ÉCOLE NATIONALE SUPÉRIEURE DE L'AÉRONAUTIQUE ET DE L'ESPACE

PROJET DE FIN D'ÉTUDE

Amélioration de l'interprétabilité d'un algorithme d'induction de règles floues (OLS) et comparaison avec d'autres méthodes d'induction

Stage effectué au LASB de l'INRA de Montpellier

Par

Sébastien DESTERCKE

Sous la direction de

Brigitte CHARNOMORDIC de l'INRA et

Serge GUILLAUME du Cemargref

Année scolaire 2003-2004

Table des matières

Remerciements

Je tiens à remercier avant tout Brigitte Charnomordic et Serge Guillaume, qui m'ont guidé tout au long du travail que j'ai effectué. Leur disponibilité, leurs conseils, leur expérience, leur patience m'ont permis d'avancer à mon rythme et de ne pas me fourvoyer dans des chemins trop tortueux. Leur sympathie, leur gentillesse, leur modestie ont contribué en très grande partie à l'intérêt et à la joie que j'ai pu retirer de ce stage. C'est grâce à eux si j'ai pu aborder le domaine de la recherche et en retirer autant de plaisir.

Merci à Laurent Lardon pour m'avoir initié au secret de la digestion anaérobie avec patience et gaieté.

Merci à Vincent Fromion pour m'avoir fait partager sa vision passionnée mais néanmoins objective du monde de la recherche. J'ai toujours trouvé un grand intérêt dans les discussions que nous avons eues sur le sujet.

Merci à Jean-Pierre Vila et à sa bienveillance, sans laquelle je n'aurais peut-être pas trouvé ce stage.

Merci à Manuel Samuelides pour avoir accepté d'être mon responsable d'école.

Merci à Laurent Germain, mon responsable d'appronfondissement à Supaero et à Bernard Lecussan, mon responsable d'option dans la même école, pour avoir accepté que mon projet de fin d'étude ne soit pas en rapport direct avec les domaines abordés lors de cette année d'étude.

Merci à Pierre Manneback, professeur à la faculté polytechnique de Mons, sans qui je n'aurais pas pu passer cette année à Toulouse puis à Montpellier.

Merci à l'ensemble des rapporteurs pour leur effort et à l'ensemble des membres du jury pour leur présence.

Merci enfin à l'ensemble de l'équipe du laboratoire dans lequel j'ai été accueilli. J'ai apprécié autant leur diversité de caractères que leur gentillesse. Les pauses café et les repas étaient toujours l'occasion d'échanges intéressants sur des sujets variés et très souvent fort éloignés des préoccupations du laboratoire.

Présentation du stage

Présentation l'INRA

C'est en 1946 que l'Institut National de la Recherche Agronomique (INRA) voit le jour, pour devenir en 1984 un établissement public à caractère scientifique et technologique. Il est placé sous la double tutelle du ministère s'occupant de la recherche et de celui s'occupant de l'agriculture.

Concrètement, les actions de l'INRA sont basées sur 3 grands principes :

- Oeuvrer pour l'intérêt public, tout en veillant à maintenir un équilibre entre les exigences des demandes de la société et du monde de la recherche.
- Innover ainsi qu'augmenter et compléter les connaissances de la recherche, principalement dans les domaines de l'agriculture, de l'environnement et de l'alimentation, et assurer la diffusion de toutes ces productions.
- Apporter sa contribution au monde scientifique et technique en terme de formation, de sensibilisation, de promotion et d'expertise

De façon plus concrète, l'INRA regroupe 14 départements de recherche répartis dans toute la France à travers 200 sites. Plus de 8500 personnes sont employées par l'organisme, dont 2800 chercheurs et ingénieurs aux compétences diverses.

Présentation du centre de Montpellier

En termes de chiffres, le centre INRA de Montpellier possède 13 implantations dans tout le Languedoc-Roussillon, qui emploient en tout 650 agents, dont 300 scientifiques et ingénieurs.

C'est dans le campus de l'ENSAM (Ecole Nationale Supérieure Agronomique de Montpellier) que sont regroupés la plupart des laboratoires et services. L'ensemble fait lui-même partie du complexe international Agropolis, qui regroupe de nombreux établissements de recherche et d'enseignement supérieur.

Parmi l'ensemble des unités, un grand nombre d'entre elles sont engagées dans des activités mixtes de recherche (UMR) en partenariat avec d'autres organismes de recherche comme le Cirad, le CNRS ou encore le Cemagref.

Au niveau des activités du centre, un éventail impressionnant de disciplines est développé, ce qui lui permet de s'impliquer dans de nombreuses problématiques régionales et d'entretenir des collaborations étroites avec de nombreux acteurs des mondes agricole et agroalimentaire.

Présentation du LASB

L'ensemble du stage s'est déroulé au sein du Laboratoire d'Analyse des Systèmes et Biométrie (LASB) de l'INRA de Montpellier, avec quelques visites occasion-

nelles au Cemagref de Montpellier. Le LASB se situe au coeur du campus de l'ENSAM. L'intérêt principal du laboratoire se situe dans l'analyse et le contrôle de systèmes dynamiques d'intérêts biologiques. Dans cette optique, l'informatique, les mathématiques appliquées et l'asservissement font naturellement partie des matières qui y sont privilégiées. Les problèmes rencontrés y sont nombreux et complexes, qu'il s'agisse de modélisation, de statistique, d'aide à la décision ou d'apprentissage. De plus, dans un monde où les hommes sont de plus en plus préoccupés par l'avenir de la planète, les applications ne manquent pas (dépollution, contrôle de canaux, bioinformatique, ...). C'est dans cette atmosphère que mon stage a pris place.

Chapitre 1

Introduction

Dans de nombreux domaines, et sans doute plus particulièrement dans le domaine des procédés biologiques, le contrôle des processus ainsi que la connaissance de ces derniers revêt une importance de premier ordre. La théorie de la logique floue, quand à elle, a vu le jour pour pouvoir exprimer et représenter facilement la connaissance humaine. Suite à l'avènement de l'informatique et de l'intelligence artificelle, logique floue et méthodes d'apprentissage se sont naturellement mariées pour fournir des systèmes performants numériquement et dont il était facile d'extraire de la connaissance.

Le travail présenté ici, résultat de 4 mois passés au LASB de l'INRA de Montpellier, concerne plusieurs applications de cette théorie. A l'origine, l'objectif du stage était, dans un premier temps, de programmer une version modifiée d'un algorithme d'induction de règles appelé OLS, en vue de l'intégrer à un logiciel d'apprentissage puis de comparer les résultats de cet algorithme à d'autres méthodes d'apprentissage selon les critères de l'interprétabilité et de la performance numérique.

Au cours de l'étude de l'algorithme, de son implémentation et de sa comparaison avec d'autres méthodes, de nombreuses questions se sont posées qui sont, pour certaines d'entre elles, encore loin d'être résolues. L'ensemble du travail a de plus pu être appliqué à un problème réel de dépollution provenant du Laboratoire de Biotechnologie de l'Environnement (LBE) de Narbonne. Ce problème, à lui seul, est à l'origine de bon nombre des questions soulevées. Pour répondre à une partie de ces questions, le travail s'est rapidement étendu à l'étude d'un algorithme de simplification de règles, étude qui avait pour but de l'améliorer par de nouvelles propositions.

Le travail est organisé de la façon suivante. Le premier chapitre contient des éléments concernant la théorie de la logique floue et les méthodes d'apprentissage, éléments qui sont indispensables à la bonne compréhension de la suite. Cette partie concerne principalement les lecteurs peu familiers de ces deux théories. Le

chapitre 3, lui, contient une présentation complète de l'algorithme OLS et des modifications qui y ont été apportées en vue de le rendre interprétable, ainsi qu'une analyse des résultats obtenus sur un jeu de données connu. Il contient également un court passage sur l'implémentation de l'algorithme. Ensuite, le chapitre 4, après une présentation sommaire de différentes méthodes d'induction de règles floues, contient une comparaison et une analyse de ces méthodes. Le chapitre 5 contient, quand à lui, l'étude réalisée sur l'algorithme de simplification, suivie des propositions de changements faites en vue de l'améliorer. Le chapitre 6 présente l'application concernant le processus de dépollution ainsi que quelques résultats obtenus. Finalement, les principales conclusions sont présentées dans le chapitre 7.

Chapitre 2

La logique floue

2.1 Introduction

Cette première partie consiste en un rappel des principaux éléments de logique floue. Ces éléments seront par la suite nécessaires aux lecteurs non familiers avec la théorie de la logique floue. La section 2.2 comprend un court historique de la logique floue ainsi que des motivations à l'origine de son apparition.

La section suivante (2.3) est constituée des divers éléments théoriques de base nécessaires à la compréhension du reste de ce document.

2.2 Historique et motivations

Discipline et théorie récente (elle est apparue en 1965 à l'université de Berkeley, en Californie, sous l'impulsion de Monsieur Lofti A. Zadeh), la logique floue suscite chez les chercheurs, ingénieurs et industriels un intérêt de plus en plus vif.

Cet intérêt provient sans aucun doute du fait que la logique floue permet de manipuler des notions très difficiles à définir par des notions mathématiques simples. Ainsi, l'imprécision (exprimée par des mots comme *très, assez, environ, petit, chaud, etc.*) et l'incertitude (exprimée par des mots comme *plus grand que, il est possible que, il est improbable que, etc.*) sont deux concepts très difficiles à définir mathématiquement et auxquels nous sommes pourtant confrontés chaque jour.

Ainsi, la souplesse de la logique floue s'oppose à la rigidité de la logique booléenne, et permet donc de représenter la connaissance de façon beaucoup plus humaine. Contrairement à la logique classique où l'appartenance à un ensemble est soit tout (1), soit rien (0), la logique floue permet de construire des ensembles où la notion d'appartenance sera déterminée par une fonction (qui variera entre 0 et 1). Un élément pourra ainsi appartenir à plusieurs sous-ensembles flous (SEF)

en même temps. Par exemple, une personne pourra appartenir simultanément à l'ensemble *grand* et à l'ensemble *très grand*.

Il faut aussi noter que les sous-ensembles flous (SEF), définis par des fonctions d'appartenance, n'expriment pas du tout les mêmes notions que des fonctions de probabilité. Alors que la probabilité d'appartenir à un ensemble exprimera le pourcentage de chance d'en faire partie ou pas, le degré d'appartenance flou à ce même ensemble exprimera le niveau d'appartenance à cet ensemble. Par exemple, imaginons une bouteille d'eau, dont on veut savoir si on peut la boire ou pas. Si la probabilité qu'elle soit potable est de 0.7, la boire résultera trois fois sur dix dans l'intoxication de l'infortuné assoifé. Si, par contre, cette bouteille à un niveau d'appartenance 0.7 à l'ensemble flou *potable*, alors le même assoifé pourra boire l'eau, il sera certain de ne pas être totalement intoxiqué, mais devra en payer le prix par une légère indisposition due au fait que cette eau appartenait aussi en partie à l'ensemble flou *non potable* (par exemple, à un niveau 0.3). Une autre des différences avec la théorie des probabilités est que la somme des degré d'appartenances sur l'ensemble d'un domaine n'est pas forcément égale à un.

2.3 Eléments de logique floue et de méthodes d'apprentissage

Ci-dessous se trouvent les principales notions relatives à la logique floue ainsi que quelques notions relatives aux méthodes d'apprentissage en général. Le lecteur déjà familier avec ces théories pourra sans doute se passer de ces quelques rappels.

– Ensemble flou : Un ensemble flou est défini par sa fonction d'appartenance. Un point de l'univers, x, appartient à un ensemble, A avec un degré d'appartenance, $0 \leq \mu_A(x) \leq 1$.

  La figure 2.1 montre un ensemble de forme triangulaire.

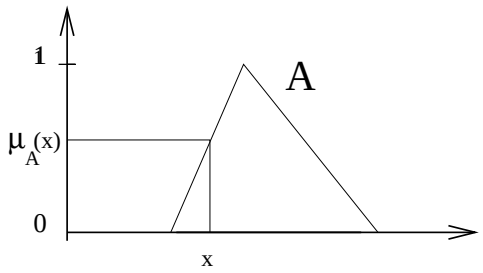


FIG. 2.1 – Un ensemble flou de forme triangulaire

– Prototype d'un ensemble : un point est un prototype d'un ensemble flou si son degré d'appartenance à cet ensemble vaut un.

– Ensemble flou normal : un ensemble flou est dit normal si au moins un de ses points est un prototype ($\exists x$ tel que $\mu(x) = 1$).

– Sous-Ensemble flou : dans la suite, la référence a des sous-ensembles flous se fera par l'abbréviation **SEF**.

– Cardinalité d'un ensemble flou : la cardinalité d'un ensemble flou correspond à la somme des degré d'appartenances des exemples (pour un ensemble discret, la cardinalité est définie comme le nombre d'exemples qui lui appartiennent).

– Opérateurs :
  – ET : opérateur de conjonction, noté $\wedge$, les plus employés sont le minimum et le produit.
  – OU : opérateur de disjonction, les plus employés sont le maximum et la somme.
  – est : la relation $x\ est\ A$ est quantifiée par le degré d'appartenance de la valeur x au sous-ensemble flou A.

– Partitionnement : Le découpage du domaine de définition d'une variable en sous-ensembles flous (SEF) est appelé partitionnement. Ces ensembles sont notés A_1, A_2, ...

– Partition floue forte : Une partition sera dite forte si $\forall x\ \exists j$, $1 \leq j \leq M^j$, tel que $\mu_{M^j}(x) \geq 0.5$, si une valeur donnée appartient au plus à 2 SEF distincts et si $\sum_{j=1}^{M^j} \mu_{M^j}(x) = 1$. La figure 2.2 illustre le concept de partition floue forte.

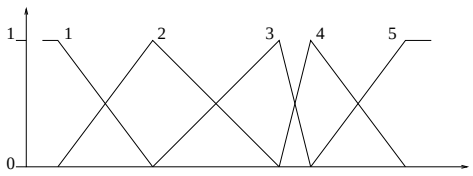


FIG. 2.2 – Une partition floue forte

– Exemple : un exemple ou individu est formé d'un vecteur d'entrée x de dimension p et, dans le cas général, d'un vecteur y, de dimension q.

– Règle floue : Une règle floue est de la forme ***Si*** *je rencontre telle situation* ***Alors*** *j'en tire telle conclusion*. La situation, appelée prémisse ou antécédent de la règle, est définie par une combinaison de relations de la forme $x\ est\ A$ pour chacune des composantes du vecteur d'entrée. La partie conclusion de la règle est appelée conséquence, ou encore simplement conclusion.

– Les règles sont de deux types :

1. Mamdani : Une règle floue de type Mamdani, dont la conclusion est un ensemble flou, s'écrit :

   $$SI\ x_1\ est\ A_1^i\ ET\ \ldots\ ET\ x_p\ est\ A_p^i\ ALORS\ y_1\ est\ C_1^i\ \ldots\ ET\ y_q\ est\ C_q^i$$

   où $A_j^i\ et\ C_j^i$ sont des ensembles flous qui définissent le partitionnement des espaces d'entrée et de sortie.

2. Takagi-Sugeno : Dans le modèle de Sugeno la conclusion de la règle est nette. Celle de la règle i pour la sortie j est calculée comme une fonction linéaire des entrées : $y_j^i = b_{jo}^i + b_{j1}^i x_1 + b_{j2}^i x_2 + \cdots + b_{jp}^i x_p$, également notée : $y_j^i = f_j^i(x)$. Dans des systèmes où l'on cherche à préserver l'interprétabilité des résultats (ce qui sera toujours le cas ici), cette conclusion se ramène à une constante et on à alors $y_j^i = b_{jo}^i$.

– Règle incomplète : Une règle floue sera dite incomplète si sa prémisse est définie par un sous-ensemble des variables d'entrée seulement. La règle, $SI\ x_2\ est\ A_2^1\ ALORS\ y\ est\ C_2$, est incomplète car la variable x_1 n'intervient pas dans sa définition. Les règles formulées par les experts sont principalement des règles incomplètes. Formellement, une règle incomplète est définie par une combinaison implicite de connecteurs logiques ET et OU opérant sur l'ensemble des variables. Si l'univers de la variable x_1 est découpée en 3 sous-ensembles flous, la règle incomplète ci-dessus peut aussi s'écrire de la façon suivante :

$$SI\ (x_1\ est\ A_1^1\ OU\ x_1\ est\ A_1^2\ OU\ x_1\ est\ A_1^3)\ ET\ x_2\ est\ A_2^1\ ALORS\ y\ est\ C_2.$$

– Degré de vérité : Pour une règle donnée, i, son degré de vérité pour un exemple, également appelé poids, et noté w_i, résulte d'une opération de conjonction des éléments de la prémisse : $w_i = \mu_{A_1^i}(x_1) \wedge \ldots \wedge \mu_{A_p^i}(x_p)$, où $\mu_{A_j^i}(x_j)$ est le degré d'appartenance de la valeur x_j à l'ensemble flou A_j^i.

– Activité : Un exemple active une règle, ou bien une règle active un exemple, si le degré de vérité de la règle pour l'exemple est non nul.

– Prototype d'une règle : un exemple est un prototype d'une règle si son degré de vérité pour cette règle vaut un.

– Système d'inférence floue (SIF) : Un système d'inférence floue est formé de trois blocs comme indiqué sur la figure 2.3. Le premier, l'étage de fuzzification transforme les valeurs numériques en degrés d'appartenance aux différents ensembles flous de la partition. Le second bloc est le moteur d'inférence, constitué de l'ensemble des règles. Enfin, un étage de défuzzification permet, si nécessaire, d'inférer une valeur nette, utilisable en commande par exemple, à partir du résultat de l'agrégation des règles. Le nombre de règles du système est noté r.

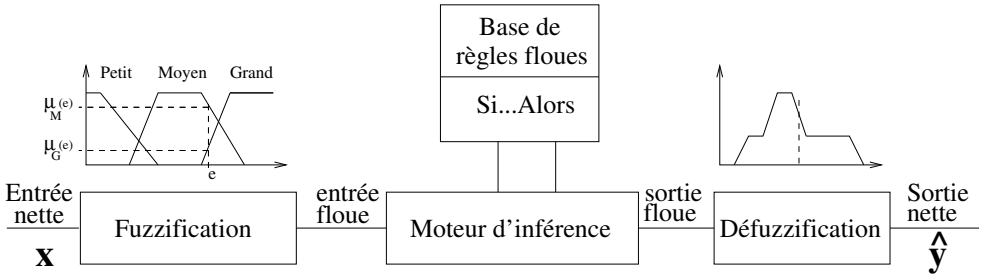


FIG. 2.3 – Un système d'inférence floue

– Sortie inférée par le système : la valeur inférée, pour une entrée donnée, dépend bien entendu de la base de règles mais aussi des opérateurs d'agrégation et de défuzzification, ainsi que de la nature de la sortie (nette ou floue). Nous ne considérerons ici que le cas de sorties nettes.
– Agrégation : c'est l'opération qui va permettre de calculer les niveaux d'activation pour un exemple donné. Dans la suite de ce rapport, l'opérateur d'agrégation utilisé sera toujours du type **somme**, qui s'exprime comme :

$$\forall j = 1, \ldots, m$$

$$W^j = \sum_r (w^r(x)) \mid C^r = j$$

où C^r est la conclusion de la règle r, m le nombre de valeurs différentes de conclusions et W^j le niveau d'activation.

– Défuzification : cet opérateur utilise les résultats de l'agrégation pour calculer la conclusion de l'exemple donné et dépend également de la nature de la sortie. Par la suite, nous utiliserons toujours un opérateur de type **sugeno** pour une sortie nette, qui s'exprime comme :

$$\widehat{y_i} = \frac{\sum_{j=1}^{m} W^j C^j}{\sum_{j=1}^{m} W^j} \tag{2.1}$$

Pour plus d'informations sur la nature des sorties et les opérateurs d'agrégation et de déffuzification, le lecteur pourra se reporter par exemple à [Gac97], à [BM93], ou encore à la documentation du logiciel FisPro [Fis]

– Vocabulaire de sortie : le vocabulaire de sortie est l'ensemble formé des conclusions distinctes des règles du système d'inférence. Sa taille sera souvent élevée pour des problèmes de régression, et correspondra le plus souvent aux labels des classes pour des problèmes de classification.

– Apprentissage et validation : de manière générale, la construction d'un système d'inférence se fera toujours en deux étapes, avec deux jeux de données contenant des exemples différents (mais provenant du même problème). L'apprentissage consistera à construire le système à partir du jeu de données réservé à cet effet. La validation, elle, consistera comme son nom l'indique à valider le système, en vérifiant qu'il infère de manière satisfaisante les exemples du jeu de validation.

– Généralisation et surapprentissage : un système sera doté d'une bonne capacité de généralisation s'il fournit de bons résultats pour d'autres exemples que ceux qui ont servi à son apprentissage. Il pourra donc se " généraliser " facilement à l'ensemble des cas possibles du problème traité. A l'inverse, le surapprentissage est le terme qui désigne le fait que le système construit ne puisse s'appliquer efficacement qu'aux exemples qui ont servi à le construire. Un système qui aura " surappris " fournira donc de très bons résultats pour les exemples qui auront servis à son apprentissage, mais de très mauvais pour tout autre exemple (d'où la nécessité d'une validation pour se prémunir de cet effet).

– Validation croisée : par validation croisée, on désigne le fait de pratiquer plusieurs fois l'apprentissage et la validation sur un même jeu de donnée, et ce pour s'assurer que la méthode est générale et ne risque pas de générer un surapprentissage. Pour chaque répétition (le nombre de ces dernières étant

variable), le jeu de données complet est séparé en deux jeux de données, l'un pour l'apprentissage, l'autre pour la validation. La proportion d'exemples réservés pour l'apprentissage est variable, elle dépend fortement du problème et du nombre d'exemples disponibles. Le recours à la validation croisée nécessite qu'un jeu de données assez important par rapport à la complexité et à la dimension du problème soit disponible. Il est possible de recourir à d'autres méthodes dans le cas où trop peu d'exemples sont disponibles (Leave-One-Out, Bootstrap), mais elles ne seront pas utilisées ici.

– Apprentissage supervisé : l'apprentissage supervisé consiste à induire des relations entre les entrées et la sortie, de dimension un, d'un système à partir d'un ensemble d'exemples. L'ensemble d'apprentissage comprend N exemples.

– Erreur quadratique moyenne : notée EQM, elle est calculée comme :

$$EQM = \frac{1}{N}\sum_{i=1}^{N} \|\widehat{y_i} - y_i\|^2$$

où $\widehat{y_i}$ est la valeur inférée par le système pour l'exemple i.

– Erreur Moyenne : contrairement à la précédente, elle est homogène à une mesure. Son expression est :

$$ErM = \frac{1}{N}\sqrt{\sum_{i=1}^{N} \|\widehat{y_i} - y_i\|^2} = \frac{\sqrt{EQM}}{\sqrt{N}} = \frac{RMSE}{\sqrt{N}}$$

– Erreur de classification : il s'agit de la somme des éléments mal classés sur l'ensemble des c classes :

$$Erreur = \sum_{i=1}^{c} m_i$$

où m_i est le nombre d'éléments mal classés pour la classe i

Chapitre 3

Etude et implémentation de l'Algorithme OLS

3.1 Introduction

Cette première partie consiste en une étude de l'algorithme OLS (Orthogonal Least-Squares) et des modifications qui y ont été apportées pour rendre ses résultats interprétables.

L'ensemble du codage de l'algorithme et de ses modifications a été réalisé en utilisant le language C++, sur base du travail déjà réalisé par Serge Guillaume du Cemagref, Brigitte Charnomordic de l'INRA et Nicolas Bonnet, stagiaire à l'INRA. En ce qui concerne les opérations matricielles du programme, nous avons utilisé la librairie de fonctions GSL (GNU scientific library). L'ensemble du code a été réalisé dans l'optique de l'intégrer ultérieurement au programme FisPro développé conjointement par l'INRA et le Cemagref de Montpellier. Pour quelques détails supplémentaire sur cet aspect, voir la section 3.6.

La section 3.2 présente l'algorithme d'origine et les notions théoriques qui l'accompagnent. La section 3.3 résume les critères d'interprétabilité retenus ainsi que les indices de performance choisis. La section suivante (3.4) décrit les diverses modifications apportées à l'algorithme d'origine. Enfin, la section 3.5 consiste en une brève étude des diverses performances de l'algorithme OLS, et brosse un tableau des avantages et inconvénients qu'il présente.

3.2 Présentation de l'algorithme d'origine

L'algorithme OLS (Orthogonal Least-Squares) est une méthode qui, à partir d'un ensemble d'exemples d'apprentissage, construit un ensemble de règles floues et sélectionne ensuite les plus importantes d'entre elles. Cette sélection s'opère

par le biais d'une régression linéaire et d'une orthogonalisation par le procédé de Gram-Schmidt. La méthode a d'abord été présentée dans [WM92a] puis plus tard dans [HM94] avec de légères modifications. Nous allons maintenant détailler la méthode telle qu'elle a été décrite dans [HM94]. Une description plus sommaire pourra être trouvée dans [Gui01b].

