Institut National des Sciences Appliquées et de Technologie

UNIVERSITÉ DE CARTHAGE

Projet de Fin d'Année

Filière : Génie Logiciel

MC1 : Détection d'Anomalies à l'aide d'Arbres de Décision Flous

---

Présenté par

SLAMA Mohamed Taieb

SFAR GANDOURA Karem

LANDOULSI Mohamed Iheb

Encadré par

Mme CHATER Meriem

Année Académique : 2020/2021

Table des matières

Table des figures

Liste des tableaux

1

Chapitre Introductif

Introduction

De nos jours, la quantité d'information qui circule dans les différents systèmes est de plus en plus importante. De ce fait, la détection d'anomalies est devenue un problème des plus importants, afin de garantir l'intégrité des données qui circulent. Pour assurer la supervision, maintes solutions se basant sur les algorithmes de Machine Learning ont été développées. Ces modèles requièrent souvent l'intervention d'experts et de moyens de mesures précis pour être créés. Dans le but de contourner ce problème, il est possible d'utiliser la logique floue qui se rapporte plus au raisonnement du cerveau humain en limitant les outils de mesures nécessaires. Nous présenterons lors de ce rapport un état de l'art de la logique floue, ainsi que celle des algorithmes d'apprentissage non supervisés du Machine Learning. Nous nous focaliserons, par la suite, sur un algorithme précis qu'est l'isolation Forest. Enfin, nous proposerons une solution pour combiner la logique Floue avec cet algorithme dans le but de détecter les anomalies présentes dans notre jeu de données.

1 La Logique Floue

1.1 Concept général

La logique floue a été introduite en mathématique en tant qu'un changement paradigmatique dans la perception de l'incertitude. La logique floue a été proposée par le professeur Lotfi A. Zadeh de L'université de Californie à Berkeley en 1965. La logique floue diffère de la logique classique en ce que les déclarations ne sont plus noir ou blanc, vrai ou faux, activé ou désactivé. Dans la logique traditionnelle, un objet prend une valeur de zéro ou un. En logique floue, une instruction peut prendre n'importe quelle valeur réelle entre 0 et 1, représentant le degré d'appartenance d'un élément à un ensemble donné.

Un exemple pouvant mettre en valeur le besoin dont est né la logique floue serait tel qui suit ; Considérons un système critique (S), avec un capteur C lisant une température T. Pour ce système (S), si T dépasse les 90°C, alors il est considéré comme défaillant, sinon il fonctionne parfaitement :

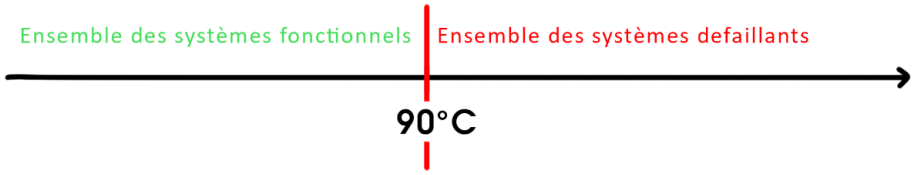


FIGURE 1.1 – Exemple d'un système critique (S)

Cela implique qu'un système avec une température infinitésimalement supérieure à la valeur barrière (90°C) est considéré défaillant qui n'est pas toujours le cas. Il serait utile donc de juger un "**degré de défaillance**" :

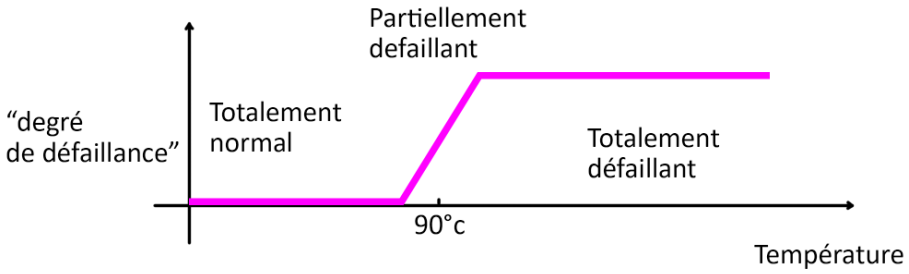


FIGURE 1.2 – Le système (S) avec un degré de défaillance

1.2 Introduction aux fonctions d'appartenance

Dans la logique classique « **crisp logic** », la fonction caractéristique, qui détermine l'appartenance à un ensemble A, affecte 1 ou 0 à chaque élément.

Cette fonction peut être généralisée en tant qu'une fonction qui renvoie un intervalle [0,1]. Une valeur plus large indique une appartenance plus importante à l'ensemble A. Cette fonction est appelée dans ce cas « **fonction d'appartenance** » ou « **membership function** » et l'ensemble correspondant **Fuzzy set**. La membership function est notée :

$$u_A : X \to [0, 1]$$

La figure ci-dessous représente quelques exemples de membership fonctions :

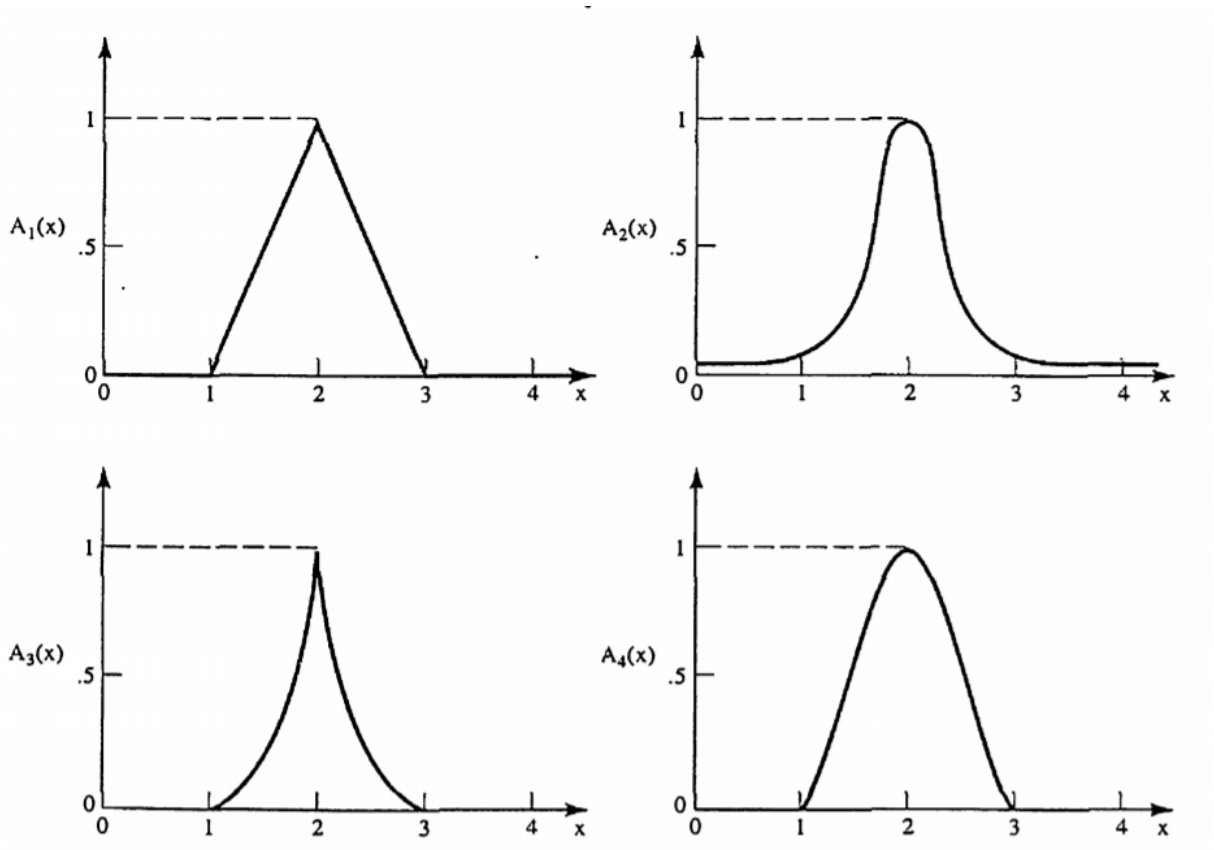


FIGURE 1.3 – Différents exemples de fonctions d'appartenance pouvant servir à caractériser des Fuzzy Sets dans différents contextes [1]

On appelle le complémentaire du fuzzy set A dont l'univers correspondant est X , le

fuzzy set tel que :

$$\forall\, x \in X,\ \bar{A}(x) = 1 - A(x)$$

1.3 Opérateur α-cut

Étant donnée un Fuzzy set A, on appelle « **α-cut A**» l'ensemble défini par :

$$\alpha A = \{x | A(x) \geq \alpha\}\ avec\ \alpha \in [0, 1]$$

Autrement dit, l'α-cut d'un ensemble flou A est l'ensemble « crisp » αA qui contient tous les éléments de l'ensemble universel X, dont les grades d'appartenance à A, sont supérieur ou égal à la valeur spécifiée de α.

1.4 Fuzzy Inference System (F.I.S)

Dans le domaine de l'ingénierie, la logique floue est utilisée pour créer des systèmes de contrôles appelées « fuzzy inference system ». L'objectif de ce système est de produire un ensemble de valeurs de sortie « outputs» à partir de valeurs d'entré « inputs ». Dans les cas des systèmes de contrôle conventionnel, le système utilise soit des formules mathématiques qui peuvent être complexes ou même impossible à implémenter soit une table pour sauvegarder tous les cas, ce qui peut être très coûteux. Le Fuzzy Control System surmonte ces deux désavantages et rend le développement plus simple.

Le Fuzzy Inference process représente les étapes nécessaires dans la création du Fuzzy Control System. Les 5 blocs fonctionnels suivants sont les composant du fuzzy interface process :

— Rule Base : Contient des règles « IF-THEN » floues.
— Data Base : Définit les fonctions d'appartenance des ensembles flous utilisés dans les règles floues.
— Decision-making Unit : Exécute des opérations sur les règles.
— Fuzzification Inference Unit : Convertit les quantités nettes en quantités floues.
— Defuzzification inference Unit : Convertit les quantités floues en quantités nettes.

Voici un schéma de principe du système d'interférence floue :

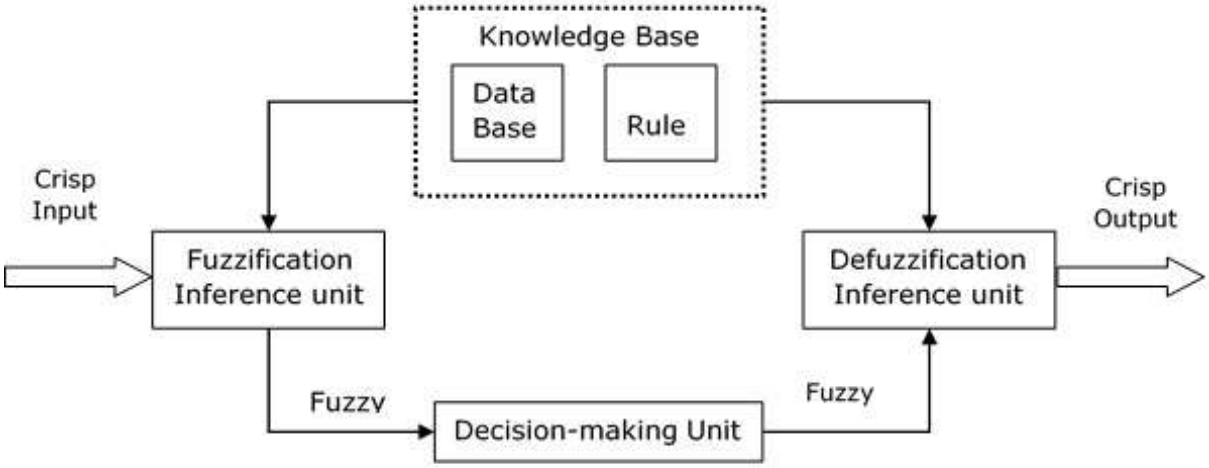


FIGURE 1.4 – Structure d'un Fuzzy Inference System [2]

Le F.I.S est composé de plusieurs composants :

1. Antécédents

   Les antécédents représentent les inputs classiques de la base de données. Le Fuzzy Inference System détermine les fuzzy sets correspondants à chaque antécédent. Afin de déterminer ces fuzzy sets, il faut définir les membership functions correspondantes. Cette tâche est généralement réalisée par un expert du domaine ou le système est implémenté. ;

2. Conséquents

   Les Conséquents représentent les outputs classiques qu'on doit déterminer dans la base de données. Le Fuzzy Inference System détermine les fuzzy sets correspondants à chaque conséquent. Afin de déterminer ces fuzzy sets, il faut définir les membership functions correspondantes. Cette tâche est généralement réalisée par un expert du domaine ou le système est implémenté. ;

3. Rules

   Les Rules représentent les relations qui lient les antécédents aux conséquents. Elles sont sous la forme d'une condition « IF Antécédent i IS A_i , ..., Antécédent j IS A_j THEN Conséquent IS C_j » .

