

Document de travail

Laura Nguyen

4 septembre 2018

1 Introduction

Dans beaucoup de problèmes de classification, les valeurs des attributs et de la classe sont ordinaux. De plus, il peut exister une contrainte de monotonie : la classe d'un objet doit croître/décroître en fonction de la valeur de tout ou partie de ses attributs. A savoir, étant donné deux objets x, x' , si $x \leq x'$ alors $f(x) \leq f(x')$. Les variables dépendantes, $f(x)$ et $f(x')$, sont des fonctions monotones des variables indépendantes, x et x' . On parle alors de problèmes de classification monotone, ou problèmes de classification avec contrainte de monotonie. Cette contrainte indique que les objets ayant de meilleures valeurs d'attributs ne doivent pas être assignés à de moins bonnes valeurs de classe.

L'ajout de cette contrainte de monotonie permet d'introduire des concepts sémantiques tels la préférence, la priorité, l'importance, qui nécessitent une relation d'ordre.

Il existe de nombreux domaines se prêtant à ce type de tâches, tels la prédiction du risque de faillite [6], l'analyse de la satisfaction des clients [7], le diagnostic médical [10]. L'importance de la prise en compte d'une relation graduelle entre les valeurs d'attributs et la classe a été démontrée [12] : les classifieurs auxquels sont imposés la contrainte de monotonie sont au moins aussi performants que leurs homologues classiques, et les experts sont plus enclins à utiliser les règles générées par les modèles monotones.

Afin d'extraire des règles à partir de données monotones, on décide d'utiliser les arbres de décision, dont l'efficacité et l'interprétabilité en classification a été prouvée [15]. Cependant, les algorithmes de construction d'arbres de décision standards (générés par CART [9]) ne produisent pas de classifieurs sensibles à la monotonie, même si la base utilisée est complètement monotone. En revanche, il est montré dans [3] que les classifieurs purement monotones ([2], [1], [5]) sont, en terme de taux de bonne classification, statistiquement indiscernables de leurs homologues non-monotones. Dans le même article, il est expliqué que ce phénomène est dû à la sensibilité de ces classifieurs au bruit non-monotone présent dans les données réelles.

Ce stage a pour but d'étudier la construction et l'évaluation d'arbres de décision prenant en compte une relation graduelle susceptible d'exister entre les valeurs d'attributs et la classe, tout en étant suffisamment robuste au bruit non-monotone. On reprend, en particulier, [11] pour la construction d'arbres de décision monotones paramétrés par une mesure de discrimination d'ordre. Une étude théorique des propriétés des mesures présentées dans le même article est également effectuée.

2 Etat de l'art

Dans cette partie, on étudie et compare les méthodes proposées par Hu et al. [8], Marsala et Petturiti [11], Qian et al. [14] et Pei et Hu [13].

2.1 Notations

On considère un ensemble $\Omega = \{\omega_1, \dots, \omega_n\}$ d'éléments définis par un ensemble d'attributs $A = \{a_1, \dots, a_m\}$, où pour tout $j = 1, \dots, m$, a_j est une fonction de Ω vers $X_j = \{x_{j_1}, \dots, x_{j_t}\}$. On note aussi $\lambda : \Omega \rightarrow C$ la fonction d'étiquetage, où $C = \{c_1, \dots, c_k\}$ est un ensemble de classes totalement ordonné.

Pour $\omega_i \in \Omega$, l'ensemble dominant de ω_i généré par a_j est défini de la façon suivante :

$$[\omega_i]_{a_j}^{\leq} = \{w_h \in \Omega : a_j(\omega_i) \leq a_j(\omega_h)\}$$

De même, l'ensemble dominant de ω_i généré par λ s'écrit :

$$[\omega_i]_{\lambda}^{\leq} = \{w_h \in \Omega : \lambda(\omega_i) \leq \lambda(\omega_h)\}$$

Les notions de *dominance rough sets* sont également utilisées. On cherche à approximer l'ensemble c_q^{\geq} , ie l'ensemble des éléments de Ω dont la valeur de classe est inférieure à c_q .

Soit $B \subseteq A$ et c_q une valeur de classe. Les approximations inférieure et supérieure de c_q^{\geq} sont définis de la façon suivante :

$$\underline{R}_B^{\geq}(c_q^{\geq}) = \{\omega \in \Omega : [\omega]_B^{\geq} \subseteq c_q^{\geq}\}$$

$$\overline{R}_B^{\geq}(c_q^{\geq}) = \{\omega \in \Omega : [\omega]_B^{\leq} \cap c_q^{\geq} \neq \emptyset\}$$

2.2 Arbre de décision basé sur l'entropie d'ordre

Il est montré dans [3] que les classifieurs purement monotones ne procurent pas de meilleurs résultats que leurs homologues classiques lorsque les données contiennent un taux élevé de bruit non-monotone.

Afin de résoudre ce problème, les auteurs de [8] proposent un algorithme robuste pour la classification monotone.

Pour cela, ils définissent une mesure de degré de monotonie entre deux attributs, qu'ils nomment information mutuelle d'ordre, ou *rank mutual information* (RMI) :

$$RMI^{\leq}(a_j, a_{j'}) = -\frac{1}{n} \sum_{i=1}^n \log \frac{|[\omega_i]_{a_j}^{\leq}| \times |[\omega_i]_{a_{j'}}^{\leq}|}{n \times |[\omega_i]_{a_j}^{\leq}| \cap |[\omega_i]_{a_{j'}}^{\leq}|}$$

RMI, sensible à la monotonie et robuste face aux données bruitées, est utilisé comme mesure de sélection d'attribut dans leur algorithme de construction d'arbres de décision monotones, REMT (*Rank Entropy Based Decision Tree*).

Dans cet article, on considère seulement les arbres binaires dans lesquels est affecté à chaque noeud un seul attribut. Concernant les données, les attributs doivent être numériques et la classe, ordinale.

À chaque étape de l'induction :

- si $|\Omega_\alpha| = 1$ ou tous les éléments de Ω_α partagent la même classe, une feuille est créée.
- Sinon, pour chaque attribut a_j , pour chaque valeur d'attribut x_{j_s} , l'ensemble des éléments est divisé en deux sous-ensembles à partir desquels on calcule $RMI_{x_{j_s}} = RMI(a_j^{x_{j_s}}, \lambda)$. On récupère, pour chaque a_j , le seuil de coupure x_j^* maximisant l'information mutuelle d'ordre : $x_{j_s}^* = \arg \max RMI_{x_{j_s}}$. L'attribut a^* utilisé pour le partitionnement est celui dont le seuil x^* maximise $RMI(a_j^{x_{j_s}^*}, \lambda)$, pour $j = 1, \dots, m, s = 1, \dots, t_j - 1$.
- Si $RMI(a^*, \lambda) < \epsilon$, une feuille est créée.
- Sinon, un noeud est construit et la procédure est répétée sur les sous-ensembles induits par a^* et x^* .

Concernant le critère d'étiquetage, si tous les exemples de la feuille possèdent la même classe, on lui assigne cette dernière. Sinon, si les exemples proviennent de classes différentes, on assigne la classe médiane. Dans le cas où deux classes possèdent le même nombre d'exemples et la feuille courante provient de la branche gauche de son noeud parent, on lui assigne la pire classe. Sinon, on lui affecte la meilleure.

L'approche de construction gloutonne du classifieur ne permet pas d'obtenir un arbre globalement monotone, même si les données utilisées sont monotones. En revanche, cette méthode garantit une monotonie plus faible : il est montré que si la base de données utilisée lors de la construction

de l'arbre est monotone-consistante, alors les règles générées par REMT sont monotones.

Afin d'évaluer la performance de leur algorithme, Hu et al. utilisent l'erreur absolue moyenne, ou *Mean Absolute Error* (MAE) :

$$MAE = \frac{1}{n} \sum_{i=1}^n |f(\omega_i) - \lambda(\omega_i)|$$

où $f(\omega_i)$ est la classe prédite par REMT.

