Uniformity	(-inf-1.5],(1.5-2.5],(2.5-4.5],(4.5-inf)
Cell-Shape-Uniformity	(-inf-1.5],(1.5-2.5],(2.5-4.5],(4.5-inf)
Marginal-Adhesion	(-inf-1.5],(1.5-3.5],(3.5-inf)
Single-Epi-Cell-Size	(-inf-2.5],(2.5-3.5],(3.5-inf)
Bare-Nuclei	(-inf-1.5],(1.5-2.5],(2.5-5.5],(5.5-inf)
Bland-Chromatin	(-inf-2.5],(2.5-3.5],(3.5-inf)
Normal-Nucleoli	(-inf-2.5],(2.5-9.5],(9.5-inf)
Mitoses	(-inf-1.5],(1.5-inf)

TAB. 5.38 – Les valeurs discrètes pour les attributs de la base *Breast-w*

Car	Seuil	Bien classés	Mal classés	50%
	0.2	**93.18%**	**06.81%**	
PAT	**0.3**	**88.63 %**	**11.36%**	
C4.5		**75%**	**25%**	
AAO		**90.90%**	**09.09%**	

TAB. 5.39 – Résultats de test de *AAP*, *C4.5* et *AAO* sur la base *Breast-w*

nous avons utilisé le *Root mean squared error* [Breiman *et al.*, 1984, Witten Ian et Frank, 2005].

Pour une instance x le *RMES* est donné par l'équation suivante :

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{j=n}(t(j|x) - P(j|x))^2} \tag{5.8}$$

où x est l'instance ; j est la valeur de classe ; $t(j|x)$ est la vraie probabilité de la classe j pour x et $P(j|x)$ est la probabilité estimée par la méthode pour l'instance x et la classe j. Dans la base de test, les vraies classes sont connues, mais pas leurs probabilités. Donc, $t(j|x)$ est défini pour valoir 1 si la classe de l'instance x est j et 0 autrement.
Le RMSE de l'ensemble de la base de test est la racine carrée de *la somme des RMSE de toutes les instances divisée par la taille de la base.*
Pour chaque base de test utilisée, le tableau 5.40 présente les *RMSE* des méthodes *AAP, C4.5 et AAO. RMSE* est une mesure d'erreur. Donc, plus il est petit, plus la méthode est meilleure. Nous remarquons à partir du tableau 5.40 que les RMSE des méthodes C4.5 et *AAO* sont plus grands que celui de la méthode *AAP*.

5.5 Conclusion

Dans ce chapitre, nous avons testé notre approche sur plusieurs bases d'apprentissage. Nous avons partitionné ces bases en trois catégories : 1) des bases ayant des attributs dépendants ; 2) des bases ayant des attributs indépendants ; 3) des bases ayant des attributs continus. Les

DataBase	AAP	C4.5	AAO
Zoo	**0.133817**	0.44812	0.245
Mushroom	**0.412543**	0.643147	0.535865
Dermatology	**0.25183339**	0.4108025	0.3068860
Vote	**0.310443**	0.52039	0.315079
Breast-cancer	**0.525338**	0.632477	0.6238318
Lymphography	**0.260477**	0.477835	0.420603
Splice	0.35292	0.38507	**0.22929**
Nursery	0.4456728	0.44999	**0.436149**
Car	0.3292286	**0.32743**	0.35666
Iris	0.290535	0.534436	**0.226616**
Breast-w	**0.29610**	0.55673	0.32818

Tab. 5.40 – The Root Mean Squared Error (L'Erreur Quadratique Moyenne)

attributs continus sont discrétisés pour pouvoir appliquer notre approche.

Nous avons comparé les résultats de classement donnés par notre approche avec ceux donnés par C4.5 et *AAO*. Les matrices de confusion sont également calculées pour chaque méthode sur chaque base de test. Nous avons montré que notre approche est meilleure que C4.5 et *AAO*.

Nous avons remarqué que la performance de notre approche est meilleure quand les attributs sont dépendants. En cas d'indépendance entre les attributs, les résultats de notre approche ne s'améliorent pas en diminuant le seuil. De plus, il n'y a pas de différence significative entre la performance de notre approche et celle de C4.5.
Nous avons utilisé la matrice de confusion pour détailler les résultats de classement ; à partir de ces matrices nous avons comparé les résultats donnés par chaque méthode pour chaque valeur de classe.

Nous avons vu que la performance de notre approche et la performance de *AAO* sont proches pour certaines bases. Cependant, dans d'autres cas, nos résultats sont meilleurs que ceux donnés par *AAO*, parce que le fait d'utiliser la dépendance entre les attributs pour prédire la probabilité d'un attribut inconnu donne un résultat plus précis que la méthode *AAO*, qui remplace un attribut manquant par sa valeur la plus probable.

En utilisant C4.5, nous avons constaté que le fait de calculer la probabilité d'un attribut manquant A, qui est la racine de l'arbre de décision et qui est le plus dépendant de la décision, en prenant en compte seulement sa fréquence dans la base d'apprentissage conduit à un mauvais classement.

En utilisant *AAO*, nous avons remarqué que le fait de calculer la probabilité d'un attribut manquant A en prenant en compte seulement les attributs qui ont des IM inférieures à celle de A et en ignorant les autres attributs qui dépendent de A conduit également à un mauvais classement.

Nous avons également remarqué que notre approche donne une erreur quadratique moyenne (RMSE) plus petite que celles des autres méthodes lorsque les bases d'apprentissage possèdent des attributs dépendants.

Dans les deux derniers chapitres, nous allons mesurer la qualité de nos résultats de classement

et nous allons montrer l'importance de fournir un résultat probabiliste de classement. Nous calculons la complexité de notre algorithme de classement ainsi que celle de C4.5 et *AAO*.

Chapitre 6

Comparaison et analyse des résultats de classement

6.1 Introduction

Nous avons proposé une approche qui construit pour chaque attribut dans la base d'apprentissage un arbre de décision probabiliste en utilisant les attributs dont il dépend. Ainsi, le résultat du classement d'un objet incomplet est probabiliste. Dans le chapitre précédent, nous avons détaillé pour chaque base de test les matrices de confusion qui nous ont aidés à analyser les résultats de chaque classe.

L'objectif de ce chapitre est d'analyser le résultat de classement de chaque objet dans la base de test. Nous calculons la fréquence de chaque objet de la base de test dans la base d'apprentissage. Pour calculer cette fréquence, nous devons chercher les autres objets qui sont proches de l'objet en question. C'est-à-dire les objets ayant un nombre maximum de mêmes valeurs d'attributs que l'objet en question. Pour cela, nous devons calculer la distance entre l'objet en question et chaque objet dans la base d'apprentissage. Ensuite, nous calculons la fréquence de cet objet et nous la comparons avec les résultats de classement donnés par notre approche ainsi que par C4.5 et *AAO*. Cette idée, basée sur la notion de distance conduit à considérer la méthode des k plus proches voisins.

Dans ce cadre et pour trouver la fonction de distance appropriée, nous nous sommes intéressés à un algorithme appelé *Relief*, proposé par [Kira et Rendell, 1992], car il est basé sur l'analyse statistique pour sélectionner les attributs pertinents dans la base d'apprentissage. *Relief* est une mesure d'impureté ; il est utilisé comme un pré-processeur pour éliminer les attributs non-pertinents dans la base d'apprentissage avant la phase d'apprentissage [Kononenko et Simec, 1995].

Contrairement aux mesures classiques[56] qui sont utilisées lors de la construction d'un arbre de décision pour évaluer la qualité d'un attribut relativement à la classe et qui ne prennent pas en compte les dépendances éventuelles entre les attributs utilisés pour prévoir le concept final (la classe), *Relief* et ses extensions peuvent effectivement prendre en compte les dépendances entre les attributs lors de l'évaluation de la qualité d'un attribut relativement à la classe.

Nous nous sommes inspirés de *Relief*[57] et ses extensions pour calculer la *Distance* entre deux

[56] Comme l'information mutuelle, l'index de Gini, le gain de ratio, etc. On les appelle mesures "myopic".

[57] Dans ce chapitre, nous n'utilisons pas *Relief* comme mesure d'impureté.

instances. En nous appuyant sur la fonction de *Distance*, nous pouvons calculer, pour chaque instance à classer, la fréquence de ses instances voisines les plus proches de chaque classe. Cette fréquence sera ensuite comparée avec la distribution de probabilités obtenue en utilisant les arbres *AAPs* et *AAOPs* pour classer la même instance.

Dans ce chapitre, nous commençons par expliquer la méthode des k plus proches voisins. Ensuite, nous expliquons *Relief* et ses extensions en détail. Nous présentons également la fonction de *Distance* que nous utilisons pour calculer la distance entre chaque couple d'instances. Ensuite, nous décrivons l'algorithme que nous avons proposé pour calculer pour chaque instance dans la base de test la fréquence de ses instances voisines les plus proches de chaque classe. Finalement, nous montrons les résultats de test de cet algorithme sur des bases réelles et nous comparons ces résultats avec les résultats de classement donnés par *AAP*, C4.5 et *AAO* pour les mêmes bases.

6.2 La méthode des K plus proches voisins (K nearest neighbor)

La méthode de k plus proches voisins [Cover et Hart, 1967] est une méthode de l'apprentissage supervisé de type apprentissage à base d'instances (instance-based learning). Son principe est pour une nouvelle instance I de prédire les k objets dans la base d'apprentissage qui sont les plus semblables à I. Cela met en jeu la notion de distance ou similarité entre deux objets. Cet algorithme utilise les objets trouvés pour classer l'instance I en lui attribuant la classe la plus probable.

On peut résumer l'algorithme comme suit :

```
Début
   On cherche à classer l'instance  I
     pour chaque objet J de l'ensemble d'apprentissage faire
       calculer la distance D(J,I) entre J et  I
     fin pour
  Dans les k objets les plus proches de  I
      calculer le nombre d'occurrences de chaque classe
  Attribuer à  I la classe la plus probable
fin
```

Les propriétés de la distance sont :

- La fonction de distance n'est pas négative et D(A,A) = 0 ;
- D(A,B) = D(B,A) ;
- $D(A,B) \geq D(A,C) + D(B,C)$.

Plusieurs fonctions de distance sont proposées, on trouve :

- La distance euclidienne : soit X= $(x_1, x_2, .., x_n)$ et Y= $(y_1, y_2, ..., y_n)$ deux instances, la distance euclidienne entre X et Y est :

$$D(X,Y) = \sqrt{\sum_{i=1}^{j=n} (x_i - y_i)^2} \tag{6.1}$$

- La distance de Manhattan :

$$D_{manh}(X,Y) = \sqrt{\sum_{i=1}^{j=n} |x_i - y_i|} \tag{6.2}$$

Le cas le plus simple est k=1 (1-ppv), où on cherche l'objet le plus proche de l'instance *I*.
Quand k=n [58], la classe retenue sera la classe la plus probable dans toute la base d'apprentissage, indépendamment de l'instance *I*.

6.3 Relief

L'évaluation de la qualité d'un attribut est une des tâches les plus difficiles dans l'apprentissage automatique. Plusieurs mesures ont été développées pour estimer la qualité des attributs[59] pour prévoir la décision[60]. Par exemple, pour un attribut discret ou qualitatif, on peut utiliser le gain d'information, le gain ratio, l'index de Gini, Relief, ReliefF, MDL, etc.
Nous nous sommes intéressés à *Relief* parce que d'une part, il est basé entièrement sur l'analyse statistique pour sélectionner les attributs pertinents et il utilise peu d'heuristiques, et d'autre part il calcule la distance entre les instances ; cette distance est calculée dans l'espace des attributs. Selon [Kononenko, 1994], cet algorithme s'est montré d'une grande efficacité pour l'estimation des attributs. *Relief* prend en compte la différence entre les valeurs d'un attribut dans deux instances, la différence entre les valeurs de classe ainsi que la distance entre les instances.

L'idée de l'algorithme *Relief* est d'estimer les attributs selon la manière dont ses valeurs se distinguent parmi les instances qui sont proches l'une de l'autre. Pour cela, Relief cherche pour une instance donnée R ses deux instances les plus proches (nearest neighbours), l'une H qui appartient à sa classe et l'autre M qui appartient à une classe différente[61].
L'algorithme *Relief* choisit aléatoirement m instances à partir de l'ensemble d'apprentissage (m est défini par l'utilisateur). Cet algorithme est défini comme suit [Kira et Rendell, 1992] :

```
set all weights W[A]:=0.0;
for i:=1 to n do
   begin
     randomly select an instance R;
     find nearest hit H and nearest miss M;
     for A := 1 to all-attributes do
        W[A]= W[A] - diff(A,R,H)/m + diff(A,R,M)/m;
  end;
```

Relief utilise une fonction *diff* pour calculer la différence entre les valeurs d'un attribut dans les deux instances. Pour un attribut discret ou symbolique, cette différence est 1 quand les valeurs sont différentes et 0 quand elles sont égales. Le poid *W[A]* est l'estimation de la qualité d'un attribut A.

À partir de l'algorithme présenté ci-dessus, on peut remarquer que :

- Si la valeur d'un attribut A est différente dans les deux instances R et M qui appartiennent à des classes différentes, on ajoute $\frac{1}{m}$ au poids de cet attribut.
- Si la valeur d'un attribut A est la même dans R et M appartenant à des classes différentes, on n'ajoute rien au poids de cet attribut.

[58] n est la taille de l'ensemble d'apprentissage

[59] Le problème de la sélection des attributs est de choisir le plus petit sous-ensemble des attributs qui sont nécessaires et suffisants pour décrire le concept final.

[60] Dans notre vocabulaire, la décision est l'attribut classe dans les arbres de décision.

[61] Par exemple : si la classe prend deux valeurs C_1 et C_2, et si l'instance H appartient à la classe C_1 alors l'instance M doit appartenir à la classe C_2.

- Si la valeur d'un attribut A est différente dans les deux instances R et H appartenant à la même classe, on soustrait $\frac{1}{m}$ du poids de cet attribut.
- Si la valeurs d'un attribut A est la même dans R et H appartenant à la même classe, on ne soustrait rien au poids de cet attribut.

La fonction de différence utilisée par *Relief* est donnée dans l'équation 6.3. Elle calcule la différence entre chaque couple d'instances sans valeurs manquantes et seulement pour les attributs discrets ou symboliques.

$$diff(A, I_1, I_2) = \begin{cases} 0 & \texttt{if} \quad V^{(A,I_1)} = V^{(A,I_2)} \\ 1 & \texttt{if} \quad V^{(A,I_1)} \neq V^{(A,I_2)} \end{cases} \tag{6.3}$$

En conséquence, la qualité d'un attribut augmente s'il prend deux valeurs différentes dans les deux instances appartenant à des classes différentes et s'il prend la même valeur dans les deux instances qui appartiennent à la même classe. En revanche, la qualité d'un attribut diminue s'il prend deux valeurs différentes dans les instances qui appartiennent à la même classe et s'il prend la même valeur dans les instances qui appartiennent à des classes différentes.

À partir de l'algorithme *relief* présenté ci-dessus, *Relief* calcule le poids moyen de chaque attribut (le niveau de pertinence)W[j] où $-1 \leq W[j] \leq 1$. Il choisit ceux ayant un niveau de pertinence supérieur à un seuil t ($W[j] \geq t$). D'après [Kira et Rendell, 1992] le niveau de pertinence (le poids moyen d'un attribut) est positif si l'attribut est pertinent, et il est négatif ou proche de zéro si l'attribut est non-pertinent. Le seuil t est déterminé en utilisant une méthode statistique pour l'estimation de l'intervalle.

Nous considérons que deux objets sont des proches voisins si la distance entre eux est inférieure à un seuil *near*. La fonction de Distance utilisée est la *Distance de Manhattan* :

$$Distance(I_1, I_2) = \sum_{j=1}^{j=n} diff(A_j, I_1, I_2)^{62} \tag{6.4}$$

Nous pouvons également utiliser la *Distance euclidienne* :

$$Distance(I_1, I_2) = \sqrt{\sum_{j=1}^{j=n} diff(A_j, I_1, I_2)^2} \tag{6.5}$$

D'après [Robnik-Sikonja et Kononenko, 2003] et pendant l'utilisation de l'algorithme *Relief*, aucune différence significative n'a été trouvée pendant les évaluations en utilisant ces deux distances.

Dans notre expérimentation, nous allons utiliser la *Distance de Manhattan*. Par exemple, si la distance entre deux instances est 5, cela signifie qu'il y a 5 attributs dont les valeurs sont différentes dans les deux instances.

Relief traite les attributs discrets et continus. Cependant, il ne peut pas traiter les données ayant des valeurs manquantes, ni le problème de classes multiples. Pour cela, [Kononenko, 1994] a proposé une extension de *Relief* appelé *ReliefF* qui prend en compte ces problèmes [Kononenko, 1994], [Robnik-Sikonja et Kononenko, 1999].

[62] `n est le nombre d'attributs.`

6.3.1 ReliefF

L'algorithme *ReliefF* est une extension de *Relief* proposé par [Kononenko, 1994, Kononenko et Robnik-Sikonia, 1996]. Il étend l'algorithme original sur plusieurs points : 1) il n'est pas limité au problème de deux classes ;
2) il traite le problème des valeurs manquantes.

Le choix de l'instance la plus proche est d'importance cruciale dans *Relief*. L'objectif est de trouver les instances les plus proches en respectant les attributs pertinents. Pour augmenter la fiabilité de l'approximation de probabilité, *ReliefF* cherche les K instances les plus proches de l'instance à traiter de chaque classe au lieu d'une seule instance la plus proche de chaque classe. k est un paramètre défini par l'utilisateur ; l'expérimentation faite par [Kononenko, 1994] montre que quand K vaut 10, les résultats sont satisfaits empiriquement.