Exemple : Mamdani Fuzzy Inference System

Ce système a été proposé en 1975 par Ebhasim Mamdani. Cette méthode était prévue de contrôler une combinaison de machine à vapeur et de chaudière en synthétisant un

ensemble de règles floues obtenues des personnes travaillant sur le système.

Les étapes pour cette méthode :

— Étape 1 : Un ensemble de règles floues doit être déterminé dans cette étape.

— Étape 2 : Dans cette étape, en utilisant la fonction d'appartenance d'entrée, l'entrée serait rendue floue.

— Étape 3 : Établissez maintenant la force de la règle en combinant les entrées floues selon des règles floues.

— Étape 4 : Dans cette étape, déterminez le conséquent de la règle en combinant la force de la règle et la fonction d'appartenance en sortie.

— Étape 5 : Pour obtenir la distribution de sortie, combinez tous les conséquents.

— Étape 6 :Enfin, une distribution de sortie défuzzifiée est obtenue.

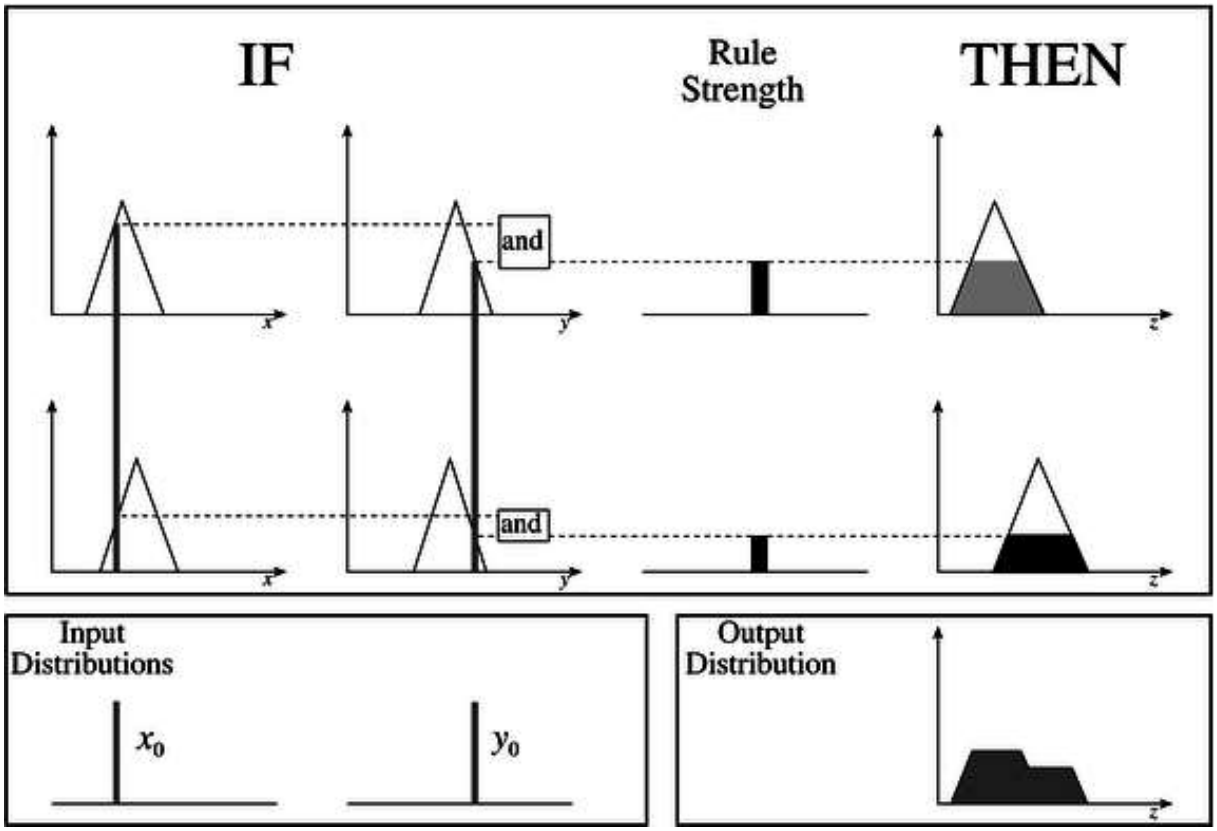


FIGURE 1.5 – Schéma de principe du système d'intérface floue Mamdani [2]

2 Le Machine Learning

2.1 Concept général

Le machine learning (apprentissage automatique) est au cœur de la science des données et de l'intelligence artificielle. Que l'on parle de transformation numérique des entreprises, de Big Data ou de stratégie nationale ou européenne, le machine learning est devenu incontournable. Ses applications sont nombreuses et variées, allant des moteurs de recherche et de la reconnaissance de caractères à la recherche en génomique, l'analyse des réseaux sociaux, la publicité ciblée, la vision par ordinateur, la traduction automatique ou encore le trading algorithmique.
À l'intersection des statistiques et de l'informatique, le machine learning se préoccupe de la modélisation des données.

2.2 Apprentissage Non Supervisé

C'est l'apprentissage automatique où les données ne sont pas étiquetées, c'est à dire n'ont pas de classe. Il s'agit donc de découvrir les structures sous-jacentes à ces données non étiquetées. Puisque les données ne sont pas étiquetées, il est impossible à l'algorithme de calculer de façon certaine un score de réussite. Mais, cette approximation reste très utile dans les domaines du quotidient.
Ce type d'apprentissage se matérialise sous forme d'algorithmes, qui, en se basant sur certaines règles de mathématiques arrivent à classer les éléments non connus.
Parmi ces algorithmes, on arrive à reconnaître celui nommé Isolation Forest qui fera le sujet de notre étude. Cet algorithme sera plus présenté en détails dans le chapitre 2 du rapport.

2

Implémentation de l'Isolation Forest

Introduction

Dans ce chapitre, l'objet portera d'abord sur ce qu'est l'isolation forest, par la suite, il faudra détailler son mode d'emploi et comment mesurer sa performance.

1 Présentation de l'algorithme Isolation Forest (IForest)

La plupart des algorithmes de détection d'anomalie tel que « Replicator Neural Network » [Williams et al. 2002], classification-based methods » [Abe et al. 2006 ; Shi and Horvath 2006], Replicator Neural Network (RNN) [Williams et al. 2002], one-class SVM [Tax et Duin 2004] et clustering-based methods [He et al. 2003] génèrent un modèle du profil normal des instances de la base de données et ensuite identifient lesquelles sont les instances qui ne sont pas conformes au modèle. Cette approche nécessite une analyse

statistique où un calcule de distance dans toute la base de données. C'est pourquoi, ces algorithmes sont très coûteux de point de vue performance. Ce qui limite la taille de la base de données à utiliser.

La capacité de détection des anomalies de cette approche est un sous-produit d'un algorithme conçu pour la classification où le clustering de données ce qui engendre une faible performance de l'algorithme, ce qui est traduit par un grand nombre d'instances mal identifiées. Contrairement à ces algorithmes, L'algorithme IForest propose une approche différente qui cherche à identifier directement les anomalies tout en limitant les calculs réalisés. Cette approche utilise la structure des arbres binaire de décision pour représenter les données.

1.1 Notion d'Arbre et d'arbre de Décision

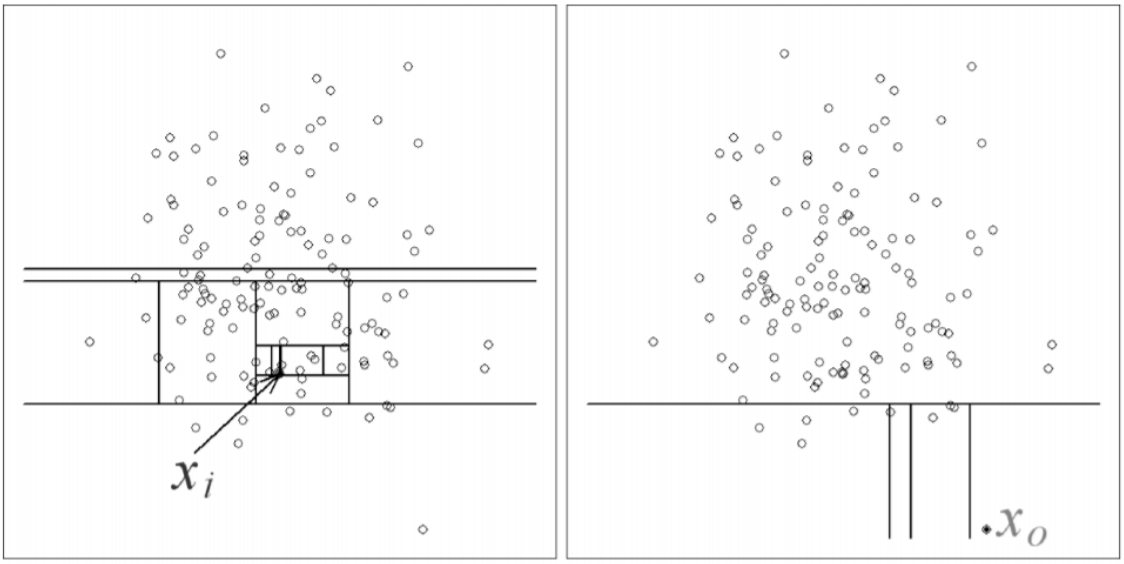
En théorie de graphe, un arbre est un graphe qui constitue des points appelés nœuds liées à travers des arc. L'ensemble des nœuds se divise en trois catégories :
- Nœud racine (l'accès à l'arbre se fait par ce nœud),
- Nœuds internes : les nœuds qui ont des descendants (ou enfants), qui sont à leur tour des nœuds,
- Nœuds terminaux (ou feuilles) : nœuds qui n'ont pas de descendant.

En informatique l'arbre correspond à une structure de donnée qui représente les données en tant que nœuds liés.
Le chemin d'un nœud représente l'ensemble des nœuds parcourue à partir de la racine.
Notons la taille du chemin d'un nœud x : $h(x)$. Il existe plusieurs types d'arbres tel que l'**arbre binaire**, arbre de décision. Nous nous intéressons dans notre projet à l'arbre binaire de recherche et à l'arbre de décision.
-> Un arbre binaire de recherche est un arbre où chaque nœud possède au plus deux nœuds fils.
-> Un arbre binaire strict est un arbre binaire où chaque nœud interne possède deux nœuds fils.

1.2 Principe

Le principe de l'approche IForest est basé sur les caractéristiques fondamentales des anomalies : ces instances sont une minorité dans la base et elles présentent des attributs distinctes des autres instances. Par conséquent, les anomalies sont très susceptibles au mécanisme d'Isolation. Ce mécanisme consiste à la séparation d'une instance des autres. En générale, une approche fondée sur l'isolation mesure la susceptibilité de l'instance à être isolée. Afin d'évaluer cette susceptibilité, l'approche IForest génère des arbres binaires strictes nommés arbre d'isolation où chacun des nœuds fils correspond à une partition de l'ensemble du nœud parent. En d'autres termes, l'algorithme réalise récursivement un split aléatoire de l'ensemble. Par conséquent les instances susceptibles à l'isolation nécessiteront généralement moins de partitionnement que les autres instances ce qui se traduit dans l'arbre par une taille du chemin plus court en moyenne que les autres.