L'algorithme est testé sur des bases de données artificielles et réelles.

Les bases de données artificielles générées sont monotones. Elles contiennent 1000 exemples à deux attributs, et un nombre de classes variant de 2 à 30.

REMT est ici comparé à CART, Rank Tree [16], OLM [2] et OSDL [4]. Les expériences montrent que REMT produit le moins d'erreurs parmi ces algorithmes, hormis dans le cas à deux classes. Les auteurs soulignent aussi que REMT est le plus précis des quatre algorithmes, peu importe le nombre d'éléments utilisés lors de la construction de l'arbre.

Sur des bases de données réelles, Hu et al. comparent leur modèle à CART et Rank Tree [16]. Les expériences menées montrent que REMT donne de meilleurs résultats en termes de MAE sur 10 bases sur les 12 collectées. De plus, les auteurs montrent également que plus la taille de la base d'apprentissage diminue, plus l'écart de performance se creuse entre REMT et les autres modèles. Enfin, dans le but de comparer les performances des algorithmes lorsque les données sont monotones, les mêmes bases sont modifiées à l'aide d'un algorithme de monotonisation. Les résultats des expérimentations montrent que REMT est plus performant que les autres modèles. De plus, les valeurs de MAE produites par tous les modèles diminuent lorsque les données sont monotonisées.

2.3 Arbre de décision paramétré par une mesure de discrimination d'ordre

Dans [11], Marsala et Petturiti étudient l'influence de la mesure de discrimination utilisée lors de la construction du classifieur, en termes de taux de bonne prédiction et de monotonie.

Comme les mesures étudiées dans l'article partagent toutes la même structure fonctionnelle, les auteurs définissent un modèle de construction hiérarchique permettant d'isoler les propriétés d'une mesure de discrimination d'ordre et d'en créer de nouvelles.

Soit H^* une mesure de discrimination d'ordre. Le pouvoir de discrimination de l'attribut a_j envers λ selon H^* peut donc s'écrire de la façon suivante :

$$H^*(\lambda|a_j) = h^*(g^*(f^*(\omega_1)), \dots, g^*(f^*(\omega_n)))$$

Marsala et Petturiti proposent un algorithme de construction d'arbre de décision nommé RDMT (*Rank Discrimination Measure Tree*) essentiellement basé sur REMT mais qui utilise, à la place de RMI, une mesure de discrimination passée en paramètre.

Comme précédemment, seuls les arbres binaires à un attribut par noeud sont considérés. Ce classifieur gère à la fois les attributs numériques et ordinaux, mais la classe doit être ordinale.

Notons H la mesure de discrimination utilisée, ϵ le seuil minimal pour H , Ω_α l'ensemble des exemples considérés à l'étape courante de construction, n_{min} la taille minimale pour Ω_α , δ la profondeur courante de la branche, et Δ la profondeur maximale que peut avoir une branche de l'arbre.

A chaque étape de l'induction :

- si $|\Omega_\alpha| < n_{min}$ ou tous les éléments de Ω_α partagent la même classe ou $\delta > \Delta$, une feuille est créée.
- Sinon, pour chaque attribut a_j , pour chaque valeur d'attribut x_{js} , l'ensemble des éléments est divisé en deux sous-ensembles à partir desquels on calcule $H(\lambda|a_j^{x_{js}})$. On récupère, pour chaque a_j , le seuil de coupure x_j^* minimisant la valeur de H : $x_j^* = \arg \min H(\lambda|a_j^{x_{js}})$. L'attribut a^* utilisé pour le partitionnement est celui dont le seuil x^* minimise $H(\lambda|a_j^{x_{js}})$, pour $j = 1, \dots, m, s = 1, \dots, t_j - 1$.
- Si $H(\lambda|a^*) < \epsilon$, une feuille est créée.

- Sinon, un noeud est construit et la procédure est répétée sur les sous-ensembles induits par a^* et x^* .

Comme REMT, RDMT ne garantit pas un classifieur globalement monotone, mais un arbre générant des règles monotones.

Les auteurs évaluent les modèles construits selon trois critères : le taux de bonne classification (mesuré par le pourcentage d'exemples correctement étiquetés, le test du K, et MAE), la monotonie et la taille des arbres (en termes de nombre de feuilles). La monotonie d'un arbre est évaluée par deux mesures :

- L'index I de non-monotonie d'un arbre \mathcal{T} :

$$I(\mathcal{T}) = \frac{\sum_{u=1}^q \sum_{v=1}^q m_{u,v}}{q^2 - q}$$

où q est le nombre de feuilles de Γ et $m_{u,v}$ vaut 1 si les feuilles l_u, l_v ne sont pas monotones, 0 sinon.

- L'index NMI1 des exemples dans chaque base de test \mathcal{D} :

$$NMI1(\mathcal{D}) = \frac{\sum_{i=1}^n \sum_{h=1}^n NMP(\omega_i, \omega_h)}{n^2 - n}$$

où $NMP(\omega_i, \omega_h)$ vaut 1 si la paire ω_i, ω_h n'est pas monotone, 0 sinon.

Les expérimentations menées dans l'article visent à comparer les arbres obtenus par différentes mesures ($H_S^*, H_S, H_G^*, H_G, H_P^*, H_{MID}^{10}, H_{ICT}$).

11 bases artificielles sont générées, dont le taux de NMI augmente de 0 à 10% avec un pas de 1%. Chaque base contient 500 exemples à 5 attributs et 5 classes.

Les tests menés sur ces bases montrent qu'il n'existe aucune différence significative entre les mesures en ce qui concerne les indices de taux de bonne classification. De plus, les mesures de discrimination d'ordre produisent des arbres plus monotones, en particulier lorsque le taux de bruit non-monotone de la base augmente. Enfin, il est également montré que ces mesures de discrimination produisent des arbres contenant plus de feuilles que les mesures classiques.

15 bases de données réelles sont sélectionnées pour évaluer les mesures sur des cas d'applications concrets. Comme précédemment, il n'y a pas de différence de taux de bonne classification significative entre les mesures. Cependant, H_S^*, H_G^* , et H_P^* génèrent des arbres plus monotones et comportant davantage de feuilles.

Les arbres produits par des mesures de discrimination d'ordre ne sont pas destinés à être directement utilisés pour la classification monotone. En revanche, étant donné leur degré de monotonie plus élevé, il est avantageux de les passer en entrée d'algorithmes de post-traitement afin renforcer la monotonie globale.

2.4 Fusion d'arbres de décision monotones

Afin d'améliorer la capacité de généralisation du modèle de classification, Qian et al. dans [14] proposent une méthode d'ensemble par fusion d'arbres de décision monotones, notée FREMT (*Fusing rank entropy based monotonic decision trees*).

Les auteurs couplent une méthode de réduction d'attributs préservant l'ordre avec REMT, utilisé pour générer les classifieurs de base.

Pour réduire l'ensemble des attributs, ils se basent sur les *dominant rough sets* et mettent à jour leur définition. Ils introduisent un paramètre β qui permet de paramétrer la sensibilité du *rough set* au bruit non-monotone. L'ensemble à approximer est :

$$c_q^{\geq} = \bigcup_{u \leq q} C_u$$

où $C_u \in \Omega/\lambda = \{C_1, C_2, \dots, C_r\}$ tel que, pour tout $q, u \leq r$, si $q \geq u$ alors les éléments de C_q sont préférés à ceux de C_u .