ReliefF met à jour les estimations de qualité des attributs de la manière suivante :

$$W[A] = W[A] - \sum_{j=1}^{k} \frac{diff(A, R_i, H_j)}{(m.k)} + \sum_{C \neq class(R_i)} \frac{[\frac{P(C)}{1-P(class(R_i))} \sum_{j=1}^{k} diff(A, R_i, M_j(C))]}{(m.k)} \quad (6.6)$$

Pour l'instance R_i, il cherche les k instances H_j les plus proches, au lieu d'une seule instance. Il prend également en compte la probabilité de chaque classe P(C), estimée à partir de la base d'apprentissage, où $C \neq class(R_i)$[63].

ReliefF a étendu la fonction *diff(A, $Instance_1$, $Instance_2$)* pour prendre en compte un attribut ayant des valeurs manquantes lors de l'estimation de sa qualité. Pour un attribut A, cette fonction devient :

$$diff(A, I_1, I_2) = \begin{cases} 0 & \texttt{if} \quad V^{(A,I_1)} = V^{(A,I_2)} \\ 1 & \texttt{if} \quad V^{(A,I_1)} \neq V^{(A,I_2)} \\ 1 - P(V^{(A,I_2)}/Class_{I_1}) & \texttt{if} \quad \texttt{A is unknown in } I_1 \end{cases} \quad (6.7)$$

où :

- $V^{(A,I_j)}$ est la valeur de A dans l'instance I_j.
- $Class_{I_1}$ est la valeur de la classe dans l'instance I_1.
- $1 - P(V^{(A,I_2)}/Class_{I_1})$ est la probabilité que deux instances I_1 et I_2 prennent des valeurs différentes pour l'attribut A dans le cas où une de ces instances (I_1 ici) possède une valeur inconnue pour A.

Nous notons que cette fonction calcule également la probabilité que deux instances I_1 et I_2 prennent deux valeurs différentes pour l'attribut A dans le cas où cet attribut est inconnu dans I_1 et I_2. Nous ne l'avons pas expliqué dans l'équation 6.7, mais nous pouvons trouver ce calcul dans [Kononenko, 1994], [Robnik-Sikonja et Kononenko, 1999].

6.4 Calculer la distance entre les instances

Notre objectif est d'analyser les résultats de classement donnés par notre approche. Pour cela, pour chaque instance test, nous voulons savoir la proportion de ses instances les plus proches dans la base d'apprentissage.

Dans ce cadre, nous nous sommes inspirés de *Relief* et *ReliefF* pour calculer la distance entre deux instances en utilisant la fonction donnée dans l'équation 6.7. La première instance

[63] La partie $\frac{P(C)}{1-P(class(R_i))}$ est égale à 1 si la classe prend seulement deux valeurs.

est obtenue à partir de la base de test et elle contient des valeurs manquantes. L'autre instance est tirée de la base d'apprentissage sans valeurs manquantes. La distance totale entre ces deux instances est simplement la somme des différences entre les valeurs des attributs dans les deux instances pour tous les attributs [Kononenko, 1994].

La fonction de distance utilisée est la *Distance de Manhattan*, présentée dans l'équation 6.4. Par exemple, si la distance entre deux instances est 5, cela signifie qu'il y a 5 attributs dont les valeurs sont différentes dans les deux instances.

Dans notre expérimentation, nous comparons chaque instance de la base de test avec toutes les instances dans la base d'apprentissage en calculant la distance entre elles. Ensuite, nous calculons pour chaque instance à classer la fréquence de ses instances les plus proches de chaque classe. Ces fréquences seront ensuite comparées avec les résultats de classement donnés par notre approche, C4.5 et *AAO*. Pour cela, nous proposons l'algorithme suivant [Hawarah *et al.*, 2006d, Hawarah *et al.*, 2006b].

6.4.1 Algorithme : Analyser-Instance

Cet algorithme prend comme entrée l'instance *Inst* de la base de test, ainsi que la base d'apprentissage complète. Il calcule la distance entre l'instance *Inst* et chaque instance *I[k]* de la base d'apprentissage. Si cette distance est inférieure à une constante *near*, qui est fixé par l'utilisateur et qui dépend de nombre d'attributs dans la base d'apprentissage[64], on considère que l'instance *I[k]* est proche de l'instance à étudier *Inst*. Ensuite, on vérifie la classe à laquelle l'instance *I[k]* appartient. Ce processus est répété m fois[65]. Finalement, cet algorithme retourne la fréquence des instances qui sont proches de l'instance *Inst* de chaque classe.

[64] Le seuil *near* ne peut pas dépasser le nombre maximum d'attributs dans la base d'apprentissage.
[65] m est la taille de la base d'apprentissage.

```
Input: test instance Inst,  m training instances I;
Output: for the Inst: frequency of nearest instances from the same class
        and frequency of nearest instances from the different class;

Function Instance-Analysis(Inst:test instance,
                  I:array[1..m] of instances):Pc:array[1..2] of real;
Const
  near=5;
var
 nbSCL, nbDCL, k, near: integer;
                   dis: real;
 begin
  nbSCL=0, nbDCL=0;
  For k:=1  to m  do
      begin
       dis= Distance(Inst,I[k])
       If dis < near   {the two instances are nearest neighbor}
       then
         If (two instances Inst and I[k] are from the same Class)
           then nbSCL++
           else nbDCL++;
      end; (*for k*)
 Pc1 = P(nearest instances from the same class)
     = nbSCL/(nbDCL+nbSCL)
 Pc2 = P(nearest instances from the different class)
     = nbDCL/(nbDCL+nbSCL)
 end;
return(Pc);
```

Nous allons comparer cette fréquence avec celle donnée par notre approche, C4.5 et AAO pour classer la même instance. Nous présentons quelques résultats dans le paragraphe suivant.

6.4.2 Résultats

Dans cette section, nous allons illustrer notre expérience en utilisant l'algorithme *Analyser-Instance* proposé ci-dessus. Nous testons cet algorithme sur plusieurs bases d'apprentissage, et nous comparons les résultats avec ceux donnés par la méthode AAP ,C4.5 et AAO :

La base *Vote*

Nous commençons par présenter le résultat de test de cet algorithme sur les exemples donnés dans le tableau 6.3 de la base *Vote*. Cette base contient 16 attributs. Nous considérons que deux instances sont proches si la distance entre elles est inférieure à une constante *near* qui peut être ici 5, 8, 9, 10, 11 ou 12. Le tableau 6.2 contient le résultat de test de cet algorithme quand *near* vaut 8, 10 et 12.

Dans le tableau 6.1, nous avons le nombre d'instances que nous avons trouvées pour chaque instance à étudier dans le tableau 6.3. Donc, 101(12, 89) signifie que nous avons trouvé dans la base d'apprentissage de *Vote* 101 instances qui sont proches de l'instance numéro 1 (tableau 6.3) dont 12 instances de la classe *Democrat* et 89 de la classe *Republican*. La vraie classe de cette

near=5	near =8	near=10	near=12
101(12, 89)	125(21,104)	148(43, 105)	175(67, 108)
80(78,2)	114(104,10)	131(111,20)	168(118,50)
24(32,1)	116(107,9)	159 (120,39)	216(124,92)

TAB. 6.1 – Détails des résultats de test de l'algorithme *Analyser-Instance* sur 3 instances de la base *Vote*

physician-fee-freeze	el-salvador-aid	education	crime	near=8	near=10	near=12
?	y	y	y	(16%,83%)	(29%,70%)	(38%,61%)
?	?	n	n	(91%,08%)	(84%,15%)	(70%,29%)
?	y	n	n	(92%,07%)	(75%,24%)	(57%,42%)

TAB. 6.2 – Résultats de test de l'algorithme *Analyser-Instance* sur 3 instances de la base *Vote*

instance dans la base de test est *Republican*.

En comparant le tableau 6.3, qui contient les résultats du classement de ces trois instances en utilisant notre approche *AAP*, *C4.5* et *AAO*, avec le tableau 6.2, qui contient le résultat donné par l'algorithme *Analyser-Instance* pour les mêmes instances, nous remarquons que nos résultats sont proches des résultats de l'algorithme *Analyser-Instance* quand *near* vaut 8. De plus, ils sont meilleurs que ceux donnés par C4.5 quand *near* vaut 8, 10 ou 12.

Nous notons que dans la base *Vote*, le seul attribut utilisé pour construire l'arbre de décision final selon la méthode C4.5 est l'attribut *physician-fee-freeze*, qui a la plus grande influence sur la décision. Cependant, si cet attribut est manquant, C4.5 calcule sa fréquence dans la base d'apprentissage sans prendre en compte les autres attributs dont il dépend. En conséquence, dans la base de test, chaque objet pour lequel la valeur de l'attribut *physician-fee-freeze* est inconnue est classé *Democrat* avec la probabilité 0.53 et *Republican* avec la probabilité 0.47.

Dans notre méthode et contrairement à C4.5, chaque objet dont l'attribut *physician-fee-freeze* est manquant est classé selon les attributs dont il dépend. Pour le seuil 0.5, notre *AAP* correspondant à la base d'apprentissage complète est construit en utilisant également l'attribut *physician-fee-freeze*. L'*AAP* de cet attribut est construit en utilisant l'attribut *el-salvador-aid*. En conséquence, quand *physician-fee-freeze* est inconnu, nous calculons sa probabilité sachant *el-salvador-aid*.

De la même manière et pour le seuil 0.4, notre *AAP* correspondant à toute la base d'apprentissage est construit en utilisant les attributs *physician-fee-freeze*, *el-salvador-aid* et *education-spending*. De plus, L'*AAP* de *physician-fee-freeze* est construit en utilisant les attributs *el-salvador-aid*, *education-spending* et *crime*. Dans ce cas, quand *physician-fee-freeze* est inconnu, nous calculons sa probabilité en fonction de ses attributs dépendants. Par exemple, dans le tableau 6.3, nous remarquons que la distribution de probabilités donnée par notre approche pour chaque objet varie selon les valeurs des attributs de l'objet. En revanche, avec la méthode C4.5, cette distribution dépend seulement de l'attribut *physician-fee-freeze*.

physician-fee-freeze	el-salvador-aid	education	crime	AAP	C4.5	AAO
?	y	y	y	(11%,89%)	(53%,47%)	(0%,100%)
?	?	n	n	(99%,01%)	(53%,47%)	(100%,0%)
?	y	n	n	(92%,07%)	(53%,47%)	(100%,0%)

TAB. 6.3 – Résultats de test de *AAP* et *C4.5* sur 3 instances de la base *Vote*

Attributs	Instance1	Instance2	Instance3	Instance4	Instane5	Instance6
lymphatics	deformed	arched	deformed	deformed	deformed	deformed
block-of-affere	?	?	?	yes	?	?
bl-of-lymph-c	no	no	?	no	no	yes
bl-of-lymph-s	no	no	?	no	no	yes
by-pass	yes	no	?	yes	?	yes
extravasates	yes	no	no	yes	yes	yes
regeneration-of	no	no	no	no	no	yes
early-uptake-in	yes	yes	no	yes	?yes	?
lym-nodes-dimin	1	1	1	1	1	3
lym-nodes-enlar	2	2	2	2	3	1
changes-in-lym	oval	oval	oval	oval	round	?
defect-in-node	?	?	lacunar	lac-central	?	lacunar
changes-in-node	?	?	lacunar	lac-margin	lac-margin	?
changes-in-stru	diluted	?	diluted	diluted	coarse	faint
special-forms	chalices	vesicles	no	?	chalices	no
dislocation-of	yes	no	no	yes	yes	yes
exclusion-of-no	yes	yes	yes	yes	yes	yes
no-of-nodes-in	4	1	2	4	2	4
class	malign-lymph	metastases	malign-lymph	malign-lymph	metastases	fibrosis

TAB. 6.4 – Les instances de la base *Lymphography*

La base *Lymphography*

De la même manière, nous allons présenter quelques résultats donnés par l'algorithme *Analyser-Instance* sur la base *Lymphography* et nous allons également les comparer avec ceux donnés par *AAP*, *AAO* et C4.5.

Les instances de la base *Lymphography* sont données dans le tableau 6.4. La classe dans cette base prend 4 valeurs : *(normal, metastases, malign-lymph, fibrosis)*. Par exemple, si le résultat de classement d'une instance est : $(1\%, 39\%, 59\%, 0)$, cela signifie que la probabilité que cette instance appartient à la classe *normal* est 0.01. La probabilité qu'elle appartienne à la classe *metastases* est 0.39 et à la classe *malign-lymph* est 0.59. La probabilité qu'elle appartienne à *fibrosis* est 0.

Le tableau 6.5 contient les résultats de classement obtenus par *AAP*, C4.5 et *AAO* sur les 6 instances données dans le tableau 6.4 de la base *Lymphography*.

La base *Lymphgraphy* contient 18 attributs, pour lesquels nous allons tester l'algorithme *Analyser-Instance* quand *near* est 3, 4, 5, 7, 9, 12, 14. Les résultats sont présentés dans les tableaux 6.6 et 6.7.

ID	AAP	C4.5	AAO
1	(0, 0, 100%, 0)	(1%, 39%, 59%, 0)	(0, 100%, 0, 0)
2	(0, 57%, 43%, 0)	(1%, 44%, 54%, 0)	(0, 0, 100%, 0)
3	(0, 0, 100%, 0)	(0, 67%, 33%, 0)	(0, 100%, 0, 0)
4	(0, 0, 100%, 0)	(0, 95%, 5%, 0)	(0, 100%, 0, 0)
5	(0, 82%, 18%, 0%)	(0, 71%, 29%, 0%)	(0, 100%, 0, 0%)
6	(0, 0, 0, 100%)	(0, 0, 0, 100%)	(0, 0, 0, 100%)

TAB. 6.5 – Les résultats de AAP, C4.5 et AAO sur des instances de la base *Lymphography*

ID	near=3	near=4	near=5
1	(0, 0, 100%, 0)	(0, 0, 100%, 0)	(0, 0, 100%, 0)
2	(0, 100%, 0%, 0)	(0%, 86%, 14%, 0)	(0, 81%, 19%, 0)
3	(0, 0, 100%, 0)	(0, 0, 100%, 0)	(0, 75%, 25%, 0)
4	(0, 0, 100%, 0)	(0, 50%, 50%, 0)	(0, 75%, 25%, 0)
5	(0, 100%, 0, 0)	(0, 100%, 0, 0)	(0, 89%, 11%, 0%)
6	(0, 0, 0, 100%)	(0, 0, 0, 100%)	(0, 0, 0, 100%)

TAB. 6.6 – Les résultats de l'algorithme Analyser-Instance sur des instances de la base *Lymphography*

ID	near=7	near=9	near=12	near=14
1	(0, 43%, 57%, 0)	(0, 49.5%, 49.5%, 1%)	(1%, 55%, 42%, 2%)	(1%, 54%, 41%, 3%)
2	(0, 63%, 37%, 0)	(1%, 58%, 41%, 0)	(1%, 57%, 42%, 0)	(1%, 55%, 41%, 2%)
3	(0, 60%, 40%, 0)	(3%, 55%, 42%, 0)	(1%, 56%, 40%, 2%)	(1%, 56%, 40%, 3%)
4	(0, 80%, 20%, 0)	(0, 53%, 44%, 2%)	(1%, 55%, 41%, 3%)	(1%, 55%, 41%, 3%)
5	(0, 63%, 37%, 0)	(0, 56%, 44%, 0%)	(1%, 56%, 42%, 1%)	(1%, 55%, 41%,3%)
6	(0, 0, 0, 100%)	(0, 0, 33%, 66%)	(0, 50%, 38%, 12%)	(1%, 52%, 43%, 4%)

TAB. 6.7 – Les résultats de l'algorithme Analyser-Instance sur des instances de la base *Lymphography*

ID	near=3	near=4	near=5	near=7	near=9	near=10	near=12	near=14
1	1	1	2	21	101	125	146	148
2	2	7	16	68	120	134	142	147
3	1	1	4	30	78	106	134	145
4	1	2	8	41	108	131	147	148
5	2	5	9	46	102	128	144	148
6	1	1	1	1	6	9	34	100

TAB. 6.8 – Le nombre d'instances les plus proches quand *near* vaut 3, 4, 5, 7, 9, 10, 12, 14

Nous remarquons que nos résultats sont proches des résultats donnés par *Analyser-Instance* quand *near* vaut 3 et 4. Quand *near* vaut 5, nos résultats et ceux de l'algorithme *Analyser-Instance* sont proches seulement pour les instances numéros : 1, 2, 5, 6. Pour les instances numéros 3 et 4, nos résultats s'éloignent des résultats de l'algorithme *Analyser-Instance*.

Les fréquences des valeurs de classe dans la base d'apprentissage de *Lymphography* ayant 148 instances sont 1% pour *normal*, 54% pour *metastases*, 41 pour *malign-lymph* et 3% pour *fibrosis*. Quand on augmente la distance (*near est au moins égal à 9*), les résultats de l'algorithme *Analyser-Instance* ont tendance à s'approcher de ces fréquences.

Les résultats obtenus par *AAO* (tableau 6.5) ne sont pas proches de ceux de l'algorithme *Analyser-Instance* quand *near* vaut 3 et 4 pour les quatre premières instances. Quand *near* vaut 5, les résultats de *AAO* pour les instances numéros 2 et 3 deviennent très proches des résultats de *Analyser-Instance* (tableau 6.6).

Pour expliquer ce résultat, nous présentons dans le tableau 6.8 le nombre d'instances les plus proches de chaque instance du tableau 6.4 pour chaque valeur de *near*. Nous pouvons expliquer les résultats donnés dans le tableau 6.8 de la manière suivante :
Near vaut 3 signifie que *Distance* $<$ *near* et cela signifie qu'il y a moins de 3 attributs dont les valeurs sont différentes dans les deux instances. Donc, pour l'instance numéro 1, le nombre d'instances les plus proches d'elle est 1. Cela veut dire que nous avons trouvé une seule instance proche de l'instance 1 dont la distance est inférieure à 3.