Dans la figure 2.1 nous observons que les instances normales nécessitent généralement plus que 10 partitions plus pour s'isoler contrairement à l'instance x_0 qui nécessite seulement. La figure 2.2 montre que l'anomalie x_0 possède une taille du chemin moyenne (taille du chemin = 4.0) bien plus petits que la moyenne des autres instances (taille du chemin moyen = 12).



(a) Isolating x_i (b) Isolating x_o

FIGURE 2.1 – Exemple de partitionnement d'isolation [3]

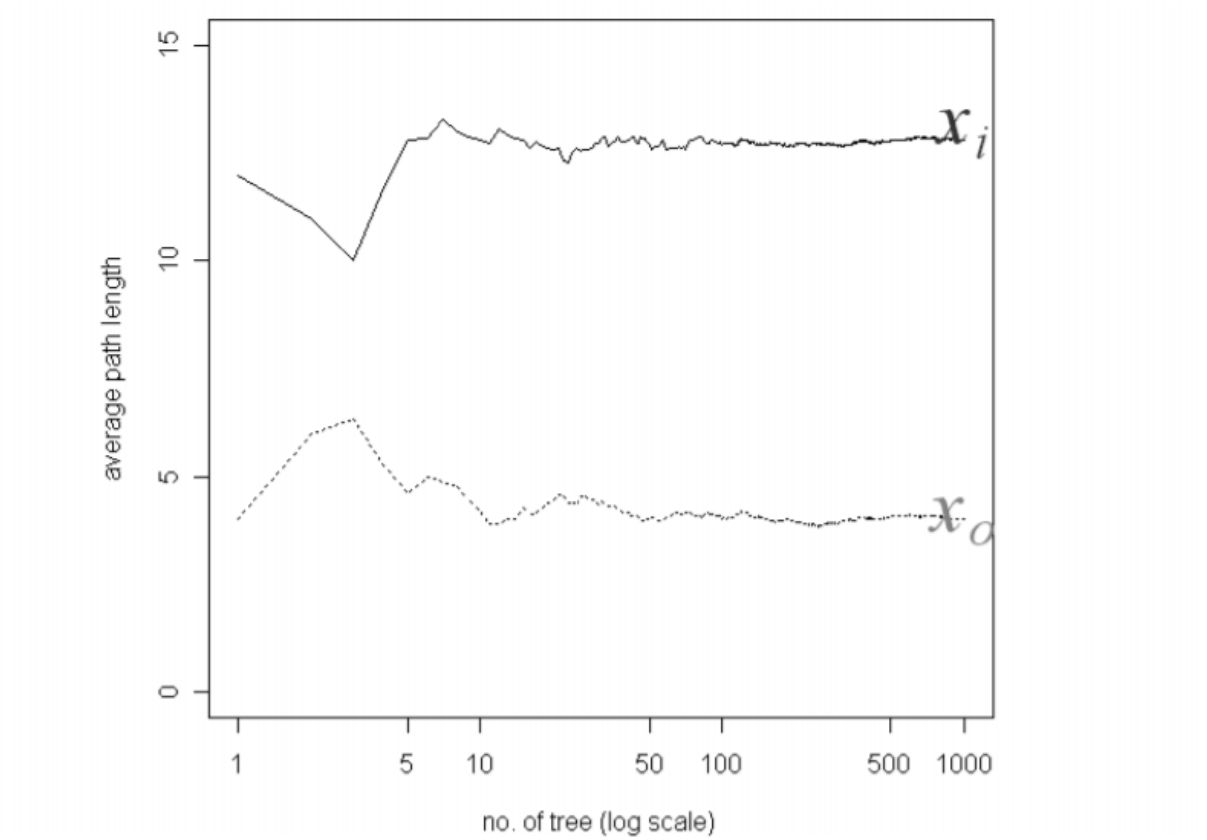


FIGURE 2.2 – Exemple de chemin moyen emprunté en fonction du nombre d'arbres [3]

Les figures 2.1 et 2.2 représentent une base de données contenant des points qui suivent une distribution Gaussienne contenant quelques anomalies. Le x_0 dans ces figures un cas d'anomalie.

1.3 Algorithme

On décrit dans cette section l'implémentation de l'algorithme IForest. Cet algorithme est divisé en deux phases. La première phase d'entraînement est responsable de la construction des arbres d'isolation. La deuxième phase, phase d'évaluation, calcule les tailles des chemins moyens de l'instance et calcule un score d'évaluation qui détermine quelles sont les anomalies.

1.3.1 Phase d'entraînement

L'algorithme IForest réalise un échantillonnage de la base données. L'échantillonnage d'une base de données consiste à un générer échantillon. C'est à dire, créer un ensemble

de taille "ψ" d'instances à partir d'un tirage avec remise de la base données.
On trouve expérimentalement que le taille $\imath\psi\jmath = 256$ est généralement suffisante pour réaliser une détection d'anomalie. Ainsi nous choisissons cette valeur comme valeur par défaut.
Ensuite il génère d'après "t" arbre binaire selon le mécanisme d'isolation indique comme la figure 2.3. Il choisit au niveau de chaque nœud un attribut aléatoire et un seuil aléatoire entre le max et le min de l'attribut dans l'ensemble du nœud pour splitter
Si nous varie le nombre d'arbre d'isolation, nous trouve expérimentalement que le chemin moyen d'une instance dans l'IForest converge généralement pour $t = 100$. Nous utiliserons cette valeur comme valeur par défaut.

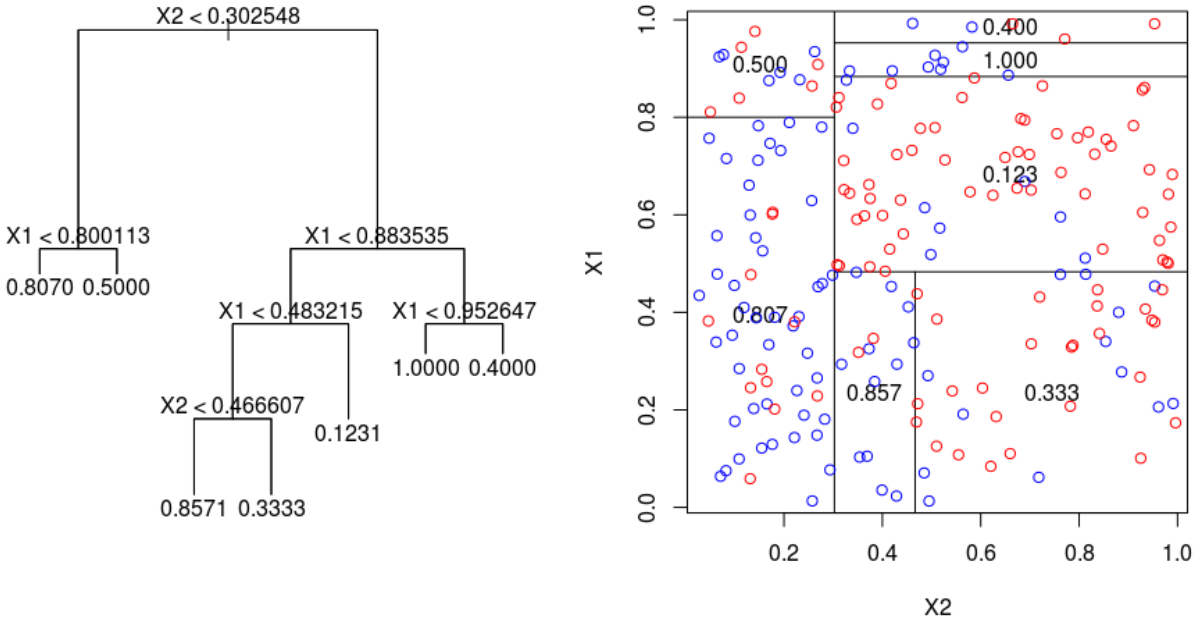


FIGURE 2.3 – Exemple d'arbre d'isolation généré à partir d'instances qui suivent une distribution normale [4]

1.3.2 Phase d'évaluation

Après avoir générer tous les arbres, l'IForest entré dans la phase d'évaluation. Il essaye de calculer une moyenne de la taille du chemin nécessaire pour atteindre la feuille où appartient l'instance.
L'Algorithme détermine d'abord au niveau de chaque arbre le nœud final où l'instance appartient et ensuite calcule la taille du chemin de ce nœud $h(x)$

Lorsque nous atteignions un seuil prédéterminé h_lim la formule de $h(x)$ devient :

$$h(x) \ = \ h(x) \ + \ c(\psi)$$

On ajoute à la valeur $h(x)$ un ajustement $c(Size)$. Cet ajustement correspond à une taille moyenne du chemin de l'instance x dans un sous-arbre aléatoire qui peut être construit au-delà de la limite de hauteur de l'arbre.
Le calcule de cet ajustement se fait à partir de la taille d'échantillon utilisé pour générer l'arbre tel que :

$$c(\psi) = \begin{cases} \dfrac{2H(\psi-1) - 2(\psi-1)}{\psi} & \text{si } \psi > 2 \\ 1 & \text{si } \psi = 2 \\ 0 & \text{sinon.} \end{cases}$$

avec $H(i)$ le nombre harmonique de i. cette valeur peut être estimé par

$$H(i) = \ln i + \gamma$$

Tel que γ représente la constante d'Euler-Mascheroni [8]

$$\gamma = \lim_{n\to\infty}(\sum_{k=1}^{n}\frac{1}{|k|} - \ln(n)) = 0.5772156649$$

Lorsque toute la taille du chemin pour chaque arbre sont calculées nous entamons le calcul de l'anomalie score. L'anomalie pour une instance x est donnée par :

$$s(x,\psi) = 2^{-\dfrac{E(h(x))}{C(\psi)}}$$

avec $E(h(x))$: la moyenne des tailles de chemins des arbres d'isolation.
On remarque qu'il existe trois cas qui résulte dans une valeur spéciale anomalie score :
(a) Si $E(h(x)) \to 0$ alors $s(x,\psi) \to 1$
(b) Si $E(h(x)) \to \psi - 1$ alors $s(x,\psi) \to 0$
(c) Si $E(h(x)) \to C(\psi)$ alors $s(x,\psi) \to 0.5$
Donc plus la valeur de $s(x,\psi)$ est proche de 1 plus sa moyenne $h(x)$ est petite c'est à dire il est probable que cette instance soit une anomalie et inversement. Plus la $s(x,\psi)$ tend

vers 0 plus il est probable que l'instance x soit une instance normale.