Cet algorithme, à l'origine pensé pour et par des problèmes de régression, a principalement été utilisé pour des problèmes de prédictions ou d'estimation de fonctions (voir [Fio02], par exemple). Nous voulons ici l'utiliser dans le cadre d'extration de connaissances à partir d'un jeu de données.

3.2.1 Fonctions de base floues

L'induction de règles floues est un problème à caractère non-linéaire. L'objectif des fonctions de base floue est de fournir un outil qui permettra de linéariser ce problème en le projetant dans un autre espace, et ce afin de pouvoir lui appliquer les méthodes propres aux problèmes linéaires.

Soit un système flou correspondant à un ensemble de règles du type

$$\text{SI } x_1 \text{ est } A_1^j \text{ et SI } x_2 \text{ est } A_2^j \text{ et SI } \ldots \text{ et SI } x_n \text{ est } A_n^j \text{ ALORS } Y = \theta \qquad (3.1)$$

Les fonctions de base floues (FBF) sont définies comme

$$p_j(x) = \frac{\prod_{i=1}^{n} \mu_{A_i^j} x(i)}{\sum_{j=1}^{M} \prod_{i=1}^{n} \mu_{A_i^j} x(i)} \qquad , j = 1, 2, \cdots, M. \qquad (3.2)$$

où les $\mu_{A_i^j} x(i)$ sont les fonctions d'appartenance du système flou. Partant des fonctions de base floues (3.2), il est possible de construire une somme qui sera équivalente à un système de règles floues. cette somme sera de la forme :

$$f(x) = \sum_{j=1}^{M} p_j(x)\, \theta_j \qquad (3.3)$$

où les θ_j seront constants et joueront le rôle des C^j d'une défuzzification de type Sugeno, alors que les $p_j(x)$ seront équivalents aux niveaux d'activation des règles SI-ALORS (voir equation 2.1 de la section 2.3). Il est également possible de montrer par le théorème de Stone-Weierstrass [WM92a] que les sommes de FBF (eq. 3.3) sont des approximateurs universels.

Dans l'algorithme OLS, nous considèrerons que les paramètres des FBF (eq. 3.2) sont constants. Les seuls paramètres qui restent donc à estimer sont les θ_j, ce qui rend la fonction (3.3) linéaire. Il sera donc possible d'utiliser une technique comme celle de l'orthogonalisation de Gram-Schmidt pour estimer au mieux ces paramètres.

3.2.2 Construction des règles

Dans l'algorithme OLS, à chaque exemple correspond une règle. Le principe de construction des prémisses de ces règles est simple : pour chaque dimension d'entrée d'un exemple, le SEF correspondant à la prémisse est celui pour lequel la valeur possède le plus grand degré d'appartenance. Ainsi, pour une règle du type $SI x_1 \; est \; A_1^i \; ET \; x_2 \; est \; A_2^i \; \ldots ET x_p \; est \; A_p^i \; ALORS \; y \; est \; C^i$, les SEF A_j^i seront ceux pour lesquels les degrés d'appartenance des x_j^i seront maximum.

Dans le cas où aucun SEF n'est pré-défini et qu'ils sont construits comme dans l'algorithme décrit dans [HM94], à chaque x_j^i correspond un SEF particulier.

3.2.3 Procédé de résolution

Soit un ensemble d'apprentissage formé de N couples $(x(k), d(k)), k = 1, 2, \cdots, N$ où $x(k)$ est de dimension p et $d(k)$ est de dimension 1.

Soit le modèle de régression suivant :

$$d(k) = \sum_{i=1}^{M} p_i(k) \; \theta_i + \epsilon(k) \tag{3.4}$$

où $d(k)$ est la sortie observée, les θ_i sont les paramètres à optimiser, les $p_i(k)$ sont les régresseurs et où $\epsilon(k)$ est une fonction d'erreur supposée non corrélée aux régresseurs.

Dans le cas présent, les régresseurs sont les fonctions (3.2) et correspondent chacun à l'ensemble des prémisses d'une règle floue (3.1), alors que les paramètres à optimiser θ_i correspondent aux conclusions. Lors de l'initialisation de l'algorithme sont construits autant de régresseurs qu'il y a d'exemples ($M = N$). Pour chaque dimension d'entrée de chaque exemple, un SEF de type gaussien est construit, dont la fonction d'appartenance est la suivante :

$$\mu_{A_i^j} x(i) = a_i^j e^{\left[-\frac{1}{2} \left(\frac{x_i - \bar{x}_i^j}{\sigma_i^j} \right)^2 \right]} \tag{3.5}$$

Les paramètres de ces fonctions sont choisis de la manière suivante : les $a_i^j = 1$, le $\bar{x}_i^j$ de chaque fonction est la valeur de l'exemple i dans la dimension d'entrée j, et le σ_i^j est égal à la variation du domaine de chaque dimension d'entrée j, à un facteur multiplicatif près. Il est précisé dans [HM94] que la valeur optimale de ce facteur est fonction du problème traité.

Mise sous forme matricielle, l'équation (3.4) devient :

$$d = P\theta + E \tag{3.6}$$

avec d et E, vecteurs colonnes de dimension N, θ vecteur colonne de dimension M et P, une matrice de dimension $N \times M$.

$$\begin{aligned} d = d(1) &\quad \cdots \quad d(N) \\ P = p_1 &\quad \cdots \quad p_M \end{aligned}$$

avec

$$p_i = p_i(1) \quad \cdots \quad p_i(N)$$

Le principe de l'algorithme OLS est de transformer les p_i en un ensemble de vecteurs orthogonaux, ce qui permettra ensuite de les évaluer individuellement en terme de variance expliquée ou d'énergie apportée.

Pour réaliser cette orthogonalisation, la matrice P est décomposée en une matrice orthogonale W et une matrice supérieure unitaire A, soit $P = WA$. La matrice W est donc de dimension $N \times M$ et telle que $W^T W = H$, où H est une matrice diagonale carrée de taille M.

Les termes généraux de la matrice A sont les α_{ij}, où α_{ij} représente la part du vecteur p_j déjà prise en compte dans le vecteur w_i qui a été précédemment construit. La matrice A est donc de ce type :

$$A = \begin{pmatrix} 1 & \alpha_{12} & \alpha_{13} & \cdots & \alpha_{1M} \\ 0 & 1 & \alpha_{23} & \cdots & \alpha_{2M} \\ 0 & 0 & & & \vdots \\ \vdots & & \ddots & & \\ & & & 1 & \alpha_{(M-1)M} \\ 0 & & \cdots & 0 & 1 \end{pmatrix}$$

L'orthogonalisation des vecteurs se fait selon le procédé de Gram-Schmidt qui permettra de projeter les vecteurs p_j sur les vecteurs w_i. Son principe est le suivant :

$$\begin{aligned} w_1 &= p_1 \\ w_2 &= p_2 - \frac{< w_1, p_2 >}{\|w_1\|^2} w_1 \\ &\vdots \\ w_i &= p_i - \frac{< w_1, p_i >}{\|w_1\|^2} w_1 - \frac{< w_2, p_i >}{\|w_2\|^2} w_2 - \cdots - \frac{< w_{i-1}, p_i >}{\|w_{i-1}\|^2} w_{i-1} \end{aligned}$$

L'équation (3.6) devient donc :

$$d = Wg + E \tag{3.7}$$

dont la solution par les moindres carrés est :

$$\widehat{g} = (W^T W)^{-1} W^T d$$

$\widehat{g}$ est un vecteur colonne de dimension M, de terme général :

$$\widehat{g_i} = \frac{w_i^T d}{w_i^T w_i}$$

Les paramètres θ s'obtiennent facilement à partir de la relation :

$$A\widehat{\theta} = \widehat{g} \tag{3.8}$$

La somme des carrés des sorties désirées, ou énergie de $d(k)$ s'écrit :

$$d^T d = \sum_{i=1}^{M} g_i^2 w_i^T w_i + E^T E$$

où $\sum_{i=1}^{M} g_i^2 w_i^T w_i$ est la part expliquée de l'énergie et $E^T E$ la part inexpliquée.

Pour minimiser l'erreur, le critère de sélection choisi sera la part de variance expliquée par un régresseur w_i normée par la variance totale. Il faudra donc maximiser le critère suivant :

$$[var]_i = \frac{g_i^2 \; w_i^T w_i}{d^T d}$$

ce qui correspond à la composante orthogonale qui maximise la variance expliquée par rapport aux sorties observées.

3.2.4 Algorithme

A la première étape, sélectionner parmi les M vecteurs celui qui explique le plus de variance. Pour cela calculer pour $1 \leq i \leq M$:

$$\begin{aligned} w_1^{(i)} &= p_i \\ g_1^{(i)} &= (w_1^{(i)})^T d / (w_1^{(i)})^T (w_1^{(i)}) \end{aligned}$$

$$[var]_1^{(i)} = (g_1^{(i)})^2 (w_1^{(i)})^T (w_1^{(i)}) / d^T d$$

et sélectionner

$$w_1 = w_1^{(i_1)} = p_{i_1} \text{ tel que } [var]_1^{(i_1)} = max\{[var]_1^{(i)}, \quad 1 \leq i \leq M\}$$

A l'étape k, $k \geq 2$, pour $1 \leq i \leq M, i \neq i_1, \cdots, i \neq i_{k-1}$, calculer :

$$\begin{aligned} \alpha_{jk}^{(i)} &= w_j^T p_i / w_j^T w_j \\ w_k^{(i)} &= p_i - \sum_{j=1}^{k-1} \alpha_{jk}^{(i)} w_j \\ g_k^{(i)} &= (w_k^{(i)})^T d / (w_k^{(i)})^T (w_k^{(i)}) \end{aligned}$$

$$[var]_k^{(i)} = (g_k^{(i)})^2 (w_k^{(i)})^T (w_k^{(i)}) / d^T d$$

et sélectionner

$$w_k = w_k^{(i_k)} = p_{i_k} - \sum_{j=1}^{k-1} \alpha_{jk} w_j$$

tel que $[var]_k^{(i_k)}$ est maximum.
La procédure s'arrête à l'étape M_s lorsque :

$$1 - \sum_{j=1}^{M_s} [var]_j < seuil$$

3.2.5 Calcul des θ

A la fin du premier passage de l'OLS, M_s régresseurs, correspondant chacun à une règle tirée d'un exemple, sont sélectionnés. D'un point de vue purement numérique, rien n'empêche alors de procéder au calcul des θ par la méthode des moindres carrés (voir eq. 3.8). Toutefois, l'ensemble des w_i et des α_{ij} qui servent au calcul de ces θ ont été eux-même calculés à partir de régresseurs p_i normalisés sur l'ensemble des exemples et à partir des M régresseurs, et non uniquement à partir des M_s régresseurs sélectionnés. Il en résulte que les valeurs des θ calculés à la fin du premier passage se trouvent pratiquement toutes totalement en-dehors du domaine de variation de la sortie à inférer, et sont par conséquent totalement incohérentes du point de vue de l'interprétabilité.

Un deuxième passage de l'algorithme est donc nécessaire pour obtenir des valeurs cohérentes qui pourront servir de base à l'interprétation. Ce second passage

servira uniquement à recalculer l'ensemble des régresseurs et des θ, sans pratiquer de sélection vu que celle-ci a déjà été faite au premier passage. La normalisation se fera cette fois à partir des M_s régresseurs, et les valeurs calculées seront pour la plupart cohérentes. Il est bien sûr possible de choisir d'autres exemples que ceux utilisés au premier passage pour effectuer le second.

3.3 Critères d'interprétabilité retenus

Par rapport à d'autres méthodes d'apprentissage de type "boîte-noire" (réseaux de neurones, algorithmes génétiques), la logique floue possède le grand avantage d'avoir la possibilité de fournir des systèmes qui peuvent être directement interprétés à un niveau humain. Malgré tout, pour que leurs résultats puissent être interprétables, les systèmes d'apprentissage flous doivent réunir un certain nombre de conditions [Gui01a].

En ce qui concernent les sous-ensembles flous, ils doivent être normaux, distinguables, le recouvrement entre deux ensembles voisins doit être suffisant et ils doivent être en nombre relativement restreint. Ce nombre peut bien sûr varier selon le problème traité, mais les capacités de la mémoire à court terme de l'être humain étant limitées [Mil56], nous nous limiterons à un nombre de 7 ± 2 sous-ensembles flous par variable.

Nous avons donc choisi, pour les partitions floues, l'implémentation proposée par [Glo99] : une partition floue forte (voir section 2.3)

L'aspect de l'interprétabilité qui concerne la présence d'un nombre réduit des règles significatives ainsi que la gestion de règles incomplètes sera abordé dans la partie 5.

Un autre critère que nous avons gardé à l'esprit est celui du niveau d'activation des règles par un exemple donné. Plus de détails sont donnés ci-dessous, lorsque l'indice de couverture est défini.

L'ensemble de ces critères représentent un ensemble de contraintes qu'il faut respecter pour obtenir des résultats interprétables. Une fois les systèmes d'inférence obtenus, il est nécessaire de calculer les performances du système, tant du point de vue des performances numériques que de l'interprétabilité. Voici le détail des principaux éléments d'estimation de la performance qui seront utilisés ultérieurement :

Indice de couverture (CI) : Il est possible qu'il existe des exemples qui n'activent que faiblement les règles. Un exemple sera déclaré inactif si son degré de vérité maximum, sur l'ensemble des règles, est inférieur à un seuil d'activation paramétrable. Un exemple inactif n'est pas géré par le système. Le nombre d'exemples inactifs est utilisé pour définir un indice de couverture : $CI = 1 - \frac{I}{N}$. I désigne le nombre d'exemples inactifs et N le nombre total d'exemples. Si un

exemple active une règle à un niveau proche de zéro, le niveau de confiance accordé à la connaissance qui en sera extraite sera très bas, il faut donc qu'un système possède un niveau de couverture suffisant compte tenu d'un seuil raisonnable d'activation des règles (de l'ordre de 10^{-1}). Si le seuil d'activation est augmenté, l'indice de couverture ne pourra que décroître ou rester égal. Il est à noter également que la valeur de cet indice est dépendant de l'opérateur de conjonction choisit. Toute chose étant égale par ailleurs, l'autilisation de l'opérateur **min** entrainera un indice de couverture plus élevé qui sera indépendant de la dimension du système, tandis que l'utilisation d'un opérateur **prod** entrainera un indice moins élevé qui évoluera de manière inversément proportionnelle à la dimension du système.

Indice de performance (PI) : Il dépend du type de sortie et mesure la performance numérique globale de la méthode. Pour un problème de régression, nous utiliserons l'ErM, tandis que ce sera le nombre d'exemples mal classés qui sera calculé pour un problème de type classification (ces deux indices ont déjà été définis dans la section 2.3).

Ratio de couverture par règle (Ratio) : Indice de couverture et nombre de règles présentes dans le système d'inférence flou sont fortement corrélés. Nous avons donc introduit la notion de part d'exemples couverts par règle, qui représente la capacité d'une règle à activer un certain nombre d'exemples. Plus ce ratio est élevé, moins il est besoin de règles pour couvrir l'ensemble des exemples, ce qui rend le système d'autant plus généraliste et interprétable. Ce ratio s'exprime comme $Ratio = \dfrac{CI}{\text{Nbre de règles}} \times 100$.

3.4 Modifications de l'algorithme OLS d'origine

Pour répondre à l'ensemble des contraintes qui rendent un système interprétable (nombre et formes des SEF, incomplétude et nombre des règles), il faut souvent accepter de dégrader légèrement la performance numérique, mais c'est là un faible prix à payer lorsque la compréhension du système revêt une grande importance. Un des enjeux des diverses méthodes d'apprentissage floues est donc de trouver un équilibre entre interprétabilité et performances numériques du système. Nous avons donc modifié l'algorithme OLS pour qu'il prenne en compte ces deux paramètres, tout aussi importants l'un que l'autre. Ce sont ces modifications qui sont maintenant présentées. Précisons que nous avons choisi d'utiliser l'opérateur de conjonction **min** plutôt que **prod** dans l'équation 3.2, et ce en regard de l'influence de cet opérateur sur le critère d'indice de couverture.

3.4.1 Algorithme d'origine et interprétabilité

Dans l'algorithme d'origine, à une dimension d'entrée de chaque exemple correpond un SEF de type gaussien (pour les détails, voir la section 3.2.3). L'utilisation de fonctions gaussiennes, dont le $\mu(x)$ est toujours positif, permet de s'assurer que chaque exemple active l'ensemble des règles du système, mais cette activation se fait à un niveau souvent très proche de zero. Il s'ensuit que si un niveau d'activation minimum est imposé pour que l'exemple soit pris en compte, l'indice de couverture diminue fortement et devient très vite insuffisant, ce qui empêche toute interprétation. La figure 3.1 montre une série de SEF obtenus par l'application de l'algorithme d'origine à une dimension du problème présenté au chapitre 6. Au vu du nombre de SEF et de la redondance qu'ils présentent entre eux, il est impossible de leur apporter une signification valable, et cette situation ne fera qu'empirer avec le nombre d'exemples d'apprentissage.

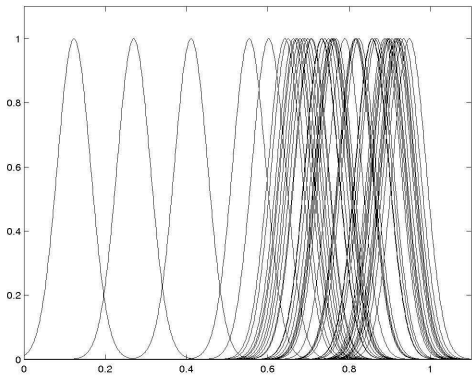


FIG. 3.1 – SEF générés pour 50 exemples d'apprentissage

3.4.2 Fonctions d'appartenance triangulaires

Notre première démarche a été de remplacer les SEF gaussiens par des SEF triangulaires équivalents, d'une part parce qu'il est plus aisé de les manipuler pour rendre les partitions floues fortes, d'autre part parce qu'ils sont plus faciles à définir par un expert. Afin de s'assurer que la substitution ne dégradait pas les résultats, nous avons choisi des fonctions triangulaires dont les couvertures étaient équivalentes à celles des fonctions gaussiennes. La figure 3.2 illustre les paramètres que nous avons retenu pour ces fonctions triangulaires (les gaussiennes sont remplacées par des triangles isocèles dont le centre est la moyenne de

la gaussienne, et dont la base vaut cinq fois l'écart-type de la gaussienne). Nous verrons plus tard que cette substitution n'entraine pas de perte de performance significative.

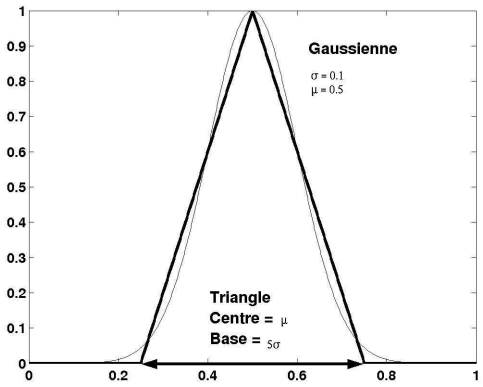


FIG. 3.2 – Fonction triangulaire de substitution

3.4.3 Diminution du nombre de SEF

L'étape suivante consiste à diminuer le nombre de SEF par variable afin de pouvoir affecter un label linguistique à chacun d'entre eux. Pour ce faire, nous avons réalisé sur chaque dimension d'entrée une opération d'agrégation élémentaire dont les détails se trouvent en annexe A. A partir de l'ensemble des centres obtenus, nous avons construit des SEF triangulaires suffisamment larges pour garantir une couverture complète de la dimension d'entrée considérée. La figure 3.3 est la partition résultant de la substitution et de la simplification de la partition de la figure 3.1. Comme nous pouvons le constater, la partition résultante n'est pas forte et le recouvrement entre deux fonctions d'appartenance voisines peut se réveler trop faible. Malgré tout, la présence d'un nombre restreint de SEF permet déjà d'apporter un début d'interprétation. Par rapport à l'algorithme d'origine, cette diminution provoquera un changement des prémisses dans les règles et une perte de performance, dont il faudra s'assurer qu'elle n'est pas trop importante.

3.4.4 Partitions floues fortes

La suite logique, en accord avec les critères d'interprétabilité choisis, est de rendre les partitions obtenues à l'étape précédente fortes, afin de pouvoir plus facilement attacher un label linguistique à chaque SEF obtenu. Cette transformations

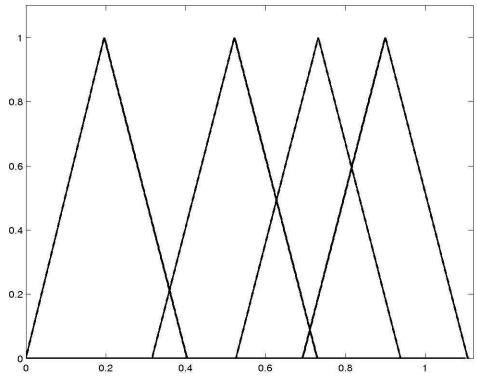


FIG. 3.3 – Substitution et simplification à partir de la même variable que celle traitée pour la figure 3.1

conservera les centres des SEF, et modifiera seulement les valeurs des extrémités de chaque SEF (les deux SEF correspondant aux bords du domaine de variation seront transformés en demi-trapèzes). La figure 3.4 montre la partition floue forte obtenue à partir de la même variable que celle traitée pour la figure 3.3. Il est évident que des labels tels que *'Très Bas '*,*'Bas '*,*'Haut '*,*'Très Haut '*deviennent, dans le cas de la figure 3.4, parfaitement compréhensibles.

3.4.5 Restriction du vocabulaire de sortie

Une fois les conditions sur les SEF remplies, une autre possibilité de simplification est la réduction du nombre d'éléments du vocabulaire de sortie. Cette réduction s'accompagnera immanquablement d'une perte de performance. Pour ce qui concerne les problèmes de régression, nous avons implémenté un algorithme itératif qui se base sur la méthode de clustering des k-means [Sap90]. A chaque itération, le nombre de classes (ou d'éléments du vocabulaire de sortie réduit) est augmenté et la méthode est appliquée à un ensemble de données relatives à la sortie. A chaque itération, chaque conclusion d'origine est remplacée par l'élement du nouveau vocabulaire dont elle est la plus proche. Le critère de sortie de l'algorithme consiste en une perte relative (exprimée en % de la performance initiale) autorisée de la performance.

L'algorithme peut aussi être utilisé avec un nombre pré-défini d'éléments du vocabulaire réduit (dans le cas de la classification, chaque label de classe correspond à un élément du vocabulaire réduit). L'algorithme et son fonctionnement sont présentés en annexe B. Cette étape modifiera l'ensemble des conclusions des

règles.

En fonction des volontés de l'utilisateur (en terme de performance ou d'interprétabilité), l'ensemble ou une partie seulement de ces modifications pourront être apportées à l'algorithme d'origine.

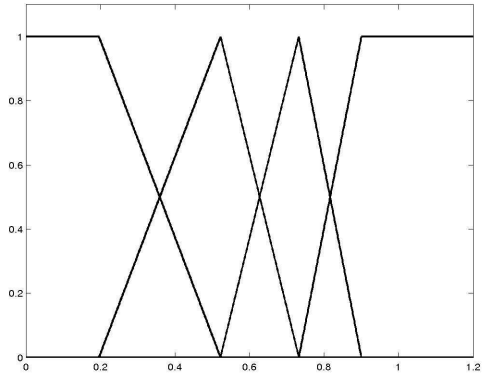


FIG. 3.4 – Partition floue forte obtenue à partir de la même variable que celle traitée pour la figure 3.1

3.4.6 Critères d'arrêt

En plus du critère concernant la part d'erreur globale expliquée, deux autres critères ont été ajoutés : un critère concernant le nombre de règles sélectionnées, et un autre concernant la part d'erreur expliquée par la dernière règle sélectionnée.

3.5 Etude de performances

3.5.1 Présentation du jeu de données

Le jeu de données utilisé est tiré des bases de données de l'UCI [BM98]. Il correspond à un problème de régression qui consiste à inférer, à partir de données diverses, la consommation en carburant d'une voiture lors d'un cycle de ville. Le jeu de données (auto-mpg) contenait à l'origine 398 exemples, dont 6 comportait des données manquantes (puissance en chevaux). Ces derniers ont été enlevés de la base, qui est donc finalement constituée de 392 exemples. Parmi les 8 variables présentées, 4 sont continues, 3 sont discrétisées et la dernière correspond à la dénomination de la voiture. Parmi ces 8 variables, j'ai éliminé celles concernant le nom des voitures et leur provenance (variable discrète non-ordonnée). Les

TAB. 3.1 – Variables d'entrée (auto-mpg)

Nom
Cylindres
Déplacement
Puissance
Poids
Accélération
Année du modèle

2 autres variables discrètes, l'année du modèle et le nombre de cylindres, ont été conservées (car ordonnées et donc fuzzifiables). Les noms des différentes variables sont présentées dans le tableau 3.1. La variable à inférer est, quand à elle, continue, et concerne le nombre de miles (1 miles = 1.609 km) parcouru par gallon (1 gallon = 3.78 litres) de carburant dépensé (Miles Per Gallon).