Soit $B \subseteq A$ et c_q une valeur de classe. Les approximations inférieure et supérieure de c_q^{\geq} sont définis de la façon suivante :

$$\begin{aligned} \underline{R}_B^{\geq}(c_q^{\geq}) &= \{\omega \in \Omega : \frac{|[\omega]_B^{\geq} \cap c_q^{\geq}|}{|[\omega]_B^{\geq}|} \geq 1 - \beta\} \\ \overline{R}_B^{\geq}(c_q^{\geq}) &= \{\omega \in \Omega : \frac{|[\omega]_B^{\geq} \cap c_q^{\geq}|}{|[\omega]_B^{\geq}|} \geq \beta\} \end{aligned}$$

où $0 \leq \beta \leq 0.5$

La frontière ascendante de c_q liée à l'ensemble d'attributs B s'écrit :

$$BND_B^{\beta}(c_q^{\geq}) = \overline{R}_B^{\geq}(c_q^{\geq}) - \underline{R}_B^{\geq}(c_q^{\geq})$$

La dépendance monotone de λ selon B est donc exprimée de la façon suivante :

$$\gamma_B^{\beta}(\lambda) = \frac{|\Omega - \bigcup_{q=1}^k BND_B^{\beta}(c_q^{\geq})|}{|\Omega|}$$

A partir de cette définition, il est possible d'établir un coefficient de pertinence d'un attribut a dans B , relativement à λ . La pertinence interne de a dans B relativement à λ est définie de la manière suivante :

$$Sig_{inner}^{\beta}(a, B, \lambda) = \gamma_B^{\beta}(\lambda) - \gamma_{B-\{a\}}^{\beta}(\lambda)$$

De la même façon, $\forall a \in A - B$, la pertinence externe de a selon B correspond à :

$$Sig_{outer}^{\beta}(a, B, \lambda) = \gamma_{B \cup \{a\}}^{\beta}(\lambda) - \gamma_B^{\beta}(\lambda)$$

À l'aide de ces mesures de pertinence d'attributs, Qian et al. proposent un algorithme de réduction d'attributs, paramétré par β , qui, étant donné Ω , A et λ , génère une réduction (*reduct*) d'attributs monotones. Le principe est le suivant :

- On récupère dans B les attributs noyaux de B en utilisant Sig_{inner}^{β} .
- Pour tout attribut $a_j \in A - B$, on calcule $Sig_{outer}^{\beta}(a_j, B, \lambda)$.
 - Si la valeur calculée est supérieure ou égale à toutes celles obtenues précédemment, alors on rajoute a_j dans B .
 - on réitère le processus jusqu'à ce que, pour tout $a_i \in A - B$, $Sig_{outer}^{\beta}(a_i, B, \lambda) = 0$ (i.e. rajouter a_i à B ne modifie pas $\gamma_B^{\beta}(\lambda)$).

Comme la taille de l'ensemble des attributs monotones sélectionnés par cette méthode est bien moindre que celle de l'ensemble original, l'arbre de décision induit par ces attributs sera moins profond et contiendra moins de noeuds, ce qui implique une meilleure aptitude à généraliser. Chaque arbre produit par cet algorithme sert de classifieur de base pour la méthode d'ensembles.

Afin d'obtenir un classifieur final performant, il est nécessaire de produire des arbres les plus diversifiés possibles. Plusieurs sous-ensembles d'attributs sont donc sélectionnés, et chacun de ces sous-ensembles est une réduction d'attributs qui préserve la monotonie des données.

Les auteurs définissent donc une méthode de recherche de multiples réductions d'attributs monotones, paramétrée par β . Étant donné Ω , A et λ , elle consiste à :

- Trouver les attributs noyaux de A en utilisant Sig_{inner}^{β} .
- Trier par ordre croissant les attributs restants $a_j \in B$ en fonction de $\gamma_B^{\beta} + \frac{|\Omega/\{a_j\}|}{|\Omega|}$.
- Stocker dans P^* les deux sous-ensembles précédents.
- À partir de P^* , trouver une réduction d'attributs RED_0 avec l'algorithme de réduction d'attributs donné plus haut.
- Pour chacun des attributs a_j se trouvant dans RED_0 mais n'étant pas des attributs noyaux,
 - Enlever a_j de P^* .
 - Calculer une réduction RED_j à partir de P^* grâce à l'algorithme précédent.

- Si RED_j n'a pas déjà été trouvée précédemment, ajouter RED_j à l'ensemble des réductions obtenues.
- Remettre a_j dans P^* .

Cette méthode permet de générer un ensemble de réductions d'attributs monotones aussi diversifié que possible. Ces sous-ensembles d'attributs servent de base pour construire des arbres de décision complémentaires, générés par REMT.

Soit $F = \{T_1, T_2, \dots, T_N\}$ une forêt d'arbres de décision générés par $RED = \{RED_1, RED_2, \dots, RED_N\}$. Étant donnés Ω , A , λ , $\beta = \{\beta_1, \beta_2, \dots\}$, et ω un objet dont la classe est à déterminer, l'algorithme final de fusion d'arbres de décision monotones, *Fusing rank entropy based monotonic decision trees (FREMT)*, est le suivant :

- Pour $i = 1, \dots, M$, trouver $RED_i = \{RED_0, RED_1, \dots, RED_{N_i}\}$ avec l'algorithme précédent
- Stocker dans RED l'union de tous les ensembles de réductions RED_i générés
- Pour chaque $RED_j \in RED$, générer un arbre T_j avec REMT.
- Construire F la forêt d'arbres $\{T_1, T_2, \dots, T_N\}$.
- Déterminer la classe de ω avec F selon le principe de fusion basé sur la probabilité maximale (*Fusing principle based on maximal probability*).

Qian et al. évaluent la performance de leur modèle selon le taux de bonne classification (mesuré par le pourcentage d'instances correctement classifiées) et l'erreur absolue moyenne.

Les tests sont menés uniquement sur des bases réelles.

Sur les 10 bases récupérées, 9 nécessitent un pré-traitement afin d'être adaptés à l'algorithme de fusion d'arbres (en transformant les problèmes de monotonie descendantes en problèmes de monotonie ascendantes). De plus, afin de comparer les performances des modèles lorsque les données sont monotones, les classes des données sont modifiées à l'aide d'un algorithme de monotonization. Les auteurs observent d'abord les performances des arbres constituant la forêt (construits à partir des réductions d'attributs des données d'origine) puis la performance du modèle final. Ils comparent leurs résultats à ceux obtenus par REMT.

D'après les expériences menées, on observe que les arbres de décision générés par les réductions d'attributs offrent de meilleurs résultats en termes de taux de bonne classification et d'erreur absolue moyenne dans la plupart des cas. De plus, les valeurs générées sont différentes les unes des autres, ce qui satisfait la contrainte de diversité nécessaire au modèle d'ensembles. Enfin, le classifieur produit par FREMT s'avère être meilleur que celui généré par REMT sur toutes les bases considérées, en termes de taux de bonne classification et d'erreur absolue moyenne. FREMT permet aussi d'améliorer la capacité de généralisation de REMT sur ces tâches de classification.

2.5 Arbres de décision partiellement monotones

Dans cette partie, on distingue deux types d'attributs : ceux dont les valeurs sont linéairement ordonnées selon les valeurs de la classe, qu'on appelle critères, et ceux n'ayant pas de relation monotone avec la classe. La plupart des tâches de classification monotone comprennent ces deux sortes d'attributs. Cependant, les algorithmes étudiés précédemment font l'hypothèse que tous les attributs sont monotones par rapport à la classe. On ne peut néanmoins pas conserver uniquement les critères car les autres attributs permettent d'améliorer la performance du classifieur. Dans [13], Pei et Hu proposent donc un algorithme de construction d'arbres partiellement monotones permettant de gérer les deux types d'attributs à la fois.