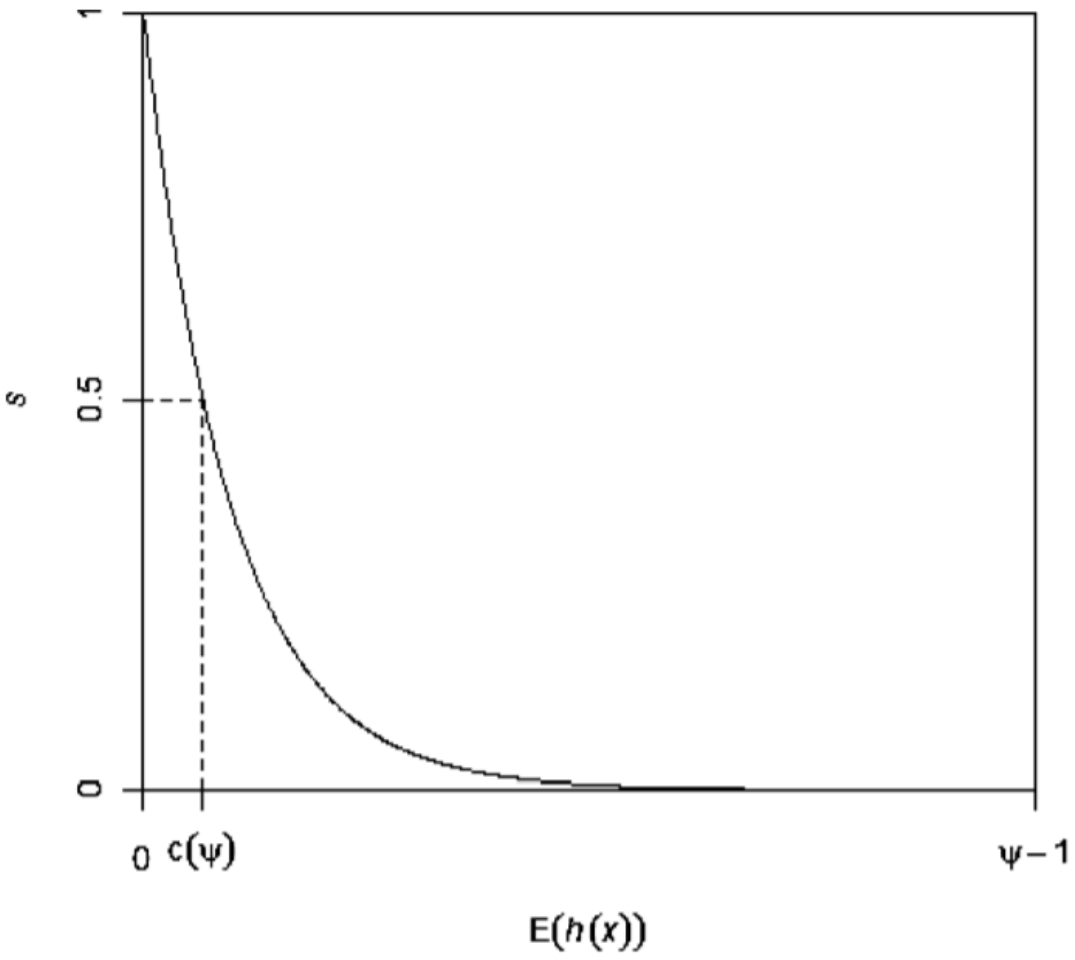


FIGURE 2.4 – Relation entre l'anomalie score s et la moyenne des tailles des chemins des arbres $h(x)$ [3]

Nous constatons d'après la figure que l'anomalie score s est inversement proportionnels à la moyenne des $h(x)$. Cela est conforme au principe illustré de l'algorithme isolation Forest.

2 Problématique

Afin de créer un algorithme modulable et personnalisable nous avons opté pour la création de notre propre algorithme IForest.
Afin de vérifier le bon fonctionnement de notre algorithme. Nous avons comparé à des implémentations préexistantes dans les librairies Scikit-learn, H2o.
Scikit-learn est un module Python qui intègre une large variété d'algorithme d'apprentis-

sage automatique. Connue pour sa fiabilité en machine Learning scikit-learn ou Sk-learn possède une large communité. Cette librairie est considérée comme une référence pour l'implémentation des algorithmes de Machine Learning.
H2o est une plateforme open-source de machine Learning. Elle est dédiée à faciliter l'accès des utilisateurs aux outils d'intelligence artificielle. Elle propose une implémentation de l'algorithme Isolation Forest.
Afin de comparer note implémentation aux implémentations de ces librairies. On doit quelles sont les métriques utilisées.

3 Métrique

Afin d'évaluer la performance d'un algorithme dans une base de données nous avons choisi 4 métriques d'évaluation.
(a) Accuracy
(b) F-score
(c) AUC ROC
Dans une classification binaire où nous cherchons à distinguer si l'instance appartient ou non à une classe. Il existe 4 cas possibles de prédictions :

	Instance appartient à la classe	**Instance n'appartient pas à la classe**
Prédit correctement	TP : true positive	TN : true negative
Prédit incorrectement	FP : false positive	FN : false negative

Tableau 2.1 – Confusion matrix

(a) Accuracy : pourcentage d'instance prédit correctement dans un ensemble

$$Accuracy = \frac{TP}{TP + TN + FN + FP}$$

(b) F-score : afin de définir cette mesure on doit parler de précision et rappel La précision représente le pourcentage de prédictions positives correctes parmi les toutes les

prédictions positives

$$prcision = \frac{TP}{TP + FP}$$

Le rappel représente le pourcentage des instances qui appartiennent à la classe des prédictions correctes

$$rappel = \frac{TP}{TP + FN}$$

F-score est une mesure engendrée de la précision et le rappel :

$$F - score = 2 * \frac{prcision * rappel}{prcision + rappel}$$

Un F-score varie entre 1 et 0 . Si le F-score vaut 1 alors mon modèle a prédit correctement toutes les instances d'une manière correct. Cependant l'Accuracy ou le F-score en lui seul n'est pas une mesure qui permet de comparer la performance des modèles. En effet ces deux métriques dépendent de la valeur du seuil utiliser pour classifier l'instance d'après son score et de l'état de la base de données c'est-à-dire si la base est déséquilibrée de point de vue nombre d'instance positive négative alors le F-score sera proche de zéro. Ces métriques restent malgré ces défauts nécessaires dans notre évaluation puisqu'il donne une valeur concrète sur la performance de notre modèle dans la base de données.

(c) AUC ROC : la courbe ROC représente la courbe associée à la variation du TPR par le FPR tel que :

$$TPR = Rappel = \frac{TP}{TP + FN}$$
$$FPR = \frac{FP}{FP + TN}$$

La AUC de la courbe ROC est la surface de l'espace limité par la courbe. Elle représente la probabilité qu'une instance qui appartient à notre classe possédera un score moins que celle d'une instance qui n'appartient pas. Si la valeur du AUC est égal à 0.5, alors notre modèle n'a pas de moyen de distinction entre l'appartenance ou non à notre classe. Son choix est aléatoire. Donc l'AUC ROC permet d'évaluer le potentiel du modèle à prédire. Il est clair que l' AUC ROC n'est pas susceptible de la valeur du seuil de distinction

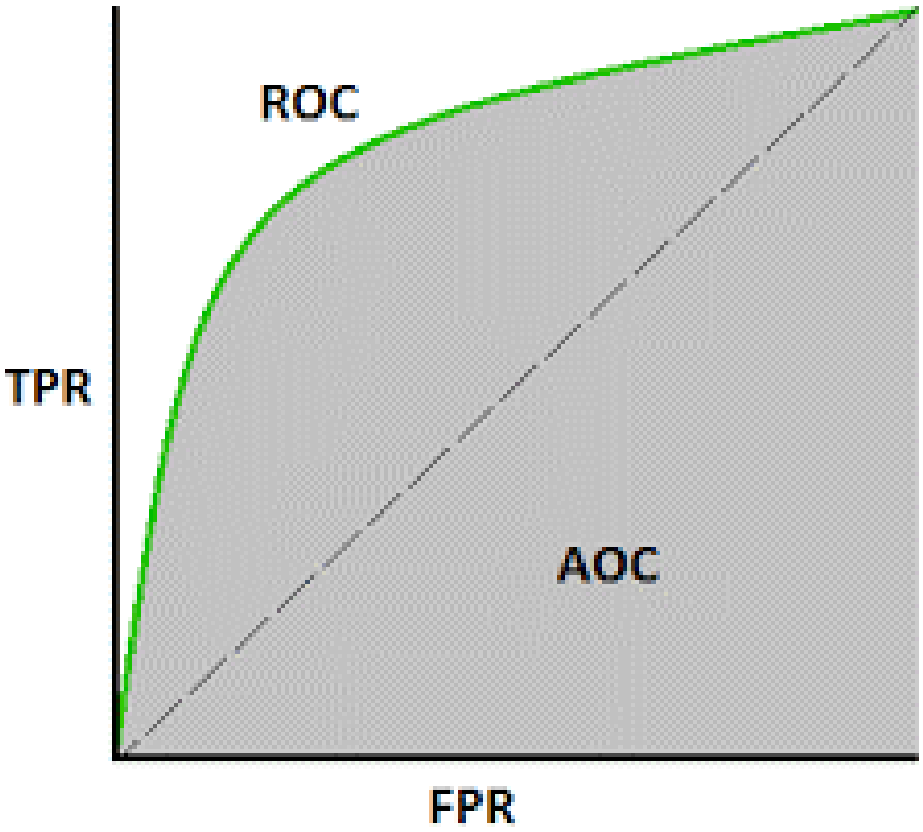


FIGURE 2.5 – Courbe AUC - ROC [5]

4 Comparaison des implémentations

Afin d'évaluer notre implémentation d'IForest nous avons choisi de travailler avec 3 bases de données :

(a) Mulcross : représente une base de données synthétiques générés à partir d'une distribution normale avec un taux de contamination égal à 10(b) Credit Cards Transactions : un ensemble transaction réalisées par des carte de crédit bancaires

(c) Health Care Heart Disease : un ensemble de caractéristiques du cœur. Les anomalies dans cette base de données correspondent au cœur susceptible à une crise cardiaque

	Accuracy	**F-score**	**AUC ROC**
Scikit-learn	0.855587	0.580694	0.949357
H2O	0.892094	0.015317	0.929219
Implémentation personnalisée	0.890701	0.242371	0.947159

TABLEAU 2.2 – Table des métriques pour la base de données Mulcross

	Accuracy	**F-score**	**AUC ROC**
Scikit-learn	0.590141	0.119089	0.676555
H2O	0.890609	0.085179	0.533987
Implémentation personnalisée	0.889387	0.225392	0.649005

TABLEAU 2.3 – Table des métriques pour la base de données Health Care Heart disease

	Accuracy	**F-score**	**AUC ROC**
Scikit-learn	0.964604	0.074373	0.950720
H2O	0.997001	0.225045	0.934328
Implémentation personnalisée	0.988490	0.157326	0.947561

TABLEAU 2.4 – Table des métriques pour la base de données Credit Card transactions

On remarque que pour toutes les bases de données la performance de notre implémentation est conforme à celle de Scikit-learn et H2O au niveau du AUC ROC on note une marge de différence de 0.01 pour le AUC et 0.02 pour l'Accuracy.
On remarque que le F-score pour tous les algorithmes dans toutes les base de données est très proche de 1. Cela s'explique par le déséquilibre de la base. En effet comme nous avons indiquée précédemment le nombre d'anomalie est très petit par rapport à celui des instances normales.

Conclusion

Pour conclure l'algorithme IForest est un algorithme qui est fondamentalement différent des autres approches de détection d'anomalie. Il offre une bonne performance sans pour autant être très coûteux. En utilisant les principes expliquées précédemment on a réussi à créer notre propre implémentation de l'IForest. Nous pouvons maintenant entamer à le modifier sans se soucier des erreurs qui peuvent s'engendrer de l'utilisation d'une librairie extérieur au cours de nos prochaines sections.

3

Implémentation de la Logique Floue

Introduction

Dans ce chapitre, l'objet portera d'abord sur la comparaison données classiques contre données floues, par la suite, il on verra comment se fait la conversion.

1 Introduction Transition Crisp vers Flou

Une première étape à considérer pour l'arrangement de l'algorithme d'isolation Forest sur la logique floue serait, vraisemblablement, la conversion de nos données Crisp vers de données floues, la fuzzification. Cela dit, bien que les mathématiques basées sur la logique floue ont un pouvoir expressif bien plus prononcé que les mathématiques usuelles (Crisp), leur puissance dépend énormément de notre aptitude à construire les fonctions d'appartenances adéquates au contexte spécifique de l'application.

Le problème, cependant, de la construction de fonctions d'appartenances pertinentes est un problème difficile et plusieurs méthodes peuvent être employées à cet égard notam-

ment des méthodes directes avec un ou plusieurs experts [book ref chap 10-], la méthode du Least Square Curve Fitting, et même en utilisant des réseaux de neurones [book ref].

Ces méthodes sont malheureusement soit trop compliquées soit dépendantes de l'apport d'un expert, et sortent du cadre de ce projet de fin d'année. Ce que nous pouvons faire cependant est de donner la main à l'expert de choisir ses fonctions d'appartenances, parmi un set des fonctions les plus utilisées pour les données à convertir.