3.5.2 Présentation des résultats

Tous les résultats présentés ici sont des moyennes obtenues sur une validation croisée de 4 répétitions. Pour chaque répétition, l'ensemble du jeu de données a été divisé en deux, le jeu d'apprentissage et de validation contenant chacun 196 exemples. Le tableau 3.2 présente les résultats obtenus. Pour chaque cas on été calculés le nombre de SEF moyen par variable (NSef), le nombre moyen de règles par système (Nr), les indices déjà présentés dans la section 3.3 (CI, PI, Ratio), ainsi que le nombre moyen de conclusions distinctes par système de règle (NConc). Le tableau comporte deux valeurs de l'indice de performance : PI_{app} concerne la performance sur le jeu d'apprentissage et PI_{val} sur le jeu de validation. L'indice de couverture, lui, concerne toujours les tests réalisés sur les jeux de validation (pour l'apprentissage, il reste toujours très proche de 100%).

Présentation des cas

Il est à noter qu'un cas reprend l'ensemble des modifications survenues pendant les cas précédents. Durant l'ensemble des tests, et sauf indication contraire, le seuil d'activation d'un exemple a été fixé à 0.1

- **cas 1 et 2** : Pour le cas 1 et 2, les SEF ont été construits d'après l'algorithme d'origine (c'est-à-dire 1 SEF par exemple, par dimension d'entrée et par valeur distincte, voir eq. 3.5). Pour le cas 1, aucun degré de vérité minimum n'a été imposé pour qu'un exemple soit jugé actif. Pour le cas 2, un niveau

TAB. 3.2 – tableau récapitulatif de résultats

Cas	NSef	Nr	Ratio	PI_{app}	PI_{val}	CI(%)	NConc
1	68,21	141.25	0.71	0.09	0.24	100	141.25
2	68.21	141.25	0.56	0.09	0.22	80	141.25
3	68.21	126.5	0.64	0.10	0.24	81.2	128.5
4	6.52	94.5	0.86	0.13	0.26	81.7	94
5	4.7	63.75	1.44	0.16	0.23	92	63.75
6	6.52	101.5	0.80	0.12	0.26	81.5	101
7	4.7	65	1.40	0.15	0.23	90.7	64.75
8	6.52	101.5	0.80	0.15	0.27	81.5	8.75
9	4.7	65	1.40	0.17	0.24	90.7	8.5

de 0.1 a été imposé (toute connaissance extraite en-deçà de ce niveau étant jugée non fiable).

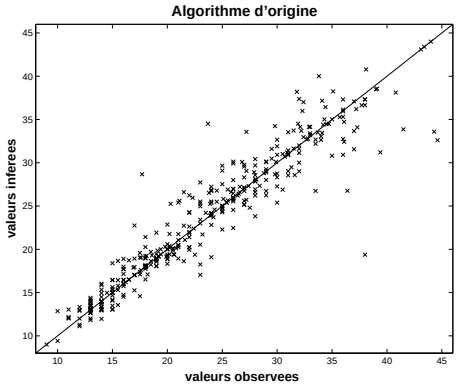
- **cas 3** : Les SEF gaussiens sont remplacés par des SEF triangulaires.
- **cas 4 et 5** : Ces cas correspondent à une diminution progressive du nombre de SEF (moyenne de 6.5 SEF pour le **cas 4** et de 5 SEF pour le **cas 5**) par l'algorithme d'agrégation cité dans la section 3.4. Les partitions floues résultantes ne sont pas fortes.
- **cas 6 et 7** : Ils correspondent respectivement aux cas 4 et 5 avec des partitions floues rendues fortes.
- **cas 8 et 9** : Ils correspondent respectivement aux cas 6 et 7 avec un vocabulaire de sortie réduit, la dégradation de performance étant limitée à 10% de la performance d'origine (dans les cas où ce critère n'a pas pu être respecté, le dernier système construit avec un vocabulaire réduit a été gardé).

Analyse des résultats

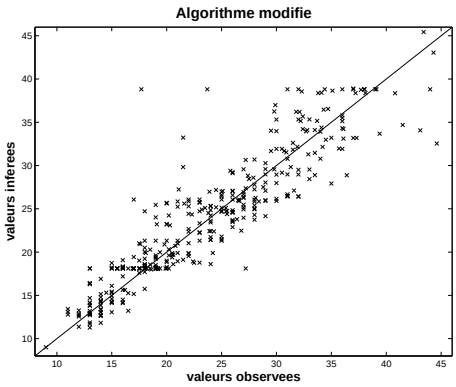
Il y a de nombreuses choses à retirer de l'observation des résultats obtenus dans les différents cas. La première constatation, la plus globale et la plus importante, est que l'indice de performance, au cours de tous les changements, n'a pas changé significativement, alors que le gain d'interprétabilité entre les deux cas extrêmes est, lui, indiscutable. Il y a même une diminution de l'écart entre l'indice de performance d'apprentissage et de validation, ce qui semble indiquer que les derniers cas présentent une meilleure capacité de généralisation que les premiers (phénomène auquel on pouvait s'attendre vu la perte de spécificité des SEF à partir du cas 3).

Comme prévu, l'élévation du seuil d'activation à 0.1 (passage du cas 1 au

FIG. 3.5 – Performances obtenues pour une même répétition



(3.5.a) Algorithme OLS d'origine



(3.5.b) Algorithme OLS modifié

cas 2) entraine une perte de couverture. Du fait que le problème compte peu de variables et que ces dernières sont distribuées de façon relativement homogène sur leur domaine de variation, cette diminution est faible mais suffit à démontrer que ce problème existe. De plus, ce ne sera pas le cas pour d'autres problèmes (par exemple, la même élévation pour le problème traité au chapitre 6 fait passer l'indice de couverture de 100% à moins de 20%). Le passage aux SEF triangulaires, lui, ne provoque aucun changement significatif, et valide donc la substitution opérée.

Pour les cas suivants, outre l'augmentation de la lisibilité des partitions, il y a une diminution notable du nombre de règles sélectionnées (de 141 règles en moyenne, on passe à 64 !) ainsi qu'une augmentation significative du ratio (par exemple, entre les cas 4 et 5, où la couverture augmente alors que le nombre de règles diminue).

Au niveau du nombre de conclusions distinctes, les légères différences jusqu'au cas 7 ne proviennent que d'approximations faites par le programme (jusqu'au cas 7, NConc $\simeq$ Nr). Par contre, les cas 8 et 9 prouvent l'utilité de la réduction de vocabulaire, qui réduit le nombre de conclusions à moins d'une dizaine sans pour autant dégrader les performances.

Les figures 3.5.a et 3.5.b montrent les résultats sur l'ensemble des données de deux systèmes induits à partir des mêmes exemples, respectivement par l'algorithme d'origine (cas1) et par l'algorithme modifié (cas9). Ces graphiques montrent que même si on peut constater une légère dégradation des résultats entre les deux cas (notamment pour certains résultats autour de 20), cette dernière est négligeable par rapport au gain énorme d'interprétabilité des systèmes.

Au final, nous obtenons donc des systèmes qui offrent des performances com-

parables à celles de l'algorithme d'origine, mais qui peuvent en plus être interprétés. Il reste malgré tout évident qu'il est difficile d'interpréter une soixantaine de règles comportant chacune six prémisses. Il est donc pratiquement indispensable de rendre le système moins complexe. Pour ce faire, plusieurs solutions s'offrent à nous, parmi lesquelles se trouvent par exemple l'élimination des variables les moins significatives (par un prétraitement au moyen de méthodes statistiques, par exemple) ou la simplification des règles composant les sytèmes flous. Dans le contexte où nous nous plaçons (interpréter les systèmes et en extraire des connaissances), éliminer des variables est assez radical (toute information sur les variables éliminées disparaissant avec elles), tandis que simplifier un système en essayant de conserver les relations les plus importantes entre variables et entre règles est tout à fait adéquat. La méthode de simplification présentée dans le chapitre 5 vise à atteindre ce but.

3.6 Quelques mots sur l'implémentation

3.6.1 Base de travail

Je disposais, comme base de travail, de l'implémentation déjà réalisée sous Matlab par Nicolas Bonnet du premier passage de l'algorithme OLS, dont la validité avait été vérifiée. Ce programme m'a donc servi de référence lors de la comparaison des résultats obtenus avec le programme développé sous C++.

J'ai pu, de plus, réutiliser une poignée de fonctions déjà écrites précédemment en C++ (conversion des données d'un type GSL vers un autre type et inversément).

3.6.2 Environnement de travail

Le stage et l'ensemble du programme ont été réalisés en majeure partie sous un environnement Linux, à l'aide de logiciels libres tels qu'Emacs, Xfig, Latex, gcc... L'ensemble de ces outils m'étaient jusqu'alors inconnus, mais ils m'ont rapidement séduit de par le grand nombre de possiblités qu'ils offraient.

Le programme a été codé en langage C++, dans l'optique de l'intégrer au logiciel open source FisPro (voir la section 3.6.3 pour plus de détails sur ce logiciel). Pour résoudre les nombreuses opérations matricielles que nécessitent la mise en oeuvre de l'algorithme OLS, nous avons utilisé la bibliothèque GSL de Gnu [GNU].

3.6.3 Le logiciel FisPro

Le logiciel FisPro [Fis] (Fuzzy Inference System Professionnal) permet de créer des systèmes d'inférence flous sur la base de connaissances expertes ou encore de les induire en utilisant des jeux de données. Il a été conçu conjointement par Serge Guillaume du Cemagref et Brigitte Charnomordic de l'INRA afin de pouvoir répondre à des besoins réels. En effet, le cas des problèmes biologiques et agronomiques a cela de particulier que le fait de pouvoir fournir une interprétation à la réponse est souvent aussi important que la réponse elle-même. De plus, la complexité de ces problèmes et leur type nécessite souvent la collaboration entre des méthodes d'inférence et le savoir irremplaçable d'un expert. C'est donc dans cette idée de collaboration que FisPro a été développé. Ce logiciel, accessible aussi bien aux initiés qu'aux néophytes, permet d'utiliser un bon nombre de méthodes classiques d'inférence floue dans un environnement agréable.

Son développement a bénéficié de fonds publics provenant de l'Etat français et de la région du Languedoc-Roussillon, dans le cadre d'un projet de recherche coordonné par l'association TRIAL, en partenariat industriel avec la cave coopérative "La Malpère ", Arzens, Aude. L'interface, réalisée en Java, a été conçue par Jean-Luc Lablee du Cemagref et Pierre-Marie Boyer.

L'ensemble de ce logiciel open source comprend deux parties bien distinctes : une bibliothèque de fonctions écrites en langage C++ qui assure l'ensemble des calculs relatifs aux méthodes employées, et une interface homme-machine, implémentée en langage JAVA. Il est à noter que les fonctions C++ sont faites de manière a pouvoir être utilisées seules et sans l'aide d'une interface graphique.

Le lecteur intéressé pourra trouver plus de précisions en annexe C ainsi que dans [Fis].

Nous avons montré que les résultats de l'algorithme OLS pouvaient être rendus interprétables sans qu'ils souffrent pour autant d'une dégradation importante de performance, mais reste à savoir si cet algorithme offre quelques avantages en ce qui concerne l'extraction de connaissances. C'est ce que nous allons voir dans le chapitre suivant, qui a pour objet la comparaison de diverses méthodes d'induction de règles floues, dont l'algorithme OLS.

Chapitre 4

Comparaison des méthodes

4.1 Introduction

L'objet de ce chapitre est la comparaison de diverses méthodes d'induction floues disponibles dans la littérautre et dans FisPro. Outre l'OLS déjà présenté précédemment, nous considérerons la méthode des arbres flous [Qui86] [BFOS84], la méthode HFP (Hierarchical Fuzzy Partitionning) [GC03] et enfin l'algorithme de Wang et Mendel [WM92b]. La présentation de ces trois méthodes s'étend sur l'ensemble du chapitre.

La section 4.2 présente tout d'abord les jeux de données choisis pour réaliser l'ensemble des tests. Ensuite, nous considérerons la construction d'un système induit comme constituée de deux étapes successives et indépendantes l'une de l'autre (ce qui n'est pas la seule manière de considérer le problème, voir par exemple [Glo02]). La première étape consistera à construire les partitions floues pour chaque variable. La section 4.3 concernera le choix de la méthode de construction des partitions, et la section 4.4 décrira une méthode de choix de la granularité de ces partitions (c'est-à-dire, du nombre de SEF dont chacune sera composée). La deuxième étape consistera à induire les sytèmes flous au travers de méthodes d'inductions de règles. Dans la section 4.5, les méthodes qui n'ont pas encore été décrites le seront, et les méthodes d'induction seront ensuite comparées entre elles. La section 4.6 contient une synthèse et une analyse des résultats obtenus pour chaque méthode.

4.2 Présentation des jeux de données

Afin de pouvoir comparer les algorithmes de manière efficace, nous avons besoin de jeux de données dont la validité est éprouvée. J'ai donc parcouru le répertoire de l'UCI (University of California, Irvine) [BM98] qui comporte un en-

semble de jeux de données permettant d'évaluer des algorithmes d'apprentissage. Il s'agissait plus précisément de trouver des problèmes bien adapatés à la logique floue, dont la plupart des données seraient fuzzifiables (et donc continues). Parmi l'ensemble des jeux de donnée disponibles, j'en ai retenu plus spécifiquement deux, un problème de régression concernant la consommation des automobiles en ville ainsi qu'un problème de classification concernant des types de verres. Bien entendu, d'autres jeux de données auraient pu être retenu pour leur intérêt (Boston housing, Breast Cancer, Wine Classification, ...). D'autres, au contraire, ont été écartés d'emblée pour leur manque d'adaptabilité à la logique floue (variables discrètes ou déjà séparées en classes) ou pour leur piètre valeur (iris, notamment, car elles sont facilement séparables, hormis pour quelques exemples).

De plus, ces problèmes devaient être de dimensions relativement restreintes, les algorithmes d'induction présentés ici fournissant des règles complètes (à l'exception des arbres).

4.2.1 Problème de consommation de véhicule

Ce problème est celui déjà présenté dans la section 3.5.1, il n'est donc plus à détailler ici. Rappelons tout de même qu'il s'agit d'inférer la consommation en gallons par miles d'une voiture au moyen de 6 variables, et que le jeu de donnée nettoyé comporte 396 exemples.

Nous aurions pu choisir d'autres problèmes, mais la plupart étaient de dimension trop élevée pour que les indices de couverture obtenus restent d'un niveau significatif.

4.2.2 Problème de classification de type de verre

Le problème de classification choisi concerne le classement de types de verres. Il provient d'un centre de criminologie du Berkshire (région du sud de l'Angleterre). C'est à la fois la nature du problème (présence de variables continues), sa significativité (reconnaître la nature d'un verre peut avoir une grande importance dans le cadre d'une enquête criminelle) et la présence de résultats antérieurs qui ont motivé ce choix.

Il s'agit ici de classer les verres en deux catégories, selon qu'ils ont subi un processus float ou pas. Ce processus consiste à faire " flotter "le verre sur du laiton liquide, il permet d'accélerer la candence de production et fournit en sortie un verre parfaitement lisse et transparent (sans y avoir appliqué de polissage).

Les variables d'entrée, qui proviennent de la composition chimique du verre (en Sodium, Fer, Aluminium, Silice, . . .), sont au nombre de 9 et sont toutes continues. Le nombre d'exemples s'élève à 163 (dont 87 exemples de verres traités par processus float).

4.2.3 Méthode de validation

Problème de régression : Comme dans la section 3.5.1, la méthode de validation choisie a été une validation croisée sur 4 répétitions, où jeux d'apprentissage et de validation comptaient chacun autant d'exemples (donc 196 exemples chacun).

Problème de classification : Là aussi, nous avons choisi une validation croisée sur 4 répétitions. La seule différence provient du fait que les jeux d'apprentissage contiennent à chaque fois 75% des exemples, en raison du nombre peu élevé de ces derniers. La proportion initiale des classes est conservée dans les jeux de données séparés.

Il est à noter que ces jeux de données sont identiques pour l'ensemble des tests sur l'ensemble des méthodes.

4.3 Choix de la méthode de partitionnement

De nombreuses méthodes permettant de construire et d'optimiser des partitions sont disponibles, mais peu d'entre elles fournissent des partitions floues fortes interprétables, or il s'agit d'une des conditions que nous avons imposées. Après une brève présentation des méthodes de construction de SEF que nous avons considérées, le protocole poursuivi durant les tests destiné à sélectionner la meilleure sera détaillé. Suivront les résultats de ces tests ainsi que leur analyse.

Il est à noter que ces constructions sont monodimensionnelles, dans le sens où elles ne prennent en compte que les données relatives à l'entrée considérée (d'autres méthodes prennent en compte la sortie ou l'ensemble des variables).

4.3.1 Méthode HFP (Hierarchical Fuzzy Partitionning)

La construction de SEF par la méthode HFP suit le même principe qu'une classification hiérarchique, elle part d'un nombre de SEF élevé pour chaque dimension d'entrée et en fusionne deux à chaque étape selon un critère de proximité. Cette construction et les notions qui lui sont relatives sont maintenant présentées.

Définition d'une distance adaptée aux partionnements flous

La définition d'une distance entre points au sein d'une partition floue, qui sera en fait une semi-distance (dissimilarité qui respecte l'inégalité triangulaire), va permettre de choisir les deux SEF qui minimisent la somme des distances sur

l'ensemble des paires de points afin de les fusionner en un seul SEF à l'étape suivante. Comme un point peut appartenir à plusieurs sous-ensembles flous simultanément, il faudra définir une distance interne (cas où deux points appartiennent au même ensemble) et une distance externe (cas où ils appartiennent à deux ensembles différents), ainsi que la combinaison de ces deux distances.

Notion de distance

La notion suivante de distance sera utilisée pour s'assurer que l'ensemble des distances définies, ainsi que leur combinaison vérifient les propriétés mathématiques d'une distance.

Étant donnés trois points, $q,\ r,\ s$, une fonction d est une dissimilarité si :

$$\forall\, q, r \qquad \begin{cases} d(q,r) \geq 0 \\ d(q,q) = 0 \\ d(q,r) = d(r,q) \end{cases} \tag{4.1}$$

Une dissimilarité est dite semi-propre si :

$$d(q,r) = 0 \quad \Rightarrow \quad \forall\, s \qquad d(q,s) = d(r,s) \tag{4.2}$$

Elle est dite propre si :

$$d(q,r) = 0 \quad \Rightarrow \quad q = r \tag{4.3}$$

Une semi-distance est une dissimilarité qui respecte l'inégalité triangulaire :

$$\forall\, q, r, s \qquad d(q,r) \leq d(q,s) + d(r,s) \tag{4.4}$$

Une distance est une semi-distance propre.

Distance interne

Pour un point x_q et un SEF j, le degré d'appartenance μ_q^j peut être vu comme la proximité de x par rapport au noyau du SEF j (pour rappel, le noyau d'un SEF est tel que $\mu(x) = 1$ sur l'ensemble des points x qui forment le noyau). Si deux points x_q et x_t appartiennent au même SEF j, nous définirons la distance interne entre ces deux points comme

$$d_{int}(q,r) = |\mu_q - \mu_r| \qquad donc \quad \forall q, r \quad 0 \leq d_{int}(q,r) \leq 1$$

Il est facile de trouver des exemples de cette distance qui ne satisfont pas à la propriété de l'équation (4.3), par exemple deux points du noyau dans le cas où ce dernier ne se réduit pas à un singleton. Il est possible de montrer que cette distance interne est en fait une semi-distance.

Distance externe

Afin de pouvoir définir une distance externe, c'est-à-dire la distance entre deux points x_q et x_t appartenant à deux SEF i,j tels que $i \neq j$, il faut considérer la différence relative entre deux SEF, cette différence pouvant être symbolisée par une distance. Cette distance peut se définir de deux manières :

- La distance peut être définie de manière purement numérique. dans ce cas, ce sera la distance euclidienne qui sera considérée. Ce sera la distance entre deux sommets pour des SEF triangulaires ou gaussiens, ou encore la distance entre les extrémités les plus proches des noyaux pour des SEF de forme trapézoïdale. La distance sera donc définie comme $d_{prot}^{num}(i,j) = \sqrt{(c_i - c_j)^2}$ où c_i et c_j sont les centres des SEF respectifs.
- La distance peut aussi être considérée de manière plus symbolique, en faisant intervenir l'ordonnancement des SEF. En associant à chaque SEF un numéro d'ordre, la distance devient alors $d_{prot}^{symb}(i,j) = \frac{j-i}{m-1}$ où m est le nombre total de SEF, i et j sont les indices des SEF correspondants (avec $j \geq i$).

Une fois cette distance déterminée, il est possible d'introduire la notion de distance externe entre deux points q et t, qui se définit comme suit :

$$d_{ext}(q,t) = |\mu_q^j - \mu_t^i| + d_{prot}(i,j) + D_c$$

où D_c est une constante qui assure que deux points qui appartiennent principalement au même SEF seront toujours considérés plus proches que deux points qui appartiennent principalement a deux SEF distincts (le SEF auquel un point appartient principalement est le SEF pour lequel son degré d'appartenance est le plus élevé). Il est également possible de montrer que la distance externe, définie de cette manière, est une semi-distance.

Combinaison de distances

Pour définir une distance globale, prenant à la fois en compte distance interne et externe, nous noterons $d_{i,j}(q,t)$ la distance partielle entre les points q et t appartenant respectivement aux SEF i et j. Cette distance sera interne si $j = i$, sinon elle sera externe. Soit m le nombre de SEF de la partition. $d(q,t)$, la distance entre q et t, se définit comme la combinaison suivante d'au plus m^2 distances :

$$d(q,t) = \frac{1}{\sum_{i=1}^{m} \mu_q^i} \sum_{i=1}^{m} \left[\mu_q^i \frac{1}{\sum_{j=1}^{m} \mu_t^j} \sum_{j=1}^{m} \left[\mu_t^j \; d_{i,j}(q,t) \right] \right] \tag{4.5}$$

$d(q,t)$ étant une combinaison de semi-distances, elle est elle-même une semi-distance.

Hierarchical Fuzzy Partitionning (HFP)

Partition initiale

Théoriquement, la partition initiale optimale est celle où à chaque valeur de l'exemple de la base d'apprentissage dans la dimension considérée correspond un sous-ensemble flou. En pratique, le fait de procéder à un premier regroupement grossier accélère notablement le processus sans dégrader les performances de ce dernier. Ces méthodes visent à regrouper les valeurs de sortie en un nombre réduit de centres à partir desquels pourront ensuite être construits les SEF initiaux. Ces regroupements peuvent se faire selon deux méthodes complémentaires :

1. Regroupement par valeurs selon un seuil de tolérance :

   Ce regroupement consiste à regrouper des valeurs qui seront considérées comme une seule valeur si leur différence ne dépasse pas un seuil de tolérance (exprimé comme un pourcentage du domaine de variation). Cette opération utilise l'algorithme élémentaire d'agrégation qui se trouve en annexe A. Cette tolérance devra être choisie assez petite, afin qu'il y ait un assez grand nombre de groupes initiaux (à l'inverse de ce qui était fait pour l'OLS, où la tolérance devait être relativement élevée, vu que le but recherché était d'obtenir un nombre suffisamment restreint de SEF).

2. Groupage par méthode des Centres mobiles :

   Si le regroupement ci-dessus est jugé trop peu complexe et fournissant des résultats trop peu fiables, il est possible de recourir, pour générer la partition initiale, à la méthode des centres mobiles [Sap90]. Si la somme des variances intra-groupes reste en-deçà d'un certain seuil donné, les groupes formés seront très homogènes.

Fusion de deux SEF

Seuls deux SEF adjacents peuvent être fusionnés à chaque étape. Ces deux SEF, et les SEF qui leurs sont adjacents, sont modifiés de manière à maintenir la partition floue forte. D'un point de vue pratique, la limite inférieure du nouveau SEF est celle du premier SEF, et la limite supérieure celle du second SEF. Le nouveau centre est une somme pondérée des deux centres, avec comme pondération la cardinalité du SEF correspondant au centre considéré.

La figure 4.1 montre le résultat de la fusion de deux ensembles à partir d'un partitionnement contenant 4 SEF.