Pour cela, ils proposent une mesure d'inconsistance d'ordre, *rank inconsistency rate (RIR)*, qui permet de distinguer les attributs ordinaires des critères et de déterminer les sens des relations gaudelles entre les critères et la classe. Étant donnés $a_j \in A$ et λ , on définit :

- Le RIR ascendant (*upward rank inconsistency rate*) selon a_j :

$$URIR^{\leq}(a_j, \lambda) = -\frac{1}{|\Omega|} \sum_{i=1}^n \log_2 \frac{|[\omega_i]_{a_j}^{\leq}| \times |[\omega_i]_{\lambda}^{\geq}|}{|\Omega| \times |[\omega_i]_{a_j}^{\leq} \cap [\omega_i]_{\lambda}^{\geq}|}$$

- Le RIR descendant (*downward rank inconsistency rate*) selon a_j :

$$DRIR^{\geq}(a_j, \lambda) = -\frac{1}{|\Omega|} \sum_{i=1}^n \log_2 \frac{|[\omega_i]_{a_j}^{\geq}| \times |[\omega_i]_{\lambda}^{\leq}|}{|\Omega| \times |[\omega_i]_{a_j}^{\geq} \cap [\omega_i]_{\lambda}^{\leq}|}$$

a_j croît avec λ lorsque $URIR^{\leq}(a_j, \lambda) = 0$, et décroît avec λ quand $DRIR^{\geq}(a_j, \lambda) = 0$.
La différence entre $URIR^{\leq}(a_j, \lambda)$ et $DRIR^{\geq}(a_j, \lambda)$ est représentée par :

$$diff_{a_j} = URIR^{\leq}(a_j, \lambda) - DRIR^{\geq}(a_j, \lambda)$$

Pei et Hu établissent un seuil $\delta \in [0, 1]$ permettant de décider s'il existe une relation graduelle entre a_j et λ . Si $diff_{a_j} \in [-\delta, \delta]$, alors a_j n'est pas monotone par rapport à λ . Sinon, si $diff_{a_j} \in [\delta, \infty[$, alors a_j décroît avec λ . Au contraire, si $diff_{a_j} \in]-\infty, -\delta]$, alors a_j croît avec λ .

Ces mesures permettent de définir un algorithme capable de déterminer s'il existe une relation graduelle entre chaque attribut et la classe ainsi que son éventuel sens de variation.

Afin de partitionner au mieux les données à chaque étape de construction de l'arbre, les mesures de sélection d'attributs RMI et MI sont utilisées.

L'information mutuelle MI (*Mutual Information*) correspond au degré de consistance entre un attribut et les valeurs de la classe. L'information mutuelle entre l'attribut a_j et la classe λ est défini par :

$$MI(a_j, \lambda) = -\frac{1}{|\Omega|} \sum_{i=1}^n \log_2 \frac{|[\omega_i]_{a_j}| \times |[\omega_i]_{\lambda}|}{|\Omega| \times |[\omega_i]_{a_j} \cap [\omega_i]_{\lambda}|}$$

De plus, Pei et Hu utilisent le coefficient d'information maximale MIC (*Maximal Information Coefficient*) pour enlever les attributs impertinents. Soient X et Y deux variables aléatoires, $|X|$ et $|Y|$ représentent le nombre d'intervalles sur les axes X et Y . Le nombre total d'intervalles n'excède pas une valeur T spécifiée. MIC est défini par :

$$MIC(X, Y) = \max_{|X||Y| < T} \frac{MI(X, Y)}{\log_2(\min(|X|, |Y|))}$$

À l'aide de ces mesures, les auteurs développent PMDT (*Partially Monotonic Decision Trees*), un algorithme de construction d'arbres partiellement monotones. Comme les autres méthodes, celle-ci ne considère que les arbres binaires où à chaque noeud interne correspond un attribut. Elle ne gère que les attributs numériques, et les valeurs de classe doivent être ordonnées.

Soit Ω_α l'ensemble des exemples considérés à l'étape actuelle de construction, ϵ le seuil minimal pour MI et RMI, et n_{min} la taille minimale pour Ω_α . À chaque étape de l'induction :

- Les éléments $\omega_i \in \Omega$ sont normalisés
- Les attributs impertinents sont retirés avec MIC afin d'obtenir Ω'
- Les attributs monotones et les attributs non-monotones sont différenciés en utilisant RIR
- Si tous les éléments de Ω' appartiennent à la même classe c , alors une feuille est créée et on lui assigne la classe c
- Sinon, si $|\Omega_\alpha| < n_{min}$ alors une feuille est créée et on lui assigne la classe majoritaire dans Ω_α
- Sinon
 - si le noeud courant n'est pas plus pur que son noeud parent, alors le seuil générant la meilleure partition selon MI est récupéré
 - sinon, le seuil générant la meilleure partition selon RMI est récupéré
- Si la partition obtenue est vide ou la valeur de MI ou RMI est plus petite que ϵ , alors la construction de l'arbre est arrêtée
- Sinon, la même procédure est répétée sur les sous-ensembles de la partition

3 Etude théorique des propriétés des mesures de discrimination d'ordre

Marsala et Petturiti définissent dans [11] un modèle de construction hiérarchique de mesures de discrimination d'ordre. Ce modèle a pour but d'isoler les propriétés qu'une fonction doit avoir pour être une telle mesure. Il sert aussi de base pour en créer de nouvelles.

Soit H^* une mesure de discrimination d'ordre. On rappelle que le pouvoir de discrimination de l'attribut a_j envers la classe λ peut se décomposer de la façon suivante :

$$H^*(\lambda|a_j) = h^*(g^*(f^*(\omega_1), \dots, f^*(\omega_n)))$$

où f^* est une mesure de la monotonie locale de l'objet, g^* une mesure de non-monotonie de l'objet, et h^* une agrégation des mesures g^* .

Pour que H^* soit une mesure de discrimination d'ordre, chaque couche doit satisfaire certaines conditions.

Soient $\omega_i \in \Omega$ et f^*, g^*, h^* les trois couches de H^* . Les propriétés suivantes sont démontrées en annexe :

4 Implémentation et expérimentation de l'algorithme de construction d'arbres monotones

On implémente une librairie de fonctions permettant la construction ainsi que l'étude expérimentale d'arbres de décision. On se base essentiellement sur RDMT(H), donné dans [11], pour construire des arbres de décision monotones, que l'on évalue sur des données artificielles et réelles. Le code est implémenté en Python 3.6 et les librairies numpy, matplotlib et scikit-learn sont utilisées.

4.1 Mesures de discrimination d'ordre

On implémente le modèle de construction hiérarchique proposé par Marsala et Petturiti. On rappelle que le pouvoir de discrimination de l'attribut a_j envers la classe λ selon une mesure de discrimination H^* peut s'écrire de la façon suivante :

$$H^*(\lambda|a_j) = h^*(g^*(f^*(\omega_1), \dots, f^*(\omega_n)))$$

Toutes les fonctions f^*, g^*, h^* présentées dans l'article sont codées. Pour chaque fonction f^* , on distingue le cas où la mesure construite détecte la monotonie (i.e. elle considère les ensembles dominants) du cas contraire (i.e. elle considère les classes d'équivalences).

Les ensembles dominants et classes d'équivalence servant de base pour le calcul de $f^*(\omega_i)$, pour tout $\omega_i \in \Omega$, le stockage de ces structures joue un rôle important dans la complexité du critère de partitionnement.

Étant donnés $\Omega = \{\omega_1, \dots, \omega_n\}$, $\mathcal{A} = \{a_1, \dots, a_m\}$, λ , et $j \in \{1, \dots, m\}$ fixé, on stocke les ensembles dominants générés par l'attribut a_j (respectivement la fonction d'étiquetage λ) dans une matrice notée A (respectivement D) de taille $n \times n$. Pour tout $i, h \in \{1, \dots, n\}$,

- $A_{ih} = 1$ si et seulement si $a_j(\omega_i) \leq a_j(\omega_h)$;
- $D_{ih} = 1$ si et seulement si $\lambda(\omega_i) \leq \lambda(\omega_h)$.