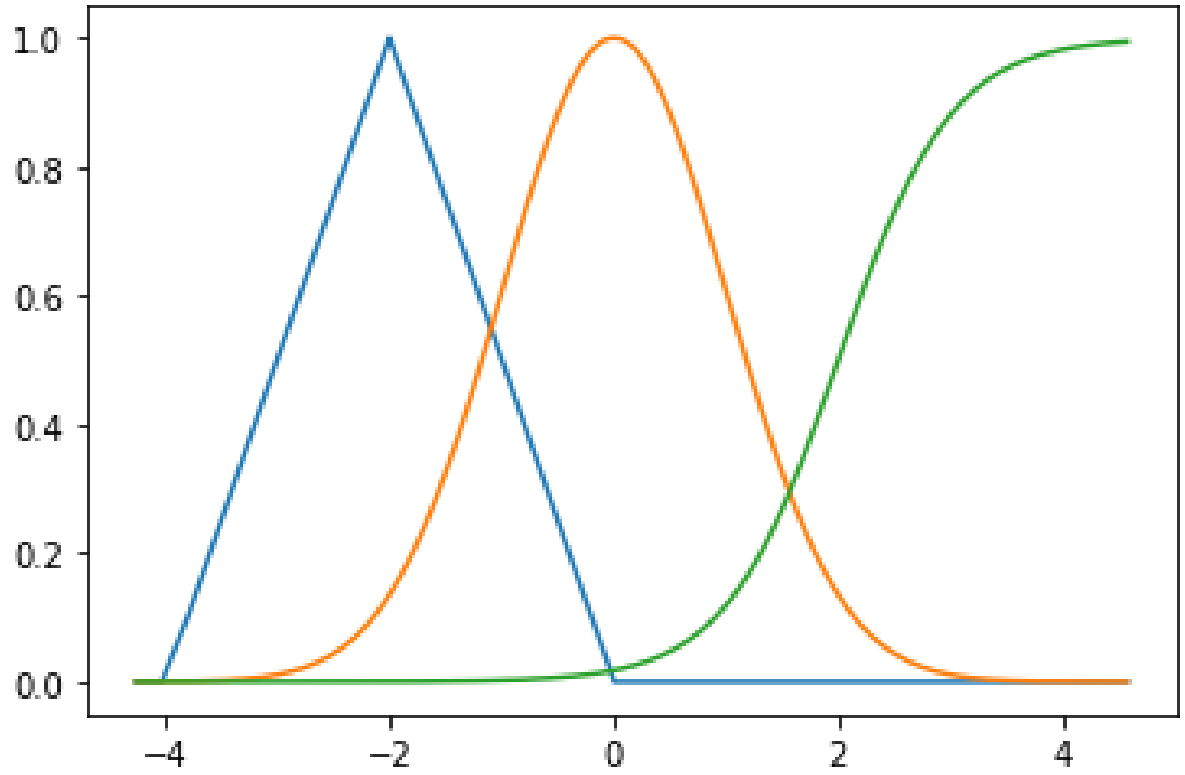


FIGURE 3.1 – Exemples de fonctions d'appartenances de différentes formes

Les fonctions d'appartenances les plus communes :

(a) Triangulaires : Ces formes de fonctions d'appartenances sont les plus communes, elles sont définies par une valeur basse a, une valeur moyenne m où l'appartenance à la modalité est égale à 1 (aussi appelée valeur modale) et une valeur haute b. On notera :

$$\mu_A(x) = \begin{cases} 0, & \text{si } x \leq a \text{ ou } x \geq b \\ \frac{x-a}{m-a}, & \text{si } a < x \leq m \\ \frac{b-x}{b-m}, & \text{si m<x<b.} \end{cases}$$

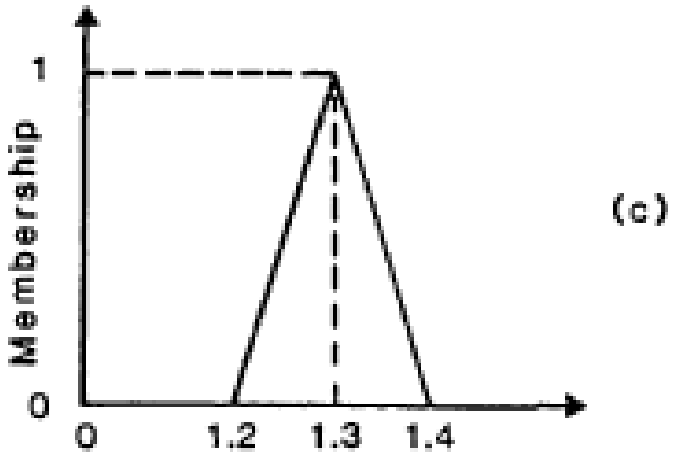


FIGURE 3.2 – Exemple de fonction d'appartenance triangulaire

(b) Trapézoïdales : ces fonctions d'appartenances sont définies avec quatre valeurs a,b,c,d tel qui suit :

$$\mu_T(x) = \begin{cases} 0, & \text{si } x \leq a \text{ ou } x \geq d \\ \frac{x-a}{b-a}, & \text{si } a < x < b \\ 1, & \text{si } b \leq x \leq c \\ \frac{d-x}{d-c}, & \text{si c < x < d.} \end{cases}$$

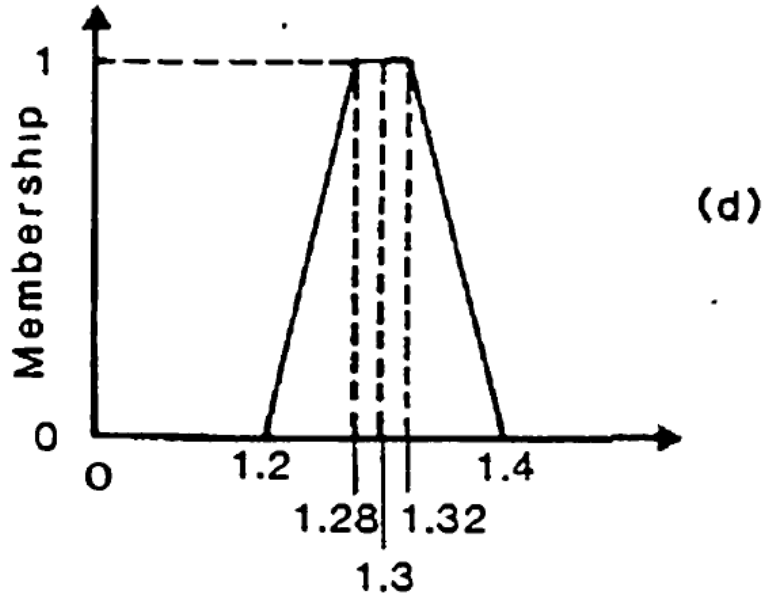


FIGURE 3.3 – Exemple de fonction d'appartenance trapézoïdale

(c) Gaussiennes : définies avec deux paramètres m et $k > 0$, elle ne prend la valeur 1 que pour la modale. Sa formule associée est :

$$\mu_G(x) = e^(- k(x-m)^2)$$

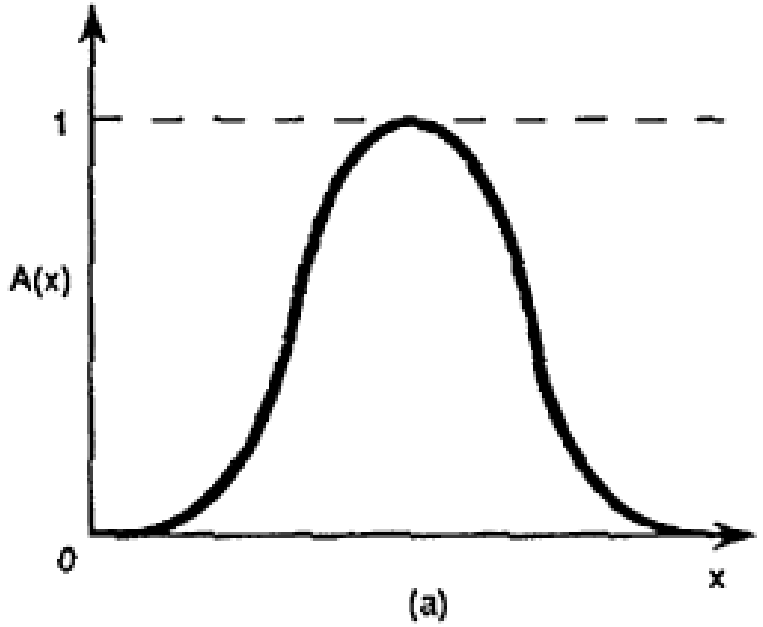


FIGURE 3.4 – Exemple de fonction d'appartenance gaussienne

2 Besoins Fonctionnels du Module de Fuzzification

Nous nous penchons donc vers la conception d'un module de fuzzification qui se préoccupera de convertir les données Crisp en Floues selon les fonctions d'appartenances vues en haut.

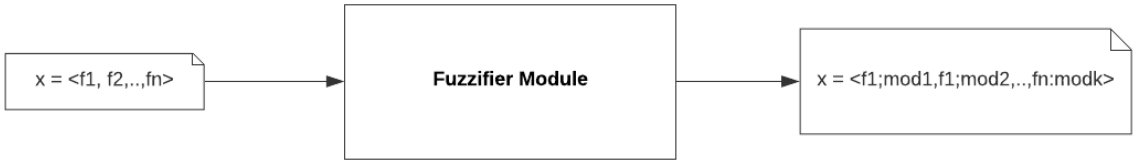


FIGURE 3.5 – Entrée / Sortie du module de fuzzification

avec mod_k correspond à la modalité relative de f_k.

	age
0	15
1	30
2	50

	age;dismal	age;poor	age;mediocre	age;average	age;decent	age;good	age;excellent
0	1.0	0.0	0.000000	0.000000	0.0	0.0	0.0
1	0.0	0.0	0.428571	0.571429	0.0	0.0	0.0
2	0.0	0.0	0.000000	0.000000	0.0	0.0	1.0

FIGURE 3.6 – Exemple de fuzzification

Il existe déjà sur le marché plusieurs outils logiciels qui s'occupent de la logique floue et du raisonnement floue. Nous en avons distingué :

- FuzzyLite : Collection de librairies c++ avec un grand nombre de fonctionnalités et flexibilité

- PyFuzzy : Librairie Python 2

- Scikit-fuzzy : Un api à but général offrant des classes et méthodes supportant la définition des systèmes fuzzy

Nous nous sommes aussi procuré un module de fuzzification en Java développé en interne dans le cadre d'une autre recherche issue à l'Institut Supérieur de l'Informatique.

Il nous revient donc à choisir une de ces librairies et utiliser ses fonctionnalités de base pour créer un module de fuzzification qui prendrait en entrée une dataset crisp et la fuzzifiera.

3 Comparaison des modules populaires

Pour l'implémentation du module de fuzzification, et parce que certaines solutions existent mais ne répondent pas tout à fait à notre problème, nous avons décidé d'utiliser quelques fonctionnalités déjà créées et de les intégrer dans notre code.

Prenant considération que python est devenu le langage de programmation le plus adapté à l'apprentissage automatique, nous nous sommes penchés vers une solution qui sera basée sur ce langage et plus précisément au stack SciPy (NumPy, SciPy,Pandas et matplotlib)

Nous dressons alors la table ci-dessous pour comparer les différentes librairies aux quelles nous avions accès.

	Comptabilité	**Fonctionnalités**	**Documentation**
Sci-kit Fuzzy	Parfaitement compatible avec le SciPy Stack	API générale et assez riche, propose des méthodes pour la création de variables floues.	Relativement documenté
Module JAVA	Codé en java, un moyen de communication (import / export csv) est nécessaire	Très limitées, propose uniquement des fonctions d'appartenance triangulaire	Mal documenté
Simpful	Codé en python 3, donc assez compatible	Assez complet, se focalise sur les systèmes d'inférence ment plus que la génération de variables floue.	Aucune documentation

TABLEAU 3.1 – Comparaison des module existants de fuzzification

=> Le choix de la librairie à utiliser est donc **Sci-kit** fuzzy pour la généralité de son API, sa facilité d'utilisation et sa meilleure documentation des trois.