Quand *near* vaut 5, le nombre d'instances les plus proches de l'instance 1 est seulement 2. La distance entre la première instance trouvée et l'instance 1 est 2.08 et la distance entre la deuxième instance trouvée et l'instance 1 est 4.672. Ces deux instances appartiennent à la classe *malign-lymph*, qui est la classe de l'instance 1.

Cependant, quand *near* vaut 7, nous avons 21 instances proches de l'instance 1, dont 9 appartiennent à la classe *metastases* avec une probabilité 0.43 et 12 appartiennent à la classe *malign-lymph* avec une probabilité 0.57. Quand *near* dépasse 7, le nombre d'instances les plus proches de l'instance 1 augmente. Par exemple, il devient 101 quand *near* vaut 9. 50 de ces instances appartiennent à la classe *malign-lymph* avec une probabilité 0.495, 50 autres instances appartiennent à la classe *metastases* avec une probabilité 0.495 et une seule instance appartient à la classe *fibrosis* avec une probabilité 0.01.

Quand *near* vaut 3 ou 4, le nombre d'instances trouvées n'est pas assez grand.

Quand on augmente *near* ce nombre augmente également, et le résultat du classement peut changer. Par exemple, les instances les plus proches des instances numéros 3 et 4 appartiennent à une autre classe que *malign-lymph*. Nous pouvons voir que quand *near* vaut 5, le nombre d'instances les plus proches de l'instance numéro 4 est 8, parmi lesquelles 6 appartiennent à la classe *metastases* et 2 appartiennent à la classe *malign-lymph*. Cela explique l'écart entre le résultat

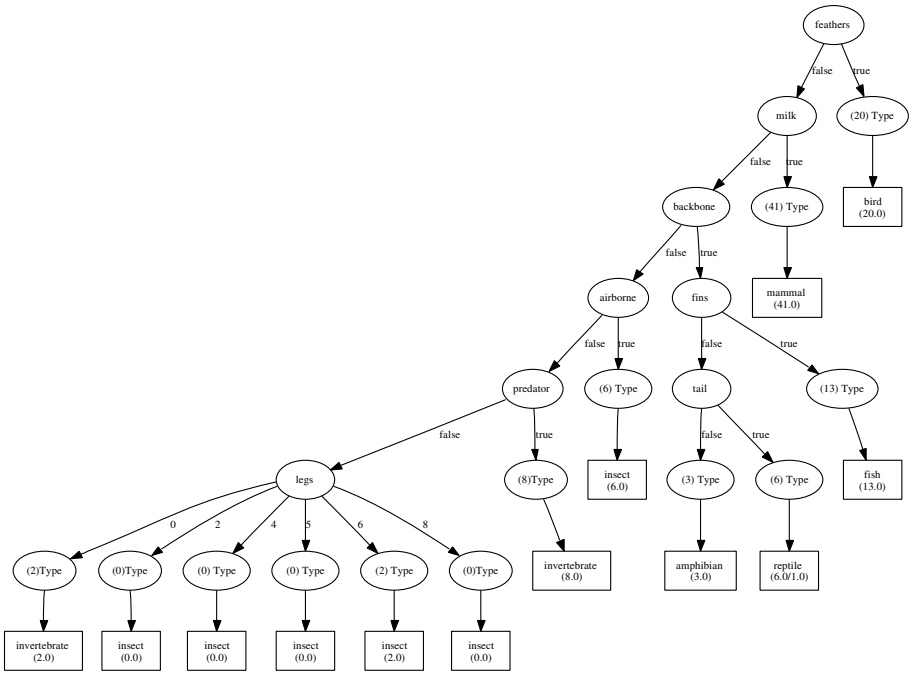

FIG. 6.1 – L'arbre de C4.5 pour la base *Zoo*

quand *near* vaut 3 et quand *near* vaut 5.

À partir du tableau 6.5, nous remarquons que AAP classe bien ces deux instances, contrairement à AAO et C4.5. Le fait de ne pas utiliser de stratégie de post-élagage lors de la construction de l'arbre de décision final nous a permis d'utiliser plus d'attributs que ceux utilisés par C4.5 et AAO.

La base *Zoo*

L'ensemble d'apprentissage de la base *Zoo* contient 101 instances sans valeurs manquantes et 17 attributs (parmi lesquels 15 attributs sont booléens). La classe est l'attribut *type*, qui prend les 7 valeurs suivantes : *mammal, bird, reptile, fish, amphibian, insect, invertebrate*. Le premier attribut est *animal-name*, qui est unique pour chaque instance. Cette base a pour but de donner le type des animaux.

Nous avons construit nos arbres $AAOPs$ et $AAPs$ en utilisant une base d'apprentissage ayant 101 instances. La figure 6.2 contient l'arbre de décision final, qui correspond à tout l'ensemble d'apprentissage et qui est construit selon l'approche AAP pour un seuil 0.1. La figure 6.1 représente l'arbre de décision final pour la même base, construit selon C4.5.

La plupart des attributs de cette base dépendent de la classe. La moyenne de l'information mutuelle normalisée entre les attributs et la classe est 0.3. Nous avons testé notre approche sur plusieurs seuils : 0.1, 0.2, 0.3 et 0.4[66].

Pour le seuil 0.1, nous trouvons que l'arbre donné par AAP (figure 6.2) classe bien toutes les instances dans la base d'apprentissage. Les attributs utilisés pour construire cet arbre sont : *milk,*

[66]Nous présentons ici seulement les résultats pour le seuil 0.1.

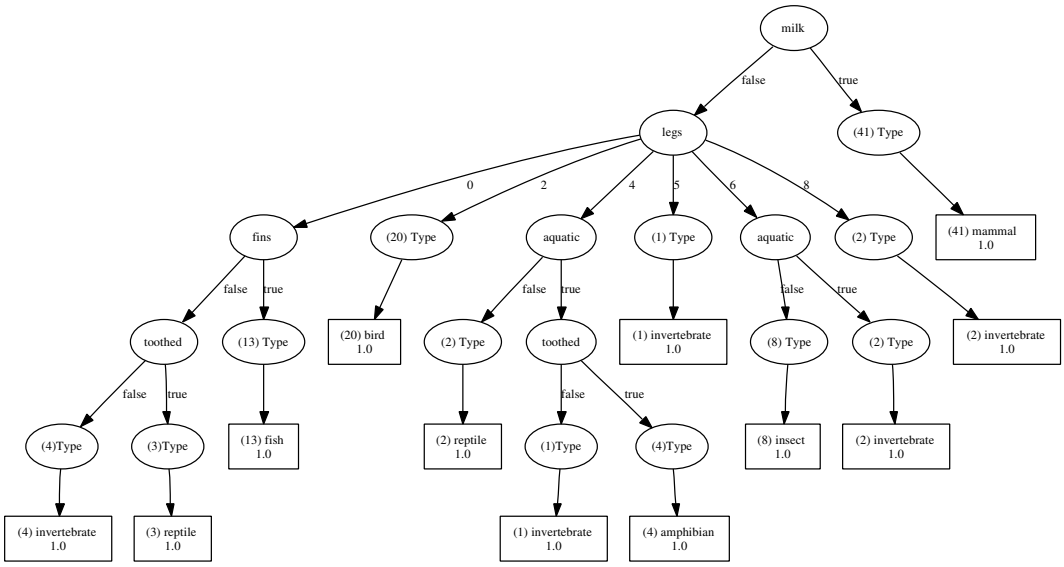

FIG. 6.2 – L'*AAP* pour *Zoo* pour un seuil 0.1

legs, fins, toothed, aquatic. La racine de cet arbre est l'attribut *milk* dont l'*AAP* est construit en utilisant les attributs : *eggs, hair, catsize, airborne*. En revanche, les attributs utilisés pour construire l'arbre de C4.5 sont : *feathers, milk, backbone, airborne, predator, legs, fins, tail*.

Nous allons maintenant comparer les résultats donnés par *AAP*, C4.5 et *AAO* avec ceux donnés par l'algorithme *Analyser-Instance* pour classer quelques instances ayant des valeurs manquantes de la base *Zoo*. Nous avons testé l'algorithme *Analyser-Instance* quand *near* vaut 3, 4, 5, 7, 9 et 12. Les instances de la base *Zoo* sont données dans le tableau 6.9. Le tableau 6.10 contient nos résultats et les résultats donnés par C4.5 pour classer les instances dans le tableau 6.9. Le tableau 6.11 contient les résultats donnés par *AAO* pour classer les mêmes instances.

Les tableaux 6.12, 6.13 et 6.14 contiennent les résultats de test de l'algorithme *Analyser-Instance* sur les instances dans le tableau 6.9 quand *near* vaut 3, 4, 5, 7, 9 et 12. En comparant ces résultats avec ceux donnés par *AAP*, *AAO* et C4.5, nous trouvons que les résultats donnés par *AAP* sont proches des résultats de l'algorithme *Analyser-Instance* quand *near* vaut 3, 4, 5. Les résultats de C4.5 ne sont pas très proches des résultats de l'algorithme *Analyser-Instance*. Par exemple, quand l'attribut *milk* est manquant, comme dans les instances 1 et 4, C4.5 calcule sa probabilité en fonction de l'attribut *feathers*. Cependant, avec *AAP*, la probabilité de *milk* est calculée en utilisant ses attributs dépendants, qui sont *eggs, hair, catsize, airborne*, pour un seuil 0.1.

En comparant les résultats obtenus par *AAO* avec les résultats de l'algorithme *Analyser-Instance*, nous remarquons que *AAO* se trompe en classant les trois premières instances et classe bien les trois dernières instances quand *near* vaut 3.

Nous remarquons que quand *near* dépasse 7, les distributions de probabilités se dispersent autour des 7 valeurs de classe de la base *zoo*. Ce qui signifie que si la distance entre deux instances est supérieure ou égale à 7, la probabilité que ces deux instances appartiennent au même type

Attributs	Instance1	Instance2	Instance3	Instance4	Instane5	Instance6
hair	true	false	false	true	false	false
feathers	false	?	false	false	false	?
eggs	true	true	?	false	true	?
milk	?	false	false	?	false	false
airborne	false	false	false	?	?	true
aquatic	true	true	?	false	true	false
predator	?	true	true	true	true	false
toothed	false	false	true	true	true	false
backbone	true	true	?	true	true	true
breathes	true	true	true	true	false	true
venomous	false	false	?	false	false	false
fins	false	false	false	false	true	false
legs	4	2	4	?	?	?
tail	?	true	?	false	true	true
domestic	false	false	false	false	false	true
catsize	true	true	false	true	false	false
classe	mammal	bird	reptile	mammal	fish	bird

TAB. 6.9 – Les instances de la base Zoo

ID	AAP	C4.5
1	(**100%**, 0%, 0%, 0%, 0%, 0%, 0%)	(**51%**, 0%, **27%**, 0%, **22%**, 0%, 0%)
2	(0%, **100%**, 0%, 0%, 0%, 0%, 0%)	(0%, **20%**, **67%**, 0%, **13%**, 0%, 0%)
3	(0%, 0%, **48%**, 0%, **52%**, 0%, 0%)	(0%, **20%**, 0%, 0%, 0%, 0%, **80%**)
4	(**100%**, 0%, 0%, 0%, 0%, 0%, 0%)	(0%, 0%,**31%**, 0%, **24%**, 0%,**45%**)
5	(0%, 0%, 0%, **100%**, 0%, 0%, 0%)	(**51%**, 0%, 0%, 0%, **49%**, 0%, 0%)
6	(0%, **67%**, **33%**, 0%, 0%, 0%, 0%)	(0%, **20%**, **67%**, 0%, **13%**, 0%, 0%)

TAB. 6.10 – Les résultats de AAP et C4.5 sur des instances de la base Zoo

ID	AAO
1	(0%, 0%, **100%**, 0%, 0%, 0%, 0%)
2	(0%, 0%, **100%**, 0%, 0%, 0%, 0%)
3	(0%, 0%, 0%, 0%, **100%**, 0%, 0%)
4	(**100%**, 0%, 0%, 0%, 0%, 0%, 0%)
5	(0%,0%, 0%, **100%**, 0%, 0%, 0%)
6	(0%,**100%**, 0%, 0%, 0%, 0%, 0%)

TAB. 6.11 – Les résultats de AAO sur des instances de la base Zoo

ID	near = 3	near=4
1	(**100%**, 0%, 0%, 0%, 0%, 0%, 0%)	(**94%**, 0%, 5%, 0%, 0%, 0%, 0%)
2	(0%, **100%**, 0%, 0%, 0%, 0%, 0%)	(0%, **100%**, 0%, 0%, 0%, 0%, 0%)
3	(0%, 0%, **50%**, 0%, **50%**, 0%, 0%)	(**18%**, **9%**, **36%**, 0%, **36%**, 0%, 0%)
4	(**100%**%, 0%, 0%, 0%, 0%, 0%, 0%)	(**100%**, 0%, 0%, 0%, 0%, 0%, 0%)
5	(0%, 0%, 0%, **100%**, 0% , 0%, 0%)	(0%, 0, **12%**, **81%**, **6%**, 0%, 0%)
6	(0%, **100%**, 0%, 0%, 0%, 0%, 0%)	(0%, **100%**, 0%, 0%, 0%, 0%, 0%)

TAB. 6.12 – Les résultats d'*Analyser-Instance* sur des instances de la base Zoo quand *near* est 3 et 4

ID	near = 5	near =7
1	(**91%**, **3%**, **3%**, 0%, **3%**, 0%, 0%)	(**64%**, 17%, 5%, 5%, 6%, 0%, 3%)
2	(**5%**, **85%**, **5%**, 0%, **5%**, 0%, 0%)	(11%, **42%**, 8%, 19%, 8%, 0%, 11%)
3	(**42%**, **12%**, **15%**, **15%**, **12%**, 0%, **3%**)	(**41%**, 21%, 5%, 14%, 4%, 4%, 10%)
4	(**100%**, 0%, 0%, 0%, 0%, 0%, 0%)	(**82%**, 2%,8%,0%, 8%, 0%,0%)
5	(**9%**, 0%, **19%**, **62%**, **9%**, 0%, 0%)	(11%, 13%, 11%, **35%**, 11%, 0%, 19%)
6	(0%, **100%**, %, 0%, 0%, 0%, 0%)	(5%, **59%**, 9%, 0%, 0%, 20%, 6%) [1]

TAB. 6.13 – Les résultats d'*Analyser-Instance* sur des instances de la base Zoo quand *near* vaut 5 et 7

[1](5%, 59%, 9%, 0%, 0%, 20%, 6%) signifie que le résultat de classement est 5% pour mammal, 59% pour bird, 9% pour reptile, 0% pour fish, 0% pour amphibian, 20% pour insect et 6% pour invertebrate.

ID	near = 9	near=12
1	(41%, 20%, 5%, 13%, 4%, 7%, 10%)	(41%, 20%,5%, 13%, 4%, 8%, 10%)
2	(35%, 23%, 6%, 15%, 4%, 4%, 11%)	(41%, 20%,5%, 13%, 4%, 6%, 10%)
3	(41%, 20%, 5%, 13%, 4%, 7%, 10%)	(41%, 20%,5%, 13%, 4%, 8%, 10%)
4	(51%, 10%, 6%, 11%, 5%, 9%, 7%)	(41%, 20%,5%, 13%, 4%, 8%, 10%)
5	(29%, 22%, 7%, 18%, 5%, 4%, 14%)	(41%, 20%,5%, 13%, 4%, 7%, 10%)
6	(22%, 36%, 7%, 7%, 5%, 15%, 7%)	(40%, 20%,5%, 13%, 4%, 8%, 10%)

TAB. 6.14 – Les résultats d'*Analyser-Instance* sur des instances de la base Zoo quand *near* vaut 9 et 12

ID	near=3	near=4	near=5	near=7	near=9	near=10	near=12
1	2	19	34	64	100	101	101
2	9	13	20	47	87	97	101
3	6	11	33	91	100	101	101
4	26	35	41	50	80	96	101
5	13	16	21	37	72	90	100
6	11	18	19	34	55	77	100

TAB. 6.15 – Le nombre d'instances les plus proches dans *Zoo* quand *near* vaut 3, 4, 5, 7, 9, 10, 12

d'animaux[67] est faible.

Nous notons que dans le tableau 6.14 quand *near* vaut 12, les résultats donnés par cet algorithme pour les 6 instances sont (41%, 20%,5%, 13%, 4%, 8%, 10%). Ces résultats sont égaux aux fréquences des valeurs de classe dans la base d'apprentissage *Zoo* ayant 101 instances.

Pour mieux expliquer les résultats de l'algorithme *Analyser-Instance*, nous présentons dans le tableau 6.15 le nombre des instances les plus proches de chaque instance du tableau 6.9. Le nombre d'instances les plus proches augmente en augmentant *near*. Quand *near* prend les valeurs 3,4, 5 et 7, les résultats de l'algorithme *Analyser-Instance* sont proches des résultats obtenus par *AAP*. Quand *near* dépasse 7, les résultats de l'algorithme *Analyser-Instance* ont tendance à s'approcher des fréquences de classe dans la base d'apprentissage.

Conclusion sur l'algorithme *Analyser-Instance*

En utilisant l'algorithme *Analyser-Instance* nous remarquons que :

- **En augmentant la distance entre les instances, la probabilité de chaque valeur de classe varie selon la situation**. Par exemple, la probabilité de *mammal* dans la première instance donnée dans le tableau 6.9 est 1 quand *near* vaut 3. Cette probabilité diminue quand *near* est supérieur ou égal à 4 (tableaux 6.12, 6.13).
- **À partir d'une certaine distance, la probabilité d'une classe est stable et égale à la fréquence de valeur de classe dans la base d'apprentissage.** La probabilité de *mammal* se stabilise et prend la valeur 0.41 quand *near* est supérieur ou égal à 9 (tableau 6.14). Cette probabilité stable de *mammal* est sa fréquence dans la base d'apprentissage.