A (respectivement D) est donc rempli en itérant sur chaque $\omega_i \in \Omega$ et en assignant à chaque ligne A_i (respectivement D_i) l'ensemble $\{\omega_h : a_j(\omega_i) \leq a_j(\omega_h)\}$ (respectivement $\{\omega_h : \lambda(\omega_i) \leq \lambda(\omega_h)\}$). Récupérer ce dernier se fait en temps $\mathcal{O}(n)$. On construit donc A et D en temps $\mathcal{O}(n^2)$. Comme les valeurs sont stockées dans une matrice de taille $n \times n$, la complexité spatiale est aussi en $\mathcal{O}(n^2)$.

En ce qui concerne les classes d'équivalence, pour tout $i, h \in \{1, \dots, n\}$,

- $A_{ih} = 1$ si et seulement si $a_j(\omega_i) = a_j(\omega_h)$
- $D_{ih} = 1$ si et seulement si $\lambda(\omega_i) = \lambda(\omega_h)$.

La procédure de construction est la même que celle donnée précédemment, en remplaçant l'inégalité par l'égalité. On obtient les mêmes complexités temporelle et spatiale. Comme les algorithmes maniant les ensembles dominants et ceux utilisant les classes d'équivalence partagent les mêmes complexités, on étudiera uniquement les fonctions faisant intervenir les ensembles dominants.

Pour a_j et λ fixés, A et D ne sont calculés qu'une seule fois car $[\omega_i]_{a_j}^{\leq}$ et $[\omega_i]_{\lambda}^{\leq}$ restent constants, pour tout $\omega_i \in \Omega$.

On s'intéresse maintenant au calcul de $f^*(\omega_i)$, qu'on appellera également $*-dsr(\omega_i)$. Afin de calculer $dsr(\omega_i)$, on récupère $[\omega_i]_{a_j}^{\leq}$ et $[\omega_i]_{\lambda}^{\leq}$ à l'aide de A et D en temps $\mathcal{O}(n)$. Le calcul de

$[\omega_i]_{a_j} \cap [\omega_i]_{\lambda}$ se faisant aussi en $\mathcal{O}(n)$, calculer $d\text{sr}(\omega_i)$ se fait en temps $\mathcal{O}(n)$. L'espace occupé est en $\mathcal{O}(n)$. En revanche, le calcul de $\text{maxd}\text{sr}(\omega_i)$, $\text{mind}\text{sr}(\omega_i)$ et $\text{avgd}\text{sr}(\omega_i)$ nécessite d'itérer sur chaque $\omega_h \in [\omega_i]_{a_j}$. Les complexités temporelle et spatiale de ces fonctions sont donc en $\mathcal{O}(n^2)$.

Afin d'observer le comportement des *-d_{sr} sur un attribut monotone, on génère une base de 100 points à deux classes, deux dimensions dont une seule est monotone par rapport à la classe. Ici, la valeur de l'attribut monotone et celle de la classe croissent en fonction de l'indice de l'élément. On duplique et bruitte la même base de sorte à obtenir des bases bruitées à 0%, 25%, 50%, 75%. Les courbes 1, 2, 3, et 4 tracent la valeur de $d\text{sr}(\omega_i)$ en fonction de i pour chaque base.

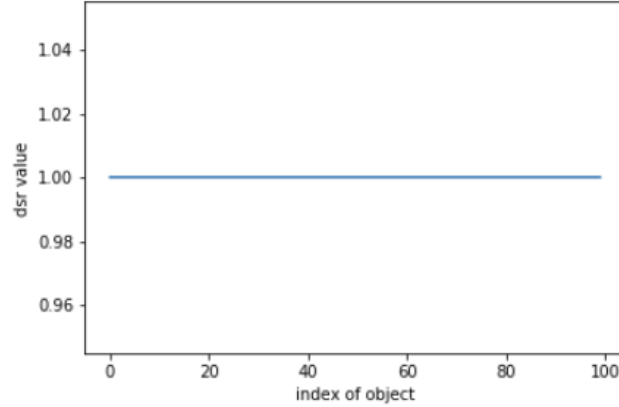


FIGURE 1 – Valeur de d_{sr} en fonction de l'indice de l'objet pour une base non-bruitée

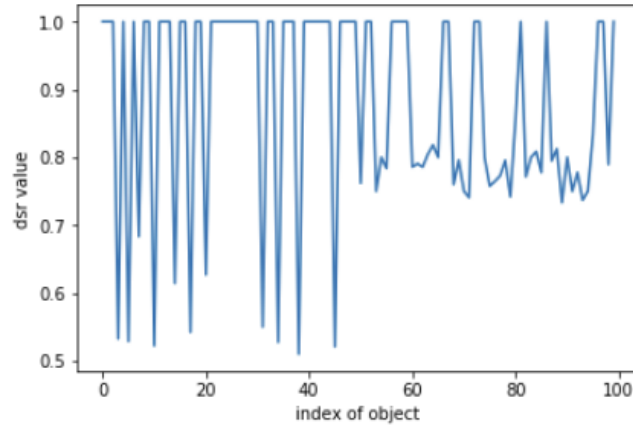


FIGURE 2 – Valeur de d_{sr} en fonction de l'indice de l'objet pour une base bruitée à 25%

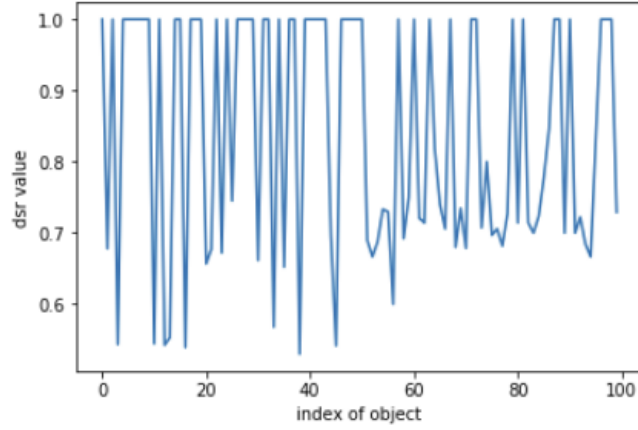


FIGURE 3 – Valeur de dsr en fonction de l'indice de l'objet pour une base bruitée à 50%

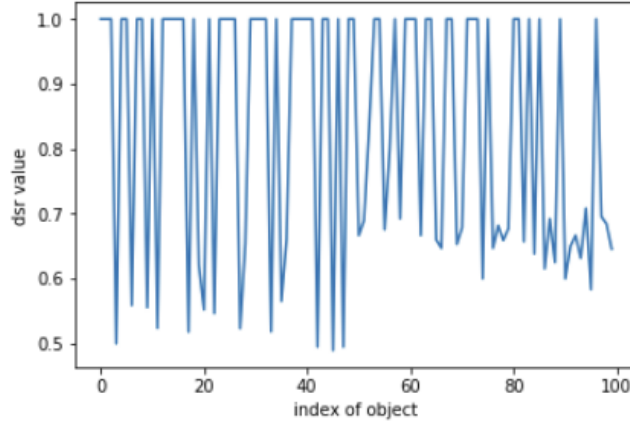


FIGURE 4 – Valeur de dsr en fonction de l'indice de l'objet pour une base bruitée à 75%

Dans la base non bruitée, $\text{dsr}(\omega_i) = 1$ pour chaque $\omega_i \in \Omega$. En effet, on a $a_j(\omega_i) \leq a_j(\omega_h) \Rightarrow \lambda(\omega_i) \leq \lambda(\omega_h)$ si et seulement si $\{\omega_h \in \Omega : \lambda(\omega_i) \leq \lambda(\omega_h) \wedge a_j(\omega_i) \leq a_j(\omega_h)\} = \{\omega_h \in \Omega : a_j(\omega_i) \leq a_j(\omega_h)\}$ si et seulement si $\frac{|\{\omega_i\}_{\lambda}^{\leq} \cap \{\omega_i\}_{a_j}^{\leq}|}{|\{\omega_i\}_{a_j}^{\leq}|} = 1$.