4 Réalisation : Implémentation du module de fuzzification

Le module a été réalisé et nous présentons ci dessous le diagramme de classe de sa conception

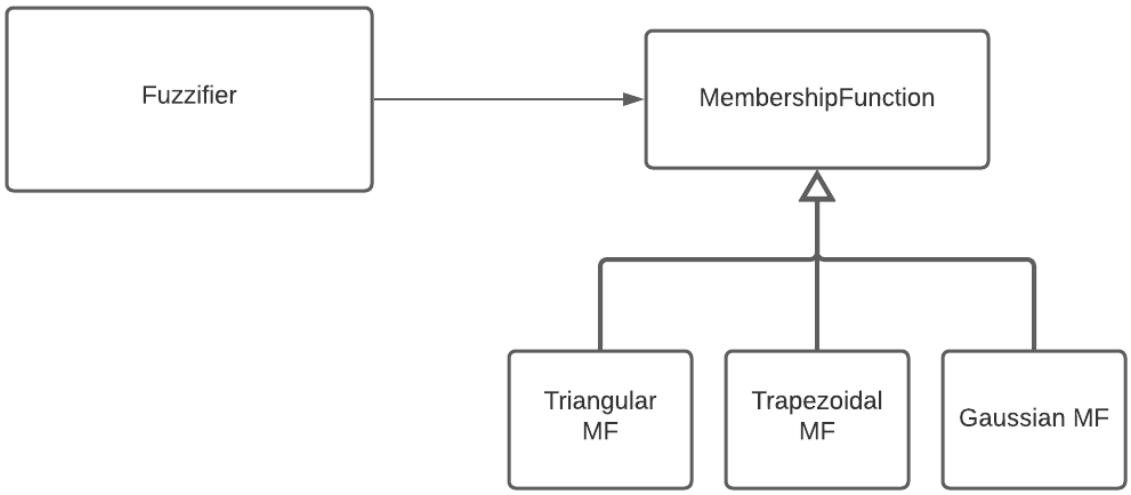


FIGURE 3.7 – structure du module de fuzzification

Le module Fuzzifier se charge donc d'itérer à travers les variables (features) de la datasets crisp donnée en entrée et de soit créer les modalités selon les exigences de l'utilisateur (l'expert domaine) soit de générer automatiquement les modalités de la variable et leurs fonction d'appartenance.

-> Modalités créées par l'utilisateur : L'utilisateur a la possibilité de créer autant de modalités qu'il veut par variable et pour chaque modalité qu'il nommera, il pourra choisir de la définir avec l'une des fonctions d'appartenances les plus communes vues précédemment.

-> Fonction d'appartenance automatique : Il a été décidé que la fonction d'appartenance auto serait la fonction triangulaire, et le nombre de modalité par défaut est de 3, comme dans la figure ci-dessous.

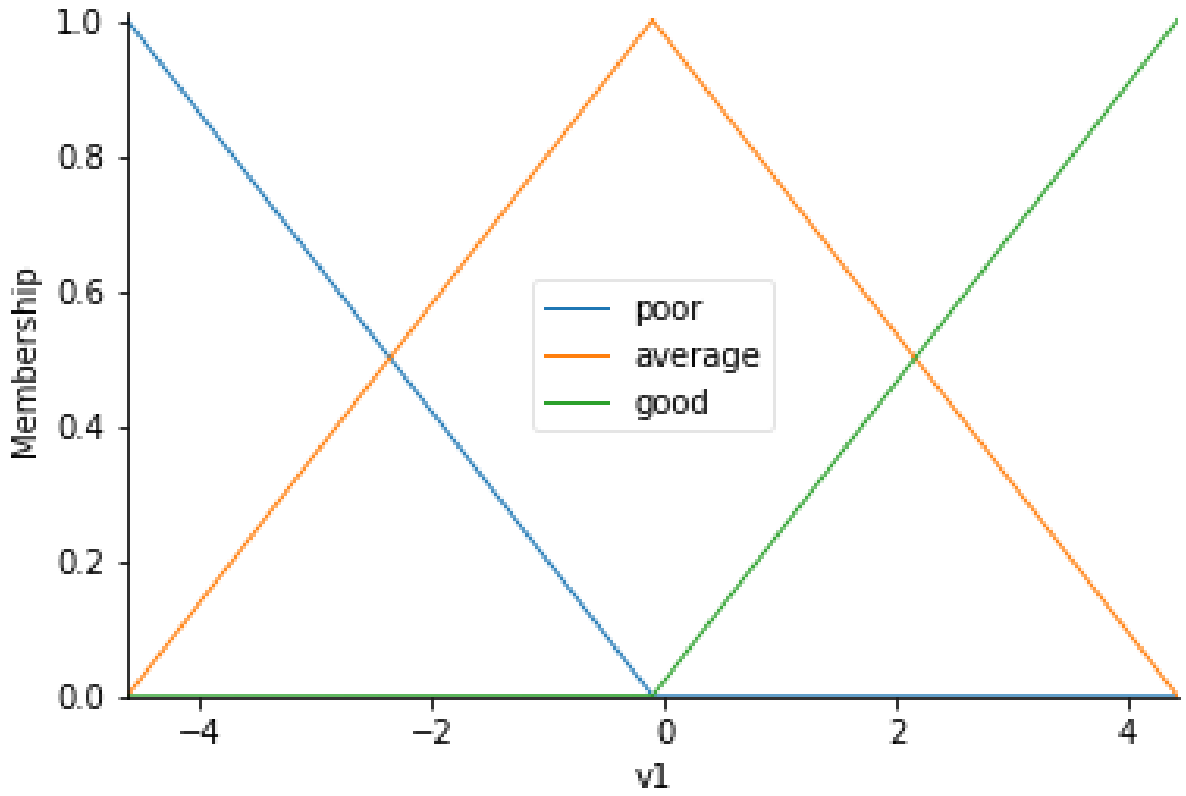


FIGURE 3.8 – Les modalité et leurs fonction d'appartenances par défaut

Comme cité précédemment, la pertinence d'une fonction d'appartenance ne peut être déterminée que dans le contexte d'une application bien spécifique. Cependant, il est à noter que plusieurs applications ne sont pas très sensibles aux variations dans la forme [fuzzy sets and fuzzy logic theory and application, George J.Klir, 1995]. Il est alors plus adéquat d'utiliser la forme triangulaire pour sa simplicité. Ces fonctions (poor, average et good dans cet exemple) sont calculées se basant sur les valeurs max, min et moyenne des données crisp en entrée.

-> Interpolation des valeurs nouvelles Le module se charge évidemment aussi de fuzzifier de nouvelles entrées qu'il n'a pas rencontrer dans le set de fuzzification. Il réalise cela avec une méthode d'interpolation entre les deux points les plus proches au nouveau point.

-> Timing de la fuzzification de la dataset Il est à noter que, dans les implémentations existantes d'algorithmes de décision à base d'arbres sur la logique floue (random forest et les classifiers floues), les approches diffèrent sur le moment de la fuzzification des données. Nous pouvons donc distinguer :

- une fuzzification de toute la base de données avant de procéder à l'échantillonnage des données floues et la création des arbres de l'ensemble

- une fuzzification de l'échantillon crisp de la base de données à la création de chaque arbre durant l'algorithme d'apprentissage.

Fuzzification de la dataset en entier	**Fuzzification séparée des échantillons**
+ Plus de précision durant l'interpolation	- Plus grande probabilité d'avoir de nouvelles valeurs en dehors du domaine de l'interpolation pour une fonction d'appartenance 'auto' (max de l'échantillon < max global)
+ Capacité à évaluer les temps d'apprentissage et les temps de fuzzification séparément	- Couplage de l'apprentissage à la fuzzification
- Plus lent	+ Nettement plus rapide, car ne gère que quelques échantillons, ne parcourt pas forcément toute la dataset

TABLEAU 3.2 – Comparaison des deux timings de la fuzzification de la dataset

Nous opterons durant cette étude pour la première méthode pour notre priorisation de la précision des résultats aux performances de l'algorithme.

Conclusion

Pour conclure, nous nous sommes chargés dans le cadre de ce projet à créer un module de fuzzification pour ingérer directement sa sortie à l'implémentation de l'algorithme de Isolation Forest sur les données floues.

4

Réalisation de la Solution

Introduction

Nous avons présenté et défini durant les chapitres précédents les concepts de base de détection d'anomalie par l'isolation et de la logique floue, mais aussi de la conversion du monde des nombre crisp au nombres floues. Nous nous focalisons désormais, finalement, sur le vif du sujet qui est l'adaptation des algorithmes Isolation Forest à la logique floue. Mais avant, un dernier pas théorique.

1 Etude Theorique

1.1 Formalisme de l'algorithme de construction d'arbre de décision floues

Un algorithme de construction des arbres de décision se formalise à l'aide d'une fonction telle que [6] :

$$\phi : (H, P, T, C, E) \rightarrow D$$

avec :

-> H est une mesure de Discrimination (nous permet de choisir l'attribut sur lequel on split, e.g. L'entropie de Shannon)

-> P une stratégie de partitionnement (suivant l'attribut déterminé grâce a H, comment partitionner un ensemble, eg. prendre une valeur aléatoire et diviser selon >, < ; mais plus couramment décomposer en sous bases chacune induite par une des modalités[*] correspondant à l'attribut)

-> T est un critère d'arrêt, T est souvent lié à la mesure de discrimination H (eg. Information = 0 => même classe => plus besoin d'avancer) mais aussi la profondeur de l'arbre etc., encore des heuristiques et variables à considérer dans l'implémentation

-> C la classe ; se départage des autres attributs par sa valeur sémantique.

-> E : ensemble d'exemple / entité / lignes dans la dataset etc.

-> D => l'arbre de décision

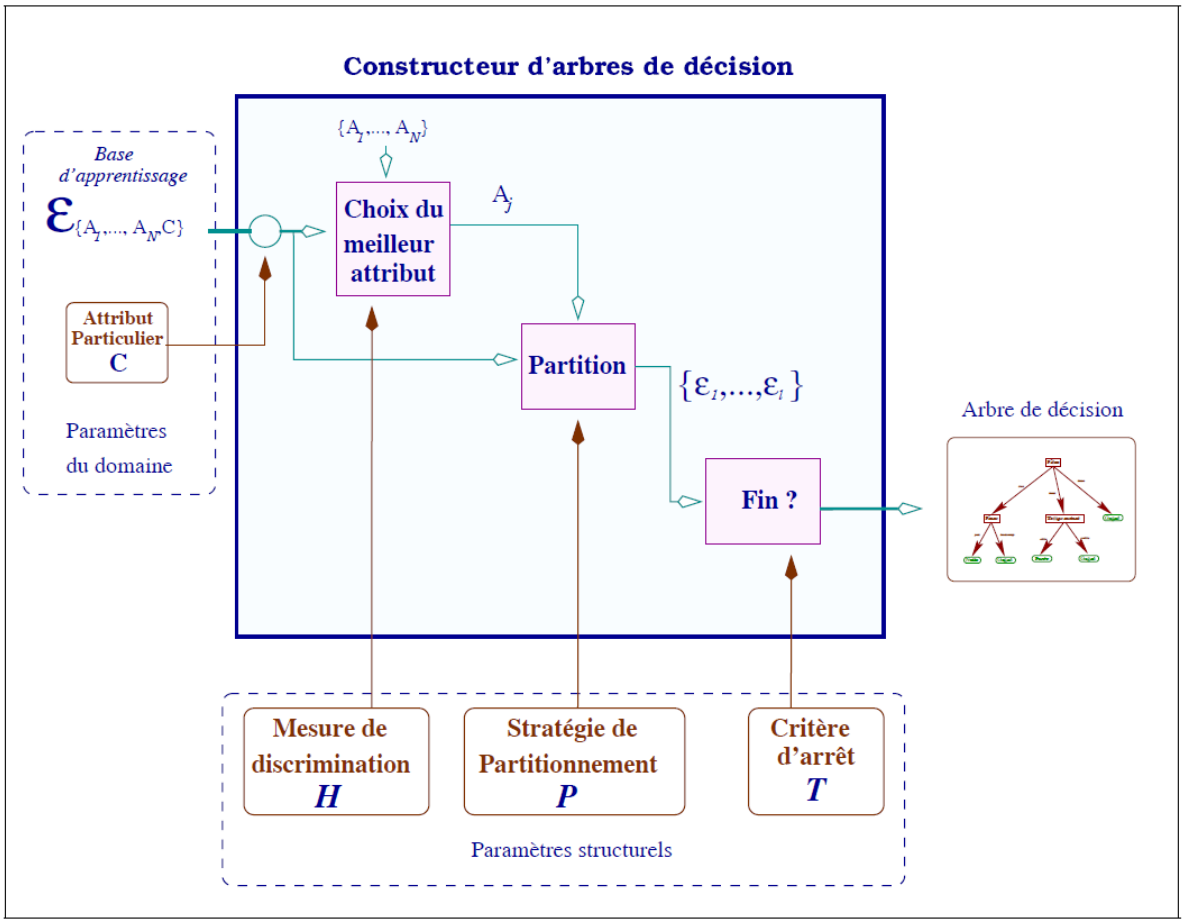


FIGURE 4.1 – Architecture générale d'un système de construction d'arbres [6]

Notons ici que nous parlons de modalité car les arbres de décision fonctionnent mieux sur des attributs symboliques et non numériques, mais ces derniers peuvent être amenés à des modalités

Notons aussi ici que dans le contexte d'arbre flous, l'appartenance aux modalités est graduelle, et le partitionnement même d'un sous-ensemble floue est un problème en soi, qu'on abordera dans le paragraphe suivant.