Par exemple, si on prend les instances 1 et 5 du tableau 6.9 :

- Instance 1 : true, false, true, ?, false, true, ?, false, true, true, false, false, 4, ?, false, true, mammal.
  - Le résultat de classement de l'instance avec AAP est : (**100**%, 0%, 0%, 0%, 0%, 0%, 0%)
  - Le résultat de classement de l'instance avec C4.5 est : (**51**%, 0%, **27**%, 0%, **22**%, 0%, 0%)

[67] même classe.

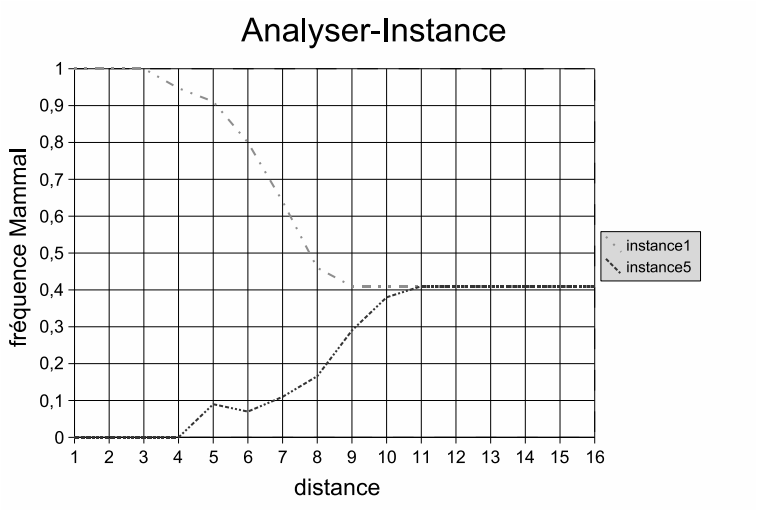

FIG. 6.3 – Les résultats d'*Analyser-Instance* pour deux instances de la base *Zoo* sur plusieurs distances

– Instance 5 : false, false, true, false, ?, true, true, true, true, false, false, true, ?, true, false, false, fish.

  – Le résultat de classement de l'instance avec AAP est : $(0\%, 0\%, 0\%, \mathbf{100}\%, 0\%, 0\%, 0\%)$

  – Le résultat de classement de l'instance avec C4.5 est : $(\mathbf{51}\%, 0\%, 0\%, 0\%, \mathbf{49}\%, 0\%, 0\%)$

Nous avons appliqué l'algorithme *Analyser-Instance* sur ces deux instances pour une distance qui varie de 3 jusqu'à 16 (figure 6.3). La première instance est de classe *Mammal*; l'autre est de classe *fish*.

À partir de la figure 6.3, nous remarquons que :

– Quand la *Distance* est au plus 3, *Analyser-Instance* classe la première instance comme *Mammal* avec une probabilité 1. Par ailleurs, la probabilité que l'instance 5 appartienne à la classe *Mammal* est 0 quand la *Distance* est au plus 4.
– En augmentant la *Distance*, la fréquence de *Mammal* pour la première instance diminue jusqu'à ce qu'elle devienne égale à 41% qui est la fréquence de la classe *Mammal* dans toute la base d'apprentissage ayant 101 instances. Dans ce cas, la *Distance* est au moins 9.
– En augmentant la *Distance*, la fréquence de *Mammal* pour l'instance 5 augmente jusqu'à ce qu'elle devienne égale à 41% qui est la fréquence de *Mammal* dans toute la base d'apprentissage ayant 101 instances. Dans ce cas, la *Distance* est au moins 11.
– La probabilité que l'instance 1 appartienne à la classe *Mammal* ne peut pas être moins que 0.41 quand la *Distance* atteint son maximum.
– La probabilité que l'instance 5 appartienne à la classe *Mammal* ne peut pas être plus que 0.41 quand la *Distance* atteint son maximum.

6.5 Conclusion

Nous nous sommes intéressés au résultat de classement de chaque objet dans la base de test parce que les valeurs des attributs changent d'un objet à un autre. Nous avons trouvé que C4.5 calcule la probabilité d'un attribut manquant A en fonction des attributs rencontrés dans le chemin de la racine de l'arbre jusqu'à A. Ces attributs ne sont pas forcément dépendants de A. En conséquence, lors du calcul de probabilité de A, C4.5 ne prend pas en compte les attributs qui dépendent de A et qui ne se trouvent pas dans son chemin.

Cependant, nous avons remarqué qu'en prenant en compte les dépendances entre les attributs, les résultats de classement sont plus fins et plus précis. Pour analyser le résultat de classement de chaque instance à classer, nous avons proposé un algorithme appelé *Analyser-Instance* qui calcule la fréquence des instances les plus proches de l'objet à classer de chaque classe dans toute la base d'apprentissage. Nous avons ensuite comparé les résultats de classement obtenus par AAP, C4.5 et AAO avec ceux donnés par l'algorithme *Analyser-Instance*. En conséquence, les résultats obtenus par AAP sont plus proches des résultats de l'algorithme *Analyser-Instance* que C4.5 et AAO.

Nous pouvons conclure que l'approche AAP, qui est basée sur la dépendance entre les attributs et qui donne un résultat de classement probabiliste, et l'algorithme *Analyser-Instance* sont deux approches proposées pour classer les objets ayant des valeurs manquantes. La première est élaborée dans le cadre des arbres de décision ; elle remplace un attribut manquant par une distribution de probabilités et associe à l'instance à classer une distribution de probabilités de classe. La deuxième est une méthode des k plus proches voisins et donne également une distribution de probabilités de classe sans traiter les attributs manquants dans l'objet à classer.

Contrairement à la méthode des k plus proches voisins, nous ne fixons pas un nombre K d'objets les plus proches de l'instance à classer. Nous prenons tous les objets ayant une distance inférieure à un seuil. Nous considérons que deux instances sont proches quand la distance entre elles est inférieure à ce seuil. Le seuil dépend du nombre d'attributs dans la base d'apprentissage. Il ne peut pas dépasser le nombre maximum d'attributs dans la base.

Nous avons remarqué qu'en augmentant la distance entre deux objets, la probabilité de chaque valeur de classe s'approche de sa fréquence dans la base d'apprentissage.

Dans le chapitre suivant, nous calculons la complexité de construction de nos arbres $AAPs$ et $AAOPs$, ainsi que la complexité de classement d'un objet ayant des valeurs manquantes avec notre approche, AAO et C4.5.

Chapitre 7

Complexité de classement d'objets avec valeurs manquantes

7.1 Introduction

Dans ce chapitre, nous allons présenter la complexité de construction d'un arbre de décision en temps d'exécution. Ensuite, nous passons au calcul de la complexité de construction des arbres de décision construits selon notre méthode. Nous finissons par présenter la complexité de classement d'un nouvel objet avec valeurs manquantes en utilisant notre approche, la méthode des *Arbres d'Attributs Ordonnés (AAO)* ainsi que en utilisant C4.5.

Avant de calculer la complexité de construction d'un arbre de décision, nous commençons par calculer le nombre de feuilles et la hauteur de cet arbre.

7.1.1 Nombre de feuilles dans un arbre de décision

Un arbre de décision est construit à partir d'un ensemble d'apprentissage selon le principe divide-and-conquer (diviser pour régner). Une fois l'arbre construit, nous pouvons calculer le nombre de ses feuilles à partir du nombre de ses noeuds internes. Les noeuds internes sont tous les noeuds dans l'arbre de décision à l'exception des feuilles.
Supposons que nous ayons N noeuds internes dans l'arbre de décision. Chaque noeud i correspond à un attribut-test A_i qui prend v_i valeurs possibles. Le nombre de feuilles dans cet arbre est :

$$L = \sum_{i=1}^{i=N} v_i - (N-1) \tag{7.1}$$

Par exemple, si nous avons 3 noeuds, dont le nombre de leurs valeurs est 3, 4, 5 respectivement, $L = (3+4+5) - (3-1) = 12 - 2 = 10$. Sur la figure 7.1, nous remarquons que nous avons toujours le même nombre de feuilles quelque soit le noeud-racine de l'arbre. Le nombre total de noeuds dans un arbre de décision est égal au nombre de ses noeuds internes + le nombre de ses feuilles :

$$TotalN = 1 + \sum_{i=1}^{i=N} v_i \tag{7.2}$$

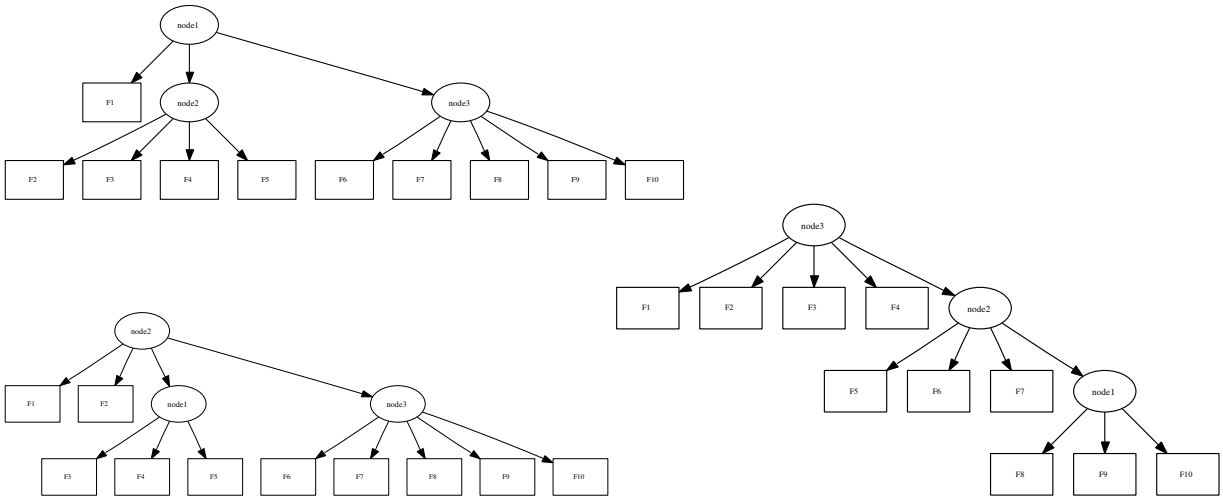

FIG. 7.1 – Quelques arbres possibles construits avec 3 noeuds

7.1.2 Hauteur d'un arbre de décision

A partir du tableau 7.1, nous remarquons que :

- Le nombre des noeuds internes dans un arbre de décision binaire dont la hauteur est h est au plus $2^h - 1$. la hauteur h d'un arbre de décision binaire, qui possède au plus N $=2^h - 1$ noeuds internes, est au moins $log_2(N + 1)$.
- Le nombre de feuilles dans un arbre de décision binaire, dont la hauteur est h, est au plus 2^h. la hauteur h d'un arbre de décision, qui a au plus L $=2^h$ feuilles, est au moins $log_2 L$.
- Le nombre de noeuds internes $_{h=i}$ = le nombre de noeuds internes$_{h=i-1}$ + le nombre de feuilles$_{h=i-1}$.
- Le nombre total de noeuds$_{h=i}$ = le nombre de noeuds internes$_{h=i}$ + le nombre de feuilles$_{h=i}$.
  Pour h = k : le nombre total de noeuds $tn = 2^k - 1 + 2^k = 2^{k+1} - 1 \Longrightarrow tn + 1 = 2^{k+1}$ $\Longrightarrow \frac{2^{k+1}}{2} = \frac{tn+1}{2} \Longrightarrow 2^k = \frac{tn+1}{2}$ mais 2^k est le nombre de feuilles dans l'arbre. Donc, si le nombre total de noeuds dans un arbre de décision binaire est tn, le nombre de feuilles L est au plus $\frac{tn+1}{2}$.
  Nous déduisons que le nombre de feuilles dans un arbre de décision binaire est égal au nombre de noeuds internes plus 1. L = N+1.
  Donc, $tn = L + N = N + 1 + N = 2N + 1 \Longrightarrow tn - 1 = 2N \Longrightarrow N = \frac{tn-1}{2}$.

De la même manière, nous pouvons calculer la hauteur d'un arbre de décision n-aire. Supposons que L est le nombre de feuilles dans l'arbre. N est le nombre de noeuds internes dans l'arbre où $v_1, v_2, ..v_N$ sont les nombres de valeurs de chaque noeud. Le nombre moyen de ces valeurs est $v = \frac{v_1+v_2+..+v_N}{N}$. Dans ce cas, la hauteur h de l'arbre décision est au moins $log_v L$ En conséquence, la hauteur minimale d'un arbre de décision ayant L feuilles est $log_v L$. Nous allons maintenant calculer la hauteur maximale d'un arbre de décision :

Hauteur maximale d'un arbre de décision

Un arbre de décision ayant m attributs symboliques ou discrets atteint sa hauteur maximale lorsqu'un chemin de sa racine à une feuille contient les m attributs symboliques ou discrets.

Hauteur h	nombre de noeuds internes N	nombre de feuilles F
h=0	0	1
h=1	1	2
h=2	1+2=**3**	2*2 =**4**
h=3	3+4 = 7	4*2=8
h=4	7+8=15	8*2 = 16
..	..	..
h = k	$2^k - 1$	2^k

TAB. 7.1 – Les relations entre la hauteur, le nombre de noeuds internes et le nombre de feuilles dans un arbre de décision binaire

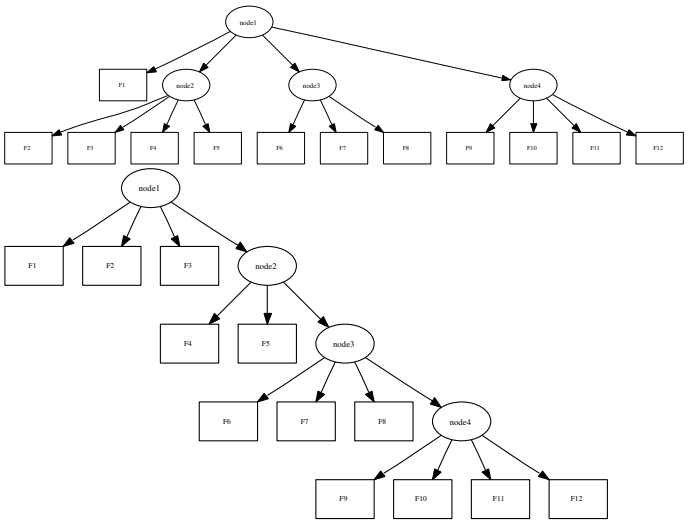

FIG. 7.2 – Hauteur minimale et maximale d'un arbre de décision avec 4 noeuds internes

Dans ce cas, la hauteur maximale est égale à m. Par exemple, supposons qu'on a un arbre de décision ayant 4 attributs discrets qui sont les noeuds internes dans cet arbre et qui possèdent 4, 3, 4, 4 valeurs, respectivement. À partir de l'équation 7.1, nous pouvons calculer le nombre de ses feuilles :$L = (4 + 3 + 4 + 4) - (4 - 1) = 12$ feuilles. Pour calculer la hauteur minimale de cet arbre, nous calculons d'abord la valeur moyenne par attribut : $v = \frac{4+3+4+4}{4} = 3.75$.

Donc, la hauteur d'un tel arbre est au moins 2 parce que : $log_v(12) = 1.88 \approx 2$ (figure 7.2). La hauteur maximale d'un arbre avec ces attributs est 4 (figure 7.2). Cet arbre atteint sa hauteur maximale (figure 7.2) lorsque les m attributs sont dans un seul chemin. Dans ce cas, la hauteur maximale d'un tel arbre est au plus 4, parce que nous ne pouvons pas avoir deux attributs discrets ou symboliques dans le même chemin dans un arbre de décision.

Dans la suite de ce chapitre et lors du calcul de la complexité de classement d'un objet dans un arbre de décision, nous utilisons la hauteur minimale de l'arbre calculée en fonction du nombre de ses feuilles. La hauteur minimale d'un arbre de décision donne la complexité minimale. Mais nous calculons cette complexité dans le pire des cas, lorsque tous les attributs d'un objet à classer sont manquants [Hawarah *et al.*, 2006a].

7.2 Complexité d'un arbre de décision

Dans cette partie, nous présentons la complexité de construction d'un arbre de décision et nous calculons la complexité de construction les arbres de décision *AAOPs et AAPs*.

7.2.1 La complexité de construction d'un arbre de décision

La complexité d'un arbre de décision dépend de[68] : 1) la tailles de la base d'apprentissage ; 2) le nombre d'attributs dans cette base ; 3) la hauteur de l'arbre de décision. Supposons que nous avons une base d'apprentissage qui contient n objets et m attributs[69], la complexité d'un arbre de décision construit en utilisant cette base d'apprentissage est [Martin et Hirschberg, 1995] :

$$O(n \times m \times log(n)) \tag{7.3}$$

Complexité de construction d'un AAOP

Dans l'approche *AAOP*, nous construisons pour chaque attribut dans la base d'apprentissage un *arbre d'attribut*. Donc, pour une base d'apprentissage qui contient m attributs (sans l'attribut classe) et n instances, nous construisons m *arbres d'attributs*. Le premier arbre construit est un seul noeud-feuille qui est construit en utilisant un seul attribut, le deuxième arbre est construit en utilisant deux attributs et ainsi de suite. Le dernier arbre est construit en utilisant m attributs[70]. Puisque nous obtenons m arbres de décision avec l'augmentation du nombre d'attributs, la complexité de ces arbres est :

$$\begin{aligned} &O(1 \times n\ log(n) + 2 \times n\ log(n) + .. + m \times n\ log(n)) = \\ &O((\sum_{i=1}^{m} i)\ n\ log(n)) = O(\frac{m(m+1)}{2}\ n\ log(n)) \end{aligned} \tag{7.4}$$

Nous remarquons que c'est également la complexité de construction des arbres d'attributs dans la méthode des *Arbre d'Attributs Ordonnés (AAO)*.