En revanche, plus le taux de bruit dans la base augmente, plus on observe de "pics". En effet, on observe davantage de ω_i tels que $\{\omega_h \in \Omega : a_j(\omega_i) \leq a_j(\omega_h)\}$ est modifié, et donc $|\{\omega_h \in \Omega : \lambda(\omega_i) \leq \lambda(\omega_h) \wedge a_j(\omega_i) \leq a_j(\omega_h)\}| \leq |\{\omega_h \in \Omega : a_j(\omega_i) \leq a_j(\omega_h)\}|$. Les courbes des autres *-dsr sont données en annexe.

Toutes les fonctions g^* présentées dans l'article ont des complexités temporelle et spatiale en $\mathcal{O}(1)$. En revanche, comme les fonctions h^* agrègent les couches g^* correspondant à chaque élément de Ω , elles sont toutes en $\mathcal{O}(n)$.

Le calcul de $H^*(\lambda|a_j)$ se fait donc en :

- $\mathcal{O}(n^2)$ lorsque $f^* = \text{dsr}$
- $\mathcal{O}(n^3)$ lorsque $f^* \in \{\text{mindsr}, \text{maxdsr}, \text{avgdsr}\}$

On étudie, pour chaque couple de mesures de discrimination (H, H') données dans [11], l'évolution de H' en fonction de H pour 2, 3 et 5 classes.

Pour cela, on génère aléatoirement, pour chaque nombre de classes, 100 bases de 100 exemples à un attribut monotone par rapport à la classe. Pour chaque base, on récupère les seuils de coupure engendrés par la discrétisation de l'attribut (étape décrite dans la section suivante) et, pour chaque mesure, on enregistre les valeurs obtenues pour chaque seuil.

Les figures 5, 6, 7 tracent les corrélations entre chaque mesure de discrimination.

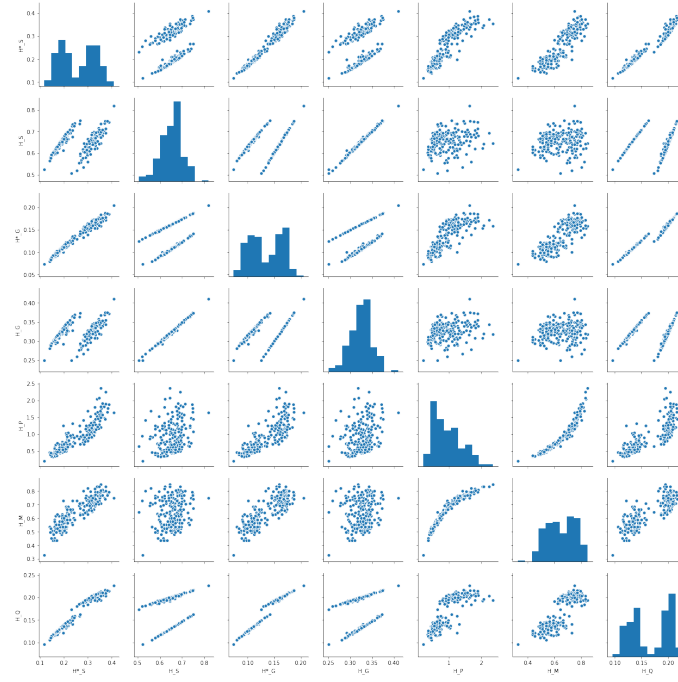


FIGURE 5 – H' en fonction de H (2 classes)

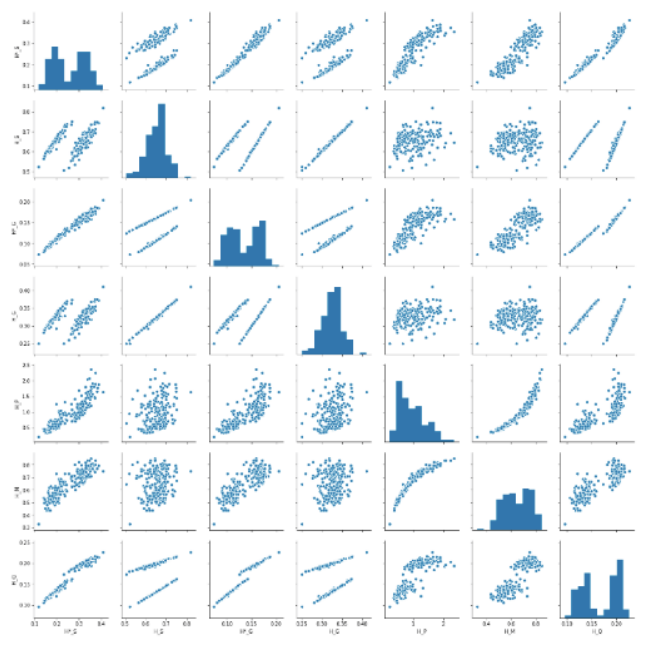


FIGURE 6 – H' en fonction de H (3 classes)

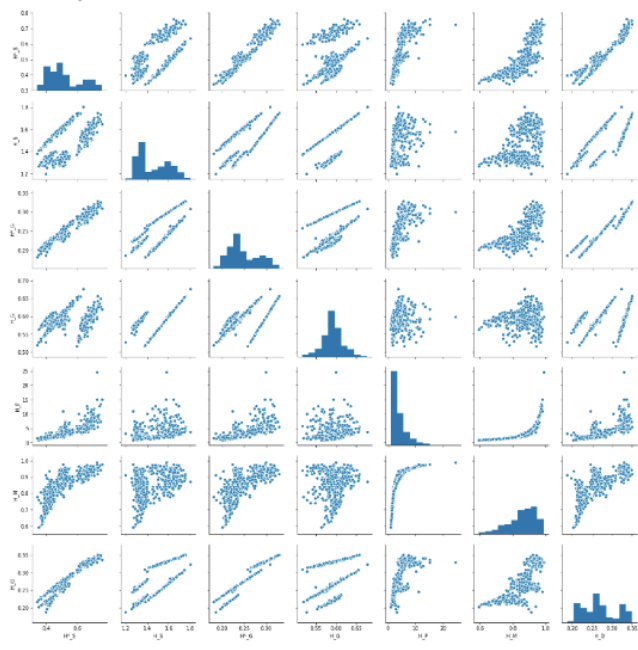


FIGURE 7 – H' en fonction de H (5 classes)

Pour chaque cas, on remarque que les mesures suivantes sont corrélées linéairement :

- H_S^* et H_G^*
- H_G et H_S

Et les mesures suivantes sont corrélées quadratiquement :

- H_P et H_M
- H_Q et H_S^*

4.2 Algorithme de construction d'arbres de décision paramétré par une mesure de discrimination

En reprenant essentiellement RDMT, l'algorithme 2 est un classifieur paramétré par une mesure de discrimination H (d'ordre ou non) et par 3 critères d'arrêt. Comme RDMT, cet algorithme ne produit pas un arbre globalement monotone. Néanmoins, si les données sont monotone-consistantes et si H tient compte de la relation graduelle entre les valeurs d'attributs et les valeurs de classe, alors cette méthode garantit une forme plus faible de monotonie (appelée *rule monotonicity*).

RDMT et notre algorithme diffèrent sur deux points :

- Lorsqu'une feuille est créée, on lui assigne la classe majoritaire parmi les exemples ayant servi à sa construction
- On distingue deux mesures de discrimination : l'une doit être minimisée pour le partitionnement et l'autre permet de déterminer l'arrêt. Dans la suite, la deuxième mesure utilisée sera toujours H_S .