Il nous est utile d'étudier ce formalisme, car il nous permet de nous rendre compte des éléments d'études de la théorie des arbres de décisions, et ainsi de mieux étudier les choix à faire lors de l'implémentation des algorithmes. Ce formalisme nous aide aussi à extraire les points communs des algorithmes de construction d'arbre de décision entre la logique floue et classique, nous permettant de nous inspirer au maximum des implémentation déjà faites sur la logique classique, notamment L'isolation forest de scikit-learn comme vu dans le chapitre 2.

1.2 Partitionnement d'un sous-ensemble flou

Nous rappelons que le principe fondamentale des sous-ensembles floues, est qu'une entité x de l'univers de la variable n'est plus évaluée à si elle appartient à un ensemble ou non, mais plutôt au degré auquel elle appartient au dit ensemble.
Cela en découle alors directement la problématique de partitionner les sous-ensembles flous.

"Une stratégie de partitionnement P est donc de diviser E en utilisant la partition floue, certains exemples pouvant alors appartenir à plusieurs sous-bases avec des degrés d'appartenances différentes" - Marsala [6], Apprentissage inductif en présence de données imprécises.

Il existe plusieurs stratégies de partitionnement flou, nous nous focaliserons durant cette étude sur les deux plus courantes :

-> Création de partition plus classiques (crisp) en utilisant des alpha-coupes [6]

-> Répartition des exemples dans toutes les sous-bases avec des degrés d'appartenances différentes [7]

1.2.1 Partitionnement plus simple en utilisant les alpha coupes

Suivant cette approche, nous partitionnons un ensemble d'éléments flous par des alpha-cuts sur leurs fonctions d'appartenances, suivant une modalité donnée.

Le partitionnement des nœuds ici pourrait se faire de sorte que suivant un attribut, les fils d'un nœud donné forment l'ensemble de toutes les modalités.
Par exemple considérons l'attribut Taille avec les modalité Grande – Moyenne – Petite avec L'alpha-cut = 0.5

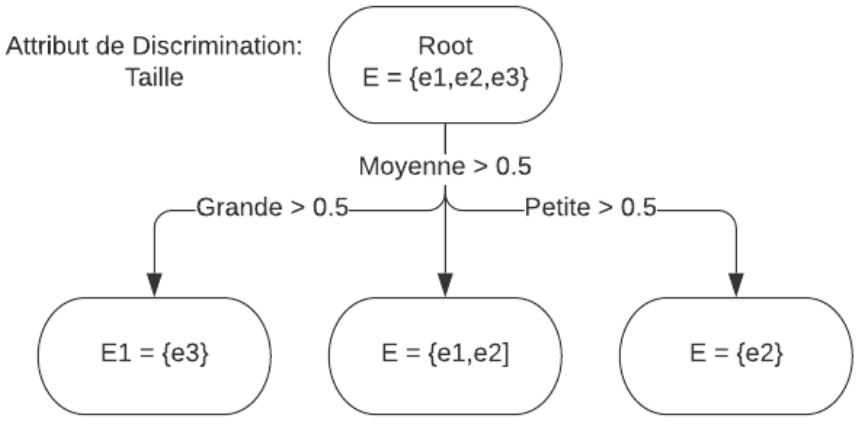


FIGURE 4.2 – Partitionnement alpha cut sur toutes les modalité d'un attribut

L'inconvénient avec cette technique est que nous perdons les propriétés de l'arbre binaire, notamment le calcul direct de son hauteur moyenne utilisé dans les Isolation Forest sur des données crisp.

Pour y remédier, nous pourrions choisir à chaque fois un Attribut mais aussi la modalité avec laquelle on partitionne l'ensemble (de façon aléatoire aussi) et puis diviser l'ensemble selon que l'appartenance à la modalité est supérieure à une valeur d'alpha-cut donnée :

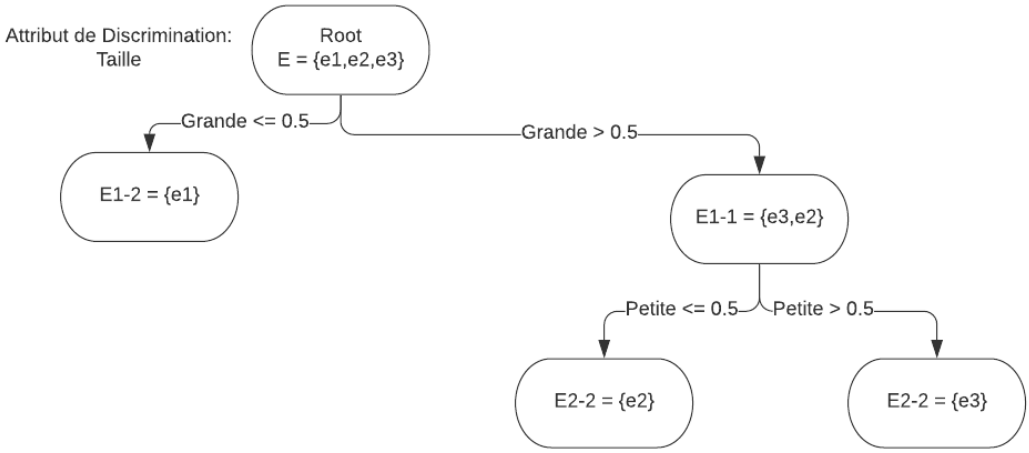


FIGURE 4.3 – Partitionnement binaire d'un sous ensemble flou avec des alpha-cuts

Un autre inconvénient à ce partitionnement est que nous perdons en partie la puissance de la logique flou, car nous nous redirigeons vers une classification classique et crisp.

1.2.2 Appartenance de tous les exemples à toutes les sous-bases

:

-> On commence avec tous les exemples ayant le même poids, $w_j = 1$

-> En chaque nœud, on calcule l'appartenance d'un exemple au NOEUD (non à la modalité relative au nœud, suivant l'attribut de division) ;

-> Cette dernière est calculée grâce à l'appartenance à la modalité et à l'appartenance au nœud parent ;

-> Notons la fonction qui prend l'appartenance à la modalité et au nœud parent et renvoie l'appartenance au nœud courant f_2 ;

-> Un exemple ayant une appartenance nulle à qui que ce soit, le nœud ou la modalité est considéré comme exclu du nœud.

-> On calcule le gain d'information des attributs pour choisir avec lequel splitter (ce dernier est ajusté pour les valeurs manquantes dans ce papier, mais nous pourrions omettre ce détail dans notre implémentation pour débuter)

-> On partitionne selon la variable avec le plus grand Gain (comme dans les arbres de décision classiques) et répètent la procédure.

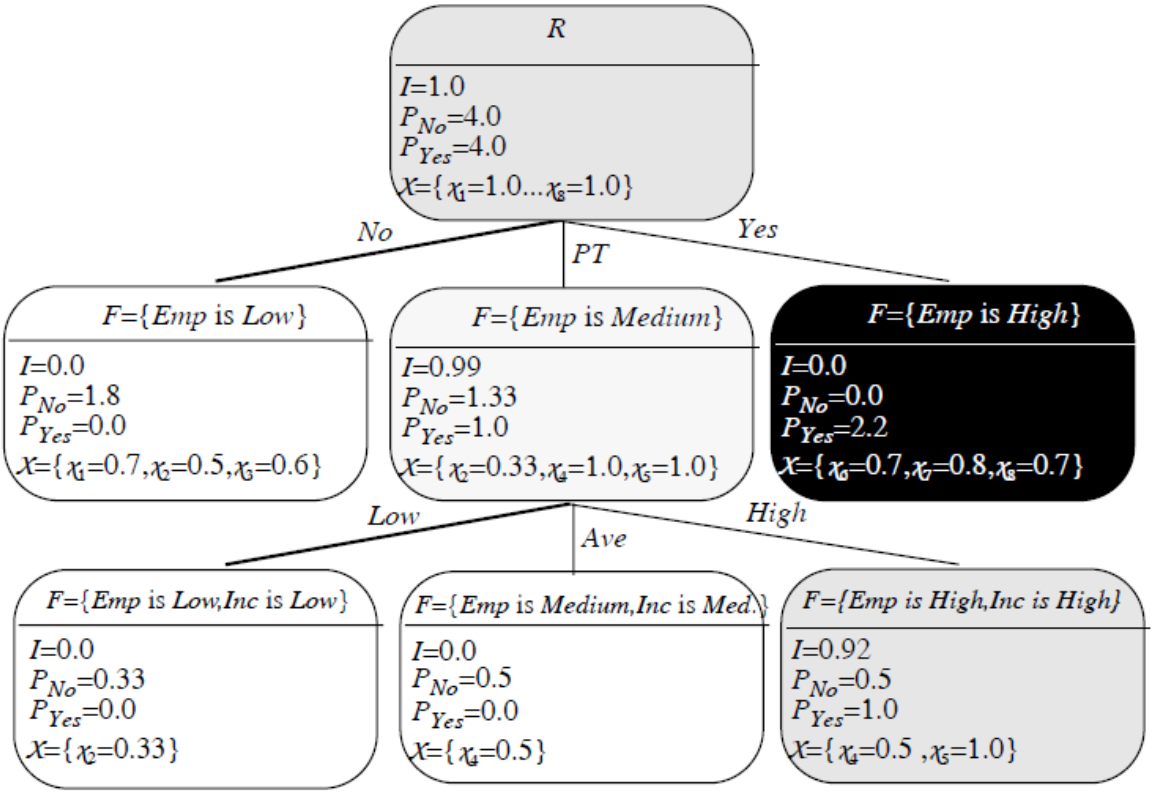


FIGURE 4.4 – Exemple de partitionnement avec la méthode Janikow [7]

Cette méthode part donc du principe où tous les éléments se propagent aux sous-ensembles, mais avec des degrés d'appartenances différents avec le cas appartenance = 0 comme cas extrême où l'élément n'appartient pas à ce sous ensemble.

Néanmoins, cette méthode aussi présente le problème où les arbres qu'elle engendre ne sont pas binaires. Or les algorithmes de Isolation forest se basent fondamentalement sur le caractère strictement binaire de la structure de l'arbre (pour la normalisation de la profondeur d'un élément dans l'arbre lors du calcul du score).
Cette méthode serait donc légèrement modifiée pour répondre à ces besoins. Ces modifications sont :
-> Pour un noeud N, on choisit l'attribut de discrimination et la modalité de discrimination, ce noeud aura donc deux fils, un pour la modalité et un deuxième "autre" -> Le noeud "autre" aura les appartenances de son nœud parent propagé directement telle quelles au fils.