Critère de fusion

Maintenant que la notion de distance et le principe de fusion ont été définis, il reste à choisir, à chaque étape, les deux ensembles qui seront fusionnés. Le but de la méthode est de choisir les deux ensembles qui, une fois fusionnés, fourniront la

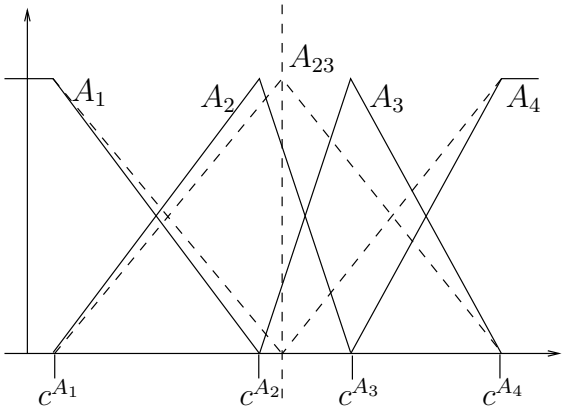


FIG. 4.1 – Processus de fusion de A_2 et A_3 en A_{23}

partition la plus proche de la partition précédente. Il faut donc introduire un critère qui permettra de satisfaire cette contrainte.

Soit la somme des distances, D_s, définie comme :

$$D_s = \frac{1}{n(n-1)} \sum_{q,t=1,2,\ldots,n,\ q \neq t} d(q,t) \tag{4.6}$$

où s est la taille de la partition, n le nombre d'exemples présents dans la base d'apprentissage, et $d(q,t)$ la distance telle que définie par l'équation 4.5.

Afin de conserver au mieux l'homogénéité de la structure, la fusion choisie sera celle qui minimisera la variation de D_s. La combinaison f de fusion retenue parmi les $s-1$ fusions possibles sera celle telle que :

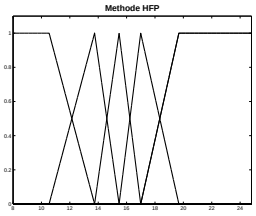
$$D_{s-1} = argmax\left(\frac{D_{s-1}^f}{D_s}\right) \qquad \forall f = 1,\ldots,s-1 \tag{4.7}$$

Les fusions seront menées jusqu'a obtenir une partition comportant un seul SEF (ou deux, une partition se réduisant au SEF universel revenant à la même chose que supprimer la variable considérée).

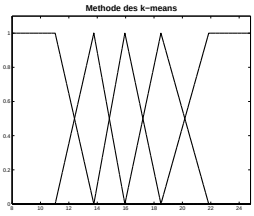
4.3.2 Méthode des k-means

Cette méthode consiste à appliquer l'algorithme des k-means sur l'ensemble des données relatives à une dimension, et à construire les SEF à partir des k centres obtenus. A chaque centre correspondra le noyau d'un futur SEF. Ces mêmes noyaux serviront ensuite d'extrémité aux SEF adjacents. Chaque SEF sera donc défini par trois des centres trouvés par la méthode, un centre définira son noyau,

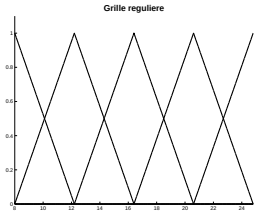
FIG. 4.2 – Partitions issues de chaque méthode pour une même variable



(4.2.a) méthode HFP



(4.2.b) méthode des kmeans



(4.2.c) Grille régulière

et les deux centres adjacents définiront ses extrémités. Pour les centres de valeur minimale et maximale, ce seront des demi-trapèzes qui seront construits.

La figure 4.2.b est un exemple de partition obtenue par la méthode des k-means

4.3.3 Grilles régulières

Il s'agit d'une méthode très simple, qui consiste simplement à répartir les centres de manière uniforme sur l'espace de sortie et à ensuite construire les SEF à partir de ces derniers. Hormis pour connaître les extrémités du domaine, il n'y donc pas besoin, dans le cas des grilles régulières, de recourir aux données pour construire les partitions.

Les figures 4.2.a, 4.2.b et 4.2.c proviennent du partitionnement d'une variable du problème de régression. Il est visible que chaque méthode conduit a des partitions assez différentes qui ne sont pas équivalentes entre elles.

4.3.4 Protocole de test

Le problème est ici de choisir la méthode de partitionnement à utiliser pour obtenir les meilleurs résultats. Afin de comparer ce qui est comparable, toute chose, hormis la méthode de partitionnement, doit être égale par ailleurs. Les diverses méthodes qui ont servi à évaluer la qualité des partitions ont déjà été évoquée dans l'introduction, et seront explicitée dans la suite du chapitre (sauf pour l'OLS qui a déjà été expliqué en détail).

- **Paramètres des méthodes** : ils sont variables et choisis de manière à assurer un indice de couverture d'au moins 90% pour un seuil d'activation de 0.1. Il s'agit par exemple du *MuMin* de la méthode HFP ou du critère d'arrêt de la méthode OLS.

- **Partitionnements flous** : ils sont identiques pour chacune des méthodes. Nous avons choisi d'en prendre un nombre identique pour chaque variable, même si ce n'est pas optimal.
  - Problème de régression : chaque dimension d'entrée est divisée en 5 SEF, ce qui semble être une bonne moyenne.
  - Problème de classification : chaque dimension d'entrée est divisée en 3 SEF, ce qui ne fournit pas des résultats optimaux, mais augmenter ce nombre amène des problèmes d'indice de couverture pour certaines méthodes, à cause du trop grand nombre de variables.
- **perte de performance** : En ce qui concerne la réduction du vocabulaire de sortie pour l'algorithme OLS et l'élagage des arbres flous, la perte relative autorisée de performance a été fixée à 10% (seulement pour le problème de régression en ce qui concerne l'OLS).

Une fois ces paramètres fixés, nous pouvons analyser la sensibilité des différentes méthodes au partitionnement choisi afin de pouvoir sélectionner la meilleure façon de construire ce partitionnement. Ce sont les résultats de cette analyse qui sont maintenant présentés.

4.3.5 Analyse des méthodes de partionnement

Les tableaux récapitulatifs présentés dans cette section comportent les résultats de chaque méthode pour chaque type de partitionnement, sur l'ensemble de chaque jeu de données. Les labels des méthodes sont les suivants :

- OLS : algorithme OLS modifié, sans réduction du vocabulaire de sortie
- OLSr : algorithme OLS modifié, avec réduction du vocabulaire de sortie
- WM : méthode de Wang & Mendel
- HFP : méthode décrite dans la section 4.4.2 et utilisée pour évaluer et induire les système.
- Ar : Arbre non-élagué
- ArEl : Arbre élagué, avec un élaguage par feuille limité à une perte relative de performance (10%)
- Moy : moyenne des performances obtenues sur l'ensemble des méthodes

L'objet de cette section n'étant pas l'étude des performances de chaque méthode d'induction, ce n'est que plus loin que nous détaillerons le fonctionnement de ces méthodes. Les chiffres présents dans les tableaux sont les indices de performance déjà définis précédemment (ErM pour la régression, nombre de mal classés pour la classification). Ces indices sont calculés pour l'ensemble des jeux de données.

TAB. 4.1 – Performances (PI) des méthodes pour chaque partitionnement, problème de régression

	OLS	OLSr	WM	HFP	Ar	ArEl	Moy
HFP	0.128	0.137	0.125	0.138	0.134	0.136	0.133
K-means	0.129	0.138	0.147	0.145	0.137	0.141	0.140
Regular	0.142	0.147	0.157	0.154	0.151	0.157	0.151

TAB. 4.2 – Performances (PI) des méthodes pour chaque partitionnement, problème de classification

	OLS	OLSr	WM	HFP	Ar	ArEl	Moy
HFP	13	26.25	20.25	36.5	24.5	27.5	24.6
K-means	28.25	30.75	35.25	33.5	30	31.25	31.5
Regular	13.75	32.75	27.5	12.75	46.75	43	29.5

Problème de régression

Le tableau 4.1 résume les résultats obtenus pour le problème de régression. Pour l'ensemble des méthodes, le classement est le même : la méthode HFP fournit les meilleurs résultats, suivie de la méthode des k-means puis des grilles régulières dont les performances sont toujours les moins bonnes. La faible variation entre les méthodes peut s'expliquer par la nature du problème (il est en effet très facile d'atteindre de tels ordres de performances dans ce problème, uniquement en sélectionant deux variables). De ce fait, les différences peuvent être jugées significatives, et nous ne pouvons pas ignorer que l'ordre des méthodes est toujours le même.

Problème de classification

Le tableau 4.2 résume les résultats obtenus pour le problème de classification. Rappelons que le nombre de SEF par variable (3) était peu élevé et loin d'être optimal. Malgré tout, les partitions HFP fournissent presque partout des résultats nettement meilleurs que les deux autres méthodes (sauf pour, justement, la méthode d'évaluation acompagnant HFP).

En ce qui concerne la méthodes k-means et les grilles régulières, on pourrait de prime abord les croire équivalentes au vu des résultats moyens. Malgré tout, en poussant l'analyse un peu plus loin, on s'apercevra rapidement que les résultats des grilles régulières ont une variance bien plus grande, et que la méthode dont les résultats sont les plus dépendants des partitions (les arbres) fournit des résultats

nettement meilleurs avec la méthode k-means. Tout cela indique qu'il est très hasardeux de se fier aux grilles régulières, qui tantôt fourniront de bons résultats, tantôt en fourniront de mauvais.

4.3.6 Méthodes de partitionnement : conclusion

Les résultats indiquent que c'est la méthode HFP qui fournit les meilleurs SEF, résultats qui intuitivements semblent corrects. Cette tendance a été confirmée sur d'autres jeux de données tout au long de ce stage (notamment pour le problème du chapitre 6), sans pour autant qu'une vérification systématique telle que celle présentée ici soit faite. C'est donc la méthode HFP que nous retiendrons parmi les trois présentées.

Une fois la méthode sélectionnée, reste à choisir la granularité des partitions (ou nombre de SEF par variables). La suite présente une méthode asociée à la méthode HFP dans [GC03], et qui permet d'évaluer assez rapidement les systèmes induits par chaque partitionnement.

4.4 Sélection de la granularité des partitionnements

La sélection de la granularité des partitionnements est une tâche qui sera tantôt assez aisée, tantôt très délicate et complexe. Il est parfois très difficile d'évaluer le nombre de SEF qui permettront d'induire des systèmes d'une interprétation aisée et qui auront des performances valables. Il est donc très important de pouvoir rapidement évaluer les différentes combinaisons de granularité possibles, et c'est dans ce but qu'a été développée la méthode explicitée ici.

4.4.1 Sélection du meilleur partitionnement

La méthode, présentée dans [Gui01b], est la suivante : elle consiste, en partant d'un partitionnement comptant un nombre très réduit de SEF (par exemple, un SEF universel pour chaque dimension d'entrée), à raffiner successivement les partitions des variables sélectionnées et à observer l'évolution des critères de performance, de couverture et d'interprétabilité. A chaque étape, le raffinement opéré est celui qui apporte le meilleur gain de performance par rapport au système précédent.

En disant raffinement, nous entendons augmenter le nombre de SEF d'une unité dans la partition de la variable considérée. La méthode HFP, de par sa philosophie hiérarchique, facilite ce travail en gardant en mémoire les SEF produits pour chaque niveau de granularité, mais il est parfaitement envisageable de reprendre cette démarche pour une autre méthode de construction de SEF.

A chaque étape de ce raffinement, une base de règle complète (c'est-à-dire qui comprend toutes les combinaisons possibles entre SEF) est générée et les conclusions sont calculées pour un certain nombre d'entre elles grâce à l'algorithme décrit ci-dessous. Une fois l'ensemble de tous les systèmes évalués de cette manière, il est possible de sélectionner le meilleur d'entre eux.

4.4.2 Algorithme d'évaluation des systèmes

Tout d'abord, l'ensemble des règles possibles est construit, soit $r_p = \prod_{i=1}^{p} s_i$ le nombre total de règles, avec s_i le nombre de SEF pour la dimension d'entrée i.

La conclusion de la règle r est ensuite calculée en considérant l'ensemble des exemples appartenant à E_r (E_r sera défini plus loin). Dans le cas d'un problème de type classification, la conclusion est la classe majoritaire des exemples de E_r. Dans le cas d'un problème de régression, le calcul est le suivant :

$$C_r = \frac{\sum_{i \in E_r} \mu_r(x_i) * y_i}{\sum_{i \in E_r} \mu_r(x_i)}$$

où C_r est la conclusion de la règle r, $\mu_r(x_i)$ le degré de vérité de l'exemple i pour la règle r, et y_i la sortie observée pour l'exemple i.

L'ensemble E_r, quand à lui, peut être construit selon deux stratégies différentes, l'une appelée *décrémentale*, l'autre *minimum*. Dans les deux cas, elles nécessite la définition de deux paramètres, *EffectifMin* (qui correspondra à l'effectif minimum de E_r pour que la règle r soit conservée) et *MuMin* (qui correspondra au degré minimum d'activation de la règle par un exemple pour que celui-ci puisse appartenir à E_r).

La stratégie dite *décrémentale* ne retiendra que les exemples qui activent le plus la règle. D'abord, seuls les exemples qui activent la règle avec $\mu(x_i) = 0.7$ seront retenus. Ensuite, si le nombre d'exemples correspondant à ce critère est inférieur à *EffectifMin*, le seuil d'activation sera décrémenté petit à petit jusqu'à ce que le nombre d'exemple sélectionnés atteigne *EffectifMin* ou que le degré du seuil atteigne *MuMin*.

La stratégie dite *minimum* consiste à retenir d'emblée l'ensemble des exemples qui activent la règle r avec un degré supérieur à *MuMin*.

Dans tous les cas (stratégie *décrémentale* ou *minimum*), si l'effectif de l'ensemble E_r est inférieur à *EffectifMin* lors de l'arrêt de la stratégie, la règle r n'est pas sélectionnée car considérée comme trop peu influente (on considère que les exemples qui activent peu la règle en activeront d'autres à un niveau plus élevé, qui seront donc dominantes dans leur inférence).

TAB. 4.3 – Granularités retenues pour les partitions du problème de régression

Variable	1	2	3	4	5	6
Granularité	2	6	6	5	6	4

TAB. 4.4 – Granularités retenues pour les partitions du problème de classification

Variable	1	2	3	4	5	6	7	8	9
Granularité	3	2	4	2	3	6	3	6	2

Les deux paramètres *EffectifMin* et *MuMin* sont donc très importants, et doivent être choisis avec soin. En effet, le système de règle résultant dépend directement d'eux, ainsi donc que son évaluation dans la méthode HFP (évaluation qui pourra se faire sur base de critères et d'indices classiques).

4.4.3 Résultats de la sélection sur les problèmes traités

Nous avons appliqué la méthode de sélection aux deux problèmes. Pour chacun d'entre eux, les paramètres d'évaluation *EffectifMin* et *MuMin* ont respectivement été fixés à 2 et à 0.1. En ce qui concerne l'indice de couverture, nous lui avons imposé de rester au-dessus de 80% dans les deux cas. la stratégie appliquée était de type *décrémentale*.

Les tableaux 4.3 et 4.4 contiennent, pour chaque variable de chaque problème, les granularités choisies. Ces granularités sont celles qui ont présenté les meilleurs indices de performance. Cette sélection est basée sur le seul critère de l'indice de performance et ne fait que respecter les contraintes de l'interprétabilité, sans chercher à maximiser cette dernière. En effet, nous aurions tout aussi bien pu choisir des granularité moins importantes, ce qui aurait dégrader l'indice de performance numérique tout en facilitant sans doute l'interprétation.

4.5 Présentation des méthodes et comparaison

Comme pour la sélection de la méthode de construction des partitions, les méthodes utilisées seront d'abord présentées avant que le détail du protocole suivi durant les tests ne soit détaillé. Les résultats obtenus seront ensuite donnés et commentés.

Les méthodes OLS et HFP d'induction ont déjà été présentées. Pour la méthode HFP, l'induction de règles se fera par le même algorithme qui a servi à évaluer les

partitions. Les méthodes présentées ici sont donc celles des arbres de décisions flous et de Wang & Mendel.

4.5.1 Arbres de décision flous

Les arbres de décision flous tirent leur origine de la théorie des arbres binaires [Qui86],[BFOS84] classiques. Ils requièrent, pour être mis en œuvre, un partitionnement flou pré-défini (ce dernier peut par exemple provenir de la méthode HFP décrite dans la section 4.3.1).

Dans les arbres flous, à chaque nœud sera associé une variable, et à chaque branche sera associé un SEF de cette variable, ce qui fait qu'à chaque chemin conduisant à une feuille de l'arbre correspondra une règle floue. (la figure 4.3 illustre la notion d'arbre flou).

Les différentes notions qui suivent sont expliquées pour des problèmes de classification. Les mêmes notions pour les problèmes de régression étant très proches, nous ne les développerons pas ici.

Entropie

L'entropie d'un noeud n sera définie telle que :

$$H_n = -\sum_k p_k^n * log(p_k^n) \tag{4.8}$$

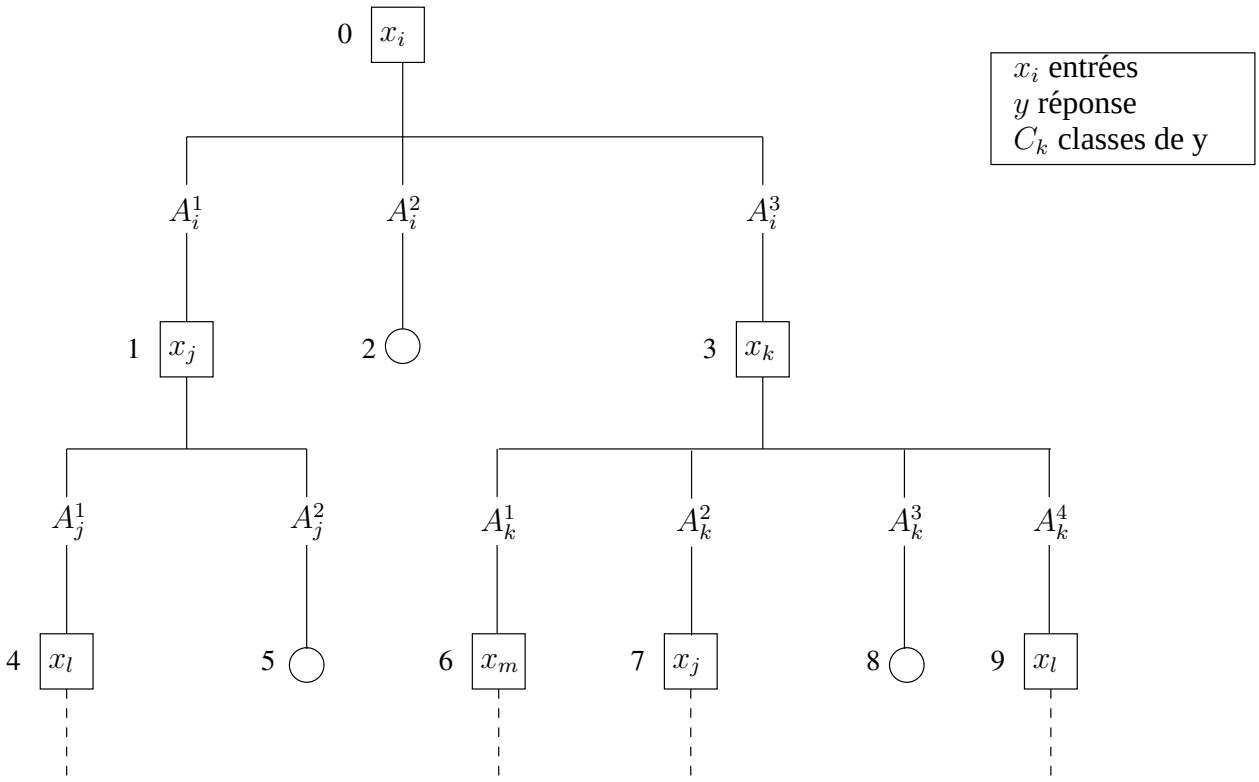
p_k^n étant la proportion floue d'exemples au noeud n qui appartiennent à la classe k (dans les arbres non flous, p_k^n est la probabilité pour un exemple du noeud n d'appartenir à la classe k) . p_k^n sera calculée d'après les degrés d'appartenance et s'écrira :

$$p_k = \frac{\sum_{i=1}^{N} \mu_k(y_i) \wedge \mu_n(x_i)}{\sum_{c=1}^{K} \sum_{i=1}^{N} \mu_c(y_i) \wedge \mu_n(x_i)}$$

où K est le nombre de classes, N le nombre d'exemples, $\wedge$ l'opérateur de conjonction choisi, $\mu_k(y_i)$ le degré d'appartenance de l'exemple i à la classe k et $\mu_n(x_i)$ le degré d'appartenance de l'exemple i au noeud n, qui sera défini comme :

$$\begin{array}{ll} \mu_1(x_i) = 1 & \textit{racine}, \quad i = 1 \ldots N \\ \mu_n(x_i) = \mu_{n-1}(x_i) \wedge \mu_n(x_i^j) & n > 1, \quad i = 1 \ldots N \end{array}$$

avec $\mu_n(x_i^j)$ l'appartenance de l'exemple x_i au SEF n de la variable j.



Règle noeud 2 : Si x_i est A_i^2 Alors y est C_2

Règle noeud 5 : Si x_i est A_i^1 et x_j est A_j^2 Alors y est C_5

FIG. 4.3 – Exemple d'arbre flou

L'entropie représente le désordre global du système. Par exemple, pour deux classes et dans le cas d'un arbre classique, l'entropie sera maximum si la probabilité d'appartenir à une classe est de 0.5, et minimum dans le cas où la probabilité d'appartenir à une classe est de 1 (et donc de 0 pour l'autre classe). Le but d'un arbre de décision, flou ou classique, est de diviser successivement l'espace des données en vue de diminuer au maximum l'entropie.

Gain d'entropie

A chaque noeud d'un arbre de décision flou, il faudra donc déterminer quelle variable, parmi celles qui n'ont pas encore été sélectionnées dans la branche considérée, est susceptible d'apporter la plus grande diminution d'entropie (et donc le plus grand apport d'information).

C'est dans ce but qu'est introduit la notion de gain d'entropie, qui représente le gain d'information apporté par la scission d'un noeud par une variable précise. Le gain d'entropie apportée par la scission d'un noeud n sur la variable j est :

$$G_n^j = H_n - \sum_{l=1}^{M_j} w_l H_l \tag{4.9}$$

où H_n est l'entropie du noeud n (voir équation 4.8), H_l l'entropie du noeud correspondant au SEF l de la variable j, et w_l le poids relatif du SEF, défini comme :

$$w_l = \frac{\sum_{k=1}^{K} \sum_{i=1}^{N} \mu_k(y_i) \wedge \mu_{n+l}(x_i)}{\sum_{k=1}^{K} \sum_{s=1}^{M_j} \sum_{i=1}^{N} \mu_k(y_i) \wedge \mu_{n+s}(x_i)} \tag{4.10}$$

Elagage

Une fois l'arbre construit dans son ensemble, il est possible de l'élaguer en transformant certains noeuds en feuilles. Un élagage est autorisé à la condition que les performances ne soit pas trop dégradées par ce dernier. Il peut bien entendu y avoir plusieurs critères permettant de décider si un élagage se fera ou non (basés sur la performance numérique, l'augmentation d'entropie, ...).

4.5.2 Wang & Mendel

Cette méthode dont les détails peuvent être retrouvés dans [WM92b] consiste à générer, à partir d'un jeu de données et de SEF pré-définis, un ensemble de règles où chaque règle correspond à un exemple (à l'instar de l'algortihme OLS, voir section 3.2.2).

Les conclusions des règles correspondent aux sorties observées des exemples correspondants. Si deux exemples correspondent à une règle identique, ce sera celui qui active cette règle avec le plus grand degré qui sera conservé pour initialiser la conclusion.

Il s'agit là d'une méthode très simple qui permet de facilement intégrer des exemples supplémentaires, même une fois le système construit.

4.5.3 Protocole de test

Comme lors de la sélection de la méthode de construction des partitions, nous avons fixé un certain nombre de paramètres pour pouvoir ensuite comparer les

performances de chaque méthode, à la fois du point de vue de l'interprétabilité et de la performance numérique.

La méthode de validation, les jeux de données, ainsi que la méthode de construction et de sélection des partitions ont déjà été détaillés. Jeux de données comme partitions floues seront les mêmes pour l'ensemble des tests sur chaque problème.

Opérateurs de conjonction, d'agrégation et de défuzzification seront ceux précisés au début de ce rapport (à savoir, respectivement **min**,**somme** et **sugeno**).

Ci-dessous sont précisés les paramètres pris en compte lors des tests. Il s'agit de paramètres qui fournissent en général de bons résultats, et qui ne sont pas optimaux pour chacun des problèmes, vu que le but recherché est de décrire le comportement des méthodes, et non de trouver la solution optimale.