Complexité de construction d'un AAP

Pour la même base d'apprentissage, nous construisons également m autres arbres de décision en utilisant l'approche *AAP*. Chaque arbre est construit en utilisant ses attributs dépendants. Dans le pire des cas, tous les attributs dans la base d'apprentissage sont dépendants. Dans ce cas, chaque arbre de décision est construit en utilisant m attributs. Donc, la complexité de ces m arbres est :

$$O(m\ n\ log(n) + O(m\ n\ log(n) + ... + O(m\ n\ log(n) = O(m\ m\ n\ log(n)) = O(m^2\ n\ log(n))$$

[68] Un arbre de décision représente les comparaisons effectuées par un algorithme de tri lorsqu'il traite une entrée de taille donnée. Dans un tel arbre, le nombre de feuilles est n !, parce que nous avons n ! permutations possibles de n éléments. Donc, la hauteur de l'arbre est log n !. D'après la formule de Stirling on a $n! \geq (\frac{n}{e})^n$. La longueur d'un chemin de la racine à une feuille dans l'arbre de décision est égal au nombre de comparaisons nécessaires au tri pour parvenir à la réponse souhaitée. Ce qui donne pour un chemin qui est égale à la hauteur de l'arbre : $log(n!) \approx nlogn$.

[69] L'attribut classe est inclus dans les m attributs.

[70] Au pire quand les *(m-1)* attributs dépendent de l'attribut m.

En conséquence, la complexité de tous les arbres construits selon $AAOP$ et AAP est :

$$O(\frac{m(m+1)}{2}\ n\ log(n)) + O(m^2\ n\ log(n)) = O(\frac{(3m^2+m)}{2}\ n\ log(n))$$

Nous ajoutons à la complexité précédente, la complexité de construction de l'arbre de décision final qui possède *(m+1)* attributs[71] et n instances. à partir de l'équation 7.3, la complexité de construction de l'arbre de décision final est :

$$O((m+1)\ n\ log(n))$$

En conséquence, la complexité de construction de tous les arbres d'attributs et l'arbre de décision final devient :

$$\begin{aligned}
&O(\frac{(3m^2+m)}{2}\ n\ log(n)) + O((m+1)\ n\ log(n)) = \\
&O(\frac{(3m^2+3m)}{2}\ n\ log(n)) + O(n\ log(n)) = \\
&O(\frac{3(m^2+m)}{2}\ n\ log(n)) + O(n\ log(n)) = \\
&O(\frac{3}{2}\ m^2\ n\ log(n)) + O(\frac{3}{2}\ m\ n\ log(n)) + O(n\ log(n))
\end{aligned}$$

Dans l'équation précédente, nous pouvons ignorer le terme $\frac{3}{2}$ parce que les facteurs constants ne sont pas importants [Aho et Ullman, 1993].
Dans la théorie de la complexité, la somme de deux fonctions O(g(n)), O(f(n)) est égale à O(max(O(g(n)), O(f(n))). En conséquence, nous pouvons ignorer la partie $O(n\ log(n))$. D'autre part, nous avons $m^2 > m$. Les termes d'ordre inférieur sont négligeables. Donc, nous pouvons également ignorer le terme $O(m\ n\ log(n))$ parce qu'il est plus petit que $O(m^2\ n\ log(n))$. Finalement, cette complexité devient :

$$O(m^2\ n\ log(n)) \tag{7.5}$$

Cette complexité est quadratique en fonction du nombre d'attributs m dans la base d'apprentissage et quasi-linéaire en fonction de taille n de la base d'apprentissage.

7.3 Complexité de classement d'objet dans un arbre de décision

Quand l'arbre de décision est construit, le classement d'un nouvel objet est très rapide avec une complexité $O(h)$, où h est la profondeur maximale de l'arbre [Tan *et al.*, 2006]. Cela est vrai quand nous classons une nouvelle instance sans valeurs manquantes, parce qu'on parcourt seulement un chemin dans l'arbre de décision depuis sa racine jusqu'à une feuille en fonction des valeurs des attributs-noeuds dans ce chemin. Si L est le nombre de feuilles dans l'arbre, la hauteur de cet arbre est supérieure ou égale à $log_v(L)$.

Nous allons calculer la complexité de classement dans un arbre de décision en fonction de la hauteur minimale de l'arbre. La complexité de classement d'un nouveau cas est $O(log_v(L))$.

[71] Donc, l'attribut classe est inclus.

Cette complexité change quand la nouvelle instance contient des valeurs manquantes pour certains attributs, parce que nous parcourons plusieurs chemins dans l'arbre au lieu d'un seul. Dans le pire des cas, nous parcourons tous les chemins possibles dans l'arbre de décision. Nous notons que le nombre de chemins (le nombre de sorties possibles) dans un arbre de décision est égal au nombre de ses feuilles. Donc, la complexité de parcours tous les chemins possibles dans l'arbre devient $O(L\ log_v(L))$.

Dans l'approche *AAOP*, nous avons construit m arbres. Le premier arbre est constitué d'un seul noeud-feuille, et sa complexité est $O(log1) = O(0)$.

Dans notre travail, chaque attribut de la base d'apprentissage possède deux *arbres d'attributs* (*AAOP* et *AAP*). La complexité de classement d'une nouvelle instance en utilisant un *AAOP* ou un *AAP* est :

- Dans le meilleur des cas, quand l'instance ne contient pas de valeurs manquantes, nous parcourons un seul chemin dans cet arbre : $O(log_v(L))$.
- Dans le pire des cas, quand tous les attributs sont manquants, nous parcourons tous les chemins dans l'arbre : $O(\ L\ log_v(L))$.

7.3.1 Complexité de notre algorithme de classement

Pendant le processus de classement, quand nous rencontrons une valeur manquante pour un attribut-noeud, nous appelons son *arbre d'attribut*[72] pour calculer sa probabilité. De la même manière, si nous rencontrons un autre attribut manquant dans cet arbre, nous devons appeler son arbre d'attribut, et ainsi de suite. A la fin du processus de classement, nous remarquons que nous avons appelé plusieurs *arbres d'attributs*. Le calcul de probabilité de chaque attribut manquant n'est pas arbitraire. Nous commençons toujours par traiter l'attribut le moins dépendant de la classe.

Exemple. Nous allons expliquer la complexité de notre algorithme de classement en utilisant un exemple de la base *météo* [Quinlan, 1993].

Dans cette base, nous avons 5 attributs : 1) *Temps*, qui prend les valeurs {ensoleillé, couvert, pluvieux} ; 2) *Température*, qui prend également 3 valeurs {élevée, moyenne, basse} ; 3) *Humidité* prend 2 valeurs {haute, normale} ; 4) *Vent* prend les valeurs {vrai, faux} ; 5) *Décision* est l'attribut classe et prend les valeurs {A, B}.
Considérons le classement d'un objet dont *Temps*, *Température* et *Humidité* sont manquants, *Vent* est *faux*. Nous commençons de la racine de l'arbre donné dans la figure 7.5 en parcourant les chemins correspondant aux valeurs de l'attribut manquant *Temps* jusqu'à ce qu'on arrive à une feuille. Parce que *Humidité* est également manquante, nous parcourons 4 chemins dans cet arbre.

Nous calculons la probabilité que cette instance appartienne à la classe A comme suit :

$$
\begin{aligned}
P(A) = {} & \texttt{P(A|ensoleillé, haute) * P(ensoleillé, haute)} \\
& \texttt{+ P(A|ensoleillé, normal) * P(ensoleillé, normal)} \\
& \texttt{+ P(A|couvert) * P(couvert)} \\
& \texttt{+ P(A|pluvieux, faux) * P (pluvieux, faux)}
\end{aligned}
\tag{7.6}
$$

[72] AAOP ou AAP selon la situation.

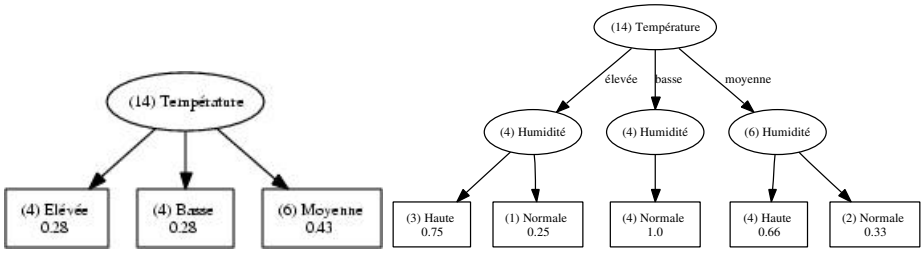

FIG. 7.3 – L'*AAOP de Température* et l'*AAOP d'Humidité*

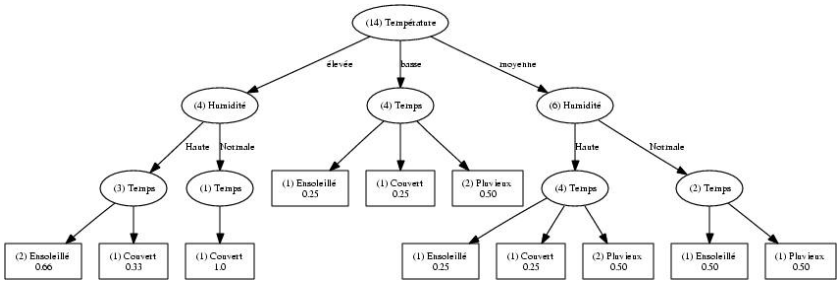

FIG. 7.4 – L'*AAP de Temps*

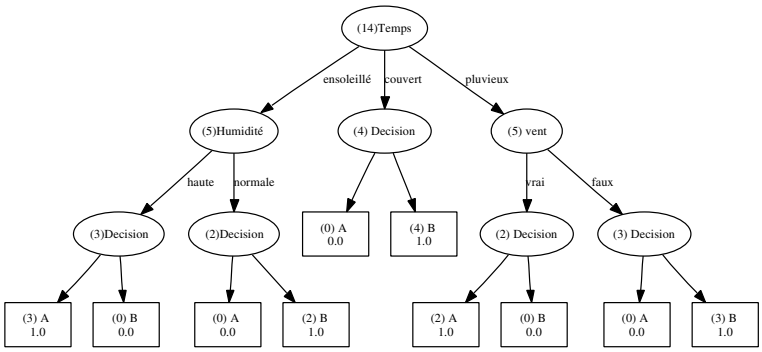

FIG. 7.5 – L'arbe de décision final pour la base *météo*

Dans le premier chemin, nous avons deux valeurs manquantes {ensoleillé et haute}. La probabilité jointe P(ensoleillé, haute) est calculée de la manière suivante :

```
P(ensoleillé,haute) = P(haute) * P(ensoleillé|haute)          (7.7)
```

Parce que la valeur *haute* est moins dépendant de la classe que *ensoleillé*, la probabilité P(haute) est calculée à partir de l'*AAOP* de *Humidité*. Cet arbre contient un autre attribut manquant qui est *Température*. *Température* dépend également de *Temps*. Donc, la probabilité dans l'équation 7.7 devient :

```
P(Temps = ensoleillé, Humidité = haute) =                      (7.8)
P(Température=élevée)*
P(Humidité=haute|Température=élevée)*
P(ensoleillé|Température = élevée,Humidité= haute)+
P(Température = moyenne)*
P(Humidité=haute|Température=moyenne)*
P(ensoleillé|Humidité = haute,Température = moyenne)+
P(Température = basse)*
P(Humidité = haute|Température=basse)*
P(ensoleillé|Humidité = haute,Température = basse)
```

Pour calculer la probabilité que *Humidité* est *haute*, nous devons calculer la probabilité que *Température* est {élevée, moyenne ou basse}. La probabilité de *Température* est calculée de son arbre *AAOP* donné dans la figure 7.3[73].

Nous remarquons que pour calculer les probabilités :
P(Température=élevée), P(Température = basse) and P(Température = moyenne) de l'équation 7.8, nous appelons l'*AAOP* de *Température* 3 fois, correspondant aux trois valeurs possibles de l'attribut *Température* .

Dans l'*AAOP* de *Température*, nous parcourons seulement un chemin et cet arbre est formé d'un seul noeud-feuille. Donc, la complexité de parcours l'*AAOP* de *Température* est $O(log1) = 0$.

Pour calculer la probabilité que *Humidité* est *haute*, nous appelons l'*AAOP* d'*Humidité* 3 fois parce que *Température* est inconnue. Donc, nous calculons la probabilité que *Humidité* est *haute* sachant que *Température* est *élevée*; la probabilité que *Humidité* est *haute* sachant que *Température* est *moyenne* et la probabilité que *Humidité* est *haute* sachant que *Température* est *basse*. Dans ce cas, nous parcourons tous les chemins dans l'*AAOP* d'*Humidité*. Donc, la complexité de parcours tous les chemins dans l'*AAOP* d'*Humidité* est $O(3log(3))$.

Finalement, pour calculer la probabilité que *Temps* soit *ensoleillé* sachant que *Humidité* est *haute*, nous devons calculer les trois probabilités suivantes : La probabilité que *Temps* soit *ensoleillé* sachant que *Humidité* est *haute* et *Température* est *élevée*; La probabilité que *Temps* soit *ensoleillé* sachant que *Humidité* est *haute* et *Température* est *moyenne*; La probabilité que *Temps* soit *ensoleillé* sachant que *Humidité* est *haute* et *Température* est *basse*. Donc, nous

[73] L'*AAOP* de *Température* est un noeud-feuille avec la distribution de probabilités de ses valeurs.

parcourons 3 chemins dans l'AAP de *Temps* en fonction des valeurs de *Température* avec la complexité $O(3log(5))$. La complexité totale est $O(3log(3) + 3log(5))$.

En conséquence, pour calculer la probabilité jointe dans l'équation 7.7, nous appelons l'AAP de *Temps* 3 fois, chacune pour un seul chemin ; nous appelons l'$AAOP$ de *Humidité* 3 fois, à chaque fois l'arbre est appelé pour un seul chemin. De la même manière, nous appelons l'$AAOP$ de *Température* 3 fois en fonction du nombre de ses valeurs.

En conséquence, nous avons appelé 9 *arbres d'attributs* ; la complexité de ces 9 arbres est la somme des complexités de chacun d'eux. En comparant avec l'équation 7.8, nous trouvons que :

- La complexité de parcours l'$AAOP$ de *Température* pour la valeur *élevée* est O(0).[74]
- La complexité de parocurs l'$AAOP$ de *Température* pour la valeur *basse*est O(0).
- La complexité de parocurs l'$AAOP$ de *Température* pour la valeur *moyenne*est O(0).
- La complexité de parcours l'$AAOP$ de *Humidité* pour la valeur *haute* sachant que *Température* est *élevée* est $O(log(3))$.
- La complexité de parcours l'$AAOP$ de *Humidité* pour la valeur *haute* sachant que *Température* est *moyenne* est $O(log(3))$.
- La complexité de parcours l'$AAOP$ de *Humidité* pour la valeur *haute* sachant que *Température* est *basse* est $O(log(3))$.
- La complexité de parcours l'AAP de *Temps* pour la valeur *ensoleillé* sachant que *Humidité* est *haute* et*Température* est *élevé* est $O(log(3))$.
- La complexité de parcours l'AAP de *Temps* pour la valeur *ensoleillé* sachant que *Humidité* est *haute* et *Température* est *moyenne* est $O(log(3))$.
- La complexité de parcours l'AAP de *Temps* pour la valeur *ensoleillé* sachant que *Humidité* est *haute* et*Température* est *basse* est $O(log(3))$.

Donc, la complexité totale devient : $O(3log(3) + 3log(5))$.

Nous pouvons calculer la complexité de notre algorithme de classement comme suit :
Supposons qu'à la fin de notre processus du classement, nous ayons appelé $nbAT$ *arbres d'attributs*. La complexité de ces $nbAT$ arbres est :

$$O(\sum_{i=1}^{i=nbAT} ComplexityTree(Tree_i)) \tag{7.9}$$

$$ComplexityTree(Tree) = \begin{cases} \texttt{0 if one leaf/*}L_{Tree} = 1\texttt{*/} \\ O(log(L_{Tree})) \ \texttt{else} \end{cases} \tag{7.10}$$

où L_{Tree} est le nombre de feuilles de cet arbre.

Pour calculer $nbAT$, qui est le nombre d'arbres d'attributs appelés pendant le processus du classement, nous supposons que l'instance à classer contient m attributs manquants et que l'attribut i prend v_i valeurs. Dans le cas où tous ces attributs sont dépendants, nous pouvons calculer le nombre d'arbres d'attributs appelés pendant le processus de classement comme suivant : $nbAT = m \times (v_1 \times v_2 \times ... \times v_m)$
où $\bar{v} = \frac{v_1+v_2+...+v_m}{m}$ est le nombre moyen des valeurs possibles pour chaque attribut. Le nombre des arbres appelés devient :

$$nbAT = m \times \bar{v}^m \tag{7.11}$$

[74]Parce que cet arbre est constitué d'une seule feuille.

En conséquence, le nombre d'arbres d'attributs appelés est exponentiel en fonction du nombre d'attributs manquants dans l'instance à classer. À partir des équations 7.9, 7.10 et 7.11, nous pouvons calculer la complexité de tous les arbres appelés au cours du classement comme suit :

$$\begin{aligned}
&O(\sum_{i=1}^{i=nbAT} ComplexityTree(Tree_i)) \\
&= O(\sum_{i=1}^{i=nbAT} log(L_{Tree_i})) \\
&= O(log(L_{Tree_1}) + log(L_{Tree_2}) + ... + log(L_{Tree_{nbAT}})) \\
&= O(log(L_{Tree_1} \times L_{Tree_2} \times ... \times L_{Tree_{nbAT}})) \\
&= O(log(\prod_{i=1}^{i=nbAT} L_{Tree_i})) = O(log(\prod_{i=1}^{i=nbAT} \bar{L}_T)) \\
&= O(log(\bar{L}_T^{nbAT})) = O(nbAT \times log(\bar{L}_T)) \\
&= O(m \times \bar{v}^m log(\bar{L}_T))
\end{aligned} \tag{7.12}$$

où :

- $\bar{L}_T$ est le nombre moyen des feuilles dans un *arbre d'attribut*.
- m est le nombre d'attributs manquants dans l'instance.
- $\bar{v}$ est le nombre moyen des valeurs pour chaque attribut manquant.