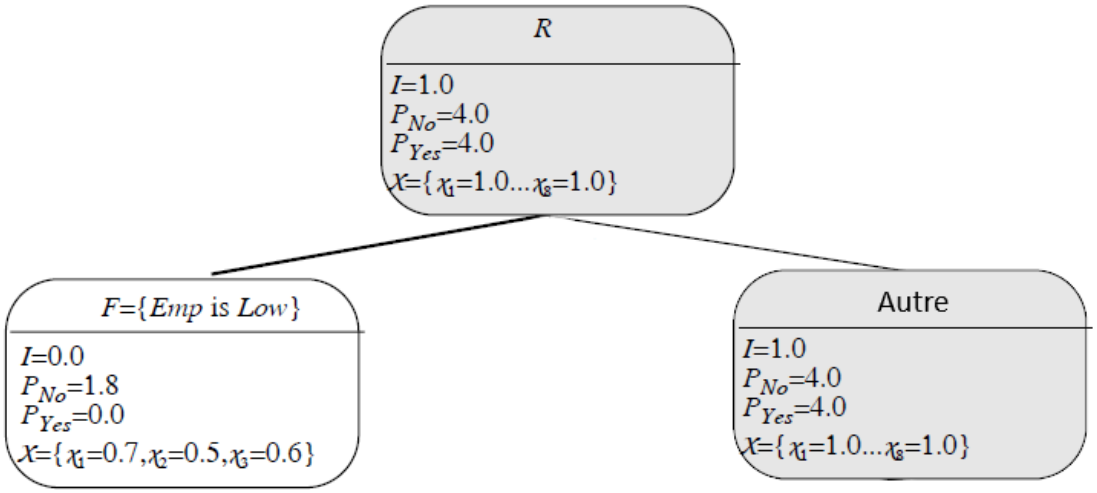


FIGURE 4.5 – Exemple de partitionnement en utilisant la méthode d'appartenance binarisée

1.2.3 Evaluation des algorithmes d'isolation forest pour les méthodes de partitionnement présentées

Grâce à la nature classique du partitionnement avec les alpha cut pour la génération d'arbre binaire comme vu précédemment, le calcul du score de prédiction d'anomalie s'avère extrêmement similaire, même identique au calcul de score vu dans le chapitre 2. On ne peut cependant en dire de même pour le calcul du score selon la deuxième méthode (Janikow, 1994) [7]. En effet, ce calcul récurrent dans chaque nœud de l'appartenance de tous les exemples à ce nœud, nous donne donc un ensemble partitionné en plusieurs sous-ensembles qui ont des propriétés intéressantes :

=> La sommes des cardinalités des sous-ensembles peut être supérieure, égale, ou même inférieure à la cardinalité de l'ensemble initial.

=> Un exemple pouvant appartenir à plusieurs sous-ensembles et avec des degrés d'appartenances différentes, nous aurons, encore, plusieurs possibilités à tester dans le cadre d'isolation forest.

On considère alors pour un x donnée, on considère tous les nœuds où la valeur d'appartenance à x est non nulle, noté N_x et $s(x, n)$ le score calculé pour un exemple x et un

nœud n appartenant à N_x.

Nous pouvons donc choisir une fonction d'agrégation sur les $sn(x)$ tel que n appartient à N_x

-> Moyennes des profondeurs : $Score(x) = Sum(ninNx)sn(x,n)/|Nx|$

-> Moyenne Pondérée des profondeurs On considère aussi $A_n(x)$, l'appartenance de x au noeud n, on peut alors calculer $score(x) = Sum(ninNx)sn(x) * An(x)/|Nx|$

-> Moyenne des profondeurs de tous le noeuds où l'appartenance est supérieur à la médiane des appartenances

-> Etc ..

2 Implémentation

2.1 Diagramme de Classe

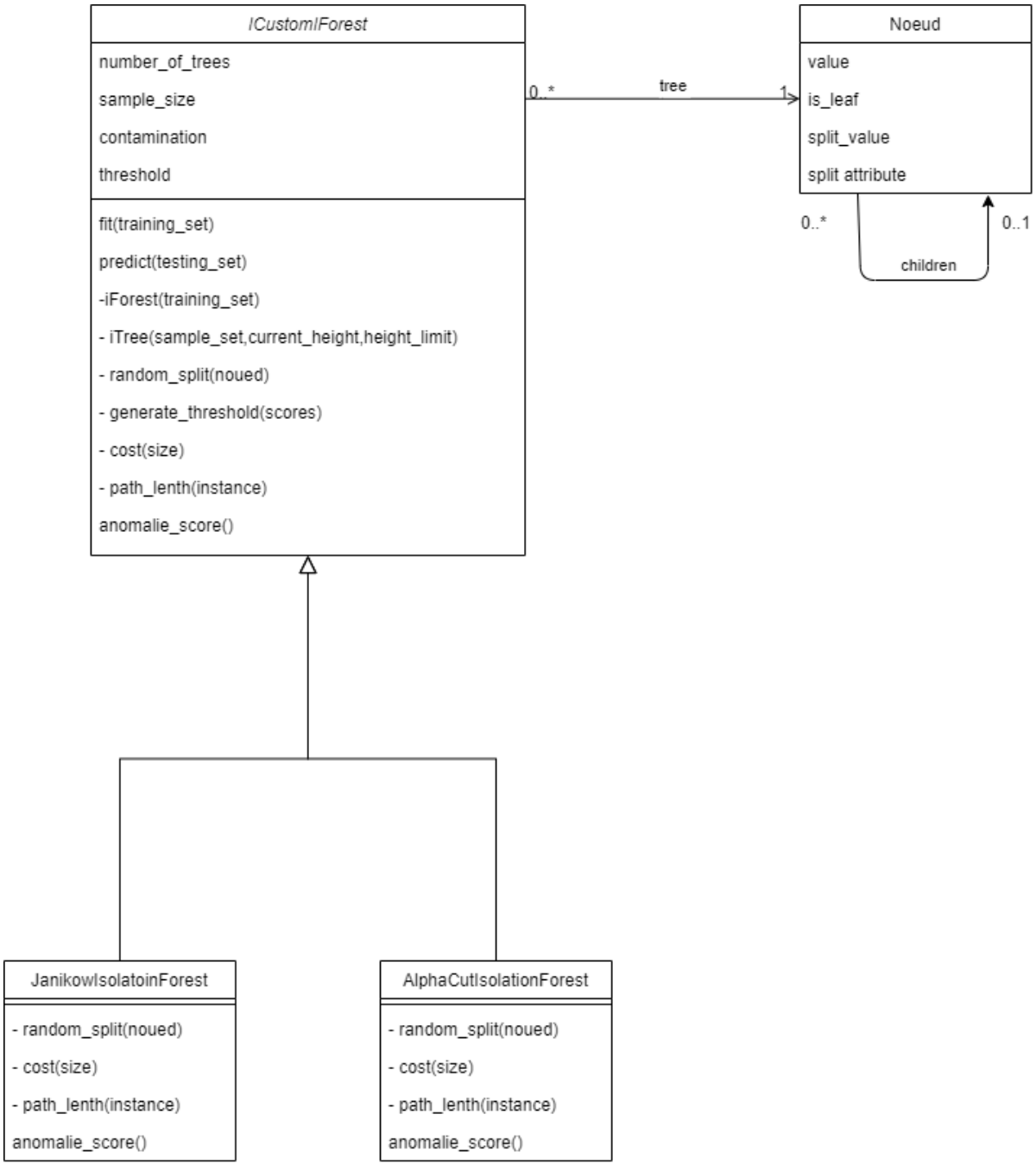


FIGURE 4.6 – Diagramme de classe Fuzzy IForest

2.2 Algorithme dans le cas abstrait

Afin de créer une version modulable et plus personnalisable de l'IForest, Nous avons choisi de découper les étapes de l'algorithme IForest en différentes méthodes suivant le Single Responsibility Principal. Nous avons créé une classe abstraite qui implémente les méthodes communes entre les différentes versions du Fuzzy Isolation Forest tel que la méthode iTree, IForest et predict.

Algorithme: iTree(X)

Inputs: train_set - input

Output: une iTree

Si train_set n'est pas divisible alors

Return nœud[is_leaf ← true, value← train_set]

Sinon Soit q un attribut aléatoire du training set

Soit p une valeur aléatoire de l'ensemble Q entre le max et le min de l'attribut q dans la base de données

Current ← Nœud[value ← train_set,split_value ← p, split_attribute← q]

Pour ensemble dans random_split(current) faire

    Ajoute iTree(ensemble) dans noeud.children

Return current

2.3 Considération Alpha cut

Etape de Split :

L'étape de split dans l'implémentation alpha-cut isolation Forest est presque identique à celle de l'IForest. Puisque les alphas-cut dans la logique floue représentes un pont entre les données crispe et les données fuzzy. Cependant en utilisant la base de données fuzzy on peut se trouver dans le cas ou tout l'ensemble d'un nœud possède la même valeur(0 par Example). Dans ce cas nous avons opté à limiter le calcule et à considérer ce nœud en tant que feuille.

Etape d'évaluation :

Puisque l'arbre d'isolation généré par l'alpha Cut isolation Forest est un arbre binaire complet donc Nous allons utiliser les mêmes étapes de calcules que l'IForest.

		Accuracy	AUC ROC
Breast Cancer	Scikit learn	0.667838	0.815760
	Custom Crisp IForest	0.643234	0.819790
	Fuzzy iForest	0.664323	0.834007
healthcare	Scikit learn	0.578122	
	Custom Crisp IForest	0.891017	
	Fuzzy iForest	0.884702	

3 Evaluation empirique

Cette section présente finalement quelques expériences utilisée pour évaluer la version Fuzzy avec alpha-cuts d' Isolation Forest

Nous commençons par comparer les scores définis au chapitre 2 de l'implémentation fuzzy avec notre propre implémentation crisp et l'implémentation de sci-kit Isolation Forest comme :

		Accuracy	AUC ROC
Breast Cancer	**Scikit learn**	0.667838	0.815760
	Custom Crisp IForest	0.643234	0.819790
	Fuzzy iForest	0.664323	**0.834007**
healthcare	**Scikit learn**	0.578122	0.6618
	Custom Crisp IForest	0.891017	0.667
	Fuzzy iForest	0.884702	**0.672**
Mulcross	**Scikit learn**	0.8787	0.9724
	Custom Crisp IForest	0.8950	0.9536
	Fuzzy iForest	**0.9168**	0.9562
Diabetes	**Scikit learn**	0.6718	0.685
	Custom Crisp IForest	0.65364	0.664
	Fuzzy iForest	0.660156	0.679
Mulcross	**Scikit learn**	0.964432	0.9465
	Custom Crisp IForest	0.901361	0.9497
	Fuzzy iForest	0.90136	**0.9523**

Conclusion

Nous remarquons que les Isolation Forest fuzzy ont une performance pratiquement identique aux algorithmes crisp, même parfois légèrement mieux. Nous pourrions hypothétiser que cela est dû à la similarité des algorithmes suite au partitionnement avec des alpha-cuts. Nous considérons ce résultat comme un bon signe et laissant grande ouverte une porte de nombreuse perspectives à suivre, car durant ces expériences, la fuzzification de toutes les datasets a été fait au mode auto et pensons que l'apport d'un expert de domaine, dans le cadre de l'application dans le monde réel de l'algorithme fuzzy, aura des résultat encore plus attrayants.

Conclusion générale et perspectives

La détection d'anomalies en se basant sur des données floues peut donc s'avérer très utile vu que ça apporte un tout nouvel angle de gestion des données.

D'autres combinaisons de solutions peuvent être imaginées dans ce même sens, tel que l'utilisation des arbres n-aires au lieu de celles binaires, utilisation d'autres algorithmes de Machine Learning, ou même combiner certaines des idées présentées lors de ce rapport.

Bibliographie

[1] George J. Klir/Bo Yuan. Fuzzy sets and fuzzy logic, theory and applications. page 12.

[2] Fuzzy logic - inference system. https://www.tutorialspoint.com/fuzzy_logic/fuzzy_logic_inference_system.htm.

[3] Zhi-Hua Zhou Fei Tony Liu, Kai Ming Ting. Isolation-based anomaly detection. *ACM Transactions on Knowledge Discovery from Data*, 6(1), 2012.

[4] Isolation forest. https://donghwa-kim.github.io/iforest.html.

[5] Understanding auc - roc curve. https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5.

[6] Christophe Marsala. Apprentissage inductif en presence de donnees imprecises : construction et utilisation d'arbres de decision flous. 1998.

[7] Cezary Z. Janikow. Fuzzy decision trees : Issues and methods. 1994.

[8] Mascheroni constant. https://en.wikipedia.org/wiki/Euler%E2%80%93Mascheroni_constant.