- **OLS** : le critère d'arrêt choisi est la part de variance non-expliquée, fixée à 10^{-2} (au-delà, il y a de gros risques de surapprentissage). La perte relative de performance autorisée pour la réduction de vocabulaire dans le cas de la régression est de 10%
- **HFP** : La stratégie choisie est de type *décrémentale*, afin de limiter le nombre de règles générées. Deux séries de tests ont été réalisées : dans la première, les paramètres *MuMin* et *EffectifMin* ont été fixés à 0.1 et 3, afin d'assurer un bon indice de couverture. Dans la deuxième, ils ont été fixés à 0.4 et 3, afin de limiter le nombre de règles induites. Ces deux séries sont respectivement labellisées **HFP1** et **HFP2** dans les tableaux.
- **Arbres flous** : la perte relative de performance lors de l'élagage d'une feuille a été fixée à 10%. Là aussi, deux séries de tests ont été faits : dans la première, la profondeur de l'arbre (le nombre maximal de noeuds successifs dans une branche donée) a été fixée au maximum (c'est-à-dire la valeur de la dimension d'entrée), afin d'explorer l'arbre de la façon la plus complète possible. Dans la deuxième, cette profondeur a été fixée au tiers de la valeur de la dimension d'entrée. Les deux tests de la première série (respectivement pour les arbres complets et élagués) ont été labellisés **Ar1** et **ArEl1**, tandis que ceux de la deuxième ont été labellisés **Ar2** et **ArEl2**
- **Wang & Mendel** : cette méthode n'est pas paramétrée.

4.5.4 Résultats et commentaires

Les labels des méthodes testées sont restés identiques à ceux utilisés lors des partitionnements. Les tableaux, quand à eux, sont très similaires à celui qui est présenté dans la section 3.5.2. Ils contiendront, outre les critères présentés dans la section 3.3 (Ratio, PI, CI), le nombre de règles que chaque méthode induira (Nr). Comme pour les tableaux de la section 3.5.2, les valeurs présentées ici sont des moyennes sur l'ensemble des répétitions. Les indices de performances seront ici aussi séparés en PI_{app} (pour l'apprentissage) et PI_{val} (pour la validation).

TAB. 4.5 – tableau récapitulatif des résultats, problème de régression

Méthode	Nr	Ratio	PI_{app}	PI_{val}	PI_{glob}	CI(%)
OLS	87	1.08	0.136	0.265	0.144	93.7
OLSr	87	1.08	0.153	0.266	0.149	93.7
WM	132.75	0.73	0.156	0.233	0.141	96.5
HFP1	510	0.19	0.179	0.235	0.147	97.5
HFP2	54.25	1.38	0.237	0.267	0.178	75
Ar1	591.25	0.17	0.158	0.228	0.137	99.7
ArEl1	278.75	0.35	0.162	0.23	0.140	99.2
Ar2	24.25	4.12	0.205	0.246	0.160	100
ArEl2	12.75	7.51	0.228	0.264	0.174	95.7

L'indice de performance (PI_{glob}) sur l'ensemble du jeu de données est également calculé. L'indice de couverture CI est l'indice de couverture sur le jeu de données complet.

Problème de régression

Le tableau 4.5 résume les résultats obtenus pour le problème de régression. Celui-ci met en évidence certains caractères propres à chaque méthode. En ce qui concerne la performance numérique, nous tempèrerons notre jugement, vu qu'il a été prouvé à plusieurs reprises (voir [ARMF01] et [DSZ04]) que ce problème pouvait très bien se traiter en ne considérant que 2 variables et en n'induisant que 2 ou 4 règles. Nous nous garderons donc d'analyser les différences de performances entre méthodes, ces dernières n'étant pas vraiment significatives dans ce problème. Par contre, il sera possible d'analyser l'évolution des performances au sein d'une méthode.

Au niveau du nombre de règles, nous pouvons déjà écarter la méthode HFP aux paramètres lâches ainsi que les arbres à profondeur maximale. Il est en effet impossible de discerner une quelconque interprétation dans un tel nombre de règles. Pour la méthode HFP, des paramètres plus stricts semblent permettre de remédier à ce phénomène, mais il suffit d'observer indice de couverture et indice de performance pour se rendre compte que l'un baisse dangereusement alors que l'autre augmente significativement, ce qui fait que les performances globales du système s'en trouvent fortement dégradées .

Quand aux arbres flous à profondeur limitée, ils présentent à la fois de bonnes performances numériques et une interprétabilité accrue du fait de leur structure et du nombre très restreint de règles. Malgré tout, si l'on exclut la méthode HFP

TAB. 4.6 – tableau récapitulatif des résultats, problème de classification

Méthode	Nr	Ratio	PI_{app}	PI_{val}	PI_{glob}	CI(%)
OLS	59.25	1.61	5.5	7.5	13	95.5
OLSr	59.25	1.61	11.75	8.5	20.25	95.5
WM	94.25	1.03	5.5	6	11.5	97.2
HFP1	560.50	0.16	17.75	10.25	28	90
HFP2	34.25	2.12	14.25	6.5	20.75	72.7
Ar1	234.75	0.43	4.75	7.25	12	100
ArEl1	131.75	0.76	4.75	7.75	12.5	100
Ar2	33.75	2.96	14	5.25	19.25	100
ArEl2	12.75	6.74	29.75	9.25	39	86

à paramètres restrictifs, ce sont les méthodes qui présentent les moins bonnes performances sur l'ensemble du jeu de données.

Les méthodes relatives à l'algorithme OLS et à Wang & Mendel fournissent toutes deux des résultats intermédiaires, l'OLS induisant un nombre moins élevé de règles ainsi que de meilleures performances d'apprentissage lorsque le vocabulaire de sortie n'est pas réduit.

En terme de généralisation, ce sont les arbres flous à profondeur limitée qui présentent les meilleurs résultats (la méthode HFP à paramètres restrictifs présentant un indice de couverture trop faible pour qu'on puisse en dire qu'elle a de bonnes qualités de généralisation). A l'autre Extrémité se trouve l'OLS sans réduction du vocabulaire de sortie. Ces deux résultats n'ont rien de surprenant, au vu de la structure des arbres et de la façon dont l'OLS calcule les conclusions des règles. En ce qui concerne l'OLS, il faut aussi noter que le calcul des conclusions (et donc le second passage) s'est fait avec le même jeu de données que la sélection de règles (premier passage). Comme précisé dans [HM94], nous aurions pu prendre un autre jeu de données à l'effectif plus important, ce qui aurait sans doute pu améliorer les capacités de généralisations (et donc de performance).

Notons au passage que si notre but était la sélection de variables, les arbres de décision ainsi que la méthode HFP sélectionnerait le poids et l'année du modèle comme les deux variables les plus importantes, à l'instar de [ARMF01]

Problème de Classification

Le tableau 4.6 a la même structure que celui présenté pour le problème de régression. Les indices de performance correspondent au nombre d'exemples mal classés.

Les résultats constatés pour le problème de régression restent ici valables. En terme de performance, les résultats sont sensiblement équivalents pour toutes les méthodes, hormis pour la méthode HFP à paramètres restrictifs et pour les arbres à profondeur limitée où les performances sont nettement dégradées. Pour la méthode HFP, cette dégradation, à la fois en terme de couverture et en terme numérique, n'amène pas vraiment d'amélioration d'interprétabilité, vu que le Ratio n'est pas beaucoup plus élevé que pour l'OLS (qui a l'avantage de présenter des performances nettement meilleures). A l'inverse, les arbres à profondeur limitée ne présente une dégradation qu'en terme de performances numériques, dégradation contrebalancée par des règles comprenant au plus trois prémisses et des Ratio nettement plus élevés que pour les autres méthodes.

Si seules les performances numériques sont considérées, l'OLS dispute la palme aux arbres à profondeur maximum, mais dès que le critère d'interprétabilité intervient, les arbres à profondeur maximale sont écartés d'emblée.

Dans le problème de classification comme dans celui de régression, la méthode de Wang & Mendel semble fournir de très bons résultats (même si son nombre de règles est assez élevé). S'il est vrai que cette méthode est simple et efficace pour des problèmes peu complexes accompagnés de bon jeux de données, sa simplicité jouera contre elle lorsque le problème deviendra plus complexe et que la validité du jeu de donnée pourra être mise en doute (ce qui est très souvent le cas dans des applications pratiques, particulièrement lorsque l'on veut en extraire de la connaissance).

A titre de comparaison avec des méthodes classiques, les performances obtenues par analyse discriminante et par l'algorithme du plus proche voisin sont respectivement de 43 et de 28 mal classés. Les performances des méthodes présentées ici sont donc tout à fait honorables.

4.6 Analyse des méthodes

Les analyses qui suivent concerne les avantages et inconvénients présentés par chacune des méthodes. Il est important de se rappeler que nous considérons à chaque fois le double critère de l'interprétabilité et de la performance numérique. Les considérations qui suivent n'auraient aucun sens si un seul de ces deux crtières était pris en compte. Les critères que j'ai retenu m'ont semblé les plus aptes à rendre compte de la qualité de chaque méthode, ils me sont tout à fait personnel et il se peut qu'ils se rencontrent rarement, voir jamais, dans la littérature.

4.6.1 Algorithme OLS

L'algorithme OLS se caractérise avant tout par sa méthode de sélection des règles et par les bonnes peformances numériques qu'il offre.

Avantages

- **Sélection des règles** : la méthode de sélection des règles les plus importantes de l'OLS fait qu'il en induira beaucoup moins que la plupart des autres méthodes. De plus, en orthogonalisant l'ensemble des règles restantes après chaque sélection, il s'assure de ne pas sélectionner de règles redondantes. Parmi l'ensemble des méthodes, c'est celle qui génère le moins de règles, avec les arbres de décision flous (lorsque leur profondeur est limitée par un paramètre).
- **Performances numériques** : les conclusions générées par l'algorithme OLS sont optimales au sens des moindres carrés, ce qui fait que l'OLS fournit en général de très bonnes performances numériques. Il faut malgré tout garder à l'esprit que ces conclusions sont optimales pour les données à partir desquelles elles sont calculées, il faut donc s'attendre à une dégradation des performances avec d'autres données, même si l'OLS possède de bonnes capacités de généralisation. Il faut noter que plus le jeu de données utilisé pour calculer les conclusions (second passage de l'algorithme) sera important, plus cette dégradation aura de chance d'être réduite.
- **Règles significatives** : les règles que l'OLS sélectionne " expliquent "de manière significative le problème, ce qui représente un grand intérêt du point de vue de l'interprétabilité.

Inconvénients

- **Conclusions hors-domaine** : les conclusions générées par l'OLS ne se situent pas forcément à l'intérieur du domaine de variation de la sortie, ce qui peut parfois poser de nombreux problèmes d'interprétation, principalement pour les problèmes de classification. Pour remédier à ce problème, nous avons introduit la réduction du vocabulaire de sortie, mais si celle-ci augmente l'interprétabilité du système, elle en dégrade aussi les performances numériques.
- **Complétude des règles** : les règles induites par l'OLS sont complètes, ce qui pose rapidement des problèmes de couverture des données si jamais le problème dépasse les petites dimensions (≥ 5 variables)
- **Besoin de SEF pré-définis** : l'algorithme ne construit pas les SEF dont il a besoin, du moins pas si l'on veut que ses résultats soient interprétables. Il faut donc utiliser une autre méthode pour lui founir ces SEF.

4.6.2 Méthode HFP et algorithme d'évaluation de partitions

Cette méthode se démarque principalement par sa manière de construire les SEF et par le fait qu'elle soit autonome (c'est-à-dire qu'elle est capable d'induire un système interprétable complet à partir des seules données).

Avantages

- **Construction de SEF** : les SEF construits par la méthode HFP sont totalement guidés par les données, ce qui fait d'une part qu'ils sont très intéressants d'un point de vue interprétabilité, et que d'autre part ils permettent dans la plupart des cas d'induire des systèmes à la performance accrue par rapport à d'autres méthodes de construction de SEF. Même si c'est le seul avantage significatif de la méthode, il est de taille. De plus, le fait que chaque partition soit construite de manière incrémentale permet de garder une trace de l'ensemble des systèmes de partitions disponibles, et donc de pouvoir revenir sur son choix par la suite si cela s'avère nécessaire.

Inconvénients

- **Sensibilité aux paramètres** : il est difficile de trouver des paramètres *Mu-Min* et *EffectifMin* qui induisent de bons systèmes flous. S'ils sont trop restrictifs, l'indice de couverture du système résultant sera souvent trop faible, tandis que s'ils sont trop lâches, ils induiront un nombre de règles gargantuesque et donc un système aux bonnes performances numériques mais ininterprétable. Cette méthode est donc peu adaptée à l'induction d'un système à la fois interprétable et performant.
- **Temps global d'éxecution** : mener la méthode a son terme (de la construction des SEF au système induit) est très long par rapport à la qualité résultante (en considérant la qualité globale du système). En effet, une fois que le problème atteint une certaine complexité (en nombre de variables ou de SEF), l'exploration des différentes partitions peut prendre un temps assez impressionnant, sans pour autant fournir de bons résultats.

4.6.3 Arbres de décision flous

Parmi les méthodes présentées, celle des arbres de décision flous est la seule à induire des règles incomplètes, ce qui présente à la fois avantages et inconvénients. Au vu des résultats obtenus, nous n'analyserons ici que les arbres dont la profondeur d'exploration est limitée, par exemple par le nombre de noeuds d'une branche ou par le gain d'entropie minimum pour qu'un noeud soit scindé en plusieurs branches.

Avantages

- **Extraction de connaissance** : les arbres de décisions flous, de par leur structure même et de par leur méthode d'exploration, fournissent de très bons outils lorsqu'il s'agit d'extraire de la connaissance d'un problème. Chaque feuille et chaque noeud représente en effet une mine d'information pour l'oeil avisé.
- **Hiérarchisation des variables** : le fait que chaque noeud de l'arbre se sépare sur la variable la plus informative à son niveau est d'un grand intérêt en ce qui concerne l'interprétabilité et la sélection des variables. Cette hiérarchisation permettra de se rendre compte de l'importance de chaque variable, ou de l'inutilité de certaines d'entre elles.
- **Incomplétude des règles induites** : le nombre de règles induites par l'arbre est comparable au nombre de celles induites par l'algorithme OLS, à ceci près que ces dernières sont incomplètes, et que leur interprétation en est donc facilitée.

Inconvénients

- **Problèmes de régression** : les arbres flous sont moins bien adapatés aux problèmes de régressions (notament pour l'extraction de connaissances), même s'ils fournissent de très bons résultats pour ces derniers.
- **Performances numériques** : même s'ils ont des performances numériques très honorables, ces dernières sont généralement moins bonnes. Cela vient du fait que leur exploration est volontairement limitée pour des raisons d'interprétabilité.
- **Perte d'informations** : il faut noter que le passage des arbres à leur système flou équivalent (c'est-à-dire, une règle par feuille) provoque une perte d'information importante.
- **Besoin de SEF pré-définis** : comme pour l'OLS, la méthode des arbres de décision flous implique qu'on possède des SEF déjà construits via une autre méthode.

4.6.4 Wang & Mendel

Wang & Mendel est une méthode d'induction très simple et qui fournit pourtant des résultats assez performants, à la condition d'être certain de la qualité du jeu de données.

Avantages

- **Simplicité** : la méthode de Wang & Mendel se passe de paramètres, ce qui fait qu'elle ne souffre pas de sensibilité à ces derniers. De plus, cette simplicité fait que de nouveaux exemples peuvent très facilement contribuer au système existant.
- **Rapidité** : la méthode, du fait de sa simplicité, est très rapide, et est donc très efficace pour estimer le comportement d'un système ou le niveau des performances qu'on peut en attendre.

Inconvénients

- **Nombre de règles** : il peut être assez élevé, et de plus les règles sont complètes, ce qui entraîne un certain manque d'interprétabilité des systèmes.
- **Besoin de SEF pré-définis** : même si la méthode originale propose une construction de SEF (comme pour l'OLS), ces derniers ne sont pas interprétables.
- **Gestion des conflits** : la gestion des conflits entre règles de même prémisses est très basique, et peut parfois conduire à des aberrations et a de mauvaises performances pour peu que le jeu de données contienne des exemples erronés ou soit de piètre qualité.

4.7 Une méthode idéale ?

Comme vous l'avez lu, chacune de ces méthodes présente des avantages et des inconvénients. En pratique, il s'agit souvent de trouver le meilleur système qui soit un compromis entre interprétabilité et performance. ce problème est loin d'être simple, et trouver un juste équilibre est souvent long et hardu. La meilleure chose à faire, selon moi, est de tirer parti des qualités de chaque méthode.

Ainsi, la méthode HFP permettra de construire des partitions floues qui correspondront bien au problème posé. Le système de partitionnement hiérarchique permettra de choisir la granularité - ou la complexité - la mieux adapatée à nos besoins.

L'utilisation des arbres flous, elle, permettra d'affiner les résultats, d'éliminer des variables qui n'apportent pas d'informations, ou encore de mieux comprendre le fonctionnement du système en donnant son comportement général.

La méthode de Wang & Mendel pourra être utilisée pour se faire une idée des performances numériques qu'on peut atteindre ainsi que des tendances du système. Plus encore que pour les autres méthodes, il faut rester prudent quand aux résultats qu'elle fournit.

Finalement, l'algorithme OLS pourra être utilisé pour sélectionner les règles les plus importantes (et donc diminuer leur nombre) et calculer des conclusions optimales (dont le vocabulaire pourra éventuellement être réduit par la suite).

Il reste à régler le problème des règles complètes, ces dernières devenant rapidement ininterprétables lorsque la dimension du problème grandit. Toutes les méthodes présentées induisent des règles complètes, hormis les arbres flous (mais leurs performances sont souvent moins bonnes), il est donc indispensable, pour pouvoir extraire de la connaissance des systèmes et les interpréter, de simplifier les bases de règles obtenues. Ce problème sera abordé dans le chapitre suivant.

Le tableau 4.7 est un résumé des qualités de chaque méthode, en terme de performance numérique, d'interprétabilité des résultats, de nombre de règles induites et de temps d'éxecution.

TAB. 4.7 – Méthodes d'induction : tableau récapitulatif

Nom	Performance	Interprétabilité	Nbre règles	Temps
OLS d'origine	$++$	$--$	$+$	$-$
OLS modifié	$++$	$+$	$+$	$-$
Arbres Flous	$-$	$++$	$++$	$+$
Wang & Mendel	$+$	$-$	$-$	$++$
HFP	$--$	$+$	$--$	$--$

Rappelons aussi que la construction des partitions consiste ici en une étape distincte de l'induction de règles, et que les partitions, une fois construites, ne sont plus modifiées. Nous aurions pu envisager une autre approche que l'approche monodimensionnelle et optimiser les partitions en même temps que nous cherchions à optimiser le système de règles. Mais pour qu'une telle approche puisse être satisfaisante du point de vue de l'interprétabilité, il est indispensable de poser des contraintes, et quelques méthodes ont déjà été proposées dans ce but [Glo02].

Chapitre 5

Algorithme de simplification : étude et perspectives

5.1 Introduction

Moins nombreuses sont les prémisses des règles constituant un système flou, plus il est facile d'en extraire de la connaissance utile. C'est pour cette raison qu'a été développé l'algorithme de simplification présenté ici. De plus amples détails pourront être trouvés dans [GC03] et [Gui01b].

Cet algorithme, utilisable à partir de n'importe quelle base de règle, ainsi que les motivations qui ont été à la base de sa création, vous sont présentés dans la section 5.2. Cette présentation sera suivie d'une analyse de l'algorithme (section 5.3) et de la description des propositions faites en vue de l'améliorer (section 5.4).

5.2 Motivations et présentation de l'algorithme

De nos jours, grâce à l'avènement de l'informatique, à la rapidité croissante des processeurs et aux méthodes d'apprentissages, avoir un système possédant un grand nombre de variables devient très courant.

En ce qui concerne la logique floue, traiter beaucoup de variables peut rapidement devenir un problème si l'interprétabilité des résultats est importante (Or, si elle ne l'était pas, une méthode de type "boîte noire"pourrait très bien être utilisée) : d'une part, il est très difficile d'apporter une signification à des règles comportant plus d'une demi-douzaine de variables dans leurs prémisses, l'influence de chaque variable et/ou d'un ensemble de variables étant alors pratiquement impossible à discerner, et d'autre part, il faudra un nombre impressionnant de règles pour assurer un indice de couverture suffisant. Dans ces conditions, il est donc parfois très difficile d'extraire de la connaissance à partir des systèmes induits, en parti-

culier de ceux qui fournissent des règles complètes (une règle est dite complète si ses prémisses contiennent l'ensemble des variables).

L'algorithme qui suit a donc pour objectif de jeter les bases d'une méthode de simplification des règles, qui facilitera l'interprétation de ces dernières ainsi que l'extraction de connaissances. D'autres méthodes ont déjà été proposées, mais la plupart ont soit une vue trop globale (élimination pure et simple d'une variable), soit une vue trop particulière (suppression des variables une à une dans chaque règle) de la problématique. La méthode qui suit recherche une solution intermédiaire, en s'occupant des relations qui pourraient éventuellement exister entre plusieurs règles.

5.2.1 Notion de groupe de règles

Les groupes de règles permettront de rassembler et de simplifier par la suite les règles qui expriment des idées ou tendances similaires, mais dont les prémisses sont légèrement différentes.

Définition d'une distance entre règles

La distance entre deux règles dépend du nombre de prémisses qui diffèrent entre elles. La distance entre 2 règles a et b s'exprime comme :

$$d^r(a,b) = \sum_{j=1}^{p} d^j_{part}(a,b) \tag{5.1}$$

où $d^j_{part}(a,b)$ est la distance partielle entre a et b pour la variable j. En notant SEF^a_j le label du SEF de la variable j pour la règle a, on a :

$$d^j_{part}(a,b) = 0 \quad si \quad \begin{cases} SEF^a_j = SEF^b_j \\ ou\ SEF^a_j = ANY \\ ou\ SEF^b_j = ANY \end{cases}$$

$$d^j_{part}(a,b) = 1 \quad sinon$$

où le label ANY représente l'absence de la variable dans la règle. Le fait que la distance entre 2 labels ne soit pas prise en compte (par exemple, le fait que $d_{part}(petit, grand) = d_{part}$(*tres petit,tres grand*)) peut sembler étrange, mais convient bien à cet algorithme, vu que la distance entre labels n'a plus d'importance dès lors que la variable est éliminée des règles concernées.

Hétérogénéité d'une règle

Soit σ^r la variance sur la sortie des exemples qui activent la règle r, et σ la variance sur la sortie de l'ensemble des exemples. L'hétérogénéité de la règle r est alors définie comme :

$$H^r = \frac{\sigma^r}{\sigma} \tag{5.2}$$

Une hétérogénéité proche de 1 signifiera que la règle active des exemples dont les sorties sont très différentes, et sera par conséquent une image de l'inconsistance de la règle. Une hétérogénéité élevée signifiera en effet que des exemples de mêmes prémisses ont des conclusions très différentes. Il faut donc veiller a ce que cette hétérogénéité ne soit pas trop élevée.

Groupe de règles

Un groupe de règles est défini comme un ensemble de règles dont les prémisses ne diffèrent que sur une variable j. Au vu de la définition de la distance entre règles (voir équation 5.1), un groupe de règle est l'ensemble des règles dont la distance est 0 ou 1 (Dans le cas où deux règles ne diffèrent que sur des prémisses où l'une d'entre-elle a le label ANY, la distance serait de 0).

5.2.2 Algorithme de simplification

Le détail des algorithmes utilisés est disponible en annexe E, ou dans [GC03].

Généralités

L'algorithme se base sur trois étapes successives. Au cours de ces étapes, les simplifications successives fournissent des systèmes dont les performances et/ou l'indice de couverture sont dégradés. Plusieurs paramètres sont donc nécessaires pour vérifier que ces dégradations ne sont pas trop importantes :

- $CurErr$: la performance initiale du système
- $Loss_{thres}$: la perte de performance autorisée, exprimée en %
- CI_{thres} : l'indice de couverture minimal autorisé
- H_{thres} : le seuil d'hétérogénéité (eq. 5.2) au-delà duquel une nouvelle règle provenant d'une simplification n'est pas acceptée

En ce qui concerne la perte de performance, une boucle extérieure incrémente le pourcentage de perte autorisée, pour atteindre finalement la valeur $MaxErr = CurErr(1 + Loss_{thres})$. Cette incrémentation permet de réaliser en premier lieu les simplifications qui entraînent le moins de perte de performance, et rend l'ensemble plus progressif.

Première étape : fusion de groupes

Cette étape consiste à construire les différents groupes de règles possibles, pour ensuite tenter de les fusionner en une règle générique où le SEF qui diffère entre les règles du groupe est remplacé par le label ANY. Sont d'abord considérées les règles dont la distance est 1, puis la méthode est étendue à celles dont la distance est 0 (cas possible lorsqu'il existe des règles incomplètes comportant le label ANY pour une variable j).

Une fusion n'est opérable que si la règle résultante a une hétérogénéité (eq. 5.2) inférieure au seuil fixé. Les fusions opérables sont ensuite toutes pratiquées simultanément. Le nouveau système est accepté si la perte de performance ne dépasse pas le seuil fixé.

Deuxième étape : suppression de règles

Une fois toutes les fusions possibles réalisées, les règles sont supprimées une à une. La suppression d'une règle est rendue définitive si la perte de performance ou la perte d'indice de couverture qu'elle entraîne ne dépassent pas les seuils fixés.