Nous remarquons que la complexité dans l'équation 7.12 augmente quand le nombre des valeurs manquantes augmentent.

Finalement, la complexité de classement d'une instance dans l'arbre de décision final en utilisant notre méthode est égale à la complexité de cet arbre de décision + les complexités de tous les arbres d'attributs appelés pendant le processus du classement[75] :

$$\begin{aligned}
Complexity(Trees) = O(&nbPath_{Tree}\ \ Log(L_{Tree})) \\
&+ O(\sum_{i=1}^{i=nbAT} ComplexityTree(Tree_i))
\end{aligned} \tag{7.13}$$

$$nbPath_{Tree} = \begin{cases} \texttt{1 if no missing values} \\ L_{Tree}\ \texttt{if all are missing} \\ nbPT\ \texttt{else} \end{cases} \tag{7.14}$$

Nous remarquons dans l'équation 7.13 que $nbPath_{Tree}$ peut être L_{Tree} dans le pire des cas quand tous les attributs sont manquants. Nous notons que la complexité $O(nbPath_{Tree}\ Log_v(L_{Tree}))$ dans cette équation est quasi-linéaire en fonction du nombre de feuilles de l'arbre de décision final.

En conséquence, la complexité totale de classement d'une instance avec valeurs manquantes devient :

$$\begin{aligned}
Complexity(Trees) = O(&nbPath_{Tree}\ \ Log(L_{Tree})) \\
&+ O(m \times \bar{v}^m log(\bar{L}_T))
\end{aligned}$$

[75] Dans l'équation 7.14, L_{Tree} est le nombre de feuilles dans cet arbre et est égal au nombre de tous les chemins. $nbPT$ est le nombre de chemins que nous devons parcourir dans l'arbre de décision final.

Cette complexité est exponentielle en fonction du nombre d'attributs manquants dans l'instance à classer et quasi-linéaire en fonction du nombre de feuilles dans l'arbre de décision final.

7.3.2 Complexité de classement dans la méthode des Arbres d'Attribut Ordonnés

Dans la méthode des *Arbres d'Attributs Ordonnés (AAO)* et lors du classement d'une instance incomplète, les attributs inconnus sont ordonnés par ordre croissant selon leur Information Mutuelle avec la classe. Ils sont calculés successivement dans cet ordre. Si le nombre des attributs manquants dans l'instance à classer est m. Nous appelons pendant le processus de classement m *arbres d'attributs*, qui correspondent à ces m attributs. Nous commençons par appeler l'*arbre d'attribut* pour l'attribut manquant le moins dépendant de la classe. Nous calculons sa valeur et nous la remplissons dans l'instance à classer et ainsi de suite. Pour déterminer la valeur manquante d'un attribut i, on parcourt un seul chemin dans son arbre d'attribut ayant L_i feuilles. La complexité de parcours d'un tel arbre est : $O(logL_i)$
La complexité de traiter les m valeurs manquantes dans un objet à classer dans la méthode AAO est donc :

$$\begin{aligned} & O(log(L_1) + .. + log(L_m)) = O(log(L_1 \times L_2 \times ... \times L_m)) \\ & = O(log(\prod_{i=1}^{i=m} L_i)) = O(log(\prod_{i=1}^{i=m} \bar{L}_T)) = O(log(\bar{L}_T^m)) \\ & = O(m \times log(\bar{L}_T)) \end{aligned} \tag{7.15}$$

Où $\bar{L}_T$ est le nombre moyen de feuilles dans un arbre d'attribut. Quand on remplit toutes les valeurs manquantes dans l'instance à classer, on parcourt un seul chemin dans l'arbre de décision final pour trouver la classe de cette instance, ce qui correspond à une complexité $O(log(L))$ où L est le nombre des feuilles dans l'arbre de décision final.

7.3.3 Complexité de classement dans la méthode C4.5

Dans la méthode C4.5, quand nous classons une instance ayant des valeurs manquantes, nous parcourons les chemins possibles dans l'arbre de décision final selon les valeurs des attributs inconnus. Dans le pire des cas, nous parcourons tous les chemins dans l'arbre de décision, ce qui correspond à la complexité suivante :

$$O(L\ log_v(L))$$

Cette complexité est quasi-linéaire en fonction du nombre de feuilles dans l'arbre de décision.

7.4 Conclusion

Dans ce chapitre, nous avons expliqué le calcul de la complexité de construction des arbres d'attributs (AAOPs et AAPs) en temps d'exécution. Nous avons ensuite présenté la complexité de classement d'un objet incomplet en utilisant notre algorithme de classement, la méthode C4.5 ainsi que la méthode des *Arbres d'Attributs Ordonnés (AAO)*.

Nous avons montré que notre algorithme est exponentiel en fonction de nombre d'attributs manquants dans l'instance à classer. Plus le nombre des valeurs manquantes dans l'objet à classer est grand, plus la complexité du classement augmente. Quant à la complexité de classement d'un objet en C4.5, elle est quasi-linéaire en fonction de nombre de feuilles de l'arbre de décision final.

Nous pouvons conclure que notre méthode est plus complexe que C4.5 et *AAO* lors du classement d'un objet ayant des valeurs manquantes. Cependant, la performance de notre approche est meilleure que celles de C4.5 et *AAO*.

Conclusion

Contributions

Nous avons traité dans ce mémoire du problème des valeurs manquantes dans les données pour les arbres de décision. Les arbres de décision sont une méthode d'apprentissage supervisé qui extrait des connaissances sous forme d'arbre. Cet arbre est facile à interpréter et à utiliser comme outil d'aide à la décision pour classer de nouveaux cas. Dans un tel arbre, les nœuds représentent les attributs et les valeurs de ces attributs forment les branches de chaque nœud. Les feuilles représentent la classe.

Le problème des valeurs manquantes se pose pendant la phase de construction lors du calcul de l'information mutuelle pour choisir l'attribut nœud ainsi que lors de la partition de l'ensemble d'apprentissage selon l'attribut choisi. Pendant la phase de classement, nous rencontrons ce problème quand un attribut nœud est inconnu dans l'objet à classer. L'objectif du travail effectué dans cette thèse est de classer de manière probabiliste un objet ayant des valeurs manquantes dans un arbre de décision.

Dans un premier temps, nous avons proposé deux approches pour résoudre ce problème, la première, *AAOP*, est une extension de la méthode des *Arbres d'Attributs Ordonnés (AAO)* proposée par Lobo et Numao. Nous avons étendu cette méthode sur deux points : 1) nous gardons sur chaque feuille les valeurs de la classe avec leurs probabilités ; 2) nous construisons chaque arbre d'attribut en utilisant les attributs déjà traités et dépendants de l'attribut en question. La deuxième proposition, *AAP*, construit pour chaque attribut un arbre d'attribut en utilisant les attributs dont il dépend. La dépendance est calculée en utilisant l'Information Mutuelle.

Nous avons combiné les deux approches pour classer le même objet quand nous avons plusieurs attributs manquants et dépendants en même temps. Dans ce cas, nous utilisons pour les attributs les moins dépendants de la classe leurs *AAOPs*, et nous utilisons les *AAPs* des attributs les plus dépendants de la classe. Par exemple, considérons le cas où nous avons quatre attributs *A, B, C* et *D* manquants et ordonnés par ordre croissant en fonction de leur Information Mutuelle Normalisée avec la classe tel que *A* et *B* sont dépendants et ils dépendent de *C* et *D*. *C* et *D* ne sont pas dépendants. Dans ce cas, nous utilisons pour *A* et *B* leurs *AAOPs*, et pour les attributs *C* et *D*, nous utilisons leurs *AAPs*.

Nous avons testé cette approche sur plusieurs bases de données réelles et pour plusieurs seuils de dépendance. Les attributs dans les bases prennent des valeurs qualitatives et discrètes. Les attributs discrets sont traités comme des attributs qualitatifs en considérant chaque valeur comme une modalité.

Nous avons comparé nos résultats de classement avec ceux donnés par C4.5 et *AAO*. Nous avons montré que notre approche donne de meilleurs résultats quand les attributs sont dépendants. En revanche, quand les attributs sont indépendants, nos résultats de classement n'offrent pas d'améliorations par rapport aux résultats de C4.5 et *AAO*, puisque dans ce cas chacun de

nos arbres est constitué d'un seul nœud-feuille avec sa distribution de probabilités dans la base d'apprentissage. Or, dans la réalité, il est rare que les données soient indépendantes, et nous avons montré que la prise en compte des dépendances améliorait de manière sensible les performances du classement.

Dans un deuxième temps, nous avons proposé un algorithme statistique, appelé *Analyser-Instance*, qui est une implantation de la méthode des k plus proches voisins. Cet algorithme calcule pour chaque instance de la base de test la fréquence de ses instances les plus proches dans la base d'apprentissage. Notre expérimentation a montré que les résultats de l'approche AAP sont proches du résultat de l'algorithme *Analyser-Instance* pour certains seuils. Cet algorithme prend en compte les valeurs manquantes dans l'objet à classer lors du calcul de ses fréquences, sans traiter les valeurs manquantes elles-mêmes.

Dans le dernier chapitre de ce mémoire, nous avons calculé la complexité de notre algorithme de classement. Nous l'avons comparée avec la complexité de classement obtenue en utilisant C4.5 et AAO en présence de valeurs manquantes. Nous avons trouvé que la complexité de notre algorithme est exponentielle en fonction du nombre d'attributs manquants dans l'instance à classer.

Ainsi, si le nombre des valeurs manquantes est égal à 1 et si l'attribut manquant est la racine de l'arbre de décision final, la complexité de notre approche est proche de la complexité de C4.5 lors du classement. De plus, les résultats de classement de AAP et de AAO sont meilleurs que ceux obtenus par C4.5 parce que celui-ci considère la fréquence de cet attribut dans toute la base d'apprentissage sans prendre en compte les dépendances éventuelles.

Si l'attribut manquant n'est pas la racine de l'arbre, notre approche donne un résultat de classement meilleur que celui de C4.5 et AAO parce que AAO n'utilise pas forcément tous les attributs qui dépendent de l'attribut inconnu, et C4.5 calcule sa probabilité dans le sous-ensemble d'apprentissage associé à son nœud.

La complexité de notre algorithme de classement est plus élevée que celles de C4.5 et AAO, mais ses performances sont meilleures.

Retour sur les arbres de décision probabilistes

En 2005 et 2006, la conférence internationale IEEE en Data Mining (ICDM) a lancé deux propositions pour identifier les dix meilleurs algorithmes utilisés en Data Mining, et C4.5 a été identifié comme la méthode la plus utilisée pour le classement[76]. Ainsi, C4.5 semble être la méthode la plus efficace pour le classement en l'absence de valeurs manquantes.

Dans cette thèse, nous n'avons pas traité la question de l'amélioration de l'arbre de décision en estimation de probabilité mais nous avons étudié les arbres de décision probabilistes pour traiter le problème des valeurs manquantes dans les objets à classer. Nous avons utilisé les arbres de décision probabilistes sous leur forme la plus simple, en gardant sur chaque feuille les valeurs de la classe avec leurs probabilités.

Le problème de l'estimation de probabilité en utilisant un arbre de décision probabiliste est délicat puisque d'une part l'algorithme de construction cherche à trouver l'arbre le plus petit en appliquant la stratégie d'élagage, ce qui diminue le nombre d'attributs utilisés pour la construction, et d'autre part, le fait de laisser l'arbre croître jusqu'au bout conduit à un nombre

[76]Les algorithmes candidats en classement étaient C4.5, CART, K plus proches voisins et Naïve Bayes. `http://www.cs.uvm.edu/~icdm/algorithms/index.shtml`. C4.5 est classé premier.

d'instances assez petit par feuille, ce qui donne une estimation de probabilité non fiable. Dans notre travail, pour améliorer la probabilité estimée sur chaque feuille dans l'arbre de décision, nous avons utilisé une stratégie de pré-élagage pour avoir les attributs pertinents dans chaque arbre construit.

Toutefois, le fait d'utiliser un seul arbre de décision pour classer l'objet ayant des valeurs manquantes réduit la capacité de cet arbre en classement, et l'estimation de probabilité donnée dans ce cas devient très faible. Parce que C4.5 prend la fréquence de l'attribut manquant dans le sous-ensemble associé au nœud où cet attribut est inconnu, C4.5 n'est pas efficace pour classer un objet ayant des valeurs manquantes. De plus, les distributions de probabilités obtenues avec C4.5 sont éloignées des résultats obtenus avec l'algorithme *Analyser-Instance*.

Avec notre approche, nous associons des arbres de décision probabilistes à chaque attribut dans la base d'apprentissage, ces arbres vont servir à calculer la distribution de probabilités de l'attribut manquant. Nous pouvons conclure que la combinaison des distributions de probabilités données par ces arbres pour classer l'objet conduit à un résultat de classement plus précis et plus fin que celui donné par C4.5.

Perspectives

Nous envisageons plusieurs directions pour la poursuite de ce travail.

D'une part, nous souhaiterions approfondir le travail sur les arbres de décision probabilistes pour améliorer leur performance en estimation de probabilité, sans changer la structure de l'arbre, et en prenant en compte le fait qu'il faut avoir un nombre minimum d'objets par feuille et qu'il faut utiliser le maximum d'attributs pertinents.

D'autre part, nous pensons étudier en détail les méthodes statistiques qui traitent les valeurs manquantes dans les données et les comparer avec notre approche. Il serait intéressant de comparer les distributions de probabilités obtenues par notre approche avec celles d'autres méthodes de fouille de données, comme les règles d'association [Ragel et Cremilleux, 1998, Tsumoto *et al.*, 2005] et les réseaux bayésiens [Naïm *et al.*, 2004, Heckerman, 1997].

Il n'existe pas à notre connaissance une "meilleure" méthode pour la gestion des valeurs manquantes, qui soit applicable à toutes les bases de données. Nous avons montré que les relations entre les attributs changent d'une base à l'autre. Il existe des bases où la corrélation entre les attributs est forte et d'autres où on rencontre des attributs indépendants.

Ainsi, pour que la méthode des *Arbres d'Attributs Ordonnés* soit applicable, les attributs des bases d'apprentissage doivent vérifier certaines conditions ; notre approche fonctionne mieux que *AAO* et C4.5 quand la corrélation entre les attributs est forte ; la méthode de majorité n'est applicable que pendant la phase d'apprentissage.

Dans notre travail, nous avons montré comment construire deux familles d'arbres pour les attributs dans la base d'apprentissage, ainsi que l'arbre de décision final. Puisque la complexité de classement avec notre approche est exponentielle en fonction du nombre d'attributs manquants dans l'objet à classer, nous cherchons des moyens pour diminuer cette complexité et augmenter le taux de bon classement probabiliste.

Nous pensons que construire une seule famille d'arbres *AAPs* qui prenne en compte les dépendances éventuelles entre les attributs peut apporter des simplifications significatives. Nous avons également été confrontés à un problème de cycle quand deux attributs sont dépendants et manquants en même temps. Pour éviter ce problème, nous pensons à une solution hybride qui combine la famille des *arbres d'attributs probabilistes AAP* avec l'algorithme *Analyser-Instance*,

qui donne un résultat proche de celui de notre approche. Ainsi, pour l'attribut le moins dépendant de la classe, nous proposons de calculer sa distribution de probabilités en appelant l'algorithme *Analyser-Instance* et pour l'attribut le plus dépendant de la classe utiliser son AAP.

Nous pensons que ces améliorations sont susceptibles de diminuer la complexité de calcul et aussi augmenter la qualité de l'estimation probabiliste.

Nous avons dans ce travail abordé un des aspects du vaste problème de la gestion des valeurs manquantes dans les données, qui dépasse largement le cadre des arbres de décision. Ce problème, qui se répète sur toutes les étapes du processus d'ECD, fait partie de thèmes traités en Fouille de Données Complexes.

Annexe A

A.1 La validation Croisée

La validation croisée est une méthode d'estimation non biaisée du taux d'erreur [Cornuéjols *et al.*, 2002]. L'idée de la validation croisée (N-fold cross-validation) consiste à :

- Diviser les données d'apprentissage S en N sous-échantillons de taille égale.
- Retenir l'un de ces échantillons, disons de numéro i, pour le test et construire le classifieur en utilisant les *N-1* autres.
- Mesurer le taux d'erreur empirique e_i sur l' échantillon i.
- Recommencer n fois en faisant varier l'échantillon i de 1 à N.

L'erreur estimée finale est donnée par la moyenne des erreurs mesurées.

$$e = \frac{1}{N} \sum_{i=1}^{N} e_i$$

A.2 Paired test

C'est un test statistique pour comparer deux algorithmes d'apprentissage sur la même base de test. Soient deux hypothèses de classement h_1 et h_2. Notons :

- n_{00} le nombre d'exemples mal classés par h_1 et h_2.
- n_{01} le nombre d'exemples mal classés par h_1, mais pas par h_2.
- n_{10} le nombre d'exemples mal classés par h_2, mais pas par h_1.
- n_{11} le nombre d'exemples correctement classés par h_1 et h_2.

Considérons alors la statistique Z connue sous le nom de test couplé (paired test) de McNemar [Dietterich, 1998, Cornuéjols *et al.*, 2002] :

$$z = \frac{|n_{01} - n_{10}|}{\sqrt{n_{10} + n_{01}}}$$

L'hypothèse que h_1 et h_2 aient le même taux d'erreur peut être rejetée avec une probabilité supérieure à 95% si $|z| > 1.96$.