Comme l'on se restreint à des arbres de décision binaires, les données doivent être partitionnées en deux sous-ensembles à chaque étape de l'induction. Cette partition se fait en divisant l'ensemble courant Ω_α selon l'attribut a^* qui respecte le plus la contrainte de monotonie, i.e. celui dont la valeur $x_{j_s}^*$ minimise $H(\lambda|a_j^{x_{j_s}^*})$, pour $j = 1, \dots, m, s = 1, \dots, t_j - 1$.

Etant donné une mesure de discrimination H^* , *DISCRETIZE* (algorithme 1) permet de récupérer, pour chaque $a_j \in \mathcal{A}$, le seuil de coupure minimisant la valeur de H ainsi que cette dernière. On parcourt tous les seuils de coupure x_{j_s} de a_j pour trouver x_j^* minimisant $H(\lambda|a_j^{x_{j_s}^*})$, soit :

$$x_j^* = \arg \min \{H(\lambda|a_j^{x_{j_s}^*}), s = 1, \dots, t_j - 1\}$$

Algorithm 1 Discrétisation

procedure DISCRETIZE(H^*, Ω) $\triangleright H^*$: mesure de discrimination construite de façon hiérarchique $\triangleright \Omega$: base de données étiquetées $\triangleright a_j$: attribut à discrétiser $n \leftarrow |\Omega|$ $T \leftarrow$ matrice dont la première colonne contient les valeurs de $a_j(\omega_i)$ triées par ordre croissant, la deuxième, les valeurs de $\lambda(\omega_i)$ triées selon $a_j(\omega_i)$, et la troisième, les $i = 1, \dots, n$ triés selon $a_j(\omega_i)$, pour tout $\omega_i \in \Omega$. $U \leftarrow \{a_j(\omega_i) : \forall \omega_i \in \Omega\}$ \triangleright valeurs uniques de $a_j(\omega_i)$ pour tout $\omega_i \in \Omega$ $a_j^b(\omega_i) \leftarrow 1, \forall \omega_i \in \Omega$ \triangleright valeur de l'attribut a_j binarisée i.e $a_j^b(\omega_i) = 0$ si $a_j(\omega_i) \leq x_{j_s}$, 1 sinon, pour x_{j_s} fixé $[\omega_i]_{a_j^b}^{\leq} \leftarrow \Omega, \forall \omega_i \in \Omega$ $[\omega_i]_{a_i}^{\leq} \leftarrow \{\omega_h \in \Omega : \lambda(\omega_i) \leq \lambda(\omega_h)\}$ $S \leftarrow \emptyset$ \triangleright seuils de coupure considérés $E \leftarrow \emptyset$ \triangleright valeurs de H^* obtenues pour chaque seuil de coupure $s \in S$ $V \leftarrow \emptyset$ \triangleright ensemble des $\omega \in \Omega$ déjà visités**for** $v \in U$ **do** $\Omega_v \leftarrow \{\omega_i \in \Omega : a_j(\omega_i) = v\}$ $C_v \leftarrow \{c \in C : \exists \omega_h \in \Omega_v, \lambda(\omega_h) = c\}$ $v' \leftarrow$ valeur suivante dans U $\Omega_{v'} \leftarrow \{\omega_i \in \Omega : a_j(\omega_i) = v'\}$ $C_{v'} \leftarrow \{c \in C : \exists \omega_h \in \Omega_{v'}, \lambda(\omega_h) = c\}$ $V \leftarrow V \cup \{\Omega_v\}$ $a_j^b(\omega_i) \leftarrow 0$ pour tout $\omega_i \in \Omega_v$ $x_{j_s} \leftarrow \frac{v+v'}{2}$ **if** $C_{v'} \neq C_v$ **then** $[\omega_i]_{a_j^b} \leftarrow \Omega, \forall \omega_i \in V$ $\bar{V} \leftarrow \Omega \setminus V$ \triangleright ensemble des $\omega \in \Omega$ non visités $[\omega_i]_{a_j^b}^{\leq} \leftarrow \bar{V}, \forall \omega_i \in \bar{V}$ $S \leftarrow S \cup \{x_{j_s}\}$ $E \leftarrow E \cup \{H^*(\lambda|a_j^b)\}$ **return** $\min E, \arg \min_{s \in S} E$ \triangleright retourner la valeur d'entropie minimale et le seuil de coupure permettant de l'obtenir

Algorithm 2 Construction de l'arbre

```
procedure BUILD_DT( $\Omega_\alpha, H^*, H, \epsilon, \Delta, n_{min}, \lambda, \delta$ )  
     $\triangleright \Omega_\alpha$  : base de données étiquetées  
     $\triangleright H^*$  : mesure de discrimination utilisée pour le partitionnement  
     $\triangleright H$  : mesure de discrimination utilisée pour déterminer l'arrêt  
     $\triangleright \epsilon$  : limite inférieure pour H  
     $\triangleright \Delta$  : longueur maximale d'un chemin de la racine à une feuille  
     $\triangleright n_{min}$  : taille minimale pour  $\Omega_\alpha$   
     $\triangleright \lambda$  : fonction d'étiquetage  
     $\triangleright \delta$  : longueur du chemin de la racine au noeud courant  
  
     $h \leftarrow H(\Omega_\alpha, \lambda)$   
    if  $h < \epsilon$  or  $|\Omega_\alpha| < n_{min}$  or  $\forall \omega_i, \omega_h \in \Omega_\alpha, \lambda(\omega_i) = \lambda(\omega_h)$  or  $\delta > \Delta$  then  
         $c_\alpha \leftarrow c$  tel que  $|\{\omega_i \in \Omega_\alpha : \lambda(\omega_i) = c\}| \geq |\{\omega_i \in \Omega_\alpha : \lambda(\omega_i) \neq c\}|$   
        return LEAF( $c_\alpha, \Omega$ )  
  
     $m \leftarrow$  nombre d'attributs dans  $\Omega$   
     $S \leftarrow \emptyset$   $\triangleright$  pour chaque  $a_j \in \mathcal{A}$ , contient  $x_{j*} = \arg \min\{H(\lambda|a_j^{x_{js}}), s = 1, \dots, t_j - 1\}$   
     $E \leftarrow \emptyset$   $\triangleright$  pour chaque  $a_j \in \mathcal{A}$ , contient  $\min\{H(\lambda|a_j^{x_{js}}), s = 1, \dots, t_j - 1\}$   
  
    for  $a_j=0$  to  $m-1$  do  
        if  $a_j(\omega_i) = a_j(\omega_h), \forall \omega_i, \omega_h \in \Omega_\alpha$  then  
             $S \leftarrow S \cup \{\infty\}$   
             $E \leftarrow E \cup \{\infty\}$   
        else  
             $x_{j*}, h \leftarrow \text{DISCRETIZE}(H^*, \Omega_\alpha, a_j)$   
             $S \leftarrow S \cup \{x_{j*}\}$   
             $E \leftarrow E \cup \{h\}$   
  
     $x_* \leftarrow \arg \min_{s \in S} E$   
     $a_*^{x_*} \leftarrow \arg \min_{a_j \in \mathcal{A}} E$   
  
     $\Omega_{\leq}, \Omega_{\geq} \leftarrow \text{DIVIDE}(\Omega_\alpha, a_*^{x_*}, x_*)$   
    if  $|\Omega_{\leq}| = 0$  then  
        return  $\{c\}$  tel que  $|\{\omega_i \in \Omega_{\geq} : \lambda(\omega_i) = c\}| \geq |\{\omega_i \in \Omega_{\geq} : \lambda(\omega_i) \neq c\}|$   
    if  $|\Omega_{\geq}| = 0$  then  
        return  $\{c\}$  tel que  $|\{\omega_i \in \Omega_{\leq} : \lambda(\omega_i) = c\}| \geq |\{\omega_i \in \Omega_{\leq} : \lambda(\omega_i) \neq c\}|$   
     $\mathcal{T}_{\leq} \leftarrow \text{BUILD\_DT}(\Omega_{\leq}, H^*, H, \epsilon, \Delta, n_{min}, \lambda, \delta + 1)$   
     $\mathcal{T}_{\geq} \leftarrow \text{BUILD\_DT}(\Omega_{\geq}, H^*, H, \epsilon, \Delta, n_{min}, \lambda, \delta + 1)$   
     $\mathcal{T} \leftarrow \text{TREE}(\{a_*^{x_*}\}, \mathcal{T}_{\leq}, \mathcal{T}_{\geq})$   
    return  $\mathcal{T}$ 
```

Considérons la base de la figure 8 contenant 60 points à deux dimensions x et y et à 3 classes. x est l'attribut dont les valeurs sont monotones par rapport aux valeurs de la classe. Chaque classe est représentée par le même nombre de points. La classe 1 est représentée par les points bleus, la 2 par les points oranges, et la 3 par les points verts.