Troisième étape : suppression de variables

Une fois les deux premières étapes passées, toutes les variables de toutes les règles sont enlevées une à une de chaque règle. Chaque fois qu'une variable est supprimée, sa suppression est rendue définitive si la nouvelle règle ne dépasse pas le seuil d'hétérogénéité fixé et si la dégradation de la performance et de l'indice de couverture ne dépassent pas non plus les seuils fixés.

5.3 Analyse de l'algorithme

Cette partie analyse quelques défauts de l'algorithme de simplification dont je me suis rendu compte en l'utilisant. Cette liste n'est sans doute pas exhaustive, mais comprend selon moi des points importants à corriger qui pourraient améliorer significativement l'algorithme.

5.3.1 Conclusions du système résultant

Lorsque plusieurs règles sont simplifiées, d'autres supprimées, leurs conclusions en viennent à changer ou à disparaitre. Dans l'algorithme de simplification tel qu'il est décrit dans la section 5.2, il est apporté très peu de soin à la conservation des conclusions.

Il en résulte que le système résultant de la simplification peut souffrir d'un manque de couverture de l'espace de sortie (la couverture de l'espace d'entrée étant assurée par le critère d'indice de couverture). Ce problème survient principalement lorsqu'une zone de l'espace de sortie ne concerne que peu d'exemples. Lorsque peu d'exemples occupent une zone, le fait que ces derniers soient systématiquement inférés à une valeur légèrement supérieure ou inférieure à leur valeur réelle n'entraine qu'une faible dégradation de la performance globale du système. C'est le cas, par exemple, des problèmes de diagnostic de défaut, où les exemples sont pour la plupart sans défaut, mais où la zone d'inférence la plus importante est justement celle des défauts. Les deux figures 5.1.a et 5.1.b, qui proviennent de l'application traitée au chapitre 6, illustrent ce fait. Les graphiques montrent que les résultats sont principalement dégradés pour des valeurs proches de un (qui sont inférées pour la plupart à des valeurs proches de 0.5), alors que c'est cette zone qu'on cherche à inférer le mieux ! L'exemple donné ici est encore clément, et il peut arriver que la dégradation, même pour des contraintes relativement fortes (perte de perf inférieure à 10^{-1}, seuil d'hétérogénéité inférieur à 0.5, couverture minimale fixée à 90%) soit beaucoup plus importante. Il est même arrivé que des systèmes simplifiés ne comportent que des règles aux conclusions très proches de zero !

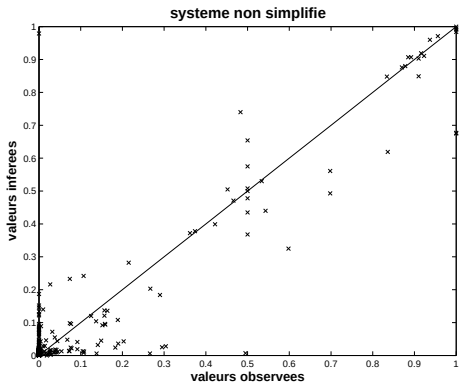
Pour corriger cet état de fait, une légère modification a déjà été apportée à l'algorithme : elle consiste à garder la dernière règle présentant une conclusion distincte des autres. Par exemple, dans le cas d'un problème de classifcation, cela assure que toutes les règles ayant une même classe pour conclusion ne soient pas supprimées. Cette modification s'applique malgré tout assez mal à un cas de régression où chaque règle aurait une conlusion distincte des autres. De plus, il est rare en pratique qu'une classe puisse être caractérisée par une seule règle.

5.3.2 Sensibilité locale

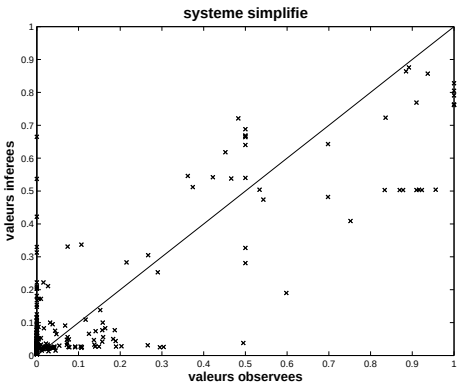
Un des autres problèmes de l'algorithme de simplification actuel est qu'une perte de performance dans une zone peut très bien être compensée par un gain de performance dans une autre zone. Il se peut que la suppression ou la simplification de certaines règles "ramène "certaines valeurs inférées vers les valeurs observées alors que le phénomène inverse sera observé dans d'autres zones de l'espace de sortie. Cet "équilibre "entre dégradation et amélioration de la performance aboutira à une performance globale peu dégradée, voire même améliorée.

L'apparition de ce problème ou de celui de la section 5.3.1 est fortement conditionnée par le choix du fichier utilisé pour calculer la performance résultante des différents systèmes. Le fichier choisi doit être équilibré, comporter un échantillon représentatif d'exemples et n'être ni trop pauvre en exemples (risque de particulariser le système pour ces exemples), ni trop volumineux (risque que la dégradation

FIG. 5.1 – Simplification : performances du système d'origine et du système simplifié



5.1.a Système d'origine



5.1.b Système simplifié

de performance soit "noyée "dans le nombre d'exemples).

5.3.3 Grandes dimensions et indice de couverture

Un autre problème se présente lorsque le système a de très grandes dimensions. De façon générale, plus un problème est de grande dimension, moins il va être facile de couvrir l'intégralité de son espace d'entrée. Le nombre de règles possibles croît en effet exponentiellement avec le nombre de variables, et il est souvent difficile dans les problèmes dépassant une taille moyenne (> 5 variables) d'inférer un ensemble de règles qui couvre tout l'espace de sortie tout en veillant à conserver leur interprétabilité (de plus, il se peut que certaines règles ne se rencontrent jamais en pratique). Ce phénomène prend bien entendu encore plus d'importance lorsque le nombre de SEF de chaque variable d'entrée augmente.

Il en résulte que l'indice de couverture initial d'un système est parfois assez bas, or le paramètre de couverture choisi pour l'algorithme est un **seuil absolu**. Remarquons également que l'indice de couverture ne pourra qu'augmenter lors d'une fusion de règle ou lors de la suppression d'une variable (2 cas qui rendent les règles plus générales), tandis que ce même indice de couverture ne pourra que diminuer lors d'une suppression de règles. Il en résulte deux défauts qui s'opposent lorsque l'indice de couverture initial est peu élevé : soit un seuil de couverture élevé est imposé et il y a de fortes chances qu'aucune simplification ne soit opérée, soit le seuil imposé est proche de l'indice initial, et il y alors un gros risque qu'une amélioration (par fusion de règles ou par élimination de variables) de cet indice soit suivie d'une forte dégradation (par suppression de règles) de cet

indice (et donc du système) qui sera quand même acceptée, puisque le seuil est absolu et ne tient pas compte de l'évolution de l'indice de couverture au cours de la simplification.

5.3.4 Sensibilité aux paramètres

La simplification, telle que présentée ici, est très sensible à ses divers paramètres. Augmenter ou diminuer quelque peu l'un d'eux (par exemple, diminuer l'hétérogénéité de 0.1, la perte de performance autorisée de 1 ou 2%), peut conduire à deux extrémités : soit le processus de simplification ne simplifie strictement rien, soit il simplifie tellement le système que les résultats de ce dernier deviennent totalement absurdes.

5.4 Propositions et perspectives d'amélioration

5.4.1 Changements structurels

Il s'agit ici de légers changements qui devraient permettre à la simplification de mieux s'adapter à divers cas de figures :

1. Suppression du test de couverture : En ce qui concerne la simplification, la programmation avait été faite de sorte que l'indice de couverture soit aussi testé lors des fusions de groupes et lors de la suppression de variables. Or, ces deux algorithmes ne peuvent qu'augmenter le seuil de couverture, vu qu'ils rendent les règles présentes dans le système plus générales. Le test a donc été supprimé car inutile dans ces deux parties de l'algorithme.
2. Test sur la couverture : dans la version originale, le test sur l'indice de couverture se fait sur un seuil absolu et non sur un pourcentage relatif comme cela peut être le cas pour la performance. Cela pose un problème dans le cas où le système d'origine a un indice de couverture faible, car dans ce cas il faut soit accepter de limiter fortement la simplification (dans le cas où un seuil plus élevé que l'indice initial, donc très difficile à atteindre, est imposé), soit prendre le risque d'obtenir finalement un système aux performances très dégradées (cas où le seuil rejoint l'indice d'origine). L'idée est de permettre à la couverture de se dégrader d'un certain pourcentage pendant la suppression de règle, et ce sans prendre en compte le seuil absolu si l'indice de couverture ne l'a jamais dépassé au cours de l'algorithme. La suppression de règle pourrait ainsi être permise, avec comme condition que la dégradation provoquée par l'ensemble cumulé des suppressions soit limitée à un pourcentage de l'indice de couverture obtenu juste avant ces

suppressions (de cette manière, la suppression de règles ne provoquerait pas la perte du gain de couverture réalisé lors de la fusion et de la suppression de variable).

3. Fusion individuelle : jusqu'a présent, l'ensemble des fusions possibles étaient réalisées avant qu'un test ne soit fait pour accepter ou refuser ces fusions selon la dégradation de performance qu'elles provoquaient. Il s'agit là d'un principe de " tout ou rien " qui ne m'a pas semblé très en accord avec la volonté de simplifier la base. Le changement proposé consiste donc à effectuer le test sur la dégradation de performance après chaque fusion et non pas après qu'elles aient toutes été réalisées. Cela devrait permettre de réaliser une partie des fusions là où avant aucune fusion n'était faite. Ce procédé nécessite en plus l'introduction d'une hiérarchie entre les fusions (par exemple, opérer d'abord les fusions qui améliorent le plus ou dégradent le moins la performance).

5.4.2 Notion de performance locale

Comme décrit dans la section 5.3, un des problèmes majeurs de la simplification est que la performance calculée est globale et ne prend pas en compte les dégradations locales. Il se peut donc très bien qu'une partie de l'espace d'inférence subisse une forte dégradation qui sera équilibrée par le rapprochement d'autre points inférés de leur valeur observée.

De plus, il existe un autre risque dans les problèmes de diagnostic de type de défaut. Dans ce genre de problème, la plupart des exemples se trouvent dans un état " normal "jusqu'à ce que survienne le défaut, et que la sortie prenne alors des valeurs exceptionnelles. Le changement peut bien sûr être graduel, mais la plupart des valeurs n'en resteront pas moins dans un état " normal ". Or, actuellement, l'indice de performance est l'erreur moyenne (ErM), calculée de la manière suivante :

$$ErM = \frac{1}{N}\sqrt{\sum_{i=1}^{N} \|\widehat{y_i} - y_i\|^2} = \frac{\sqrt{EQM}}{\sqrt{N}} = \frac{RMSE}{\sqrt{N}}$$

où N est le nombre d'exemples, $\widehat{y_i}$ la valeur inférée par le système et y_i la valeur observée. Ce calcul est équivalent a faire une somme pondérée des carrés des erreurs, où chaque point est pondéré avec un poids $\frac{1}{N}$. Dans le cas des problèmes de détection de défaut, il est évident que les quelques exemples représentant les défauts ont un poids négligeable par rapport à la multitude d'exemples représentant la normalité. Dans ce cas de figure, si la procédure de simplification dégrade fortement les performances autour des exemples de défaut, la perte de performance

globale résultante sera négligeable, or c'est justement les défaut que l'on veut le mieux inférer ! C'est de ces deux raisons (le fait que seule la dégradation globale soit prise en compte et le fait que le nombre d'exemples disponibles dans une zone soit très petit) qu'est née l'idée d'un calcul de performance locale ou encore d'une somme pondérée de diverses performances.

Le principe est simple : l'espace de sortie est découpé en plusieurs sous-ensembles et la performance est calculée indépendamment pour chaque sous-ensemble. Chacun de ces sous-ensembles est ensuite assorti d'un poids normalisé selon son importance, et une performance pondérée est ensuite calculée à partir de la performance globale et des performances locales. Un processus similaire est bien sûr conduit pour l'indice de couverture. La pondération, selon le cas, peut être automatisée (par exemple, un poids plus grand là où les exemples sont moins nombreux) ou pré-définie (cas où un expert souhaite privilégié une zone particulière).

Sous-ensembles de sortie

Selon le type de problème, les sous-ensembles qui décomposeront l'espace de sortie sont plus ou moins évidents :

- Sortie nette de type régression : le plus facile est de découper l'espace de sortie selon les spécificités du problème et les zones qui sont jugées les plus importantes. Pour reprendre le cas des défauts, il est plus important de conserver une bonne performance dans la zone du défaut que dans les autres, de plus, les exemples y sont nettement moins nombreux.
- Sortie nette de type classification : pas trop de difficulté ici, les classes étant déjà un découpage idéal, il sera malgré tout possible de donner une plus grande importance à des classes qui comptent peu d'exemples mais pour lesquelles il serait très important de faire un bon classement (par exemple : une classe de champignons rares et toxiques).
- Sortie floue : là aussi, le découpage est tout indiqué, il suffit de prendre chaque SEF séparément. Il faudra toutefois prendre garde à pondérer aussi les résultats par le niveau d'appartenance de l'exemple au SEF.

Performance pondérée

Le calcul de la performance pondérée pourrait être le suivant :

$$Perf_{pond} = \frac{Perf_{glob} + \sum_{i=1}^{K} P_i Perf_i}{2}$$

avec

$$\sum_{i=1}^{K} P_i = 1$$

et

$$\begin{array}{lll} Perf_i = & \frac{1}{n_i} \sum_{j=1}^{n_i} \|\widehat{y_j} - y_j\|^2 & \text{(en régression)} \\ Perf_i = & Mc_i & \text{(en classification)} \end{array}$$

où n_i représente le nombre d'exemples dont les valeurs observées sont dans la zone i, Mc_i le nombre d'exemples mal classés pour la classe i, P_i le poids de la zone i, K le nombre de zones (ou de classes ou de SEF) et $Perf_{glob}$ la perf globale définie plus haut. Pour $i \rightarrow \infty$ et des poids P_i équipondérés, on retrouverait la performance globale initiale.

De manière générale, cette performance pondérée pourrait permettre de rétablir un certain équilibre sur l'ensemble du système dans le cas où les exemples sont très peu présents dans une partie du domaine de variation de ce dernier (prise en compte de cas exceptionnels, de défauts, de données manquantes, ...). De façon plus particulière, la définition d'une telle performance peut permettre de se concentrer sur une partie du système et de constater l'évolution de la performance de cette partie en particulier.

L'utilisation d'une performance pondérée sous cette forme présente à la fois avantages et désavantages. Utilisée de manière agrégée, elle est directement substituable à la performance globale, et leurs comportements respectifs peuvent être facilement comparés. Mais agréger les performances locales entre elles provoque immanquablement une perte d'information plus ou moins importante, vu qu'il devient impossible de constater leurs évolutions individuelles. D'un point de vue purement pratique (c'est-à-dire, du point de vue de l'implémentation), il est bien sûr plus facile de ne considérer que la performance pondérée. Cependant, je reste persuadé que l'utilisation de toute l'information disponible permettra, si elle est utilisée judicieusement, d'obtenir de meilleurs résultats.

5.4.3 Groupes par conclusion - extension de la fusion

Une autre idée avancée, afin de combattre le peu d'intérêt apporté aux conclusions lors de l'algorithme originel, est de grouper les règles non par distance, mais par similarité des conclusions. Il est en effet fréquent que des règles ayant des conclusions similaires aient des prémisses elles aussi très similaires, les prémisses différentes jouant un rôle mineur dans les phénomènes que les règles essaient

de décrire. Considérons l'exemple du tableau 5.1 qui reprend deux règles d'un système essayant d'inférer la température à partir d'un ensemble de données. En considérant qu'une altitude commence à être *très élevée* à partir de $4000m$ d'altitude, il est évident que le simple fait d'être à une altitude *très élevée* induit une température *basse*. Il est donc possible, dans ces deux règles, d'éliminer l'ensemble des autres variables pour créer une règle générique du type **SI altitude EST *très élevée* ALORS température EST *basse***. Par contre, il serait très dangereux d'éliminer tout autre variable que l'altitude lorsque celle-ci est moins élevée.

TAB. 5.1 – Exemple trivial de système flou

SI altitude EST *très élevée* ET SI ciel EST *couvert* $\cdots$ ALORS température EST *basse*
SI altitude EST *très élevée* ET SI ciel EST *clair* $\cdots$ ALORS température EST *basse*

Le nouveau principe de fusion proposé est donc le suivant :

1. Grouper les règles dont les conclusions sont proches ou similaires
2. Au sein de ce groupe, essayer de simplifier les règles entre elles selon un ordre de distance croissant (on va donc des règles dont les prémisses sont les plus similaires vers celles dont les prémisses sont les plus différentes)
3. Vérifier que les simplifications ne crèent pas des règles qui entrent en conflit avec d'autres groupes
4. Pour chaque simplification successive, l'accepter ou pas selon un critère choisi (par ex., la dégradation de la performance initiale)

Ce système de fusion permettrait de simplifier l'ensemble des règles dans le même esprit que la fusion déjà réalisée (c'est-à-dire considérer les relations entre variables et entre règles) tout en s'assurant que le système résultant couvre sensiblement le même domaine de sortie que le système originel (pour peu que les paramètres de fusion soient bien ajustés). Ce principe allierait donc à la fois les notions de distance et de préservation des conclusions. Sa validité reste cependant à vérifier.

5.4.4 Perspectives

D'autres améliorations et idées ont été évoquées, qui visent à améliorer la méthode ou à éviter certains écueils habituellement rencontrés dans les méthodes d'apprentissage :

1. **Fonction de coût :** La proposition était ici de guider l'algorithme par une fonction de coût classique plutôt que par des tests séparés. Le respect de contraintes comme la dégradation maximum autorisée sur un indice étant assuré par une pénalité. La fonction de coût (ou de gain) serait donc de la forme :

$$F_{gain} = \sum \Delta_{gain} - \sum \Delta_{perte} - M$$

   où Δ_{gain} pourrait par exemple être le gain en indice de couverture, Δ_{perte} la dégradation de la performance pondérée, et M un nombre arbitrairement grand si une contrainte est violée, et valant zero sinon.

2. **Agrégation des systèmes :** Les méthodes utilisées pour évaluer la qualité de méthodes d'apprentissage pour un problème donné générant souvent plusieurs systèmes à partir de différentes parties du jeu de donnée (méthode de validation croisée, de bootstrap, ...), il serait peut-être utile que la simplification se fasse sur l'ensemble de ces systèmes agrégés. Cette agrégation de l'ensemble des règles provoquerait sans doute beaucoup de redondance, mais cette dernière serait éliminée par la simplification et permettrait dans certains cas de mettre en évidence les inconsistences existantes.

Dans le cadre de systèmes complexes consitués de règles complètes, la simplification des règles constitue une étape obligatoire pour pouvoir épuré le système et n'en tirer que la connaissance vraiment utile.Dans ce cadre, les idées présentées par cette algorithme représentent une très bonnes solutions. Malgré tout, il reste de nombreux détails à affiner pour qu'il devienne vraiment performant. De nombreux tests concernant cet algorithme de simplification ont été réalisés sur l'application qui va être présentée au chapitre suivant. Ces tests ont d'ailleurs permis de mettre bon nombre des défauts cités dans ce chapitre en évidence. C'est aussi sur cette application (entre autre) que seront menés les tests de l'algorithme de simplification modifié.

Chapitre 6

Application

6.1 Introduction

Le travail réalisé n'est pas resté au niveau des cas d'école déjà évoqués, mais a aussi eu des applications pratiques. Ces dernières ont principalement concerné un processus de dépollution anaérobie dont les détails sont présentés dans la section 6.2.

Pour cette application, les discussions avec l'expert et ses avis se sont révélés extrêmement importants. C'est de cette collaboration expert-méthode d'apprentissage qu'il est question dans la section 6.3.

Enfin, les résultats obtenus à partir des données qui nous étaient fournies sont analysés dans la section 6.4.

6.2 Présentation du procédé

6.2.1 La digestion anaérobie

Après avoir allégrement saccagé la planète pendant bon nombre de siècles (tout en s'étant surpassé ces dernières décennies), l'homme devient peu à peu conscient qu'il est temps de réagir s'il ne veut pas voir la terre devenir un vulgaire caillou sans vie. Parmi les préoccupations principales des défenseurs de l'environnement se trouvent la pollution et l'utilisation des énergies renouvelables (préoccupations d'ailleurs fortement corelées entre elles), notamment la pollution de l'eau, qui a trop longtemps été considérée comme une ressource inépuisable et manipulable à souhait. Parmi les nombreuses sources de pollution possibles (*origine thermique, bactérienne, urbaine, industrielle, toxique, ...*), certaines, dont le traitement des eaux urbaines, produisent de grandes quantités de boues organiques.

La *digestion anaérobie* (aussi appelée *fermentation mécanique*) est un procédé qui, à l'aide de bactéries, transforme des matières organiques en biogaz (principalement en méthane (60-70%) et en gaz carbonique). Ce processus biologique complexe, dépendant de micro-organismes anaérobie, se produit déjà spontanément dans les marais ou dans certains écosystèmes anaérobies (comme les systèmes digestifs des vaches, des termites ou encore de l'homme). Il a depuis été compris et maîtrisé par l'industrie, et sert principalement à la dépollution carbonique d'effluents. Ce procédé est d'autant plus intéressant que les biogaz produit par ce dernier peuvent être réutilisés comme source d'énergie. La figure 6.1 montre les principales étapes du processus de digestion anaérobie.

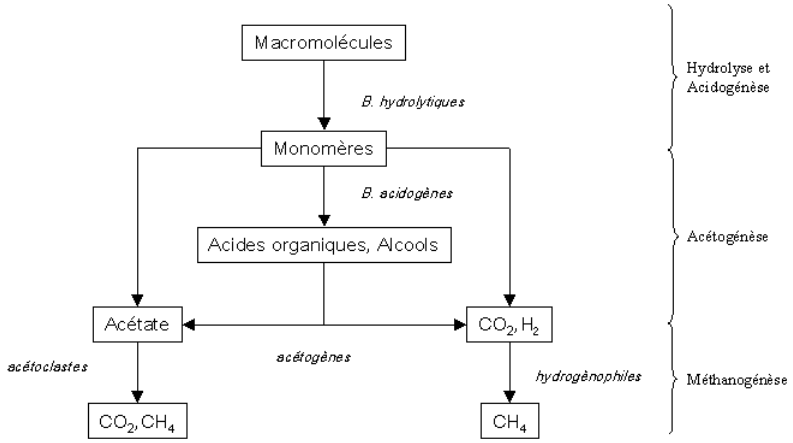


FIG. 6.1 – Etapes de la digestion anaérobie

Malgré tout, ce n'est que très récemment (dans les années 70) qu'industriels et chercheurs ont commencé à sérieusement s'intéresser à ce procédé. Cette réticence peut s'expliquer par le fait que la popualtion bactérienne nécessaire à la mise en œuvre de la digestion anaérobie est d'une croissance assez lente et d'une grande instabilité. Elle supporte en effet très mal les surcharges, l'acidité ou encore la présence de toxiques. Il est donc très important de pouvoir contrôler la stabilité du processus, vu que générer de nouvelles populations de bactéries prend un temps assez long.

De nombreux modèles mathématiques ont donc vu le jour avec pour objectif de contrôler au mieux ce phénomène, le modélisant de mieux en mieux, mais au prix d'une complexité toujours croissante. Le modèle de référence de l'IWA (International Water Association), l'ADM1 compte 26 variables d'état, 19 cinétiques biochimiques, 3 cinétiques de transfert gaz-liquide et 8 variables algébriques [BKA$^+$02].

La complexité du procédé justifie donc pleinement le recours à des méthodes

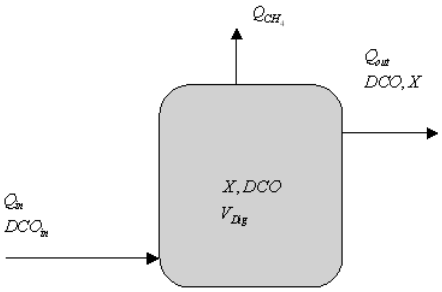


FIG. 6.2 – Représentation schématique d'un réacteur

d'apprentissage, tandis que sa nature biologique implique un besoin de compréhension important et une présence experte indispensable, deux critères qui jouent en la faveur du recours à la logique floue.

6.2.2 Présentation des données

Installation

TAB. 6.1 – Grandeurs mesurées couramment sur un bioprocédé anaérobie

Variable	Description	Unité
pH	pH au sein du réacteur	UpH
vfa	Conc. en Acides Gras Volatils au sein du réacteur	$mg.L^{-1}$
qGas	Débit de biogaz	$L.h^{-1}$
qIn	Débit d'alimentation	$L.h^{-1}$
ratio	Alcalinité partielle/Alcalinité Totale	
CH_4Gas	Taux de CH_4 dans le biogaz	$\%$
h_2Gas	Taux de H_2 dans le biogaz	ppm
cod	Conc. en Demande Chimique d'Oxygène au sein du réacteur	$gDCO.L^{-1}$

Les données sur lesquelles nous avons travaillé proviennent du laboratoire de biotechnologie de l'environnement (LBE) de Narbonne, laboratoire rattaché au centre de l'institut national de recherche agronomique (INRA) de Montpellier. L'installation expérimentale est constituée d'un réacteur de digestion anaérobie.