Pour plus de détails sur les tests statistiques utilisés pour comparer les algorithmes d'apprentissages supervisés, nous vous renvoyons à [Dietterich, 1998].

Annexe B

Les matrices de confusion

B.1 La matrice de confusion pour la base *Vote*

a	b	<-classified as
154	18	a = democrat
2	66	b = republican

Tab. B.1 – La matrice de Confusion de AAP de la base *Vote* pour un seuil 0.2

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
0.8958	0.029	0.987	0.895	0.936	democrat
0.971	0.105	0.786	0.971	0.868	republican

Tab. B.2 – Les détails des résultats pour AAP sur la base *Vote*

a	b	<-classified as
166	6	a = democrat
34	34	b = republican

Tab. B.3 – La matrice de Confusion de C4.5 sur la base *Vote*

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
0.965	0.5	0.83	0.965	0.892	democrat
0.5	0.035	0.85	0.5	0.63	republican

TAB. B.4 – Les détails des résultats de C4.5 sur la base *Vote*

a	b	<-classified as
153	19	a = democrat
1	67	b = republican

TAB. B.5 – La matrice de Confusion de AAO sur la base *Vote*

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
0.89	0.015	0.994	0.89	0.939	democrat
0.985	0.11	0.779	0.985	0.87	republican

TAB. B.6 – Les détails des résultats de AAO sur la base *Vote*

B.2 La matrice de confusion sur la base *Breast-cancer*

a	b	<–classified as
62	4	| a = no-recurrence-events
16	10	| b = recurrence-events

TAB. B.7 – La matrice de Confusion de AAP sur la base *Breast-cancer* pour un seuil 0.01

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
0.939	0.615	0.795	0.939	0.861	no-recurrence-events
0.385	0.061	0.714	0.385	0.5	recurrence-events

TAB. B.8 – Les détails des résultats de AAP sur la base *Breast-cancer*

a	b	<–classified as
63	3	| a = no-recurrence-events
24	2	| b = recurrence-events

TAB. B.9 – La matrice de Confusion de C4.5 sur la base *Breast-cancer*

B.3 La matrice de confusion pour la base *Lymphography*

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
0.955	0.923	0.724	0.955	0.824	no-recurrence-events
0.077	0.045	0.4	0.077	0.129	recurrence-events

TAB. B.10 – Les détails des résultats de C4.5 sur la base *Breast-cancer*

a	b	<–classified as
60	6	| a = no-recurrence-events
21	5	| b = recurrence-events

TAB. B.11 – La matrice de Confusion de AAO sur la base *Breast-cancer*

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
0.909	0.808	0.741	0.909	0.816	no-recurrence-events
0.192	0.091	0.455	0.192	0.27	recurrence-events

TAB. B.12 – Les détails des résultats de AAO sur la base *Breast-cancer*

a	b	c	d	<–classified as
0	0	0	0	| a = normal
0	35	4	0	| b = metastases
0	1	20	0	| c = malign-lymph
0	0	0	2	| d = fibrosis

TAB. B.13 – La matrice de Confusion de AAP sur la base *Lymphography* seuil 0.06

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
0	0	0	0	0	normal
0.897	0.043	0.972	0.897	0.933	metastases
0.952	0.098	0.833	0.952	0.889	malign-lymph
1	0	1	1	1	fibrosis

TAB. B.14 – Les détails des résultats de AAP

a	b	c	d	<–classified as
0	0	0	0	| a = normal
0	35	4	0	| b = metastases
0	9	12	0	| c = malign-lymph
0	0	0	2	| d = fibrosis

TAB. B.15 – La matrice de Confusion de C4.5 sur la base *Lymphography*

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
0	0	0	0	0	normal
0.897	0.3691	0.795	0.897	0.843	metastases
0.571	0.098	0.75	0.571	0.649	malign-lymph
1	0	1	1	1	fibrosis

TAB. B.16 – Les détails des résultats de C4.5

a	b	c	d	<–classified as
0	0	0	0	a = normal
0	33	6	0	b = metastases
0	7	14	0	c = malign-lymph
0	0	0	2	d = fibrosis

TAB. B.17 – La matrice de Confusion de AAO sur la base *Lymphography*

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
0	0	0	0	0	normal
0.846	0.304	0.825	0.846	0.835	metastases
0.667	0.146	0.7	0.667	0.682	malign-lymph
1	0	1	1	1	fibrosis

TAB. B.18 – Les détails des résultats de AAO

B.4 La matrice de confusion pour la base *splice*

a	b	c	<–classified as
26	1	1	a = EI
1	43	1	b = IE
0	0	37	c = N

TAB. B.19 – La matrice de Confusion de AAP sur la base *Splice* pour un seuil 0.03

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
0.929	0.012	0.963	0.929	0.945	EI
0.956	0.015	0.977	0.956	0.966	IE
1	0.027	0.949	1	0.974	N

TAB. B.20 – Les détails des résultats de AAP

a	b	c	<–classified as
25	1	2	a = EI
2	41	2	b = IE
0	0	37	c = N

TAB. B.21 – La matrice de Confusion de C4.5 sur la base *Splice*

B.5 La matrice de confusion pour la base *Car*

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
0.893	0.024	0.926	0.893	0.909	EI
0.911	0.015	0.976	0.911	0.943	IE
1	0.055	0.902	1	0.949	N

TAB. B.22 – Les détails des résultats de C4.5

a	b	c	<–classified as
28	0	0	| a = EI
1	42	2	| b = IE
0	0	37	| c = N

TAB. B.23 – La matrice de Confusion de AAO sur la base *Splice*

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
1	0.012	0.966	1	0.982	EI
0.933	0	1	0.933	0.966	IE
1	0.027	0.949	1	0.974	N

TAB. B.24 – Les détails des résultats de AAO

a	b	c	d	<–classified as
1793	0	0	|	a = unacc
7	24	0	0 |	b = acc
2	8	2	0 |	c = good
2	2	0	7 |	d = vgood

TAB. B.25 – La matrice de Confusion de AAP sur la base *Car* seuil 0.01

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
0.984	0.204	0.942	0.984	0.962	unacc
0.774	0.063	0.649	0.774	0.706	acc
0.167	0	1	0.167	0.286	good
0.636	0	1	0.636	0.778	vgood

TAB. B.26 – Les détails des résultats de AAP sur la base *Car*

a	b	c	d	<–classified as
1793	0	0	|	a = unacc
5	26	0	0 |	b = acc
2	8	2	0 |	c = good
2	1	0	8 |	d = vgood

TAB. B.27 – La matrice de Confusion de C4.5 sur la base *Car*

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
0.984	0.167	0.952	0.984	0.968	unacc
0.839	0.059	0.684	0.839	0.754	acc
0.167	0	1	0.167	0.286	good
0.727	0	1	0.727	0.842	vgood

TAB. B.28 – Les détails des résultats de C4.5 sur la base *Car*

a	b	c	d	<–classified as
1820	0	0	|	a = unacc
20	11	0	0 |	b = acc
5	7	0	0 |	c = good
3	7	0	1 |	d = vgood

TAB. B.29 – La matrice de Confusion de AAO sur la base *Car*

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
1	0.519	0.867	1	0.926	unacc
0.355	0.068	0.44	0.355	0.393	acc
0	0	0	0	0	good
0.091	0	1	0.091	0.167	vgood

TAB. B.30 – Les détails des résultats de AAO sur la base *Car*

B.6 La matrice de confusion pour la base *Nursery*

a	b	c	d	e	<–classified as
26	0	0	8	8	a = not-recom
0	0	0	0	0	b = recommend
2	0	1	3	0	c = very-recom
2	0	0	34	3	d = priority
4	0	0	8	27	e = spec-prior

TAB. B.31 – La matrice de Confusion de AAP sur la base *Nursery*

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
0.619	0.095	0.765	0.619	0.684	not-recom
0	0	0	0	0	recommend
0.167	0	1	0.167	0.286	very-recom
0.872	0.218	0.642	0.872	0.739	priority
0.692	0.126	0.711	0.692	0.701	spec-prior

TAB. B.32 – Les détails des résultats de AAP sur la base *Nursery*

a	b	c	d	e	<–classified as
26	0	0	8	8	a = not-recom
0	0	0	0	0	b = recommend
2	0	1	3	0	c = very-recom
2	0	0	34	3	d = priority
2	0	0	10	27	e = spec-prior

TAB. B.33 – La matrice de Confusion de C4.5 sur la base *Nursery*

B.7 La matrice de confusion pour la base *Iris*

B.8 La matrice de confusion pour la base *Breast-w*

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
0.619	0.071	0.813	0.619	0.703	not-recom
0	0	0	0	0	recommend
0.167	0	1	0.167	0.286	very-recom
0.872	0.241	0.618	0.872	0.723	priority
0.692	0.126	0.711	0.692	0.701	spec-prior

TAB. B.34 – Les détails des résultats de C4.5 sur la base *Nursery*

a	b	c	d	e	<–classified as
24	0	1	12	5	a = not-recom
0	0	0	0	0	b = recommend
0	0	6	0	0	c = very-recom
0	0	3	36	0	d = priority
0	0	0	14	25	e = spec-prior

TAB. B.35 – La matrice de Confusion de AAO sur la base *Nursery*

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
0.571	0	1	0.571	0.727	not-recom
0	0	0	0	0	recommend
1	0.033	0.6	1	0.75	very-recom
0.923	0.299	0.581	0.923	0.713	priority
0.641	0.057	0.833	0.641	0.725	spec-prior

TAB. B.36 – Les détails des résultats de AAO sur la base *Nursery*

a	b	c	<–classified as
11	0	0	a = Iris-setosa
0	7	1	b = Iris-versicolor
0	1	31	c = Iris-virginica

TAB. B.37 – La matrice de Confusion de AAP sur la base *Iris*

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
1	0	1	1	1	Iris-setosa
0.875	0.023	0.875	0.875	0.875	Iris-versicolor
0.969	0.053	0.969	0.969	0.969	Iris-virginica

TAB. B.38 – Les détails des résultats de AAP sur la base *Iris*

a	b	c	<–classified as
11	0	0	a = Iris-setosa
3	5	0	b = Iris-versicolor
17	1	14	c = Iris-virginica

TAB. B.39 – La matrice de Confusion de C4.5 sur la base *Iris*

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
1	0.5	0.355	1	0.524	Iris-setosa
0.625	0.023	0.833	0.625	0.714	Iris-versicolor
0.438	0	1	0.438	0.609	Iris-virginica

TAB. B.40 – Les détails des résultats de C4.5 sur la base *Iris*

a	b	c	<–classified as
11	0	0	| a = Iris-setosa
0	7	1	| b = Iris-versicolor
0	1	31	| c = Iris-virginica

TAB. B.41 – La matrice de Confusion de AAO sur la base *Iris*

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
1	0	1	1	1	Iris-setosa
0.875	0.023	0.875	0.875	0.875	Iris-versicolor
0.969	0.053	0.969	0.969	0.969	Iris-virginica

TAB. B.42 – Les détails des résultats de AAO sur la base *Iris*

a	b	<–classified as
50	4	| a = benign
2	32	| b = malignant

TAB. B.43 – La matrice de Confusion de AAP sur la base *Breast-w* pour un seuil 0.2

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
0.926	0.059	0.962	0.926	0.943	benign
0.941	0.074	0.889	0.941	0.914	malignant

TAB. B.44 – Les détails des résultats de AAP sur la base *Breast-w*

a	b	<–classified as
48	6	| a = benign
16	18	| b = malignant

TAB. B.45 – La matrice de Confusion de C4.5 sur la base *Breast-w*

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
0.889	0.471	0.75	0.889	0.814	benign
0.529	0.111	0.75	0.529	0.621	malignant

TAB. B.46 – Les détails des résultats de C4.5 sur la base *Breast-w*

a	b	<–classified as
46	8	a = benign
0	34	b = malignant

TAB. B.47 – La matrice de Confusion de AAO sur la base *Breast-w*

TP Rate	FP Rate	Precision	Recall	F-Measure	Class
0.852	0	1	0.852	0.92	benign
1	0.148	0.81	1	0.895	malignant

TAB. B.48 – Les détails des résultats de AAO sur la base *Breast-w*

Bibliographie

[Adriaans et Zantinge, 1996] Adriaans, P. et Zantinge, D. (1996). *Data Mining*. Addison-Wesley.

[Aho et Ullman, 1993] Aho, A. et Ullman, J. (1993). *Concepts fondamentaux de l'informatique*. Dunod, Paris.

[Boussaid *et al.*, 2005] Boussaid, O., Gançarski, P., Masseglia, F., Trousse, B., Venturini, G. et Zighed, D. A. (2005). *Fouille de donnés complexes*, volume Vol. 7. Revue des Nouvelles Technologies de l'Information (RNTI-E-4). Cépaduès Editions.

[Bratko et Knononenko, 1987] Bratko, I. et Knononenko, I. (1987). Learning diagnostic rules from incomplete and noisy data,. *Interactions in Artificial Intelligence and Statistical Methods*.

[Breiman *et al.*, 1984] Breiman, L., Friedman, J. H., Olshen, R. A. et Stone, C. J. (1984). *Classification and regression trees*. CA : Wadsworth International Group.

[Buntine, 1993] Buntine, W. (1993). Learning classification trees. *In* Hand, D. J., éditeur : *Artificial Intelligence frontiers in statistics*, pages 182–201. Chapman & Hall,London.

[Cestnik et Bratko, 1991] Cestnik, B. et Bratko, I. (1991). On estimating probabilities in tree pruning. *In EWSL-91 : Proceedings of the European working session on learning on Machine learning*, pages 138–150, New York, NY, USA. Springer-Verlag New York, Inc.

[Cestnik *et al.*, 1987] Cestnik, B., Knononenko, I. et Bratko, I. (1987). Assistant 86 : A knowledge elicitation tool for sophisticated users. *In I. Bratko and N. Lavrac, editors, Progress in Machine Learning, Sigma Press, Wilmslow, UK*, pages 31–45.

[Clark et Niblett, 1989] Clark, P. et Niblett, T. (1989). The cn2 induction algorithm. *Machine Learning*, 3(4):261–283.

[Cloppet *et al.*, 2005] Cloppet, F., Petit, J.-M. et Vincent, N. (2005). *Extraction des connaissances : Etat et perspectives (Ateliers de la conférence EGC'2005)*. Revue des Nouvelles Technologies de l'Information (RNTI-E-5). Cépaduès Editions.

[Cornuéjols *et al.*, 2002] Cornuéjols, A., Miclet, L. et Kodratoff, Y. (2002). *Apprentissage artificiel : Concepts et algorithmes*. Eyrolles.

[Cover et Hart, 1967] Cover, T. et Hart, P. (1967). Nearest neighbor pattern classification. 13:21– 27.

[Crémilleux, 1991] Crémilleux, B. (1991). *Induction automatique : aspects théoriques, le système ARBRE, Applications en médecine*. Thèse de doctorat, Universit é Joseph Fourier.

[Crémilleux, 2000] Crémilleux, B. (2000). Decision trees as a data mining tool. *Computing and Information Systems*, pages pp. 91–97.

[Crémilleux et Robert, 2000] Crémilleux, B. et Robert, C. (2000). Use of attribute selection criteria in decision trees in uncertain domains. *In Uncertainty in Intelligent and Information Systems, Advances in Fuzzy Systems, Applications and Theory*, volume vol. 20, pages pp. 150–161.

[Delavallade et Dang, 2007] Delavallade, T. et Dang, T. H. (2007). Using entropy to impute missing data in a classification task. *In FUZZ-IEEE*, pages 1–6. IEEE.

[Dietterich, 1998] Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10:1895–1923.

[Domingos et Provost, 2000] Domingos, P. et Provost, F. (2000). Well-trained pets : Improving probability estimation trees.

[Dougherty *et al.*, 1995] Dougherty, J., Kohavi, R. et Sahami, M. (1995). Supervised and unsupervised discretization of continuous features. *In International Conference on Machine Learning*, pages 194–202.

[Esposito *et al.*, 1997] Esposito, F., Malerba, D. et Semeraro, G. (1997). A comparative analysis of methods for pruning decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):476–491.

[Fayyad *et al.*, 1996] Fayyad, U., Piatetsky-shapiro, G. et Smyth, P. (1996). From data mining to knowledge discovery in databases. *AI Magazine*, 17:37–54.

[Fayyad et Irani, 1992] Fayyad, U. M. et Irani, K. B. (1992). On the handling of continuous-valued attributes in decision tree generation. *Machine Learning*, 8:87–102.

[Fortesa *et al.*, 2006] Fortesa, I., Mora-Lopezb, L., Moralesb, R. et Triguerob, F. (2006). Inductive learning models with missing values. *Mathematical and Computer Modelling*, 44. Issues 9-10.

[Fournier, 2001] Fournier, D. (2001). *Étude de la qualité de données à partir de l'apprentissage automatique. Application aux arbres d'induction.* Thèse de doctorat, Université de Caen.

[Fournier et Crémilleux, 2000] Fournier, D. et Crémilleux, B. (2000). Using impurity and depth for decision trees pruning. *Second International ICSC Symposium on Engineering of Intelligent Systems (EIS'00)*, pages pp. 320–326.

[Fournier et Crémilleux, 2002] Fournier, D. et Crémilleux, B. (2002). A quality index for decision tree pruning. *Knowledge-Based Systems journal*, 15:37–43.

[Friedman *et al.*, 1996] Friedman, J. H., Kohavi, R. et Yun, Y. (1996). Lazy decision trees. *In* Shrobe, H. et Senator, T., éditeurs : *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference*, pages 717–724, Menlo Park, California. AAAI Press.