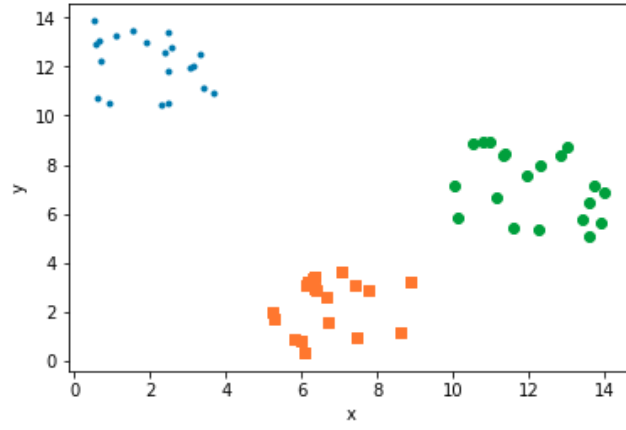
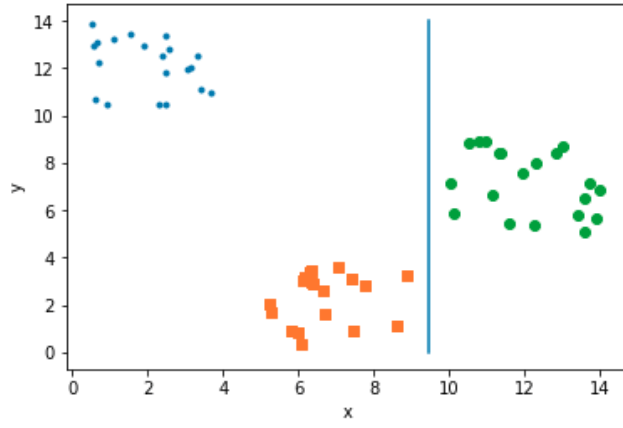
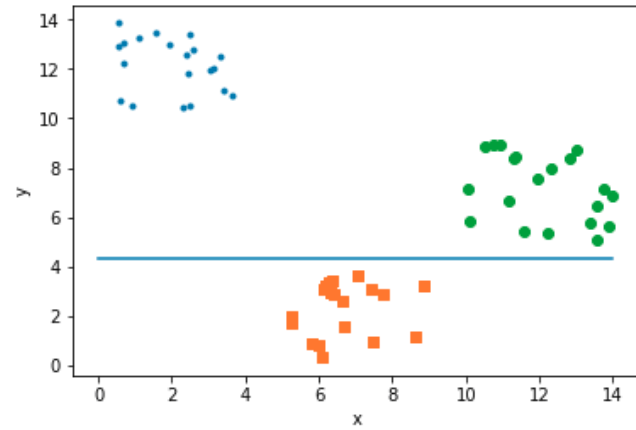


FIGURE 8 – Base de données jouet



$$H_S^*(\lambda|\mathbf{x}) = 0.19 \leq H_S^*(\lambda|\mathbf{y}) = 0.53$$

FIGURE 9 – Seuil de coupure généré par H_S^*



$$H_S(\lambda|\mathbf{x}) = H_S(\lambda|\mathbf{y}) = 0.67$$

FIGURE 10 – Seuil de coupure généré par H_S

On observe sur les figures 9 et 10 les seuils de coupures générés respectivement par H_S^* et H_S sur cette base jouet. On remarque que H_S^* tient compte de la monotonie des valeurs de la classe par rapport aux valeurs de l'attribut x : la valeur d'entropie minimale calculée pour l'attribut x est inférieure à celle calculée pour l'attribut y . Ce n'est cependant pas le cas pour H_S qui est insensible à la monotonie : la coupure optimale sur l'axe x génère la même valeur d'entropie que celle faite sur l'axe y .

Références

- [1] A. BEN-DAVID. “Monotonicity maintenance in information-theoretic machine learning algorithms”. In : *Machine Learning* 19.1 (1995), p. 29-43.
- [2] A. BEN-DAVID, L. STERLING et Y.-H. PAO. “Learning and classification of monotonic ordinal concepts”. In : *Computational Intelligence* 5.1 (1989), p. 45-49.
- [3] A. BEN-DAVID, L. STERLING et T. TRAN. “Adding monotonicity to learning algorithms may impair their accuracy”. In : *Expert Systems with Applications* 36.3 (2009), p. 6627-6634.
- [4] K. CAO-VAN. “Supervised ranking : from semantics to algorithms”. Thèse de doct. Ghent University, 2003.
- [5] K. CAO-VAN et B. DE BAETS. “Consistent representation of rankings”. In : *Theory and Applications of Relational Structures as Knowledge Instruments*. Springer, 2003, p. 107-123.
- [6] S. GRECO, B. MATARAZZO et R. SLOWINSKI. “A new rough set approach to evaluation of bankruptcy risk”. In : *Operational tools in the management of financial risks*. Springer, 1998, p. 121-136.
- [7] S. GRECO, B. MATARAZZO et R. SLOWINSKI. “Customer satisfaction analysis based on rough set approach”. In : *Zeitschrift für Betriebswirtschaft* 77.3 (2007), p. 325-339.
- [8] Q. HU, X. CHE, L. ZHANG, D. ZHANG, M. GUO et D. YU. “Rank entropy-based decision trees for monotonic classification”. In : *IEEE Transactions on Knowledge and Data Engineering* 24.11 (2012), p. 2052-2064.
- [9] B. LEO, J. H. FRIEDMAN, R. A. OLSHEN et C. J. STONE. “Classification and regression trees”. In : *Wadsworth International Group* (1984).
- [10] C. MARSALA. “Gradual fuzzy decision trees to help medical diagnosis”. In : *Fuzzy Systems (FUZZ-IEEE), 2012 IEEE International Conference on*. IEEE. 2012, p. 1-6.
- [11] C. MARSALA et D. PETTURITI. “Rank discrimination measures for enforcing monotonicity in decision tree induction”. In : *Information Sciences* 291 (2015), p. 143-171.
- [12] M. J. PAZZANI, S. MANI et W. R. SHANKLE. “Acceptance of rules generated by machine learning among medical experts”. In : *Methods of information in medicine* 40.05 (2001), p. 380-385.
- [13] S. PEI et Q. HU. “Partially monotonic decision trees”. In : *Information Sciences* 424 (2018), p. 104-117.
- [14] Y. QIAN, H. XU, J. LIANG, B. LIU et J. WANG. “Fusing monotonic decision trees”. In : *IEEE Transactions on Knowledge and Data Engineering* 27.10 (2015), p. 2717-2728.
- [15] J. R. QUINLAN. “Induction of decision trees”. In : *Machine learning* 1.1 (1986), p. 81-106.
- [16] F. XIA, W. ZHANG, F. LI et Y. YANG. “Ranking with decision tree”. In : *Knowledge and information systems* 17.3 (2008), p. 381-395.