Les données ont été prises sur une période de 4 mois située entre février et mai 2004.

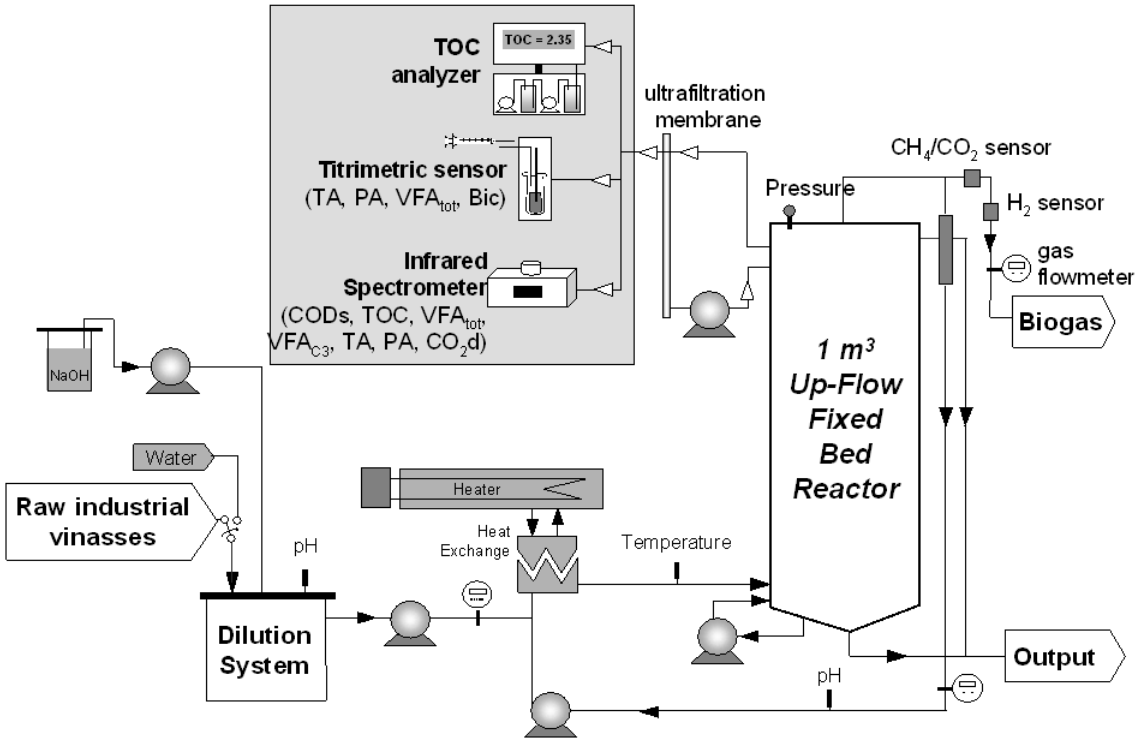


FIG. 6.3 – Synoptique du procédé pilote utilisé

Le procédé est un lit fixe, fait d'une colonne de $3,5$ mètres de haut et de $0,6$ mètre de diamètre, pour un volume utile initial de $0,948m^3$. Le réacteur, en plus d'être équipé des capteurs et appareils de mesure classiques (débits, température, pH) est doté de capteurs avancés (composition du biogaz) et expérimentaux (titrimètre, spectromètre), la figure 6.3 représente un schéma du procédé utilisé. Les principales variables d'entrée sont présentées dans le tableau 6.1, et le tableau 6.2 résume la composition moyenne des vinasses (effluents provenant de producteurs vinicoles et fortement pollués) utilisées.

Chacune des données, une fois enregistrée par un capteur, est ensuite prétraitée et analysée par un système expert complexe qui fournit, en sortie, les possiblités d'être dans différents états de sortie. Le tableau 6.3 est un résumé de ces états de sortie possibles. L'ensemble de ces états est discret et non-ordonné. En

TAB. 6.2 – Composition moyenne des vinasses utilisées

Nom	Concentration	Unité
COD totale	38.8	$gO2.L^{-1}$
COD soluble	36.2	$gO2.L^{-1}$
COT	11.5	$g.L^{-1}$
VFA totaux	9.3	$g.L-1$
NTK	484.5	$mg_{N-NTK}.L^{-1}$
NH_4	137	$mg_{N-NH4}.L^{-1}$
pH	4.3	

pratique, il est donc possible d'atteindre un état à partir de n'importe quel autre.

Nous allons maintenant détailler chacun de ces états :

– Sous-charge (UL) : Une sous-charge se caractérise par une charge massique faible et une faible production de biogaz, ainsi que par un rapport d'alcalinité et une épuration proches de 100%. Cet état indique que le bioréacteur n'est pas exploité au mieux de ses capacités. Les réacteurs à lit fixe, ayant une bonne rétention de biomasse, sont peu atteints par une sous-charge prolongée. Cet état n'est donc pas critique pour cette application particulière, mais il est souhaitable qu'il soit détecté au plus vite pour utiliser le plus efficacement possible le réacteur.

TAB. 6.3 – états de sortie possibles

1.	Sous-charge (UL)
2.	Normal
3.	Surcharge hydraulique (HO)
4.	Surcharge organique (OO)
5.	Acidogénèse
6.	Toxicité (TOX)

– Normal : Cet état se caractérise par un rapport d'alcalinité situé aux environs de 0.7 et par un Ph neutre. Cet état possède un domaine assez vaste, pouvant aller de la sous-charge à la surcharge légère.

– Surcharge Hydraulique : Dans cet état, la charge appliquée se trouve dans la zone de normalité, mais le taux de dillution trop élevé provoque un lessivage du réacteur, et donc la disparition à terme de la population bactérienne. Cet état peut se subdiviser en 3 sous-états progressifs : surchage en installation, installée, en récupération. Cet état se distingue généralement des autres par un débit d'alimentation élevé, une production de biogaz moindre et une

accumulation de substrat et de produits intermédiaires. Cette situation, si elle n'est pas traitée à temps, peut basculer en acidogénèse et devenir alors critique.

- Surcharge Organique : Cet état résulte d'une DCO trop importante. Il y a alors accumulation de matière organique et un risque grandissant de crash complet du processus. L'état de surcharge organique peut être divisé en 3 sous-états semblables à ceux de l'état de surcharge hydraulique. Cet état se caractérise par une baisse du taux d'alcalinité et du pH. Tant que les valeurs de ces derniers restent correctes (respectivement, supérieur à 0.5 et autour de 7), la surcharge ne présente pas de risques majeurs et reste rattrapable. Au-delà, elle peut devenir critique et passer en acidogénèse.
- Acidogénèse : Suite à une surcharge, il y a accumulation d'acides gras volatils, ce qui provoque une baisse du pH et donc une acidification du milieu. D'une part une concentration élevée d'AGV va inhiber la population bactérienne qui va peu à peu devenir inefficace, d'autre part l'acidité du milieu va détruire cette même population. Ces deux conséquences font que cet état, particulièrement critique, nécessite une détection très rapide et une réponse immédiate. Cet état demande généralement un temps de récupération très long.
- Toxicité : Etat critique, il est d'autant plus difficile à détecter que ses effets peuvent être variables suivant la source de toxicité. Cet état est généralement dû à une source d'inhibition qui va provoquer une chute de l'activité des bactéries, et donc une accumulation d'acides gras et des effets semblables à ceux d'une surcharge organique. Cependant, la nature du toxique peut entrainer d'autres effets (par exemple, la présence d'azote provoquera une augmentation du pH). Cet état, difficile à détecter autrement que par un expert, est tout aussi critique que l'acidogénèse et peut provoquer des dégâts importants.

6.3 L'importance de l'expert

Au cours de ce stage, quelques réunions ont eu lieu avec Laurent Lardon, actuellement thésard au LBE de Narbonne, et dont la thèse porte sur le système expert flou utilisé pour inférer l'état du bioréacteur dans lequel se déroule le procédé anaérobie. Ces réunions m'ont permis de me rendre compte du rôle important que jouait l'expert dans la résolution d'un problème.

6.3.1 Interprétation des résultats

Dans le cas de procédés biologiques ou agronomiques, il est très important de comprendre le mécanisme qui les guide. Donc, comme dit précédémment, ce sont des domaines où l'interprétation des résultats est tout aussi importante que les résultats eux-mêmes (ce qui n'est pas le cas dans d'autres domaines). Dans ce cadre, l'expert et sa connaissance du procédé jouent un rôle prépondérant. Afin de pouvoir au mieux adapter une méthode à un problème spécifique, déceler ses défauts et les corriger, il faut avant tout comprendre le phénomène étudié. Par exemple, dans le cas du procédé anaérobie, s'il est relativement facile (à condition de posséder quelques connaissances de base en chimie) d'imaginer qu'un état acidogène diminuera le pH et augmentera la quantité d'acide gras volatils, il est beaucoup moins évident que cet état entrainera une forte baisse du ratio d'alcalinité ainsi qu'une diminution de la concentration en CH_4.

L'interprétation que l'expert apporte aux résultats est donc très importante, et permet de déceler les anomalies d'une base de règles induites (exemples contradictoires, mal inférés, valeurs erronées, ...). A l'inverse, les résultats des méthodes d'apprentissage peuvent être d'une grande aide pour les experts. En effet, si ces derniers constatent facilement l'effet d'une variable sur un processus, il leur est souvent difficile d'imaginer l'influence d'interactions sur la sortie. Les méthodes d'apprentissage et les systèmes d'induction peuvent donc être d'une grande aide pour les experts, et ceux-ci, de par leurs connaissances, peuvent facilement corriger ces derniers.

6.3.2 Conception des SEF

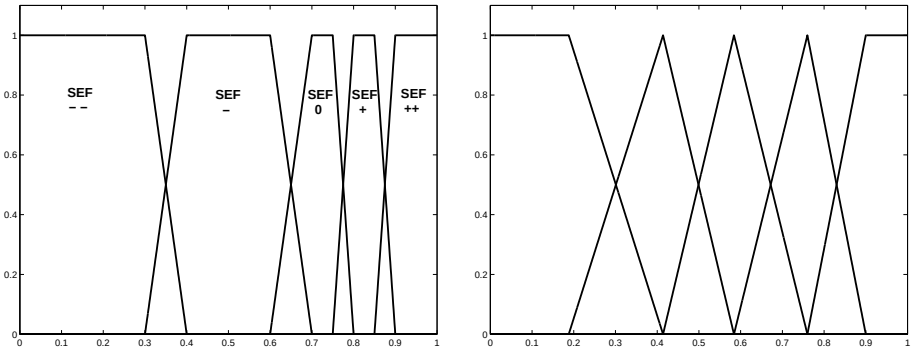
Il n'est plus à prouver que la qualité et la performance d'un système d'inférence flou est en très grande partie lié à la qualité de ses SEF. S'ils sont trop peu nombreux, le système résultant ne donnera au mieux qu'un comportement grossier du système, et fournira des résultats peu précis. A l'inverse, s'ils sont trop nombreux, il y aura risque de surapprentissage, de plus le nombre de règles et de prémisses sera trop élevé pour qu'on puisse en extraire une connaissance quelconque. Enfin, si les SEF sont mal adaptés au problème (par exemple, découper le pH en *normal, peu basique, basique, très basique* alors que le problème demande a ce qu'on différencie des niveaux d'acidité), le système résultant aura des performances médiocres. Il est donc très important que les SEF et leur nombre soient adaptés au problème considéré.

Dans ce cadre, la connaissance que l'expert a du problème représente des informations irremplaçables. D'après l'avis de ce dernier, il sera plus facile de construire des SEF bien ajustés. Si en plus l'expert assimile bien le concept de flou, il pourra construire lui-même les SEF aux moyens de diverses méthodes (k-

means, FCM,...). Une fois ces derniers obtenus, leur utilisation par les méthodes d'apprentissage et les valeurs inférées permettront de les réajuster, tout en conservant dans leur forme les informations transmises par l'expert. Grâce a cette collaboration, à la fois interprétation et performance pourront être améliorées. Dans notre application, cela a permis de mettre en évidence deux problèmes :

- Premièrement, la comparaison entre les valeurs de sortie inférées et les labels assignés par l'expert a montré que l'état normal était très souvent inféré comme une sous-charge ou une sur-charge. Après discussion avec l'expert, nous avons conclu que pour certaines variables d'entrée, le SEF correspondant à la normalité était sans doute trop spécifique et pas assez fuzzifié (par exemple, le troisième SEF de la figure 6.4, portant le label 0).
- Deuxièmement, la mauvaise qualité de certaines partitions construites par Laurent. Notamment au niveau du ratio d'alcalinité, nous nous sommes rendus compte que les partitions induites fournissaient de meilleurs résultats que les partitions expertes. Après une analyse plus poussée, nous avons préconisé un élargissement du SEF de normalité (troisième de la figure 6.4, de label 0) et une fusion des deux SEF supérieurs en un seul ($+$ et $++$), SEF qui correspondent très souvent à une sortie en sous-charge.

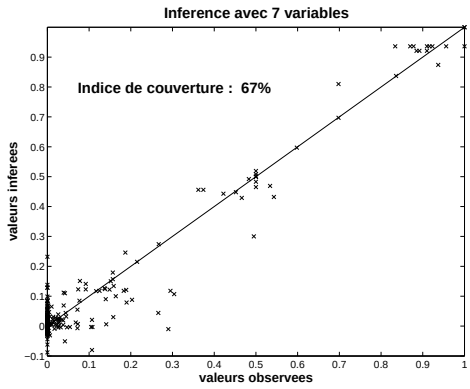
La figure 6.4.a représente l'univers de la même variable que la figure 3.4 de la section 3.4, qui est rappelée par la figure 6.4.b. Ces deux partitions sont issues du ratio d'alcalinité, et même si elles peuvent sembler assez dissemblables, elles sont assez proche du point de vue de l'interprétabilité (la principale impression de dissemblance provenant du fait que l'expert a choisi des SEF trapézoïdaux plutôt que triangulaires).



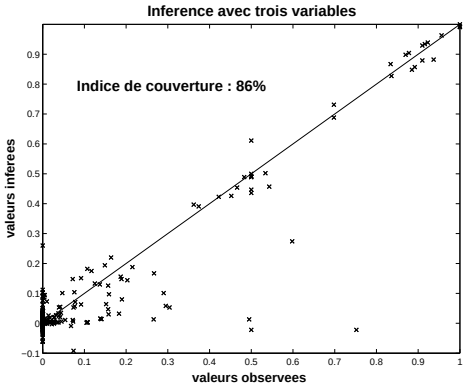
(6.4.a) Partition Experte (6.4.b) SEF de l'OLS modifié

FIG. 6.4 – Exemple de partition experte issue du ratio d'alcalinité et comparaison avec les SEF de l'algorithme OLS modifié

FIG. 6.5 – Performances de systèmes acidogènes avec partitions expertes



(6.5.a) Toutes les variables d'entrée



(6.5.b) 3 variables d'entrée

6.4 Résultats obtenus : cas acidogène

Les résultats qui seront présentés ici se limitent au cas acidogène, que nous avons étudié plus particulièrement. Ce cas a été choisi d'une part parce que, la plupart du temps, l'inférence qui en était faite par le système expert était valable, et d'autre part parce qu'il représente un des deux états critiques de l'application.

Si nous n'avons pas choisi l'état toxique, c'est parce que ce dernier, à cause de sa nature, est très mal inféré par le système expert construit par Laurent Lardon.

Le problème est donc le suivant : à partir des données disponibles, inférer le niveau de possibilité de se trouver dans la sortie " acidogène " (0 indiquant la certitude de ne pas s'y trouver et 1 la certitude de s'y trouver).

Les tests effectués ont principalement été menés sur l'ensemble des variables ainsi que sur un sous-ensemble d'entre elles qui ont été choisies en se conformant aux avis de l'expert. Ce sous-ensemble des entrées est donc formé des variables concernant la concentration en acides gras volatils, le ratio d'alcalinité et le taux de CH_4.

Le jeu de donnée complet consiste en 599 exemples récupérés au cours de 4 mois de tests dans le lit fixe. C'est à partir de ces derniers que nous avons construits les systèmes d'induction flous et les avons validés. Les résultats et les figures présentés par la suite sont des applications d'un même système à l'ensemble des variables.

TAB. 6.4 – Performances de systèmes acidogènes

Nvar	Type part.	Nrègle	PI(10^{-3})	CI(%)
7	expertes	67	2.396	67
3	expertes	32	2.916	86
3	OLS modifié	35	2.809	99

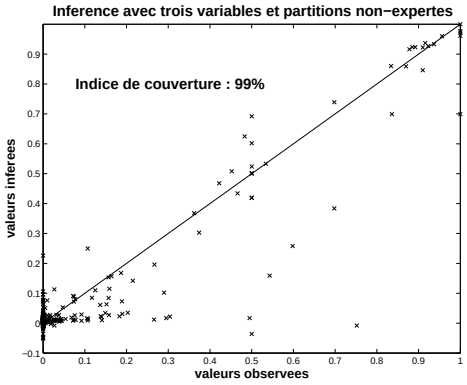
6.4.1 Analyse des résultats obtenus avec les partitions expertes

Lors de l'analyse des résultats, nous nous sommes très vite rendus compte que l'utilisation de l'ensemble des variables amenait à induire des systèmes dont l'indice de couverture était trop faible. Dans le cas de la figure 6.5.a, la couverture n'était que de 67%, ce qui n'est pas satisfaisant. Nous avons donc décidé de réduire l'ensemble des variables afin d'augmenter cette couverture. Après divers essais, un nombre de trois variables nous a semblé à la fois suffisant et nécessaire pour induire des systèmes performants, la plupart des systèmes à deux variables présentant des performances trop dégradées. Nous avons donc choisi trois des variables qui, d'après les explications de Laurent, nous semblaient pouvoir le mieux décrire le cas acidogène.

La figure 6.5.b présente les résultats obtenus par le système induit avec le même ensemble d'apprentissage que pour la figure 6.5.a, mais avec la prise en compte de seulement trois variables (agv, CH_4, ratio d'alca.). La première constatation qui s'impose est l'augmentation significative de l'indice de couverture, qui devient acceptable. De plus, l'ensemble des variables dont la valeur observée est proche de 1 sont activées, ce qui n'est pas le cas lorsque les 7 variables sont prises en compte.

Les quelques exemples très mal inférés dans le cas à 3 variables (autour des valeurs 0.5 et 0.8, par exemple) sont pour la plupart des exemples erronés, mal inférés ou correspondant a des pannes de certains capteurs (dans ces derniers cas, les valeurs de variable données correspondent à des évaluations et non à une mesure), et le recours à l'apprentissage a permis de mettre en évidence ces exemples ou ces défauts du système expert. Le tableau 6.4 comprend les résultats obtenus dans chaque cas (7 variables avec partitions expertes, 3 variables avec partitions expertes et avec partitions créée par l'algorithme OLS modifié). S'il montre une dégradation des performances entre les cas à 7 et 3 variables, nous pouvons accepter ces dernières en regard des gains réalisés en interprétabilité et en indice de couverture. De plus, cette dégradation est aussi dûe en partie aux exemples erronés présents dans le jeu de données.

FIG. 6.6 – Performances d'un système acidogène avec partitions induites



6.4.2 Comparaison avec les résultats obtenus avec partitions induites

Les résultats présentés par la figure 6.6 ont été obtenus avec des partitions construites à partir de l'algorithme présenté en annexe A et sur 3 variables. Comme pour le cas avec les partitions expertes, on retrouve les exemples mal diagnostiqués par le système expert ou erronés. Par contre, l'indice de couverture est lui nettement amélioré, les quelques exemples non couverts ayant été également reconnus comme erronés.

C'est en analysant cette différence entre indice de couverture que nous nous sommes rendus compte que les SEF experts étaient sans doute trop spécifiques, et que certains étaient peut-être superflus. En comparant les résultats des deux cas, il serait facile de conclure que l'utilisation de SEF experts n'est pas très utile (les performances ne sont pas significativement différentes, et l'indice de couverture est meilleur), mais ce serait commettre l'erreur de ne regarder le problème que par un petit bout de la lorgnette. Premièrement, les graphiques montrent que les partitions expertes permettent d'inférer nettement mieux les exemples compris dans la zone sensibles (c'est-à-dire proche de 1), ce qui est le principal effet recherché. Ensuite, il ne faut pas oublier que ce sont les mêmes SEF experts qui sont utilisés pour l'ensemble des autres sorties, il faudrait donc que les SEF construits automatiquement pour le cas acidogène soient aussi bons pour l'ensemble des sorties possibles, ce qui reste à vérifier.

6.5 L’indispensable expertise

Dans le domaine de la supervision de procédé et de la biologie, nous avons montré que remplacer totalement les experts par des systèmes artificiels relevait pour l’instant de l’utopie (il est bien sûr imaginable qu’à terme, la machine puisse “ penser ”, mais ce n’est pas le sujet ici).

Malgré tout, les méthodes d’apprentissage floues, comme nous l’avons également démontré, offrent aux experts des outils précieux qui leur sont d’une grande aide dans leur travail quotidien. D’une part, l’espert possède la connaissance des phénomènes étudié, et d’autre part, les méthodes d’apprentissage floues représentent des méthodes complexes de calculs qui lui permettent d’explorer un nombre impressionant de possiblités (nombre qu’il n’aurait jamais pu atteindre par ses seules capacités). Il serait donc totalement absurde et stupide d’écarter ou l’expert, ou les méthodes d’apprentissage par simple préjugé.

Chapitre 7

Conclusion

Initialement, le but de ce travail était d'implémenter l'algorithme OLS en vue de l'intégrer au logiciel open source FisPro pour ensuite le comparer à d'autres méthodes d'apprentissage floues. Le travail a rapidement été associé à une application pratique, et s'est étendu plus tard à l'amélioration d'un algorithme de simplification de base de règles. Sous certaines conditions, la logique floue présente le grand avantage de fournir des résultats directement interprétables au niveau humain. Cette interprétabilité ne peut se gagner qu'au prix d'une certaine perte de performance, et trouver l'équilibre entre ces deux critères est un problème en soi. L'importance de cette interprétabilité pour certains problèmes a été illustrée par l'application traitée dans ce travail. Chacune des méthodes présentée dans ce travail possède ses avantages et inconvénients, et en combinant adroitement ces avantages, il est possible d'obtenir des résultats qui relèveraient de l'utopie en cas d'utilisation d'une seule de ces méthodes. Nous avons montré que l'interprétabilité des résultats fournis par les méthodes floues est un domaine où beaucoup de choses restent à faire, principalement en ce qui concerne la gestion de règles incomplètes. Bon nombre d'avancées ou d'innovations sont proposées ici, qui ouvrent la voie à une multitude d'autres. Le stage à également débouché sur l'écriture d'un article concernant l'algorithme OLS qui a été soumis aux rencontres francophones sur la logique floue et ses applications. Il est également prévu d'en soumettre une version étendue à la revue IEEE transactions on fuzzy systems.

Bibliographie

[ARMF01] J. Abonyi, J. A. Roubos, M.Oosterom, and F.Szeifert. Compact ts-fuzzy models through clustering and ols plus fis model reduction. In *In Proc. of IEEE international conference on fuzzy systems*, Sydney, Australia, 2001.

[BFOS84] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth International Group, Belmont CA, 1984.

[BKA+02] D.J. Batstone, J. Keller, I. Angelidaki, S.V. Kalyuzhnyi, S.G. Pavlostathis, A. Rozzi, W.T.M. Sanders, H. Siegrist, and V.A. Vavilin. The iwa anaerobic digestion model n°1 (adm1). *Water Science and Technology*, 45 (10) :65–73, 2002.

[BM93] Bernadette Bouchon-Meunier. *La logique floue*. Presses Universitaires de France, 1993.

[BM98] C.L. Blake and C.J. Merz. Uci repository of machine learning databases. http ://www.ics.uci.edu/∼mlearn/MLRepository.html, University of California, Irvine, Dept. of Information and Computer Sciences, 1998.

[DSZ04] W. Duch, R. Setiono, and J. Zurada. Computational intelligence methods for rule-based data understanding. In *Proceedings of the IEEE*, volume 92 (5), pages 771–805, April 2004.

[Dur04] Esra Duru. *Arbres de décision flou : application à la supervision de procédés*. Rapport de DEA, Université Montpellier II, 2004.

[Fio02] A. Fiordaliso. About the trade-off between accuracy and interpretability of takagi-sugeno models in the context of time series forecasting. *Studies in Fuzziness and Soft Computing*, 128 :406–431, 2002.

[Fis] FisPro. http ://www.inra.fr/bia/m/fispro/. Institut National de Recherche Agronomique (INRA) et Cemagref.