[Grzymala-Busse et Hu, 2001] Grzymala-Busse, J. W. et Hu, M. (2001). A comparison of several approaches to missing attribute values in data mining. *In RSCTC '00 : Revised Papers from the Second International Conference on Rough Sets and Current Trends in Computing*, pages 378–385, London, UK. Springer-Verlag.

[Hawarah *et al.*, 2004] Hawarah, L., Simonet, A. et Simonet, M. (2004). A probabilistic approach to classify incomplete objects using decision trees. *DEXA Spain*, LNCS, 3180:549–558.

[Hawarah *et al.*, 2005] Hawarah, L., Simonet, A. et Simonet, M. (2005). Classement d'objets incomplets dans un arbre de décision probabiliste. *Deuxième atelier sur la fouille de données complexes dans un processus d'extraction des connaissances, EGC.*

[Hawarah *et al.*, 2006a] Hawarah, L., Simonet, A. et Simonet, M. (2006a). The complexity of a probabilistic approach to deal with missing values in a decision tree. *(SYNASC06) Romania,September 26-29*.

[Hawarah *et al.*, 2006b] Hawarah, L., Simonet, A. et Simonet, M. (2006b). Dealing with missing values in a probabilistic decision tree during classification. *In ICDMW '06 : Proceedings of the Sixth IEEE International Conference on Data Mining - Workshops*, pages 325–329.

[Hawarah *et al.*, 2006c] Hawarah, L., Simonet, A. et Simonet, M. (2006c). Evaluation d'une approche probabiliste pour le classement d'objets incomplètement connus dans un arbre de décision. *Troisième atelier sur la fouille de données complexes dans un processus d'extraction des connaissances, EGC, Lille*.

[Hawarah *et al.*, 2006d] Hawarah, L., Simonet, A. et Simonet, M. (2006d). Evaluation of a probabilistic approach to classify incomplete objects using decision trees. *DEXA Poland*, LNCS 4080:193–202.

[Heckerman, 1997] Heckerman, D. (1997). Bayesian networks for data mining. *Data Min. Knowl. Discov.*, 1(1):79–119.

[Holte, 1993] Holte, R. C. (1993). Very simple classification rules perform well on most commonly used datasets. *Mach. Learn.*, 11(1):63–90.

[Hunt, 1962] Hunt, E. (1962). *Concept Learning : An Information Processing Problem*. New york : Wiley.

[Hunt *et al.*, 1966] Hunt, E., Marin, J. et Stone, P. (1966). *Experiments in Induction*. New york : Academic Press.

[Kass, 1980] Kass, G. V. (1980). An exploratory technique for investigating large quantities of categorical data. *Journal of the American Statistical Association*, Vol. 29(No. 2):119–127.

[Kerber, 1992] Kerber, R. (1992). Chimerge : Discretization of numeric attributes. *In Proc. of AAAI-92*, pages 123–128, San Jose, CA.

[Kira et Rendell, 1992] Kira, K. et Rendell, L. A. (1992). A practical approach to feature selection. *In ML92 : Proceedings of the ninth international workshop on Machine learning*, pages 249–256, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Kohavi, 1996] Kohavi, R. (1996). Scaling up the accuracy of Naive-Bayes classifiers : a decision-tree hybrid. *In Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 202–207.

[Kohavi et Kunz, 1997] Kohavi, R. et Kunz, C. (1997). Option decision trees with majority votes. *In* Fisher, D., éditeur : *Machine Learning : Proceedings of the Fourteenth International Conference*. Morgan Kaufmann Publishers, Inc.

[Kohavi et Provost, 1998] Kohavi, R. et Provost, F. (1998). Glossary of terms. *Editorial for the Special Issue on Applications of Machine Learning and the Knowledge Discovery Process*, 30:271–274.

[Kononenko, 1994] Kononenko, I. (1994). Estimating attributes : Analysis and extensions of relief. *In ECML : European Conference on Machine Learning*, pages 171–182. Springer.

[Kononenko, 1998] Kononenko, I. (1998). The minimum description length based decision tree pruning. *In PRICAI'98 : Proceedings of the 5th Pacific Rim International Conference on Artificial Intelligence*, pages 228–237, London, UK. Springer-Verlag.

[Kononenko *et al.*, 1984] Kononenko, I., Bratko, I. et Roskar, E. (1984). Experiments in automatic learning of medical diagnostic rules. Technical report, Jozef Stefan Institute, Ljubljana, Yugoslavia.

[Kononenko et Robnik-Sikonia, 1996] Kononenko, I. et Robnik-Sikonia, M. (1996). Relieff for estimation and discretization of attributes in classification regression and ilp problems. *In Ramsay, A., ed., Artificial Intelligence : Methodology, Systems, Applications*, pages 31–40. IOS Press.

[Kononenko et Simec, 1995] Kononenko, I. et Simec, E. (1995). Induction of decision trees using relieff. *In : Della Riccia, G., Kruse, R., Viertl, R., (eds.) : Mathematical and statistical methods in artificial intelligence, Springer Verlag.*

[Kovacic, 1994] Kovacic, M. (1994). *Stochastic inductive logic programming.* Kovacic m. (1994) stochastic inductive logic programming, ph.d. thesis, University of Ljubljana, Faculty of electr. eng. computer sc., Ljubljana, Slovenia.

[Liang et Yan, 2006] Liang, H. et Yan, Y. (2006). Learning naïve bayes tree for conditional probability estimation. *In Canadian Conference on AI*, pages 455–466.

[Ling et Yan, 2003] Ling, C. et Yan, R. (2003). Decision tree with better ranking. *In In Proc. Int. Conf. on Machine Learning (ICML2003), AAAI Press,.*

[Liu *et al.*, 1997] Liu, W. Z., White, A. P., Thompson, S. G. et Bramer, M. A. (1997). Techniques for dealing with missing values in classification. *LNCS*, 1280:527–537.

[Lobo et Numao, 1999] Lobo, O. et Numao, M. (1999). Ordered estimation of missing values. *In PAKDD '99 : Proceedings of the Third Pacific Asia Conference on Methodologies for Knowledge Discovery and Data Mining*, pages 499–503, London, UK. Springer-Verlag.

[Lobo et Numao, 2000] Lobo, O. et Numao, M. (2000). Ordered estimation of missing values for propositional learning. *the Japanese Society for Artificial Intelligence*, 15(1):162–168.

[Lobo et Numao, 2001] Lobo, O. et Numao, M. (2001). Suitable domains for using ordered attribute trees to impute missing values. *IEICE TRANS.INF. and SYST.*, E84-D(2).

[Lobo, 2000] Lobo, O. O. (2000). *Dealing with Imperfections in the Data for Propositional Learning.* Doctor, Graduate School of the Tokyo Institute of Technology.

[MacKay, 2003] MacKay, D. J. (2003). *Information Theory, Inference, and Learning Algorithms.* Cambridge University Press.

[Margineantu et Dietterich, 2002] Margineantu, D. et Dietterich, T. G. (2002). Improved class probability estimates from decision tree models. *In Nonlinear Estimation and Classification*, Lecture Notes in Statistics 171.

[Martin et Hirschberg, 1995] Martin, J. K. et Hirschberg, D. S. (1995). The time complexity of decision tree induction. Rapport technique ICS-TR-95-27.

[Mehta *et al.*, 1995] Mehta, M., Rissanen, J. et Agrawal, R. (1995). Mdl-based decision tree pruning. *In Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD'95)*, pages 216–221.

[Michalski, 1969] Michalski, R. S. (1969). On the quasiminimal solution of the general covering problem. *in Proc. 5th Int. Symp. Information Processing*, pages 125–128.

[Mingers, 1987] Mingers, J. (1987). Expert systems-rule induction with statistical data. *The Journal of the Operational Research Society.*, 38(1):39–47.

[Mingers, 1989] Mingers, J. (1989). An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4(2):227–243.

[Mitchell, 1997] Mitchell, T. (1997). *Machine Learning.* McGraw-Hill.

[Morgan et Messenger, 1973] Morgan, J. N. et Messenger, R. C. (1973). *THAID, a sequential analysis program for the analysis of nominal scale dependent variables.* Survey Research Center, Institute for Social Research, University of Michigan.

[Morgan et Sonquist, 1963] Morgan, J. N. et Sonquist, J. A. (1963). Problems in the analysis of survey data, and a proposal. *Journal of the American Statistical Association*, Vol. 58(No. 302):415–434.

[Naïm *et al.*, 2004] Naïm, P., Wuillemin, P.-H., Leray, P., Pourret, O. et Becker, A. (2004). *Réseaux bayésiens.* Eyrolles.

[Newman *et al.*, 1998] Newman, D., Hettich, S., Blake, C. et Merz, C. (1998). Uci repository of machine learning databases : http ://www.ics.uci.edu/ mlearn/mlrepository.html.

[Niblett et Bratko., 1986] Niblett, T. et Bratko., I. (1986). Learning decision rules in noisy domains. *Research and Development in Expert Systems III*, pages 24–25.

[Paterson et Niblett, 1982] Paterson, A. et Niblett, T. (1982). Acls manual, version 1. Rapport technique.

[Provost et Domingos, 2003] Provost, F. et Domingos, P. (2003). Tree induction for probability-based ranking. *Machine Learning*, 52(3):199–215.

[Quinlan, 1979] Quinlan, J. R. (1979). Induction of decision trees. *In D. Michie, editor, Expert Systems in the Micro Electronic Age. Edinburgh University Press*, 1:168–201.

[Quinlan, 1986] Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1:81–106.

[Quinlan, 1987a] Quinlan, J. R. (1987a). Decision trees as probabilistic classifiers. *In Proceedings of the Fourth International Workshop on Machine Learning*, pages 31–37. Irvine, CA : Morgan Kaufmann.

[Quinlan, 1987b] Quinlan, J. R. (1987b). Simplifying decision trees. *Int. J. Man-Mach. Stud.*, 27(3):221–234.

[Quinlan, 1989] Quinlan, J. R. (1989). Unknown attribute values in induction. *In Proceedings of the Sixth International Machine Learning Workshop.* Morgan Kaufmann.

[Quinlan, 1990] Quinlan, J. R. (1990). Probabilistic decision trees. *in Machine Learning : an Artificial Intelligence Approach*, 3:140–152.

[Quinlan, 1993] Quinlan, J. R. (1993). *C4.5 : Programs for Machine Learning.* Morgan Kaufmann, San Diego.

[Quinlan *et al.*, 1987] Quinlan, J. R., Compton, P. J., Horn, K. A. et Lazarus, L. (1987). Inductive knowledge acquisition : a case study. *In Proceedings of the Second Australian Conference on Applications of expert systems*, pages 137–156, Boston, MA, USA. Addison-Wesley Longman Publishing Co., Inc.

[Quinlan et Rivest, 1989] Quinlan, J. R. et Rivest, R. L. (1989). Inferring decision trees using the minimum description length principle. *Inf. Comput.*, 80(3):227–248.

[Ragel et Cremilleux, 1998] Ragel, A. et Cremilleux, B. (1998). Treatment of missing values for association rules. *In Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 258–270.

[Ragel et Crémilleux, 1999] Ragel, A. et Crémilleux, B. (1999). Mvc - a preprocessing method to deal with missing values. *Knowl.-Based Syst.*, 12(5-6):285–291.

[Rissanen, 1984] Rissanen, J. (1984). Universal coding, information, prediction, and estimation. *IEEE Transactions on Information Theory*, 30(4):629–636.

[Risvik, 1997] Risvik, K. M. (1997). Discretization of numerical attributes - preprocessing for machine learning.

[Robnik-Sikonja et Kononenko, 1999] Robnik-Sikonja, M. et Kononenko, I. (1999). Attribute dependencies, understandability and split selection in tree based models. *Machine Learning : Proceedings of the Sixteenth International Conference ICML99*, pages 344–353.

[Robnik-Sikonja et Kononenko, 2003] Robnik-Sikonja, M. et Kononenko, I. (2003). Theoretical and empirical analysis of relieff and rrelieff. *Mach. Learn.*, 53(1-2):23–69.

[Roderick et Donald, 2002] Roderick, J. A. L. et Donald, B. R. (2002). *Statistical Analysis with Missing Data, Second Edition*. Hoboken (N.J), Wiley-Interscience.

[Roman, 1992] Roman, S. (1992). *Coding and information theory*. Springer-Verlag New York, Inc., New York, NY, USA.

[Russell et Norvig, 2003] Russell, S. J. et Norvig, P. (2003). *Artificial Intelligence : A Modern Approach*. Pearson Education.

[Saar-Tsechansky et Provost, 2007] Saar-Tsechansky, M. et Provost, F. (2007). Handling missing values when applying classification models. *J. Mach. Learn. Res.*, 8:1623–1657.

[Shannon, 1948] Shannon, C. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423.

[Simonoff, 1998] Simonoff, J. S. (1998). Three sides of smoothing : Categorical data smoothing, nonparametric regression, and density estimation. *International Statistical Review* ., 66(2): 137–156.

[Smyth *et al.*, 1995] Smyth, P., Gray, A. et Fayyad, U. M. (1995). Retrofitting decision tree classifiers using kernel density estimation. *In International Conference on Machine Learning*, pages 506–514.

[Su et Zhang, 2004] Su, J. et Zhang, H. (2004). Learning conditional independence tree for ranking. *In ICDM*, pages 531–534.

[Su et Zhang, 2005] Su, J. et Zhang, H. (2005). Representing conditional independence using decision trees. *In AAAI*, pages 874–879.

[Tan *et al.*, 2006] Tan, P. N., Steinbach, M. et Kumar, V. (2006). *Introduction to Data Mining*. Addison-Wesley 769 pp.

[Tsumoto *et al.*, 2005] Tsumoto, S., Yamaguchi, T., Numao, M. et Motoda, H., éditeurs (2005). *CHASE-2 : Rule based chase algorithm for information systems of type lambda*, volume 3430 de *Lecture Notes in Computer Science*. Springer.

[Wang *et al.*, 2004] Wang, L., Yuan, S., Li, L. et Li, H. (2004). Improving the performance of decision tree : A hybrid approach. LNCS 3288:327–335.

[Wehenkel, 2001] Wehenkel, L. (2001). *Théorie de l'information et du codage*. Université de Liège, Institut Montefiore.

[White, 1987] White, A. P. (1987). Probabilistic induction by dynamic part generation in virtual trees. *In Proceedings of Expert Systems '86, The 6Th Annual Technical Conference on Research and development in expert systems III*, pages 35–46, New York, NY, USA. Cambridge University Press.

[Witten Ian et Frank, 2005] Witten Ian, H. et Frank, E. (2005). *Data Mining : Practical Machine Learning Tools and Techniques (Second Edition)*. Morgan Kaufmann.

[Zadrozny et Elkan, 2001] Zadrozny, B. et Elkan, C. (2001). Obtaining calibrated probability estimates from decision trees and naive bayesian classifiers. *In Proc. 18th International Conf. on Machine Learning*, pages 609–616. Morgan Kaufmann, San Francisco, CA.

[Zhang et Su, 2004] Zhang, H. et Su, J. (2004). Conditional independence trees. *Machine Learning : ECML 2004*, LNCS 3201:513–524.

[Zheng et Low, 1999] Zheng, Z. et Low, B. T. (1999). Classifying unseen cases with many missing values. Rapport technique, In Proceedings Third Paci Conference (PAKDD99), volume 1574 of LNAI.

[Zighed et Rakotomalala, 2000] Zighed, D. et Rakotomalala, R. (2000). *Graphes d'induction : apprentissage automatique et Data mining*. Hermes.

Résumé

Nous présentons dans cette thèse une approche probabiliste pour déterminer les valeurs manquantes des objets incomplets pendant leur classement dans les arbres de décision. Cette approche est dérivée de la méthode d'apprentissage supervisé appelée *Arbres d'Attributs Ordonnées (AAO)* proposée par Lobo et Numao en 2000, qui construit un arbre de décision pour chacun des attributs, selon un ordre croissant en fonction de l'Information Mutuelle entre chaque attribut et la classe. Notre approche étend la méthode de Lobo et Numao, d'une part en prenant en compte les dépendances entre les attributs pour la construction des arbres d'attributs, et d'autre part en fournissant un résultat de classement d'un objet incomplet sous la forme d'une distribution de probabilités (au lieu de la classe la plus probable). Nous expliquons notre méthode et nous la testons sur des bases de données réelles. Nous comparons nos résultats avec ceux donnés par la méthode C4.5 et *AAO*.

Nous proposons également un algorithme basé sur la méthode des k plus proches voisins qui calcule pour chaque objet de la base de test sa fréquence dans la base d'apprentissage. Nous comparons ces fréquences avec les résultats de classement données par notre approche, C4.5 et *AAO*. Finalement, nous calculons la complexité de construction des arbres d'attributs ainsi que la complexité de classement d'un objet incomplet en utilisant notre approche, C4.5 et *AAO*.

Mots-clés: Fouille de Données, Arbres de Décision, Valeurs Manquantes, Information Mutuelle, Classement Probabiliste.

Abstract

We describe in this thesis an approach to fill missing values in decision trees during the classification phase. This approach is derived from the *ordered attribute trees (OAT)* method, proposed by Lobo and Numao in 2000, which builds a decision tree for each attribute and uses these trees to fill the missing attribute values. It is based on the Mutual Information between the attributes and the class. Our approach extends this method by taking the dependence between the attributes into account when constructing the attributes trees, and provides a probability distribution as a result when classifying an incomplete object (instead of the most probable class). We present our approach and we test it on some real databases. We also compare our results with those given by the C4.5 method and *OAT*.

We also propose a k-nearest neighbours algorithm which calculates for each object from the test data its frequency in the learning data. We compare these frequencies with the classification results given by our approach, C4.5 and *OAT*. Finally, we calculate the complexity of constructing the attribute trees and the complexity of classifying a new instance with missing values using our classification algorithm, C4.5 and *OAT*.

Keywords: Data Mining, Decision Trees, Missing Values, Mutual Information, Probabilistic Classification.