[Gac97] Louis Gacogne. *Elements de logique floue*. Editions Hermes, Paris, 1997.

[GC03] Serge Guillaume and Brigitte Charnomordic. *A new method for inducing a set of interpretable fuzzy partitions and fuzzy inference systems from data*, volume 128 of Studies in Fuzziness and Soft Computing, pages 148–175. Springer, 2003.

[Glo99] Pierre-Yves Glorennec. *Algorithmes d'apprentissage pour systèmes d'inférence floue*. Editions Hermès, Paris, 1999.

[Glo02] P.-Y. Glorennec. Constrained optimization of fuzzy decision trees. *Studies in Fuzziness and Soft Computing*, 128 :125–148, 2002.

[GNU] GNU. http ://www.gnu.org. GNU Operating System - Free Software Foundation.

[Gui01a] Serge Guillaume. Designing fuzzy inference systems from data : an interpretability-oriented review. *IEEE Transactions on Fuzzy Systems*, 9 (3) :426–443, June 2001.

[Gui01b] Serge Guillaume. *Induction de règles floues interprétables*. PhD thesis, Insa, Toulouse, 2001.

[HM94] J. Hohensohn and J. M. Mendel. Two pass orthogonal least-squares algorithm to train and reduce fuzzy logic systems. In *Proc. IEE Conf. Fuzzy Syst.*, pages 696–700, Orlando, Florida, June 1994.

[LPS04] Laurent Lardon, Ana Punal, and Jean-Philippe Steyer. On-line diagnosis and uncertainty management using evidence theory – experimental illustration to anaerobic digestion processes. *Journal of Process Control*, 14 (7) :747–763, October 2004.

[Mil56] G. Miller. The magical number seven, plus or minus two. *The Psychological Review*, 63 :81–97, 1956.

[Qui86] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1 :81–106, August 1986.

[Sap90] Gilbert Saporta. *Probabilités, analyse des données et statistiques*. Editions Technip, France, 1990.

[WM92a] Li-Xin Wang and Jerry M. Mendel. Fuzzy basis functions, universal approximation, and orthogonal least squares learning. *IEEE Transactions on Neural Networks*, 3 :807–814, 1992.

[WM92b] Li-Xin Wang and Jerry M. Mendel. Generating fuzzy rules by learning from examples. *IEEE Transactions on Systems, Man and Cybernetics*, 22 (6) :1414–1427, November/December 1992.

Annexe A

Algorithme d'agrégation

L'algorithme présenté ici peut être utilisé dans deux méthodes : comme moyen rapide de générer des SEF ultérieurement utilisés dans l'algorithme OLS (voir section 3.4), et comme un des moyens permettant de générer la partition qui sera le point de départ de la méthode HFP. Dans le premier cas (OLS), le seuil de tolérance devra être assez grand (ordre de 10^{-1}) pour qu'un nombre restreint de SEF soient générés, tandis que dans le second cas (HFP), le seuil devra être assez petit (ordre de 10^{-2}) pour que le partitionnement initial compte un nombre suffisant de SEF.

Soit un ensemble de valeurs x_i^j de la variable j avec $i = 1, \ldots, N$ et N le nombre d'exemples. Soit tol un seuil de tolérance pré-défini.

Alors, les m ensembles de points Cl_f sont construits de la manière suivante :

Ordonner les valeurs x_i^j tel que $x_1^j \leq \cdots \leq x_N^j$

On a alors :

$$\begin{aligned}
Cl_1 = &\quad \{x_i^j | x_i^j - x_1^j \leq tol\} \quad \forall i = 1, \ldots, n_1 \\
Cl_2 = &\quad \{x_i^j | x_i^j - x_{n_1}^j \leq tol\} \quad \forall i = n_1 + 1, \ldots, n_2 \\
&\quad \vdots \\
Cl_m = &\quad \{x_i^j | x_i^j - x_{n_{m-1}}^j \leq tol\} \quad \forall i = n_{m-1} + 1, \ldots, N
\end{aligned}$$

Une fois les m ensembles formés, le $f_{\text{ième}}$ centre c_j^f est alors construit selon l'équation suivante :

$$c_j^f = \frac{\sum\limits_{i \in Cl_f} x_i^j}{n_f}$$

Il est ensuite facile de construire les SEF à partir de ces centres.

Annexe B

Réduction du vocabulaire de sortie

Cette Annexe présente l'algorithme itératif utilisé pour la réduction du vocabulaire de sortie.

Algorithm 1 Algorithme de réduction de vocabulaire

PertePerf = Perte de performance autorisée (en % de la performance initiale)
Data = Valeurs observées des sorties
NbRègles = Nombre de règles du système à simplifier
NEx = Nombre d'exemples conteus dans **Data**
Sauvegarder les conclusions initiales des règles
Pour *i* allant de 1 à **NEx**
    Calculer *i* centres par une méthode k-means sur **Data**
    Ajouter les bornes inférieures et supérieures du domaine de variation de **Data** aux centres trouvés
    Pour *j* allant de 1 à **NbRègles**
        Calculer le centre le plus proche de la conclusion de la règle *j*
        Remplacer la conclusion de cette règle par ce centre
    Fin de la boucle sur les règles
    Calculer la nouvelle performance et la perte de performance
    Si la perte de performance leq **PertePerf**
        Alors Garder le nouveau système et sortir de la boucle
    Sinon
        Réinitialiser les conclusions des règles et relancer la boucle sur *i*
Fin de boucle sur *i*

Il est également possible de spécifier un nombre pré-défini de conclusions distinctes composant le vocabulaire réduit. l'algorithme est alors appliqué sans boucle et sans test sur la dégradation de performance. Dans le cas de la classification, la réduction revient à ramener la conclusion à la valeur de la classe dont elle

est la plus proche (par exemple, une valeur de conclusion de 1.125 serait ramenée à la valeur de classe 1, en supposant que les classes soient définies par des entiers successifs).

Annexe C

Le logiciel FisPro

Fispro est un logiciel open source dont la diffusion est assurée par l'INRA, vous pourrez trouver plus de rensignements à [Fis] ou à fispro@ensam.inra.fr

Toujours en cours de développement, FisPro a connu depuis sa création de nombreux changements. A la base, le logiciel n'était constitué que d'un ensemble de programmes C++ orientés objets destinés à pouvoir créer des SIF et à inférer des sorties à partir de ces derniers. Il s'est depuis enrichi de nombreux modules (méthode HFP, méthode d'optimisation d'un SIF, arbres de décisions flous) ainsi que d'une interface Java qui fournit de nombreux outils de diagnostics, et qui rend l'ensemble du programme assez intuitif pour qu'il soit utilisable très rapidement par tout un chacun.

Le logiciel est en plus localisable, et l'interface est actuellement disponible en trois langues (espagnol, français, anglais). Le logiciel étant en open source, chacun peut, s'il le souhaite, y ajouter sa propre langue. FisPro est également portable, fonctionnant tout aussi bien sous Linux, Unix ou Windows. L'ensemble des fichiers de configuration et de données sont au format texte, donc parfaitement lisibles. Les fichiers d'entrée et de sortie sont quand à eux compatibles avec les tableurs usuels.

La section C.1 présente sommairement le module de base du logiciel, tandis que la section C.2 présente la structure d'un fichier de configuration standard. La figure C.1 illustre la hiérarchie des classes du logiciel.

C.1 Module de base

Les classes de bases sont définies dans les fichiers suivants :

– *fis.h* : Ce fichier comprend les définitions des classes FIS, OUTPUT, INPUT, AGGREG, RULE et MF. La classe MF y est définie de manière abstraite, les classes de SEF particuliers étant définies dans un autre fichier. Les

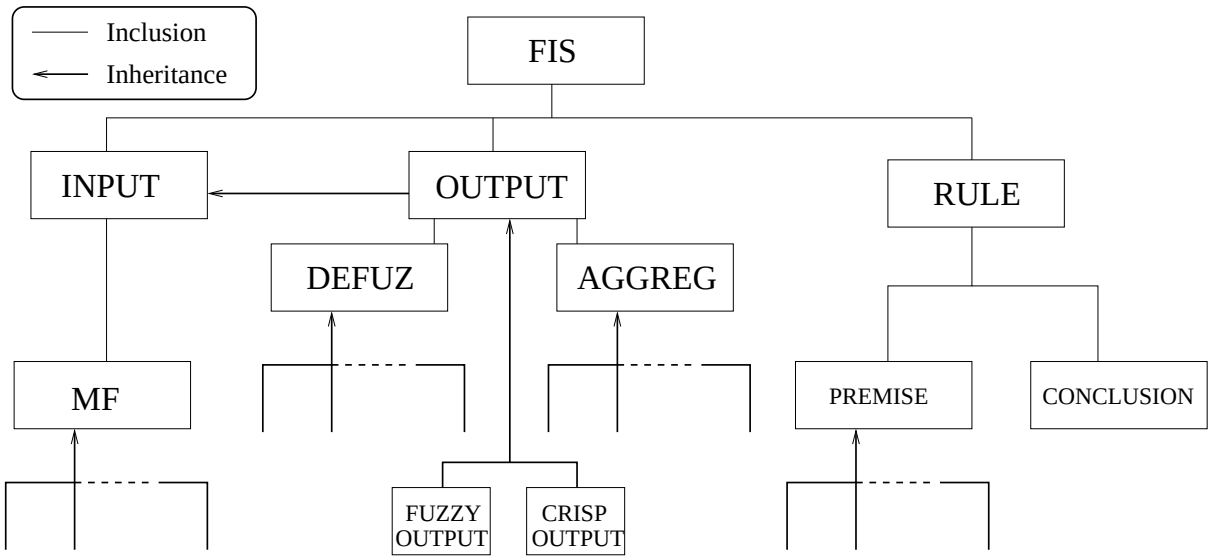


FIG. C.1 – Structure des classes de FisPro

fichiers contenant les programmes sont, eux, séparés (out.cpp, in.cpp, ...)

- *mf.h* : Ce fichier définit l'ensemble des SEF disponibles dans fispro (triangulaires, trapézoïdaux, universels, gaussiens).
- *rule.h* : Ce fichier définit les classes CONCLUSION et PREMISSE, qui permettent de manipuler les règles.
- *common.h* : Ce fichier définit un ensemble de fonctions utiles et communes à de nombreuses classes.
- *defuz.h* : Ce fichier définit l'ensemble des opérateurs de défuzzification possibles suivant le type de sortie et la défuzzification voulue (Sugeno, Mean-Max, ...).

C.2 Structure du fichier de configuration

C.2.1 L'interface

La première section d'un fichier de configuration est délimité par la balise **[Interface]**.

Elle contient le délimiteur entre deux données, qui par défaut est une virgule. Cette section est réservée aux données d'ordre général, comme des délimiteurs de chaîne de caractère.

Exemple d'interface :

[Interface]

DataSep=,

C.2.2 L'entête

La deuxième section, délimitée par **[System]** contient la structure du système.

C'est la que sont définis les dimensions des espaces d'entrée et de sortie, le nombre de règles et d'exceptions, l'opérateur de conjonction (**min** ou **prod**), la méthode de remplacement des valeurs manquantes (**mean** pour une moyenne ou **random** pour une valeur aléatoire) ainsi que le nom donné au système.

Exemple d'entête :

[System]
Name='etats1.choix'
Ninputs=7
Noutputs=1
Nrules=12
Nexceptions=0
Conjunction='min'
MissingValues='mean'

C.2.3 Les entrées

Suite à l'entête, le fichier doit contenir autant de section d'entrée que la valeur de la dimension d'entrée définie dans l'entête. ces sections sont délimitées par la balise **[Input ?]** où ? représente le numéro d'ordre de l'entrée.

Chacune de ces sections comprend l'état de la variable (active ou inactive), son nom, les bornes de son domaine de variation, le nombre de ses SEF ainsi que la définition de chacun d'entre eux (cette définition comprenant leur numéro d'ordre, leur nom, leur type et les valeurs permettant de les construire).

Exemple d'entrée :

[Input1]
Active='yes'
Name='Var1'
Range=[5.594 , 7.287]
NMFs=4
MF1='','SemiTrapezoidalInf',[5.594 , 5.817 , 6.399]
MF2='','triangular',[5.817 , 6.399 , 6.938]
MF3='','triangular',[6.399 , 6.938 , 7.287]
MF4='','SemiTrapezoidalSup',[6.938 , 7.287 , 7.287]

C.2.4 Les sorties

Pareillement aux entrées, le nombre de ces sections doit correspondrent à la dimension de sortie définie dans l'entête. Elles sont très semblables aux entrées et délimitées par **[Output ?]**.

Chaque section sortie comprend la nature de la sortie (flou ou nette), le type de defuzzification et d'agrégation choisi, la valeur par défaut en cas d'incapacité à inférer un exemple, si le problème est de type régression ou classification, et des paramètres semblables à ceux d'une entrée (activité, nom, SEF, ...).

Exemple de sortie :

[Output1]
Nature='crisp'
Defuzzification='sugeno'
Disjunction='sum'
DefaultValue= -1.000
Classif='yes'
Active='yes'
Name='Out1'
Range=[0.000 , 1.000]
NMFs=0

C.2.5 Les règles et les Exceptions

Ces sections, respectivement balisées par **[Rules]** et **[Exceptions]**, définissent l'ensemble des règles du système. Une exception est une règle inactive, qui pourrait être activée ultérieurement.

Elles contiennent l'ensemble des prémisses pour chaque variables d'entrée (0 étant utilisée si la variable est inactive au sein de la règle) ainsi que les conclusions pour chaque sortie. Selon que la sortie est nette ou floue, ces conclusions correspondent à un nombre ou à un SEF de la sortie.

Exemple de règles et d'exceptions :

[Rules]
1, 3, 1, 3, 1, 1, 1, 1.000 ,
3, 4, 4, 3, 1, 1, 4, 0.885 ,
3, 3, 4, 3, 2, 1, 4, 1.000 ,
3, 2, 3, 3, 2, 3, 2, 0.422 ,
1, 3, 1, 4, 1, 1, 2, 0.834 ,

[Exceptions]
1, 3, 1, 3, 1, 1, 2, 0.923 ,

C.3 Autres classes

La plupart des autres classes sont dérivées de la classe FIS, et correspondent aux diverses méthodes implémentées pour FisPro, parmi elles se trouve la classe FISOLS, qui implémente l'algorithme OLS (voir chapitre 3).

Annexe D

OLS : détails sur la programmation

Cette annexe précise quelques détails sur la programmation de l'algorithme OLS (voir chapitre 3 pour la théorie et l'algorithmique).

D.1 Détail des fichiers

- *maincfgols.cpp et mainols.cpp* : les fichiers mainols.cpp et maincfgols.cpp contiennent uniquement un main chargé de vérifier les arguments donnés lors du lancement du programme. maincfgols.cpp correspond au programme permettant de créer le fichier de configuration propre à l'ols, tandis que mainols.cpp correspond au programme qui éxécute l'algorithme.
- *fisols.h* : ce fichier définit l'objet FISOLS et les fonctions qui lui sont rattachées. Le programme en lui-même se trouve dans le fichier d'extension cpp du même nom. Les fonctions décrites dans ce fichier assurent la bonne marche du programme (construction des SEF, standardisation, réduction de vocabulaire, déroulement de l'algo)
- *ols.h* : utilisant la libraire GSL de GNU [GNU], ce fichier définit les fonctions qui réalisent les calculs du premier et du second passage de l'ols. Ces dernières ne sont pas orientées objet et correspondent donc plus à un codage en C qu'en C++. A nouveau, le code complet se trouve dans le fichier cpp homonyme.
- *fis.h* : quelques fonctions ont également été rajoutées à ce fichier, afin que la réduction de vocabulaire puisse être applicable à toutes les méthodes, et pas uniquement à l'algorithme OLS.

D.2 Structure des éxecutables

Chacun des éxecutables (celui de maincfgols.cpp et de mainols.cpp) doivent obligatoirement être lancés avec certains arguments, et peuvent en plus prendre d'autres arguments en option.

Création du fichier de configuration

- *Nom du fichier de donnée (obligatoire)* : cet argument permettra au fichier de connaitre les dimensions d'entrée et de sortie du problème
- *Nom d'un fichier de configuration de SIF classique (optionnel)* : si cet arguement est utilisé, le fichier de configuration intégrera les SEF pré-définis dans le fichier dont le nom a été spécifié
- *Nombre de sorties (optionnel)* : spécifie le nombre de sorties existantes (par défaut, le programme considère qu'il n'y a qu'une sortie), le nombre d'entrées est ensuite inféré à partir de ce nombre et du nombre de colonnes du fichier de données (Nombre d'entrées = Nombre de colonnes - Nombre de sorties).

Lancement de l'algorithme

- *Nom du fichier de données (obligatoire)* : ce fichier servira de fichier d'apprentissage à l'algorithme
- *Nom du fichier de configuration (obligatoire)* : nom du fichier de configuration spécifique à l'OLS et au problème traité qui permettra de paramétrer la méthode
- *Nom du fichier de données pour le second passage (optionnel)* : spécifie le nom du fichier de données qui sera utilisé pour le second passage (par défaut, ce sera le fichier d'apprentissage)
- *Nom du fichier de données pour la réduction de vocabulaire (optionnel)* : spécifie le nom du fichier de données qui sera utilisé pour la réduction de vocabulaire (par défaut, ce sera le fichier d'apprentissage)
- *Numéro de sortie (optionnel)* : spécifie la sortie à traiter (par défaut, ce sera la première)

D.3 Structure du fichier de configuration

Le fichier de configuration de l'OLS a une structure très similaire à celui d'un fichier de configuration classique d'un SIF. L'ensemble des balises sont similaires, à ceci près que le fichier de configuration de l'OLS ne contient pas de balises pour les règles et les exceptions (vu que son rôle est justement de les induire). Les entrées et les sorties sont définies de la même manière pour les deux méthodes,

ainsi que l'interface.

La principale différence se situe donc au niveau de l'entête.

D.3.1 Structure de l'entête

Voici un exemple commenté d'entête utilisée dans un fichier de configuration de l'OLS :

[Header] : balise de l'entête
NbIn=7 : dimension de l'espace des entrées
NbOut=1 : dimension de l'espace des sorties
ToleranceThresh=0.25 : Tolérance utilisée pour la construction automatique de SEF
ErrorThresh=0.01 : critère d'arrêt sur la part de variance non-expliquée
RuleErrorThresh=0 : critère d'arrêt sur la part individuelle de variance expliquée par la dernière règle sélectionnée
RuleNumberThresh=1000 : critère d'arrêt sur le nombre de règles sélectionnées
Standardization=1 : drapeau pour la transformation en partitions fortes (0=non, 1= oui)
SecondPass=1 : drapeau pour le second passage (0=non, 1=oui)
VocabularyReduction=1 : drapeau pour la réduction de vocabulaire (0=non, 1=oui)
Conjunction='min' : précise le type de l'opérateur de conjonction (prod ou min)
PerfLossAllowed=0.1 : perte de performance autorisée pour la réduction de vocabulaire (% de la performance initiale)
MuThresh=0.1 : seuil d'activation minimal pour qu'un exemple soit considéré comme actif
NVocItem=0 : cardinal pré-défini du vocabulaire réduit (minimum 2, 0=boucle incrémentale)

L'ensemble de ces paramètres n'est pas toujours nécessaire à la bonne marche du programme, mais ils assurent que ce dernier fonctionnera dans tous les cas. Par exemple, la définition d'une tolérance n'est pas utile si les SEF sont pré-définis, et le test sur la perte de performance n'aura pas lieu si le nombre de mots distincts composant le vocabulaire réduit est défini à l'avance.

En ce qui concerne les critères d'arrêt, le premier qui est satisfait provoque un arrêt de l'algorithme. les valeurs par défaut sont les valeurs de l'exemple et correspondent géénralement à des valeurs qui offrent de bonnes performances.

D.4 Fichiers de sortie du programme

Le programme fournit en sortie plusieurs fichiers, dont le nombre est variable selon sa configuration. Selon cette configuration donc, on pourra trouver en sortie :

- Un fichier de configuration classique d'un SIF contenant le résultat du premier passage de l'OLS.
- Un fichier contenant l'ensemble des règles sélectionées, l'exemple qui leur correspond, la part d'erreur que chacune explique et l'erreur expliquée cumulée.
- Un fichier de configuration classique d'un SIF contenant le résultat du second passage de l'OLS si il y a eu un second passage.
- Un fichier de configuration classique d'un SIF contenant le résultat de la réduction de vocabulaire si une réudction a été réalisée.

Annexe E

Algorithmes de la méthode de simplification

Cette Annexe présente en détail les algorithmes évoqués dans la section 5.2.2. Il s'agit en fait d'une traduction simplifiée des algorithmes se trouvant dans [GC03].

E.1 Algorithme principal

L'algorithme 2 est celui de la boucle principale. Il incrémente petit à petit l'amplitude de la dégradation autorisée (StepLoss) jusqu'à atteindre la dégradation maximale autorisée ($Loss_{thres}$). La variable $MergingPriority$ permet de faire un premier passage où ne sont autorisées que les fusions de règles, elle sera ensuite modifiée à la fin de ce premier passage pour que soient également autorisées les suppressions de règles et de variables.

L'algorithme se base sur trois étapes dont les détails seront donnés dans la suite. Afin de contrôler que les simplifications ne dégradent pas trop le système, plusieurs paramètres doivent être définis :

- $CurErr$: la performance initiale du système
- $Loss_{thres}$: la perte de performance autorisée, exprimée en %
- CI_{thres} : l'indice de couverture minimal autorisé
- H_{thres} : le seuil d'hétérogénéité (eq. 5.2) au-delà duquel une nouvelle règle provenant d'une simplification n'est pas acceptée

E.2 Algorithme de fusions de groupes

L'algorithme 3 est celui de la fusion d'un groupe de règle pour former une règle générique. Outre les paramètres défini plus haut ($Loss_{thresh}$ et H_{thresh}), les

Algorithm 2 Simplification de la base de règle : algorithme général

CurErr = InitErr (performance initiale)
MaxErr = CurErr (1 + $Loss_{thres}$)
StepLoss = StepLossInit
$MergingPriority = vrai$
while $CurErr < MaxErr$ **do**
  si $StepLoss > 1.0$
    si $MergingPriority$ **alors**
      $MergingPriority = faux$
      $StepLoss = StepLossInit$
    sinon sortir
  fin de si
  APPELER *Merging* avec $StepLoss$ en paramètre (Algorithme 3)
  si $Succes$ ou not($MergingPriority$) **alors**
    APPELER *Rule removal* avec $StepLoss$ en paramètre (Algorithme 4)
    APPELER *Variable removal* avec $StepLoss$ en paramètre (Algorithme 5)
  fin de si
  si not($Succes$) **alors** augmenter $StepLoss$
end while

variables PI, NewPI et PILoss correspondent respectivement à la performance initiale, la performance recalculée et la perte de performance induite par la fusion (en %).

La fusion de règles représentent sans aucun doute la partie la plus importante de la méthode, les autres simplifications jouant un rôle mineur.

Algorithm 3 Algorithme de fusion d'un groupe pour créer une règle générique

```
1: Compute PI ; DistanceValue = 1
2: loop
3:    Construire tous les groupes de règles avec une distance de règles = DistanceValue
4:    k = nombre de groueps construits
5:    si k = 0
6:          si DistanceValue = 1 alors DistanceValue = 0
7:          sinon sortir
8:    fin de si
9:    for 1 ≤ g ≤ k do
10:      Construire la règle générique du groupe
11:      si H_g < H_thresh alors la fusion est valide
12:      Reconstruire la base de règle initiale
13:   end for
14:   si DistanceValue = 1 et qu'il n'y a aucune fusion valide alors DistanceValue = 0
15:   sinon sortir
16:   Faire toutes les fusions valides
17:   Sauvegarder le nouveau système, calculer NewPI et PILoss
18:   si PILoss < Loss_thresh alors DistanceValue=1
19:   sinon sortir
20: end loop
```

E.3 Algorithmes de suppression de règles et de variables

Les algorithmes 4 et 5 concernent la suppression de règles et de variables. Ils sont beaucoup plus simples que les deux précédents et reprennent les indices déjà définis.

Algorithm 4 Algorithme de suppression de règles

1: Calculer PI
2: **for all** $r \in RB$ (Rule Base) **do**
3: supprimer la règle r
4: sauvegarder le système correspondant
5: calculer NewPI et PILoss
6: **si** $PILoss > Loss_{thres}$ OU $CI < CI_{thres}$ **alors** Restaurer le système initial
7: **end for**

Algorithm 5 Algorithme de suppression de variables

1: calculer PI
2: **for all** $r \in RB$ **do**
3: **for all** $v \in r$ **do**
4: Mettre son label à ANY (la rendre inactive)
5: Sauvegarder le système
6: Calculer H^r, NewPI et PILoss
7: **SI** $H^r > H_{thresh}$ OU $PILoss > Loss_{thresh}$ OU $CI < CI_{thresh}$
8: **alors** restaurer le système initial
9: **end for**
10: **end for**
11: **si** deux règles ont des prémisses identiques **alors** en supprimer une